合肥大學
HEFEI UNIVERSITY

Datenbanken

32. Konzeptuelles Schema: Kompakte Notation

Thomas Weise (汤卫思)
tweise@hfuu.edu.cn

Institute of Applied Optimization (IAO)
School of Artificial Intelligence and Big Data
Hefei University
Hefei, Anhui, China

应用优化研究所
人工智能与大数据学院
合肥大学
中国安徽省合肥市

Version: 2025-10-28

Databases

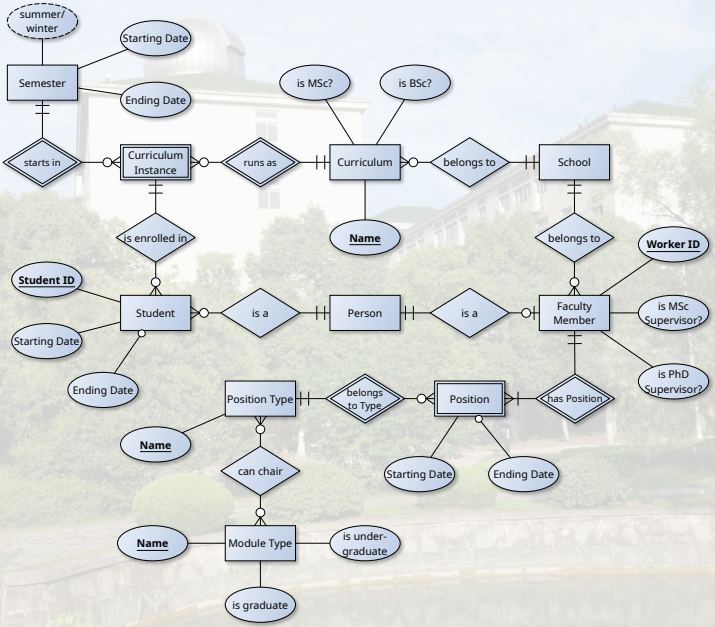
Dies ist ein Kurs über Datenbanken an der Universität Hefei (合肥大学).

Die Webseite mit dem Lehrmaterial dieses Kurses ist https://thomasweise.github.io/databases (siehe auch den QR-Kode unten rechts). Dort können Sie das Kursbuch (in Englisch) und diese Slides finden. Das Repository mit den Beispielen finden Sie unter https://github.com/thomasWeise/databasesCode.

Outline

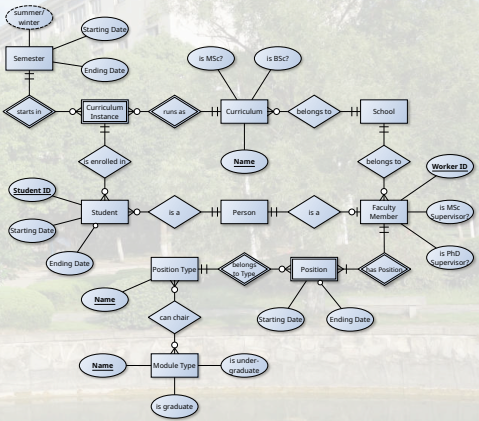
Einleitung

Einleitung


summer/ winter
Starting Date
Semester
Ending Date
is MSc?
is BSc?
starts in
Curriculum Instance
runs as
Curriculum
belongs to
School
Name
is enrolled in
belongs to
Worker ID
Student ID
Student
is a
Person
is a
Faculty Member
is MSc Supervisor?
Starting Date
Ending Date
is PhD Supervisor?
Position Type
belongs to Type
Position
has Position
Name
can chair
Starting Date
Ending Date
Name
Module Type
is under-graduate
is graduate

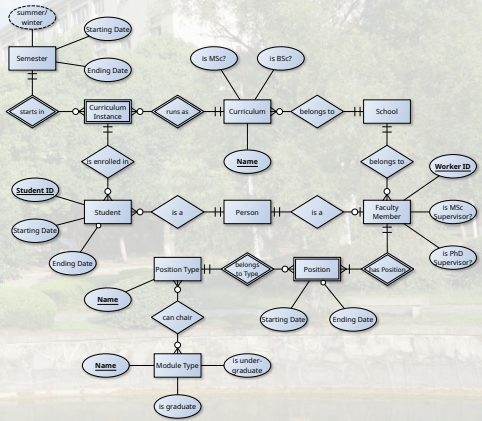
Einleitung

- Wir können zwei generelle Schlussfolgerungen über die Notation, die wir zum Modellieren verwenden, von diesem Diagramm ziehen.



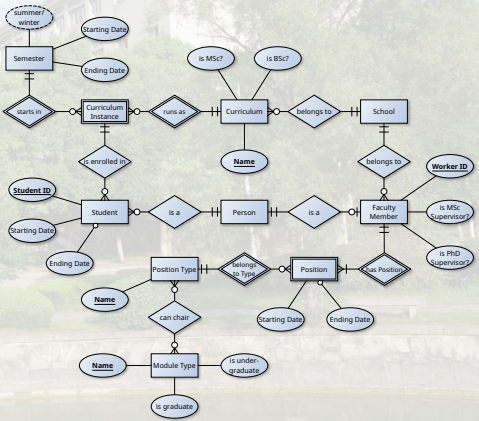
Einleitung

- Wir können zwei generelle Schlussfolgerungen über die Notation, die wir zum Modellieren verwenden, von diesem Diagramm ziehen:
  1. Wir können damit Situationen aus der wirklichen Welt modellieren.



Einleitung

- Wir können zwei generelle Schlussfolgerungen über die Notation, die wir zum Modellieren verwenden, von diesem Diagramm ziehen:
  1. Wir können damit Situationen aus der wirklichen Welt modellieren.
  2. Unsere Syntax verbraucht viel Platz.



Einleitung

- Wir können zwei generelle Schlussfolgerungen über die Notation, die wir zum Modellieren verwenden, von diesem Diagramm ziehen:
  1. Wir können damit Situationen aus der wirklichen Welt modellieren.
  2. Unsere Syntax verbraucht viel Platz.
- Wenn wir viele Attribute haben, werden die vielen Ellipsen schwer zu lesen.

Einleitung

- Wir können zwei generelle Schlussfolgerungen über die Notation, die wir zum Modellieren verwenden, von diesem Diagramm ziehen:
  1. Wir können damit Situationen aus der wirklichen Welt modellieren.
  2. Unsere Syntax verbraucht viel Platz.
- Wenn wir viele Attribute haben, werden die vielen Ellipsen schwer zu lesen.
- Wenn wir viele Beziehungen haben, dann verbrauchen die Rauten (EN: *diamonds*) viel Platz.

Einleitung

- Wir können zwei generelle Schlussfolgerungen über die Notation, die wir zum Modellieren verwenden, von diesem Diagramm ziehen:
  1. Wir können damit Situationen aus der wirklichen Welt modellieren.
  2. Unsere Syntax verbraucht viel Platz.
- Wenn wir viele Attribute haben, werden die vielen Ellipsen schwer zu lesen.
- Wenn wir viele Beziehungen haben, dann verbrauchen die Rauten (EN: *diamonds*) viel Platz.
- Es gibt aber eine Methode, um (fast) die selbe Information platzsparender auszudrücken.

Einleitung

- Wir können zwei generelle Schlussfolgerungen über die Notation, die wir zum Modellieren verwenden, von diesem Diagramm ziehen:
  1. Wir können damit Situationen aus der wirklichen Welt modellieren.
  2. Unsere Syntax verbraucht viel Platz.
- Wenn wir viele Attribute haben, werden die vielen Ellipsen schwer zu lesen.
- Wenn wir viele Beziehungen haben, dann verbrauchen die Rauten (EN: *diamonds*) viel Platz.
- Es gibt aber eine Methode, um (fast) die selbe Information platzsparender auszudrücken:
- Wir können UML Klassendiagramme[8,49,71,72] mit der Krähenfußnotation verbinden.

Einleitung

- Wir können zwei generelle Schlussfolgerungen über die Notation, die wir zum Modellieren verwenden, von diesem Diagramm ziehen:
  1. Wir können damit Situationen aus der wirklichen Welt modellieren.
  2. Unsere Syntax verbraucht viel Platz.
- Wenn wir viele Attribute haben, werden die vielen Ellipsen schwer zu lesen.
- Wenn wir viele Beziehungen haben, dann verbrauchen die Rauten (EN: *diamonds*) viel Platz.
- Es gibt aber eine Methode, um (fast) die selbe Information platzsparender auszudrücken:
- Wir können UML Klassendiagramme[8,49,71,72] mit der Krähenfußnotation verbinden.
- Das ist eine Methode, die in vielen Werkzeugen verwendet wird.

Einleitung

- Wir können zwei generelle Schlussfolgerungen über die Notation, die wir zum Modellieren verwenden, von diesem Diagramm ziehen:
  1. Wir können damit Situationen aus der wirklichen Welt modellieren.
  2. Unsere Syntax verbraucht viel Platz.
- Wenn wir viele Attribute haben, werden die vielen Ellipsen schwer zu lesen.
- Wenn wir viele Beziehungen haben, dann verbrauchen die Rauten (EN: *diamonds*) viel Platz.
- Es gibt aber eine Methode, um (fast) die selbe Information platzsparender auszudrücken:
- Wir können UML Klassendiagramme[8,49,71,72] mit der Krähenfußnotation verbinden.
- Das ist eine Methode, die in vielen Werkzeugen verwendet wird.
- Und wir verwenden sie hier.

Beispiel: Messaging System

Messaging System

- Ein Feature, das sowohl Lehrer als auch Studenten gerne in unserem System haben wollten, war ein *Messaging System*.

Messaging System

- Ein Feature, das sowohl Lehrer als auch Studenten gerne in unserem System haben wollten, war ein *Messaging System*.
- Lehrer wollen eine Möglichkeit, Studenten nachweisbar und nachvollziehbar über Anforderungen, Probleme, Aufgaben, und Erinnerungen zu informieren.

Messaging System

- Ein Feature, das sowohl Lehrer als auch Studenten gerne in unserem System haben wollten, war ein *Messaging System*.
- Lehrer wollen eine Möglichkeit, Studenten nachweisbar und nachvollziehbar über Anforderungen, Probleme, Aufgaben, und Erinnerungen zu informieren: *„Sie haben Ihre Hausaufgaben jetzt zum zweiten Mal in Folge vergessen. Wenn das wieder vorkommt, muss ich Ihnen 10 Punkte von der Endnote abziehen!“*

Messaging System

- Ein Feature, das sowohl Lehrer als auch Studenten gerne in unserem System haben wollten, war ein *Messaging System*.
- Lehrer wollen eine Möglichkeit, Studenten nachweisbar und nachvollziehbar über Anforderungen, Probleme, Aufgaben, und Erinnerungen zu informieren: *„Sie haben Ihre Hausaufgaben jetzt zum zweiten Mal in Folge vergessen. Wenn das wieder vorkommt, muss ich Ihnen 10 Punkte von der Endnote abziehen!“*
- Studenten wollen es auch, aus genau dem selben Grund – um Lehrer über Probleme nachweisbar informieren zu können.

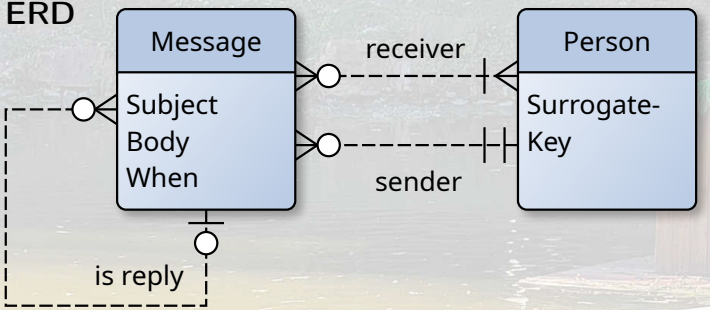
Messaging System

- Ein Feature, das sowohl Lehrer als auch Studenten gerne in unserem System haben wollten, war ein *Messaging System*.
- Lehrer wollen eine Möglichkeit, Studenten nachweisbar und nachvollziehbar über Anforderungen, Probleme, Aufgaben, und Erinnerungen zu informieren: *„Sie haben Ihre Hausaufgaben jetzt zum zweiten Mal in Folge vergessen. Wenn das wieder vorkommt, muss ich Ihnen 10 Punkte von der Endnote abziehen!“*
- Studenten wollen es auch, aus genau dem selben Grund – um Lehrer über Probleme nachweisbar informieren zu können *„Ich will die Hausaufgaben ja gerne machen, aber die PDF-Datei, die Sie zum Download anbieten, ist beschädigt und lässt sich nicht öffnen. Können Sie die nochmal hochladen?“*

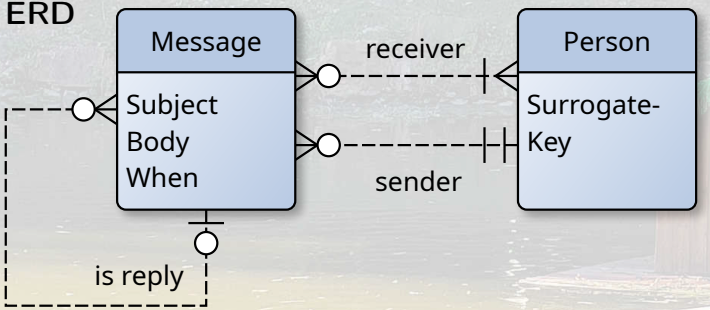
Messaging System

- Ein Feature, das sowohl Lehrer als auch Studenten gerne in unserem System haben wollten, war ein *Messaging System*.
- Lehrer wollen eine Möglichkeit, Studenten nachweisbar und nachvollziehbar über Anforderungen, Probleme, Aufgaben, und Erinnerungen zu informieren: *„Sie haben Ihre Hausaufgaben jetzt zum zweiten Mal in Folge vergessen. Wenn das wieder vorkommt, muss ich Ihnen 10 Punkte von der Endnote abziehen!“*
- Studenten wollen es auch, aus genau dem selben Grund – um Lehrer über Probleme nachweisbar informieren zu können *„Ich will die Hausaufgaben ja gerne machen, aber die PDF-Datei, die Sie zum Download anbieten, ist beschädigt und lässt sich nicht öffnen. Können Sie die nochmal hochladen?“*
- Es ist besser, wenn solche Nachrichten in unserem System beweisbar und nachvollziehbar gespeichert sind.

Messaging System: ERD

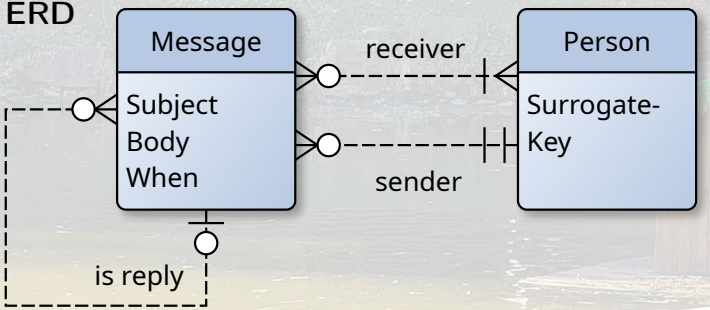


Messaging System: ERD



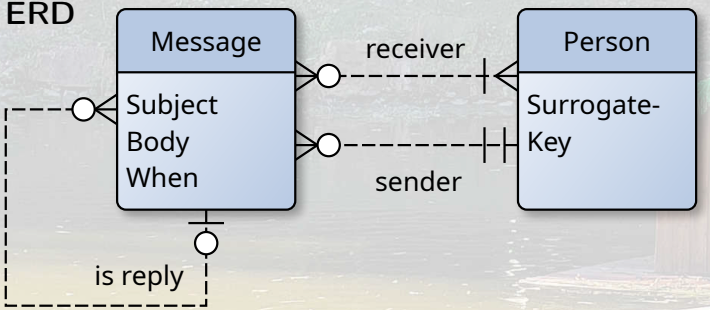
- So sieht das Messaging System in der neuen, kompakten Notation aus.

Messaging System: ERD



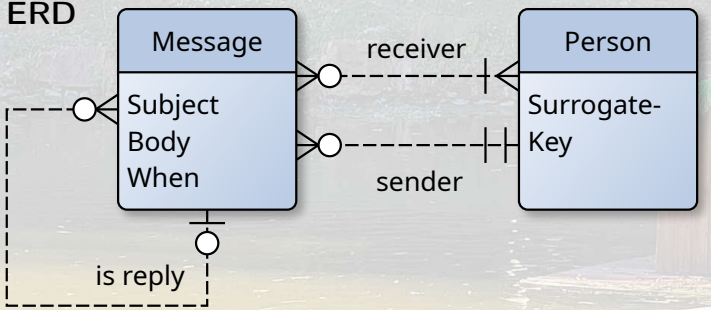
- So sieht das Messaging System in der neuen, kompakten Notation aus.
- Entitätstypen werden immer noch als Rechtecke gezeichnet.

Messaging System: ERD



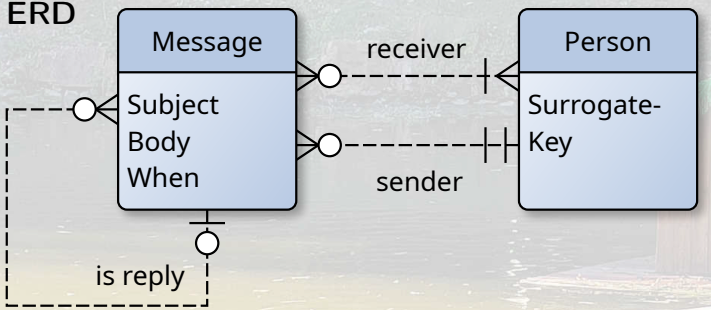
- So sieht das Messaging System in der neuen, kompakten Notation aus.
- Entitätstypen werden immer noch als Rechtecke gezeichnet.
- In der Zeile oben im Rechteck wird der name des Entitätstyps geschrieben.

Messaging System: ERD



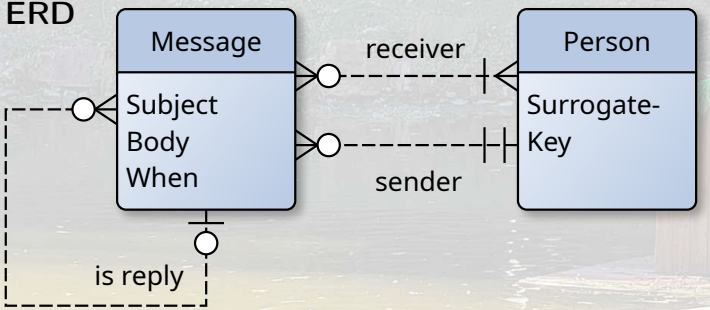
- So sieht das Messaging System in der neuen, kompakten Notation aus.
- Entitätstypen werden immer noch als Rechtecke gezeichnet.
- In der Zeile oben im Rechteck wird der name des Entitätstyps geschrieben.
- Im zweiten Teil des Rechtecks wird die Liste von Attributen geschrieben.

Messaging System: ERD



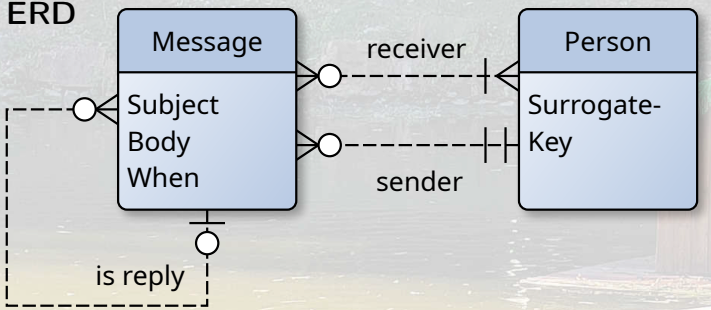
- So sieht das Messaging System in der neuen, kompakten Notation aus.
- Entitätstypen werden immer noch als Rechtecke gezeichnet.
- In der Zeile oben im Rechteck wird der name des Entitätstyps geschrieben.
- Im zweiten Teil des Rechtecks wird die Liste von Attributen geschrieben.
- In unserem Model ist *Person* ein starker Entitätstyp.

Messaging System: ERD



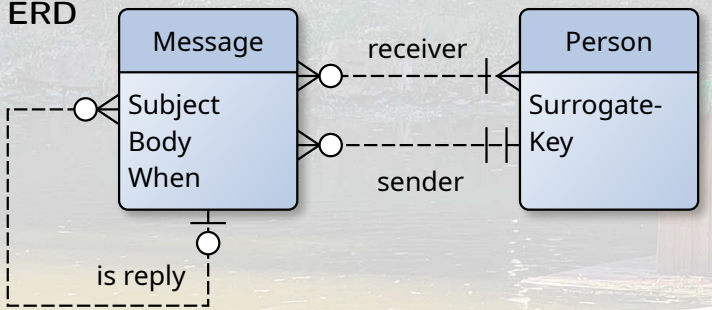
- So sieht das Messaging System in der neuen, kompakten Notation aus.
- Entitätstypen werden immer noch als Rechtecke gezeichnet.
- In der Zeile oben im Rechteck wird der name des Entitätstyps geschrieben.
- Im zweiten Teil des Rechtecks wird die Liste von Attributen geschrieben.
- In unserem Model ist *Person* ein starker Entitätstyp.
- Der neue Entitätstyp *Message* ist auch eine starke Entität.

Messaging System: ERD



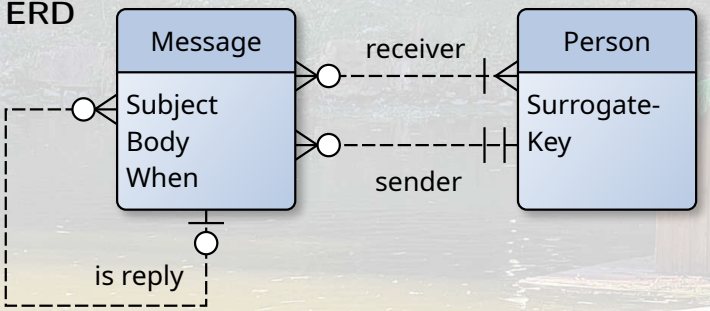
- So sieht das Messaging System in der neuen, kompakten Notation aus.
- Entitätstypen werden immer noch als Rechtecke gezeichnet.
- In der Zeile oben im Rechteck wird der name des Entitätstyps geschrieben.
- Im zweiten Teil des Rechtecks wird die Liste von Attributen geschrieben.
- In unserem Model ist *Person* ein starker Entitätstyp.
- Der neue Entitätstyp *Message* ist auch eine starke Entität.
- Er hat drei Attribute.

Messaging System: ERD



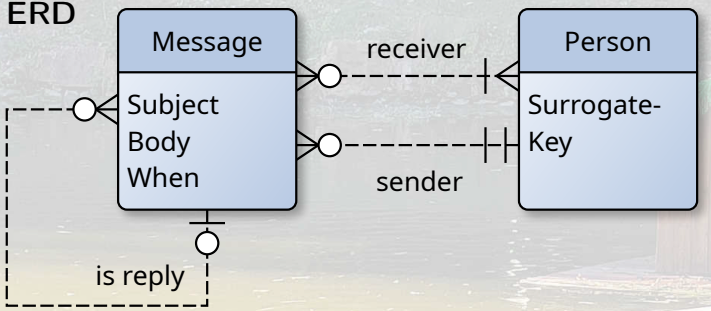
- So sieht das Messaging System in der neuen, kompakten Notation aus.
- Entitätstypen werden immer noch als Rechtecke gezeichnet.
- In der Zeile oben im Rechteck wird der name des Entitätstyps geschrieben.
- Im zweiten Teil des Rechtecks wird die Liste von Attributen geschrieben.
- In unserem Model ist *Person* ein starker Entitätstyp.
- Der neue Entitätstyp *Message* ist auch eine starke Entität.
- Er hat drei Attribute: *Subject*, also den Titel der Nachricht.

Messaging System: ERD



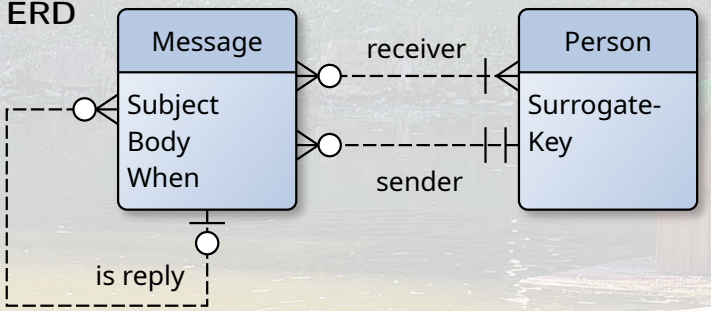
- Entitätstypen werden immer noch als Rechtecke gezeichnet.
- In der Zeile oben im Rechteck wird der name des Entitätstyps geschrieben.
- Im zweiten Teil des Rechtecks wird die Liste von Attributen geschrieben.
- In unserem Model ist *Person* ein starker Entitätstyp.
- Der neue Entitätstyp *Message* ist auch eine starke Entität.
- Er hat drei Attribute: *Subject*, also den Titel der Nachricht, *Body*, also den Text der Nachricht.

Messaging System: ERD



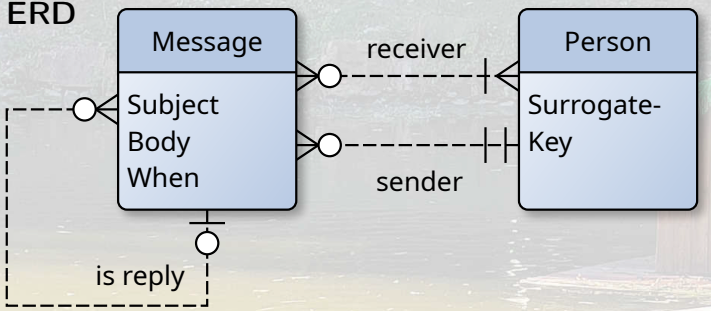
- Entitätstypen werden immer noch als Rechtecke gezeichnet.
- In der Zeile oben im Rechteck wird der name des Entitätstyps geschrieben.
- Im zweiten Teil des Rechtecks wird die Liste von Attributen geschrieben.
- In unserem Model ist *Person* ein starker Entitätstyp.
- Der neue Entitätstyp *Message* ist auch eine starke Entität.
- Er hat drei Attribute: *Subject*, also den Titel der Nachricht, *Body*, also den Text der Nachrichtund *When*, also Datum und Uhrzeit als die Nachricht abgeschickt wurde.

Messaging System: ERD



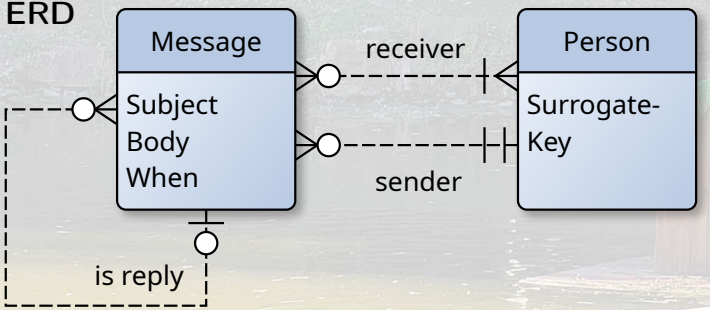
- Im zweiten Teil des Rechtecks wird die Liste von Attributen geschrieben.
- In unserem Model ist *Person* ein starker Entitätstyp.
- Der neue Entitätstyp *Message* ist auch eine starke Entität.
- Er hat drei Attribute: *Subject*, also den Titel der Nachricht, *Body*, also den Text der Nachrichtund *When*, also Datum und Uhrzeit als die Nachricht abgeschickt wurde.
- Beziehungen werden hier als einfache Linien dargestellt, die Entitätstypen direkt verbinden.

Messaging System: ERD



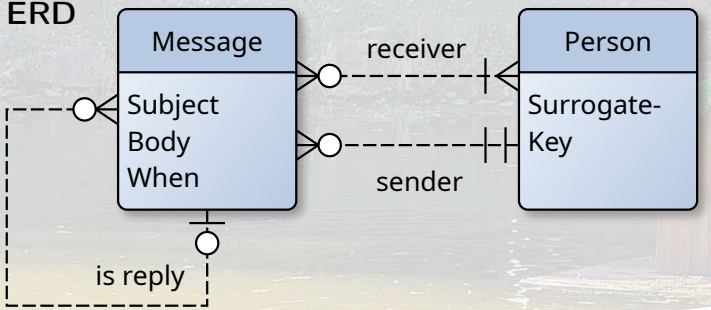
- In unserem Model ist *Person* ein starker Entitätstyp.
- Der neue Entitätstyp *Message* ist auch eine starke Entität.
- Er hat drei Attribute: *Subject*, also den Titel der Nachricht, *Body*, also den Text der Nachrichtund *When*, also Datum und Uhrzeit als die Nachricht abgeschickt wurde.
- Beziehungen werden hier als einfache Linien dargestellt, die Entitätstypen direkt verbinden.
- Die Rauten werden nicht mehr verwendet, stattdessen werden die Beziehungsnamen direkt als Labels an die Linien geschrieben.

Messaging System: ERD



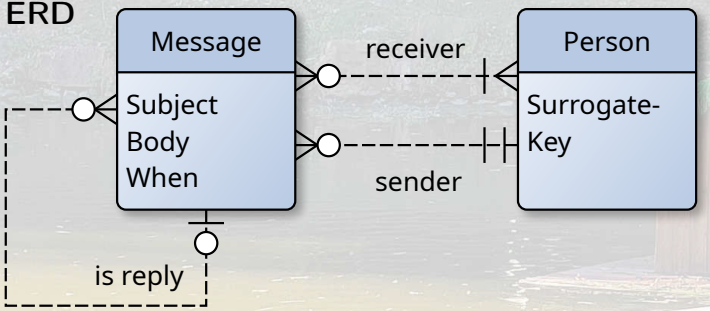
- Der neue Entitätstyp *Message* ist auch eine starke Entität.
- Er hat drei Attribute: *Subject*, also den Titel der Nachricht, *Body*, also den Text der Nachrichtund *When*, also Datum und Uhrzeit als die Nachricht abgeschickt wurde.
- Beziehungen werden hier als einfache Linien dargestellt, die Entitätstypen direkt verbinden.
- Die Rauten werden nicht mehr verwendet, stattdessen werden die Beziehungsnamen direkt als Labels an die Linien geschrieben.
- Kardinalitäten und Modalitäten werden mit der Krähenfußnotation ausgedrückt.

Messaging System: ERD



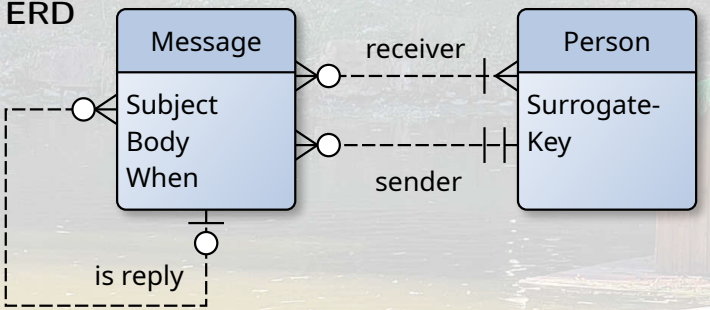
- Er hat drei Attribute: *Subject*, also den Titel der Nachricht, *Body*, also den Text der Nachrichtund *When*, also Datum und Uhrzeit als die Nachricht abgeschickt wurde.
- Beziehungen werden hier als einfache Linien dargestellt, die Entitätstypen direkt verbinden.
- Die Rauten werden nicht mehr verwendet, stattdessen werden die Beziehungsnamen direkt als Labels an die Linien geschrieben.
- Kardinalitäten und Modalitäten werden mit der Krähenfußnotation ausgedrückt.
- Jede Nachricht hat genau eine Person als Sender.

Messaging System: ERD



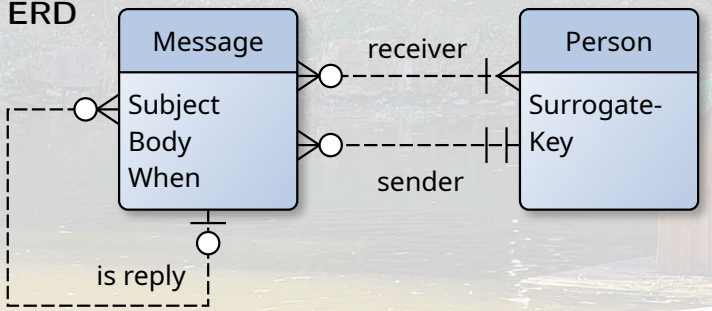
- Beziehungen werden hier als einfache Linien dargestellt, die Entitätstypen direkt verbinden.
- Die Rauten werden nicht mehr verwendet, stattdessen werden die Beziehungsnamen direkt als Labels an die Linien geschrieben.
- Kardinalitäten und Modalitäten werden mit der Krähenfußnotation ausgedrückt.
- Jede Nachricht hat genau eine Person als Sender.
- Jede Person kann Sender beliebig vieler Nachrichten sein.

Messaging System: ERD



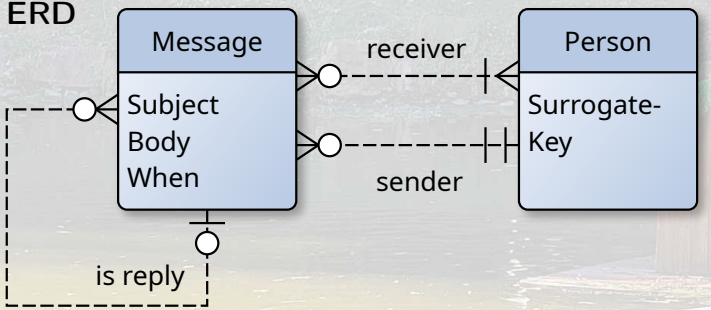
- Die Rauten werden nicht mehr verwendet, stattdessen werden die Beziehungsnamen direkt als Labels an die Linien geschrieben.
- Kardinalitäten und Modalitäten werden mit der Krähenfußnotation ausgedrückt.
- Jede Nachricht hat genau eine Person als Sender.
- Jede Person kann Sender beliebig vieler Nachrichten sein.
- Jede Nachricht hat mindestens einen aber möglicherweise auch mehrere Personen als Empfänger (EN: *receiver*).

Messaging System: ERD



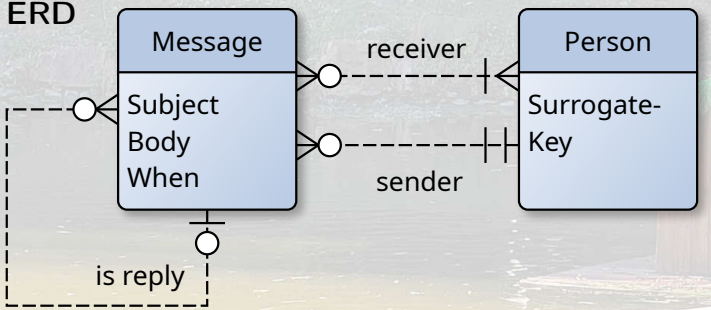
- Kardinalitäten und Modalitäten werden mit der Krähenfußnotation ausgedrückt.
- Jede Nachricht hat genau eine Person als Sender.
- Jede Person kann Sender beliebig vieler Nachrichten sein.
- Jede Nachricht hat mindestens einen aber möglicherweise auch mehrere Personen als Empfänger (EN: *receiver*).
- Jede Person kann Empfänger keiner, einer, oder mehrerer Nachrichten sein.

Messaging System: ERD



- Jede Nachricht hat genau eine Person als Sender.
- Jede Person kann Sender beliebig vieler Nachrichten sein.
- Jede Nachricht hat mindestens einen aber möglicherweise auch mehrere Personen als Empfänger (EN: *receiver*).
- Jede Person kann Empfänger keiner, einer, oder mehrerer Nachrichten sein.
- Jede Nachricht kann die Antwort (EN: *answer*) auf entweder keine oder genau eine vorherige Nachricht sein.

Messaging System: ERD



- Jede Person kann Sender beliebig vieler Nachrichten sein.
- Jede Nachricht hat mindestens einen aber möglicherweise auch mehrere Personen als Empfänger (EN: *receiver*).
- Jede Person kann Empfänger keiner, einer, oder mehrerer Nachrichten sein.
- Jede Nachricht kann die Antwort (EN: *answer*) auf entweder keine oder genau eine vorherige Nachricht sein.
- Es kann beliebig viele Antworten auf eine Nachricht geben.

Starke und Schwache Beziehungen

Starke und Schwache Beziehungen

- Wir haben ursprünglich über schwache und starke Entitäten gesprochen, die wir mit normalen oder doppelten Rechtecken dargestellt haben.

Starke und Schwache Beziehungen

- Wir haben ursprünglich über schwache und starke Entitäten gesprochen, die wir mit normalen oder doppelten Rechtecken dargestellt haben.
- Nun können Beziehungen schwach (EN: *weak*) oder stark (EN: *strong*) sein[51].

Starke und Schwache Beziehungen

- Wir haben ursprünglich über schwache und starke Entitäten gesprochen, die wir mit normalen oder doppelten Rechtecken dargestellt haben.
- Nun können Beziehungen schwach (EN: *weak*) oder stark (EN: *strong*) sein[51].

Definition: Identifizierende Beziehung

Eine *identifizierende* (starke) Beziehung ist mit mindestens einer schwachen Entität verbunden und ist notwendig um diese Entität zu identifizieren.

Starke und Schwache Beziehungen

- Wir haben ursprünglich über schwache und starke Entitäten gesprochen, die wir mit normalen oder doppelten Rechtecken dargestellt haben.
- Nun können Beziehungen schwach (EN: *weak*) oder stark (EN: *strong*) sein[51].

Definition: Identifizierende Beziehung

Eine *identifizierende* (starke) Beziehung ist mit mindestens einer schwachen Entität verbunden und ist notwendig um diese Entität zu identifizieren.

- Normalerweise verbindet eine starke Beziehung eine starke Entität mit einer schwachen Entität.

Starke und Schwache Beziehungen

- Wir haben ursprünglich über schwache und starke Entitäten gesprochen, die wir mit normalen oder doppelten Rechtecken dargestellt haben.
- Nun können Beziehungen schwach (EN: *weak*) oder stark (EN: *strong*) sein[51].

Definition: Identifizierende Beziehung

Eine *identifizierende* (starke) Beziehung ist mit mindestens einer schwachen Entität verbunden und ist notwendig um diese Entität zu identifizieren.

- Normalerweise verbindet eine starke Beziehung eine starke Entität mit einer schwachen Entität.
- Die schwache Entität kann ohne die starke Entität nicht existieren.

Starke und Schwache Beziehungen

- Wir haben ursprünglich über schwache und starke Entitäten gesprochen, die wir mit normalen oder doppelten Rechtecken dargestellt haben.
- Nun können Beziehungen schwach (EN: *weak*) oder stark (EN: *strong*) sein[51].

Definition: Identifizierende Beziehung

Eine *identifizierende* (starke) Beziehung ist mit mindestens einer schwachen Entität verbunden und ist notwendig um diese Entität zu identifizieren.

- Normalerweise verbindet eine starke Beziehung eine starke Entität mit einer schwachen Entität.
- Die schwache Entität kann ohne die starke Entität nicht existieren.
- Der Primärschlüssel der starken Entität ist teil des Schlüssels der schwachen Entität.

Starke und Schwache Beziehungen

- Wir haben ursprünglich über schwache und starke Entitäten gesprochen, die wir mit normalen oder doppelten Rechtecken dargestellt haben.
- Nun können Beziehungen schwach (EN: *weak*) oder stark (EN: *strong*) sein[51].

Definition: Identifizierende Beziehung

Eine *identifizierende* (starke) Beziehung ist mit mindestens einer schwachen Entität verbunden und ist notwendig um diese Entität zu identifizieren.

- Normalerweise verbindet eine starke Beziehung eine starke Entität mit einer schwachen Entität.
- Die schwache Entität kann ohne die starke Entität nicht existieren.
- Der Primärschlüssel der starken Entität ist teil des Schlüssels der schwachen Entität.
- Starke Beziehungen werden durch durchgezogene Linien dargestellt.

Starke und Schwache Beziehungen

Definition: Identifizierende Beziehung

Eine *identifizierende* (starke) Beziehung ist mit mindestens einer schwachen Entität verbunden und ist notwendig um diese Entität zu identifizieren.

- Normalerweise verbindet eine starke Beziehung eine starke Entität mit einer schwachen Entität.
- Die schwache Entität kann ohne die starke Entität nicht existieren.
- Der Primärschlüssel der starken Entität ist teil des Schlüssels der schwachen Entität.
- Starke Beziehungen werden durch durchgezogene Linien dargestellt.

Definition: Nicht-Identifizierende Beziehung

Eine *nicht-identifizierende* (schwache) Beziehung wird nicht benötigt, um eine Entität zu identifizieren.

Starke und Schwache Beziehungen

Definition: Identifizierende Beziehung

Eine *identifizierende* (starke) Beziehung ist mit mindestens einer schwachen Entität verbunden und ist notwendig um diese Entität zu identifizieren.

- Normalerweise verbindet eine starke Beziehung eine starke Entität mit einer schwachen Entität.
- Die schwache Entität kann ohne die starke Entität nicht existieren.
- Der Primärschlüssel der starken Entität ist teil des Schlüssels der schwachen Entität.
- Starke Beziehungen werden durch durchgezogene Linien dargestellt.

Definition: Nicht-Identifizierende Beziehung

Eine *nicht-identifizierende* (schwache) Beziehung wird nicht benötigt, um eine Entität zu identifizieren.

- Beispiele dafür sind Beziehungen, die zwei starke Entitäten verbinden.

Starke und Schwache Beziehungen

Definition: Identifizierende Beziehung

Eine *identifizierende* (starke) Beziehung ist mit mindestens einer schwachen Entität verbunden und ist notwendig um diese Entität zu identifizieren.

- Der Primärschlüssel der starken Entität ist teil des Schlüssels der schwachen Entität.
- Starke Beziehungen werden durch durchgezogene Linien dargestellt.

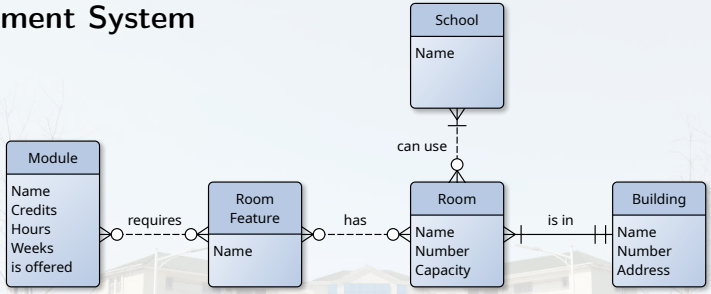
Definition: Nicht-Identifizierende Beziehung

Eine *nicht-identifizierende* (schwache) Beziehung wird nicht benötigt, um eine Entität zu identifizieren.

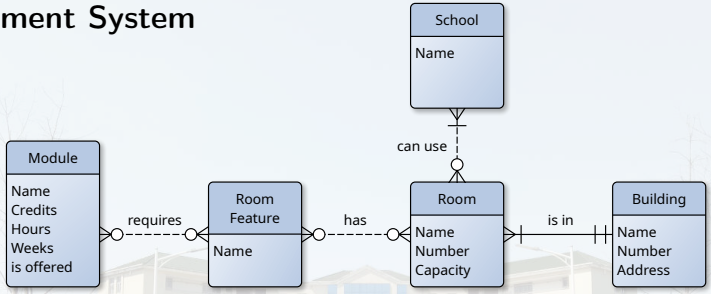
- Beispiele dafür sind Beziehungen, die zwei starke Entitäten verbinden.
- Aber auch schwachte Entitäten können mit schwachen Beziehungen verbunden sein (so lange diese Beziehungen nicht benötigt werden, um die Entitäten zu identifizieren.)

Beispiele

Raum-Management System

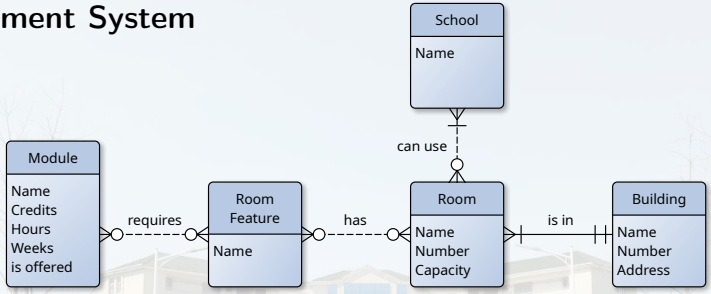


Raum-Management System



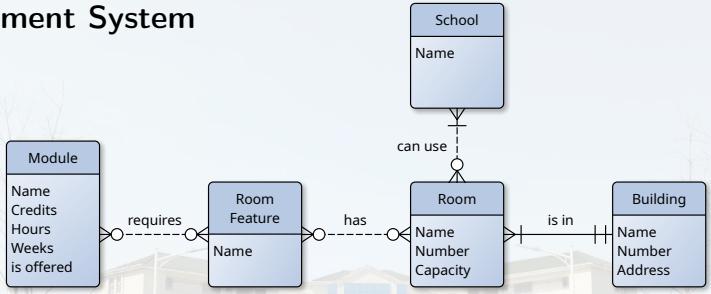
- Wir entwickeln nun ein Raum-Management System für unserer Lehre-Management-Plattform.

Raum-Management System



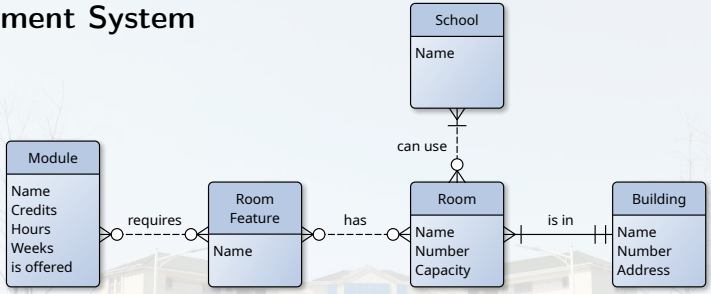
- Wir entwickeln nun ein Raum-Management System für unserer Lehre-Management-Plattform.
- Kurse finden in Räumen statt, also müssen wir wissen, welche Räume existieren, wo sie sind, und welche Features sie haben.

Raum-Management System



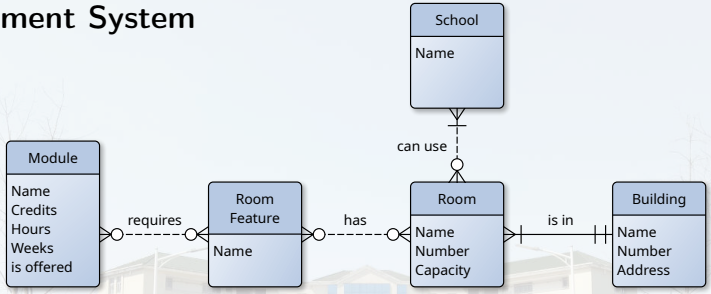
- Wir entwickeln nun ein Raum-Management System für unserer Lehre-Management-Plattform.
- Kurse finden in Räumen statt, also müssen wir wissen, welche Räume existieren, wo sie sind, und welche Features sie haben.
- In unserem Modell sind Gebäude starke Entitäten.

Raum-Management System



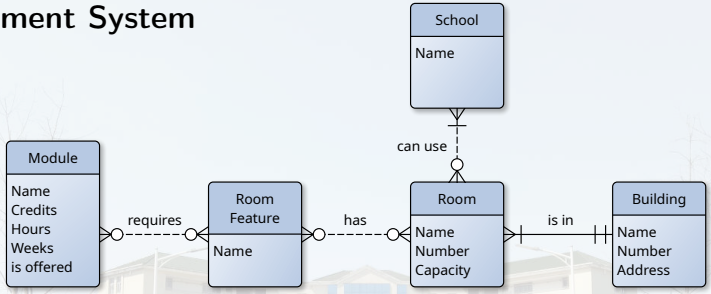
- Wir entwickeln nun ein Raum-Management System für unserer Lehre-Management-Plattform.
- Kurse finden in Räumen statt, also müssen wir wissen, welche Räume existieren, wo sie sind, und welche Features sie haben.
- In unserem Modell sind Gebäude starke Entitäten.
- Sie können einen Namen (wie z. B.合肥大学综合实验楼 haben) und eine Nummer (wie z. B. 53).

Raum-Management System



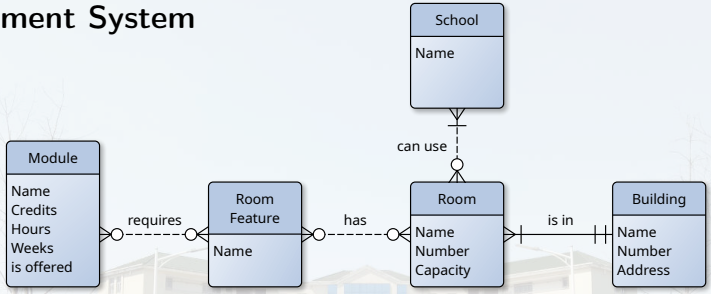
- Wir entwickeln nun ein Raum-Management System für unserer Lehre-Management-Plattform.
- Kurse finden in Räumen statt, also müssen wir wissen, welche Räume existieren, wo sie sind, und welche Features sie haben.
- In unserem Modell sind Gebäude starke Entitäten.
- Sie können einen Namen (wie z. B.合肥大学综合实验楼 haben) und eine Nummer (wie z. B. 53).
- Wir können auch eine Adresse speichern.

Raum-Management System



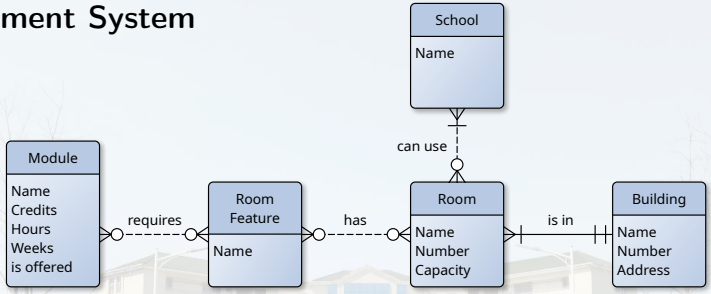
- Kurse finden in Räumen statt, also müssen wir wissen, welche Räume existieren, wo sie sind, und welche Features sie haben.
- In unserem Modell sind Gebäude starke Entitäten.
- Sie können einen Namen (wie z. B.合肥大学综合实验楼 haben) und eine Nummer (wie z. B. 53).
- Wir können auch eine Adresse speichern.
- Das muss jetzt keine so komplizierte Adresse wie bei Personen sein.

Raum-Management System



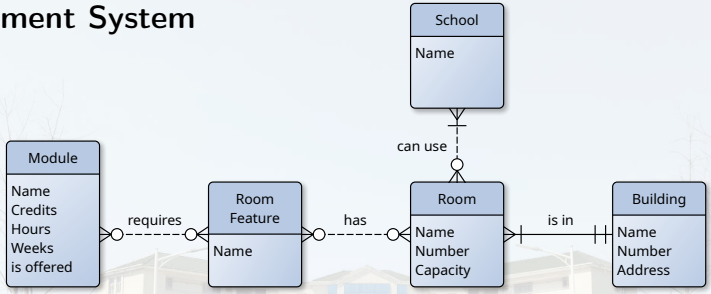
- In unserem Modell sind Gebäude starke Entitäten.
- Sie können einen Namen (wie z. B.合肥大学综合实验楼 haben) und eine Nummer (wie z. B. 53).
- Wir können auch eine Adresse speichern.
- Das muss jetzt keine so komplizierte Adresse wie bei Personen sein.
- Vielleicht würde man hier einfach die Kampusse unterscheiden, z. B. 南一区 und 南二区 oder andere nützliche Information hinschreiben.

Raum-Management System



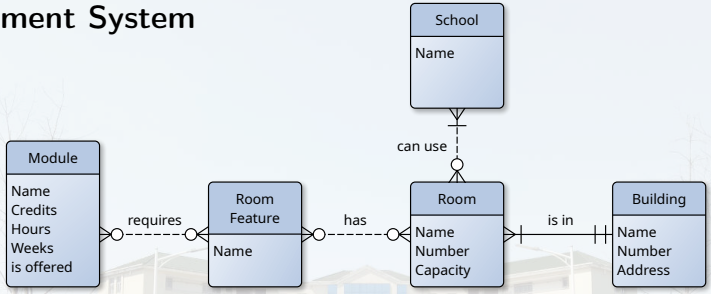
- Sie können einen Namen (wie z. B.合肥大学综合实验楼 haben) und eine Nummer (wie z. B. 53).
- Wir können auch eine Adresse speichern.
- Das muss jetzt keine so komplizierte Adresse wie bei Personen sein.
- Vielleicht würde man hier einfach die Kampusse unterscheiden, z. B. 南一区 und 南二区 oder andere nützliche Information hinschreiben.
- Räume existieren in Gebäuden und niemals ohne Gebäude, deshalb sind sie schwache Entitäten.

Raum-Management System



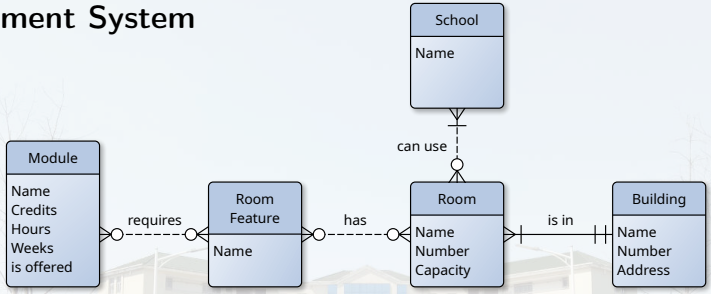
- Wir können auch eine Adresse speichern.
- Das muss jetzt keine so komplizierte Adresse wie bei Personen sein.
- Vielleicht würde man hier einfach die Kampusse unterscheiden, z. B. 南一区 und 南二区 oder andere nützliche Information hinschreiben.
- Räume existieren in Gebäuden und niemals ohne Gebäude, deshalb sind sie schwache Entitäten.
- Sie haben eine Nummer und vielleicht einen Namen.

Raum-Management System



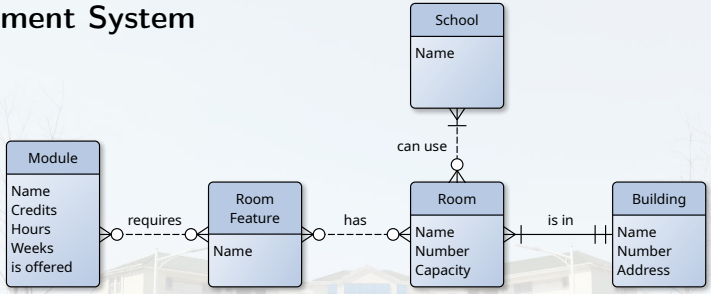
- Das muss jetzt keine so komplizierte Adresse wie bei Personen sein.
- Vielleicht würde man hier einfach die Kampusse unterscheiden, z. B. 南一区 und 南二区 oder andere nützliche Information hinschreiben.
- Räume existieren in Gebäuden und niemals ohne Gebäude, deshalb sind sie schwache Entitäten.
- Sie haben eine Nummer und vielleicht einen Namen.
- Jeder Raum hat eine Kapazität für Studenten.

Raum-Management System



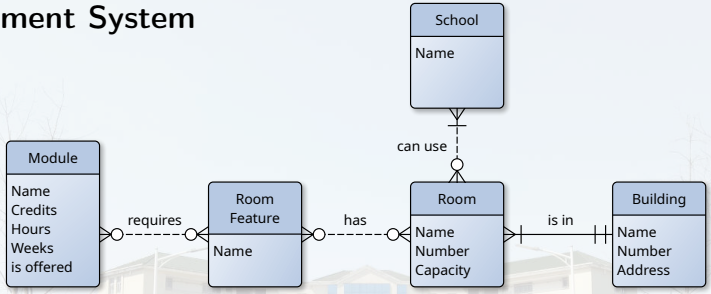
- Vielleicht würde man hier einfach die Kampusse unterscheiden, z. B. 南一区 und 南二区 oder andere nützliche Information hinschreiben.
- Räume existieren in Gebäuden und niemals ohne Gebäude, deshalb sind sie schwache Entitäten.
- Sie haben eine Nummer und vielleicht einen Namen.
- Jeder Raum hat eine Kapazität für Studenten.
- Räume sie mit Gebäuden über identifizierende Beziehungen.

Raum-Management System



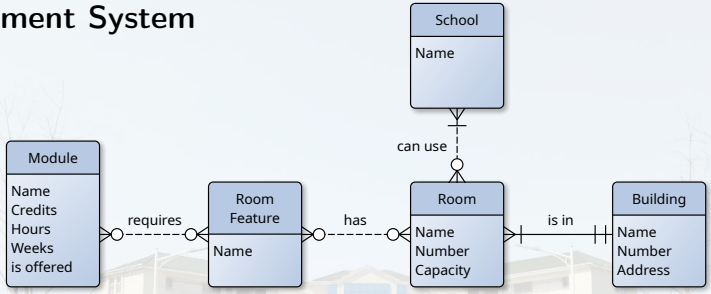
- Räume existieren in Gebäuden und niemals ohne Gebäude, deshalb sind sie schwache Entitäten.
- Sie haben eine Nummer und vielleicht einen Namen.
- Jeder Raum hat eine Kapazität für Studenten.
- Räume sie mit Gebäuden über identifizierende Beziehungen.
- Ein Raum muss in genau einem Gebäude sein und jedes Gebäude besteht aus mindestens einem Raum (sonst brauchen wir es ja nicht in der Datenbank zu speichern).

Raum-Management System



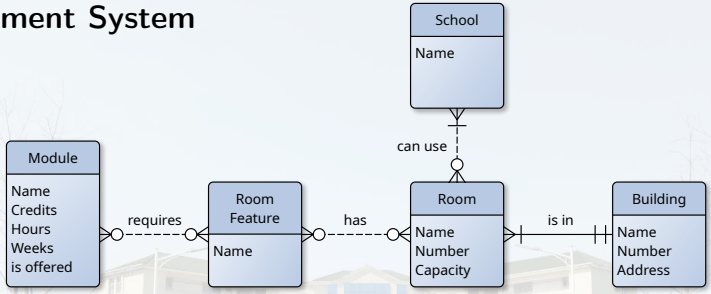
- Sie haben eine Nummer und vielleicht einen Namen.
- Jeder Raum hat eine Kapazität für Studenten.
- Räume sie mit Gebäuden über identifizierende Beziehungen.
- Ein Raum muss in genau einem Gebäude sein und jedes Gebäude besteht aus mindestens einem Raum (sonst brauchen wir es ja nicht in der Datenbank zu speichern).
- Räume sind auch über nicht-identifizierende Beziehungen mit Fakultäten (EN: *schools*) verbunden.

Raum-Management System



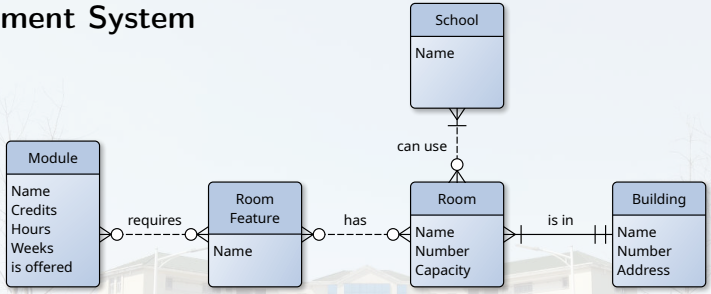
- Räume sie mit Gebäuden über identifizierende Beziehungen.
- Ein Raum muss in genau einem Gebäude sein und jedes Gebäude besteht aus mindestens einem Raum (sonst brauchen wir es ja nicht in der Datenbank zu speichern).
- Räume sind auch über nicht-identifizierende Beziehungen mit Fakultäten (EN: *schools*) verbunden.
- Einer Fakultät ist es erlaubt, keinen, einen, oder mehrere Räume zum Lehren zu verwenden.

Raum-Management System



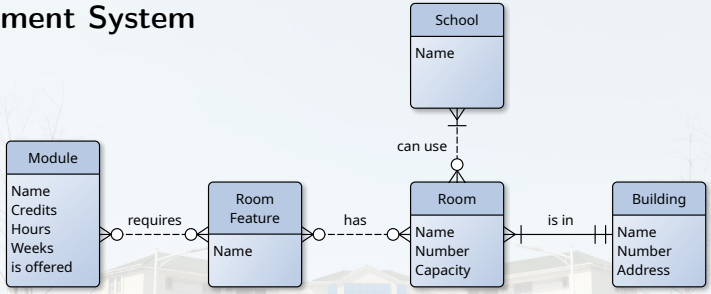
- Räume sind auch über nicht-identifizierende Beziehungen mit Fakultäten (EN: *schools*) verbunden.
- Einer Fakultät ist es erlaubt, keinen, einen, oder mehrere Räume zum Lehren zu verwenden.
- Jeder Raum ist mindestens einer Fakultät zugeordnet – sonst brauchen wir ihn nicht in der Datenbank zu speichern.

Raum-Management System



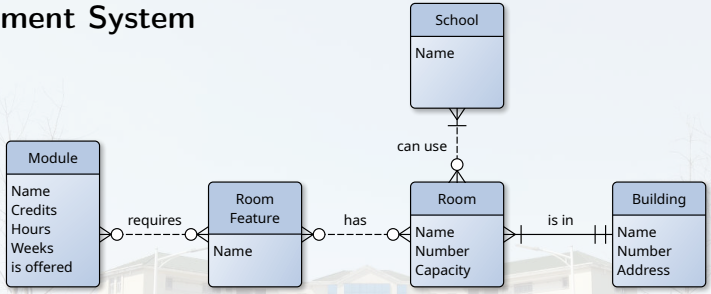
- Räume sind auch über nicht-identifizierende Beziehungen mit Fakultäten (EN: *schools*) verbunden.
- Einer Fakultät ist es erlaubt, keinen, einen, oder mehrere Räume zum Lehren zu verwenden.
- Jeder Raum ist mindestens einer Fakultät zugeordnet – sonst brauchen wir ihn nicht in der Datenbank zu speichern.
- Wir können erwarten, dass verschiedene Lehrmodule verschiedene Anforderungen an Räume haben.

Raum-Management System



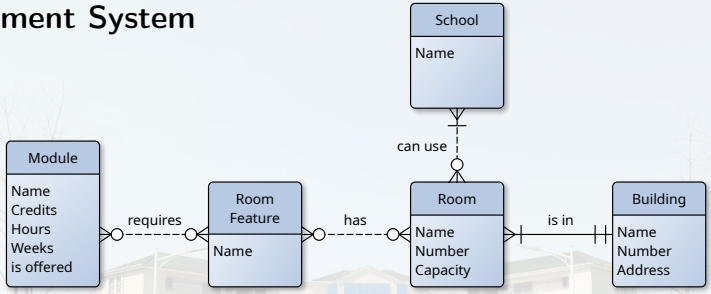
- Einer Fakultät ist es erlaubt, keinen, einen, oder mehrere Räume zum Lehren zu verwenden.
- Jeder Raum ist mindestens einer Fakultät zugeordnet – sonst brauchen wir ihn nicht in der Datenbank zu speichern.
- Wir können erwarten, dass verschiedene Lehrmodule verschiedene Anforderungen an Räume haben.
- Für normales Lehren reicht wahrscheinlich ein Beamer (EN: *overhead projector*) aus.

Raum-Management System



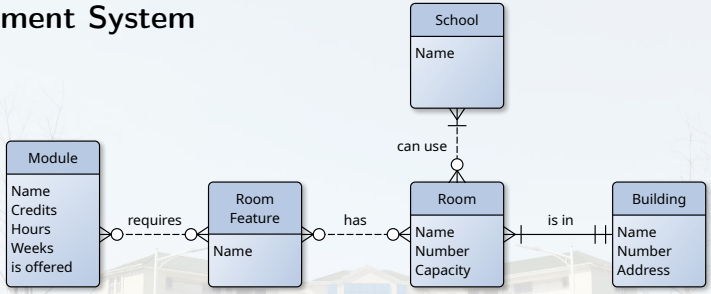
- Einer Fakultät ist es erlaubt, keinen, einen, oder mehrere Räume zum Lehren zu verwenden.
- Jeder Raum ist mindestens einer Fakultät zugeordnet – sonst brauchen wir ihn nicht in der Datenbank zu speichern.
- Wir können erwarten, dass verschiedene Lehrmodule verschiedene Anforderungen an Räume haben.
- Für normales Lehren reicht wahrscheinlich ein Beamer (EN: *overhead projector*) aus.
- Wir könnten annehmen, dass das immer so ist.

Raum-Management System



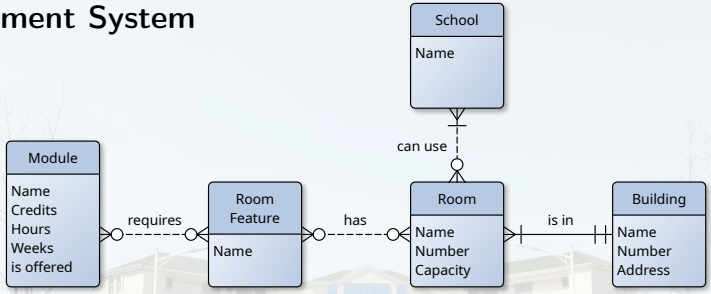
- Jeder Raum ist mindestens einer Fakultät zugeordnet – sonst brauchen wir ihn nicht in der Datenbank zu speichern.
- Wir können erwarten, dass verschiedene Lehrmodule verschiedene Anforderungen an Räume haben.
- Für normales Lehren reicht wahrscheinlich ein Beamer (EN: *overhead projector*) aus.
- Wir könnten annehmen, dass das immer so ist.
- In einer Informatik-Lab-Einheiten brauchen wir aber einen Computer pro Tisch.

Raum-Management System



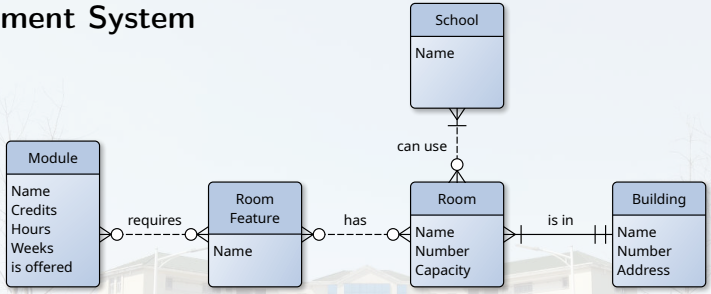
- Wir können erwarten, dass verschiedene Lehrmodule verschiedene Anforderungen an Räume haben.
- Für normales Lehren reicht wahrscheinlich ein Beamer (EN: *overhead projector*) aus.
- Wir könnten annehmen, dass das immer so ist.
- In einer Informatik-Lab-Einheiten brauchen wir aber einen Computer pro Tisch.
- Für Chemie-Experimentier-Einheiten braucht man einen Rauchabzug und irgendeine Vorrichtung für chemische Abfälle.

Raum-Management System



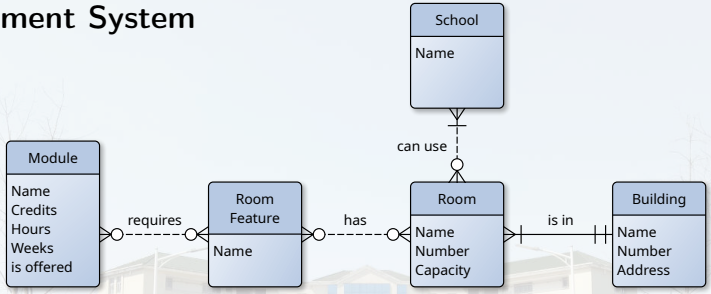
- Für normales Lehren reicht wahrscheinlich ein Beamer (EN: *overhead projector*) aus.
- Wir könnten annehmen, dass das immer so ist.
- In einer Informatik-Lab-Einheiten brauchen wir aber einen Computer pro Tisch.
- Für Chemie-Experimentier-Einheiten braucht man einen Rauchabzug und irgendeine Vorrichtung für chemische Abfälle.
- Um solche Dinge zu modellieren, erstellen wir den starken Entitätstyp *Room Feature*, der einfach nur einen erklärenden Namen speichert.

Raum-Management System



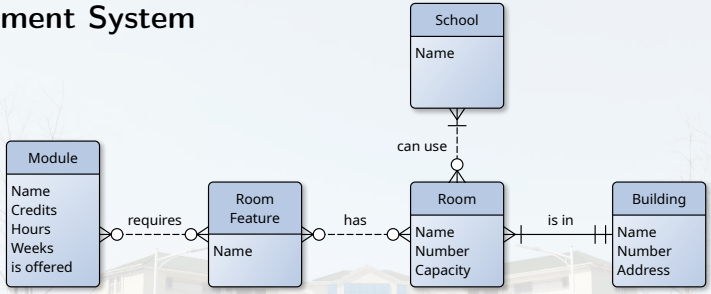
- Wir könnten annehmen, dass das immer so ist.
- In einer Informatik-Lab-Einheiten brauchen wir aber einen Computer pro Tisch.
- Für Chemie-Experimentier-Einheiten braucht man einen Rauchabzug und irgendeine Vorrichtung für chemische Abfälle.
- Um solche Dinge zu modellieren, erstellen wir den starken Entitätstyp *Room Feature*, der einfach nur einen erklärenden Namen speichert.
- Räume sind über nicht-identifizierende Beziehungen mit solchen Features verbunden.

Raum-Management System



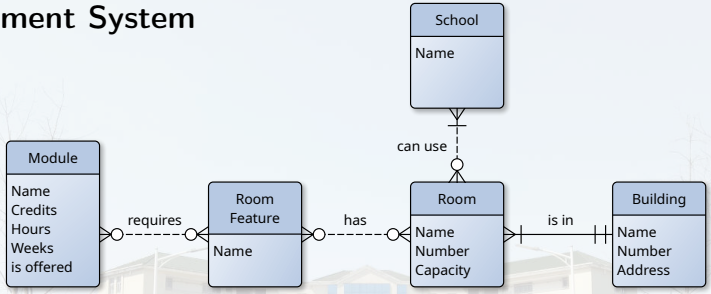
- In einer Informatik-Lab-Einheiten brauchen wir aber einen Computer pro Tisch.
- Für Chemie-Experimentier-Einheiten braucht man einen Rauchabzug und irgendeine Vorrichtung für chemische Abfälle.
- Um solche Dinge zu modellieren, erstellen wir den starken Entitätstyp *Room Feature*, der einfach nur einen erklärenden Namen speichert.
- Räume sind über nicht-identifizierende Beziehungen mit solchen Features verbunden.
- Jeder Raum kann kein, ein, oder viele solche Features haben.

Raum-Management System



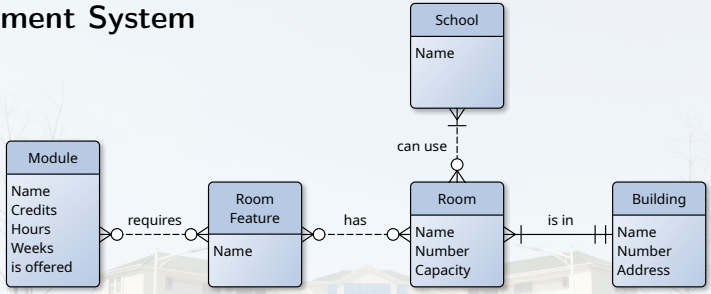
- Für Chemie-Experimentier-Einheiten braucht man einen Rauchabzug und irgendeine Vorrichtung für chemische Abfälle.
- Um solche Dinge zu modellieren, erstellen wir den starken Entitätstyp *Room Feature*, der einfach nur einen erklärenden Namen speichert.
- Räume sind über nicht-identifizierende Beziehungen mit solchen Features verbunden.
- Jeder Raum kann kein, ein, oder viele solche Features haben.
- Jedes Raumfeature kann von keinem, einem, oder vielen Räumen bereitgestellt werden.

Raum-Management System



- Um solche Dinge zu modellieren, erstellen wir den starken Entitätstyp *Room Feature*, der einfach nur einen erklärenden Namen speichert.
- Räume sind über nicht-identifizierende Beziehungen mit solchen Features verbunden.
- Jeder Raum kann kein, ein, oder viele solche Features haben.
- Jedes Raumfeature kann von keinem, einem, oder vielen Räumen bereitgestellt werden.
- Gleichzeitig können Lehrmodule beliebig viele Raumfeatures erfordern.

Raum-Management System



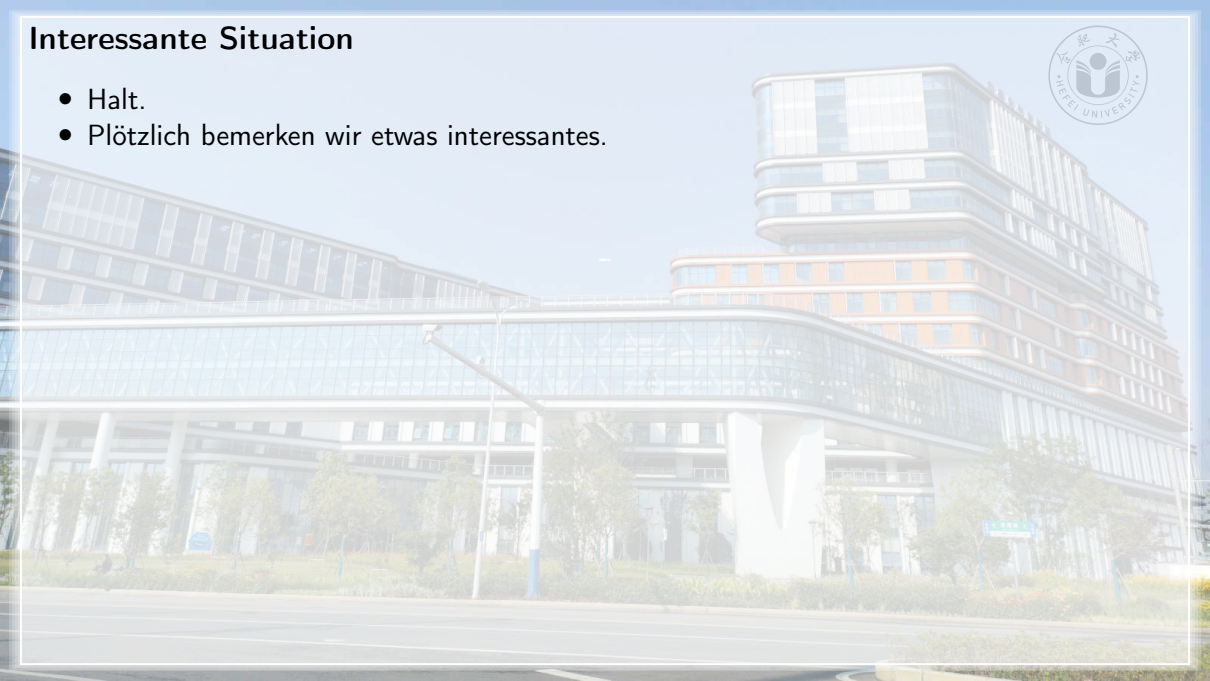
- Um solche Dinge zu modellieren, erstellen wir den starken Entitätstyp *Room Feature*, der einfach nur einen erklärenden Namen speichert.
- Räume sind über nicht-identifizierende Beziehungen mit solchen Features verbunden.
- Jeder Raum kann kein, ein, oder viele solche Features haben.
- Jedes Raumfeature kann von keinem, einem, oder vielen Räumen bereitgestellt werden.
- Gleichzeitig können Lehrmodule beliebig viele Raumfeatures erfordern.
- Ein Raumfeature kann von beliebig vielen Raummodulen erfordert werden.

Interessante Situation

- Halt.

Interessante Situation

- Halt.
- Plötzlich bemerken wir etwas interessantes.



Interessante Situation

- Halt.
- Plötzlich bemerken wir etwas interessantes.
- Wir haben gesagt: *„Ein Raum muss in genau einem Gebäude sein und jedes Gebäude besteht aus mindestens einem Raum.“*

Interessante Situation

- Halt.
- Plötzlich bemerken wir etwas interessantes.
- Wir haben gesagt: *„Ein Raum muss in genau einem Gebäude sein und jedes Gebäude besteht aus mindestens einem Raum.“*
- Das ist ein interessanter Satz, weil er fast ein philosophisches Problem darstellt.

Interessante Situation

- Halt.
- Plötzlich bemerken wir etwas interessantes.
- Wir haben gesagt: *„Ein Raum muss in genau einem Gebäude sein und jedes Gebäude besteht aus mindestens einem Raum.“*
- Das ist ein interessanter Satz, weil er fast ein philosophisches Problem darstellt.
- Wenn wir diesen Satz aus technischer Sicht anschauen – welche hier noch nicht wichtig ist – dann macht er uns nachdenklich.

Interessante Situation

- Halt.
- Plötzlich bemerken wir etwas interessantes.
- Wir haben gesagt: *„Ein Raum muss in genau einem Gebäude sein und jedes Gebäude besteht aus mindestens einem Raum.“*
- Das ist ein interessanter Satz, weil er fast ein philosophisches Problem darstellt.
- Wenn wir diesen Satz aus technischer Sicht anschauen – welche hier noch nicht wichtig ist – dann macht er uns nachdenklich.
- Was wir hier haben ist ein Building $\dashv\!\vdash\!\!\!<$ Room Schema.

Interessante Situation

- Halt.
- Plötzlich bemerken wir etwas interessantes.
- Wir haben gesagt: *„Ein Raum muss in genau einem Gebäude sein und jedes Gebäude besteht aus mindestens einem Raum.“*
- Das ist ein interessanter Satz, weil er fast ein philosophisches Problem darstellt.
- Wenn wir diesen Satz aus technischer Sicht anschauen – welche hier noch nicht wichtig ist – dann macht er uns nachdenklich.
- Was wir hier haben ist ein Building $\dashv\!\!\vdash\!\!\dashv\!\!\vdash$ Room Schema.
- Wir haben schon mehrere solcher Schemas gesehen.

Interessante Situation

- Halt.
- Plötzlich bemerken wir etwas interessantes.
- Wir haben gesagt: *„Ein Raum muss in genau einem Gebäude sein und jedes Gebäude besteht aus mindestens einem Raum.“*
- Das ist ein interessanter Satz, weil er fast ein philosophisches Problem darstellt.
- Wenn wir diesen Satz aus technischer Sicht anschauen – welche hier noch nicht wichtig ist – dann macht er uns nachdenklich.
- Was wir hier haben ist ein Building $\nparallel\!\!-\!\!\nvdash$ Room Schema.
- Wir haben schon mehrere solcher Schemas gesehen.
- Wir sind noch nicht an der Stelle angekommen, wo wir ein Datenmodell und ein DBMS auswählen müssen.

Interessante Situation

- Halt.
- Plötzlich bemerken wir etwas interessantes.
- Wir haben gesagt: *„Ein Raum muss in genau einem Gebäude sein und jedes Gebäude besteht aus mindestens einem Raum.“*
- Das ist ein interessanter Satz, weil er fast ein philosophisches Problem darstellt.
- Wenn wir diesen Satz aus technischer Sicht anschauen – welche hier noch nicht wichtig ist – dann macht er uns nachdenklich.
- Was wir hier haben ist ein Building $\dashv\!\vdash\!\!\!\!<$ Room Schema.
- Wir haben schon mehrere solcher Schemas gesehen.
- Wir sind noch nicht an der Stelle angekommen, wo wir ein Datenmodell und ein DBMS auswählen müssen.
- Aber egal welches Datenmodell wir wählen, wir werden gezwungen sein, die Datenstruktur *Room* zu instantiieren um einen Raum in unserer DB zu repräsentieren.

Interessante Situation

- Halt.
- Plötzlich bemerken wir etwas interessantes.
- Wir haben gesagt: *„Ein Raum muss in genau einem Gebäude sein und jedes Gebäude besteht aus mindestens einem Raum.“*
- Das ist ein interessanter Satz, weil er fast ein philosophisches Problem darstellt.
- Wenn wir diesen Satz aus technischer Sicht anschauen – welche hier noch nicht wichtig ist – dann macht er uns nachdenklich.
- Was wir hier haben ist ein Building $\dashv\!\!\vdash\!\!\dashv\!\!\vdash$ Room Schema.
- Wir haben schon mehrere solcher Schemas gesehen.
- Wir sind noch nicht an der Stelle angekommen, wo wir ein Datenmodell und ein DBMS auswählen müssen.
- Aber egal welches Datenmodell wir wählen, wir werden gezwungen sein, die Datenstruktur *Room* zu instantiieren um einen Raum in unserer DB zu repräsentieren.
- Wenn wir der formalen Definition oben folgen, dann brauchen wir dafür eine existierende Entität vom Typ *Building*.

Interessante Situation

- Plötzlich bemerken wir etwas interessantes.
- Wir haben gesagt: *„Ein Raum muss in genau einem Gebäude sein und jedes Gebäude besteht aus mindestens einem Raum.“*
- Das ist ein interessanter Satz, weil er fast ein philosophisches Problem darstellt.
- Wenn wir diesen Satz aus technischer Sicht anschauen – welche hier noch nicht wichtig ist – dann macht er uns nachdenklich.
- Was wir hier haben ist ein Building $\shortparallel\!\!-\!\!\!\vdash\!\!\!<$ Room Schema.
- Wir haben schon mehrere solcher Schemas gesehen.
- Wir sind noch nicht an der Stelle angekommen, wo wir ein Datenmodell und ein DBMS auswählen müssen.
- Aber egal welches Datenmodell wir wählen, wir werden gezwungen sein, die Datenstruktur *Room* zu instantiieren um einen Raum in unserer DB zu repräsentieren.
- Wenn wir der formalen Definition oben folgen, dann brauchen wir dafür eine existierende Entität vom Typ *Building*.
- Jeder Raum muss mit einem Gebäude verbunden sein.

Interessante Situation

- Wir haben gesagt: *„Ein Raum muss in genau einem Gebäude sein und jedes Gebäude besteht aus mindestens einem Raum.“*
- Das ist ein interessanter Satz, weil er fast ein philosophisches Problem darstellt.
- Wenn wir diesen Satz aus technischer Sicht anschauen – welche hier noch nicht wichtig ist – dann macht er uns nachdenklich.
- Was wir hier haben ist ein Building ⫲—⊰ Room Schema.
- Wir haben schon mehrere solcher Schemas gesehen.
- Wir sind noch nicht an der Stelle angekommen, wo wir ein Datenmodell und ein DBMS auswählen müssen.
- Aber egal welches Datenmodell wir wählen, wir werden gezwungen sein, die Datenstruktur *Room* zu instantiieren um einen Raum in unserer DB zu repräsentieren.
- Wenn wir der formalen Definition oben folgen, dann brauchen wir dafür eine existierende Entität vom Typ *Building*.
- Jeder Raum muss mit einem Gebäude verbunden sein.
- Daher müssen wir *Building* instantiieren und dann können wir einen *Room* instantiieren.

Interessante Situation

- Das ist ein interessanter Satz, weil er fast ein philosophisches Problem darstellt.
- Wenn wir diesen Satz aus technischer Sicht anschauen – welche hier noch nicht wichtig ist – dann macht er uns nachdenklich.
- Was wir hier haben ist ein Building ⫫⊣⩤ Room Schema.
- Wir haben schon mehrere solcher Schemas gesehen.
- Wir sind noch nicht an der Stelle angekommen, wo wir ein Datenmodell und ein DBMS auswählen müssen.
- Aber egal welches Datenmodell wir wählen, wir werden gezwungen sein, die Datenstruktur *Room* zu instantiieren um einen Raum in unserer DB zu repräsentieren.
- Wenn wir der formalen Definition oben folgen, dann brauchen wir dafür eine existierende Entität vom Typ *Building*.
- Jeder Raum muss mit einem Gebäude verbunden sein.
- Daher müssen wir *Building* instantiieren und dann können wir einen *Room* instantiieren.
- Das geht aber nicht.

Interessante Situation

- Wenn wir diesen Satz aus technischer Sicht anschauen – welche hier noch nicht wichtig ist – dann macht er uns nachdenklich.
- Was wir hier haben ist ein Building $\|\!\!-\!\!\!\prec$ Room Schema.
- Wir haben schon mehrere solcher Schemas gesehen.
- Wir sind noch nicht an der Stelle angekommen, wo wir ein Datenmodell und ein DBMS auswählen müssen.
- Aber egal welches Datenmodell wir wählen, wir werden gezwungen sein, die Datenstruktur *Room* zu instantiieren um einen Raum in unserer DB zu repräsentieren.
- Wenn wir der formalen Definition oben folgen, dann brauchen wir dafür eine existierende Entität vom Typ *Building*.
- Jeder Raum muss mit einem Gebäude verbunden sein.
- Daher müssen wir *Building* instantiieren und dann können wir einen *Room* instantiieren.
- Das geht aber nicht.
- Wenn wir das ERD streng interpretieren, dann haben wir das Problem, dass jedes Gebäude mit mindestens einem Raum verbunden seien muss.

Interessante Situation

- Was wir hier haben ist ein Building ⫫⊣⋉ Room Schema.
- Wir haben schon mehrere solcher Schemas gesehen.
- Wir sind noch nicht an der Stelle angekommen, wo wir ein Datenmodell und ein DBMS auswählen müssen.
- Aber egal welches Datenmodell wir wählen, wir werden gezwungen sein, die Datenstruktur *Room* zu instantiieren um einen Raum in unserer DB zu repräsentieren.
- Wenn wir der formalen Definition oben folgen, dann brauchen wir dafür eine existierende Entität vom Typ *Building*.
- Jeder Raum muss mit einem Gebäude verbunden sein.
- Daher müssen wir *Building* instantiieren und dann können wir einen *Room* instantiieren.
- Das geht aber nicht.
- Wenn wir das ERD streng interpretieren, dann haben wir das Problem, dass jedes Gebäude mit mindestens einem Raum verbunden seien muss.
- Das bedeutet also, dass wir *zuerst* einen *Room* erzeugen müssen, bevor wir ein *Building* erstellen können.

Interessante Situation

- Wir haben schon mehrere solcher Schemas gesehen.
- Wir sind noch nicht an der Stelle angekommen, wo wir ein Datenmodell und ein DBMS auswählen müssen.
- Aber egal welches Datenmodell wir wählen, wir werden gezwungen sein, die Datenstruktur *Room* zu instantiieren um einen Raum in unserer DB zu repräsentieren.
- Wenn wir der formalen Definition oben folgen, dann brauchen wir dafür eine existierende Entität vom Typ *Building*.
- Jeder Raum muss mit einem Gebäude verbunden sein.
- Daher müssen wir *Building* instantiieren und dann können wir einen *Room* instantiieren.
- Das geht aber nicht.
- Wenn wir das ERD streng interpretieren, dann haben wir das Problem, dass jedes Gebäude mit mindestens einem Raum verbunden seien muss.
- Das bedeutet also, dass wir *zuerst* einen *Room* erzeugen müssen, bevor wir ein *Building* erstellen können.
- Dieses philosophische Dilemma kann auf drei Arten gelöst werden.

Interessante Situation

- Wenn wir der formalen Definition oben folgen, dann brauchen wir dafür eine existierende Entität vom Typ *Building*.
- Jeder Raum muss mit einem Gebäude verbunden sein.
- Daher müssen wir *Building* instantiieren und dann können wir einen *Room* instantiieren.
- Das geht aber nicht.
- Wenn wir das ERD streng interpretieren, dann haben wir das Problem, dass jedes Gebäude mit mindestens einem Raum verbunden seien muss.
- Das bedeutet also, dass wir *zuerst* einen *Room* erzeugen müssen, bevor wir ein *Building* erstellen können.
- Dieses philosophische Dilemma kann auf drei Arten gelöst werden.
- Zuerst könnten wir uns hinstellen und sagen: *„Nun. Das konzeptuelle Modell repräsentiert die reale Welt. In der realen Welt existiert jeder Raum in einem Gebäude und jedes Gebäude hat mindestens einen Raum. Das ist wahr und das modellieren wir hier. Ob man das praktisch implementieren kann oder nicht, spielt an dieser Stelle keine Rolle.“*

Interessante Situation

- Wenn wir der formalen Definition oben folgen, dann brauchen wir dafür eine existierende Entität vom Typ *Building*.
- Jeder Raum muss mit einem Gebäude verbunden sein.
- Daher müssen wir *Building* instantiieren und dann können wir einen *Room* instantiieren.
- Das geht aber nicht.
- Wenn wir das ERD streng interpretieren, dann haben wir das Problem, dass jedes Gebäude mit mindestens einem Raum verbunden seien muss.
- Das bedeutet also, dass wir *zuerst* einen *Room* erzeugen müssen, bevor wir ein *Building* erstellen können.
- Dieses philosophische Dilemma kann auf drei Arten gelöst werden.
- Zuerst könnten wir uns hinstellen und sagen: *„Nun. Das konzeptuelle Modell repräsentiert die reale Welt. In der realen Welt existiert jeder Raum in einem Gebäude und jedes Gebäude hat mindestens einen Raum. Das ist wahr und das modellieren wir hier. Ob man das praktisch implementieren kann oder nicht, spielt an dieser Stelle keine Rolle.“*
- Und das ist auch richtig, denn konzeptuelles Modellieren soll Technologieunabhängig sein.

Interessante Situation

- Das bedeutet also, dass wir *zuerst* einen *Room* erzeugen müssen, bevor wir ein *Building* erstellen können.
- Dieses philosophische Dilemma kann auf drei Arten gelöst werden.
- Zuerst könnten wir uns hinstellen und sagen: „*Nun. Das konzeptuelle Modell repräsentiert die reale Welt. In der realen Welt existiert jeder Raum in einem Gebäude und jedes Gebäude hat mindestens einen Raum. Das ist wahr und das modellieren wir hier. Ob man das praktisch implementieren kann oder nicht, spielt an dieser Stelle keine Rolle.*“
- Und das ist auch richtig, denn konzeptuelles Modellieren soll Technologieunabhängig sein.
- Zweitens könnten wir sagen: „*Wenn es nicht anders geht, dann werden wir auf technischer Ebene einfach nicht erzwingen, dass jedes Gebäude mindestens einen Raum hat. Wir wissen, dass die Benutzer Raum-Datensätze dann später erstellen werden. Wir akzeptieren einfach, dass einen temporären Zustand der Datenbank gibt, wo diese gegenseitige Abhängigkeit verletzt wird. Es wäre ja nur für eine kurze Zeit, zwischen dem Moment, wo der Gebäudedatensatz und dem Moment wo der erste Raumdatensatz erstellt wird. We just accept that there may be a temporary state of the DB where one of the mutual dependency constraints is violated. Das spielt keine Rolle.*“

Interessante Situation

- Dieses philosophische Dilemma kann auf drei Arten gelöst werden.
- Zuerst könnten wir uns hinstellen und sagen: „*Nun. Das konzeptuelle Modell repräsentiert die reale Welt. In der realen Welt existiert jeder Raum in einem Gebäude und jedes Gebäude hat mindestens einen Raum. Das ist wahr und das modellieren wir hier. Ob man das praktisch implementieren kann oder nicht, spielt an dieser Stelle keine Rolle.*“
- Und das ist auch richtig, denn konzeptuelles Modellieren soll Technologieunabhängig sein.
- Zweitens könnten wir sagen: „*Wenn es nicht anders geht, dann werden wir auf technischer Ebene einfach nicht erzwingen, dass jedes Gebäude mindestens einen Raum hat. Wir wissen, dass die Benutzer Raum-Datensätze dann später erstellen werden. Wir akzeptieren einfach, dass einen temporären Zustand der Datenbank gibt, wo diese gegenseitige Abhängigkeit verletzt wird. Es wäre ja nur für eine kurze Zeit, zwischen dem Moment, wo der Gebäudedatensatz und dem Moment wo der erste Raumdatensatz erstellt wird. We just accept that there may be a temporary state of the DB where one of the mutual dependency constraints is violated. Das spielt keine Rolle.*“
- Die dritte Lösung ist, dass wir das tatsächlich so implementieren, wie es spezifiziert ist.

Interessante Situation

- Und das ist auch richtig, denn konzeptuelles Modellieren soll Technologieunabhängig sein.
- Zweitens könnten wir sagen: „*Wenn es nicht anders geht, dann werden wir auf technischer Ebene einfach nicht erzwingen, dass jedes Gebäude mindestens einen Raum hat. Wir wissen, dass die Benutzer Raum-Datensätze dann später erstellen werden. Wir akzeptieren einfach, dass einen temporären Zustand der Datenbank gibt, wo diese gegenseitige Abhängigkeit verletzt wird. Es wäre ja nur für eine kurze Zeit, zwischen dem Moment, wo der Gebäudedatensatz und dem Moment wo der erste Raumdatensatz erstellt wird. We just accept that there may be a temporary state of the DB where one of the mutual dependency constraints is violated. Das spielt keine Rolle.*“
- Die dritte Lösung ist, dass wir das tatsächlich so implementieren, wie es spezifiziert ist.
- Die meisten DBMSe unterstützen nämlich *Transactionen*, also Gruppen von Befehlen die als eine unteilbare (EN: *atomic*) Einheit ausgeführt werden und entweder zusammen erfolgreich sind oder zusammen fehlschlagen.

Interessante Situation

- Und das ist auch richtig, denn konzeptuelles Modellieren soll Technologieunabhängig sein.
- Zweitens könnten wir sagen: *„Wenn es nicht anders geht, dann werden wir auf technischer Ebene einfach nicht erzwingen, dass jedes Gebäude mindestens einen Raum hat. Wir wissen, dass die Benutzer Raum-Datensätze dann später erstellen werden. Wir akzeptieren einfach, dass einen temporären Zustand der Datenbank gibt, wo diese gegenseitige Abhängigkeit verletzt wird. Es wäre ja nur für eine kurze Zeit, zwischen dem Moment, wo der Gebäudedatensatz und dem Moment wo der erste Raumdatensatz erstellt wird. We just accept that there may be a temporary state of the DB where one of the mutual dependency constraints is violated. Das spielt keine Rolle.“*
- Die dritte Lösung ist, dass wir das tatsächlich so implementieren, wie es spezifiziert ist.
- Die meisten DBMSe unterstützen nämlich *Transactionen*, also Gruppen von Befehlen die als eine unteilbare (EN: *atomic*) Einheit ausgeführt werden und entweder zusammen erfolgreich sind oder zusammen fehlschlagen.
- Es gibt keinen Zwischenzustand, weil Transactionen unteilbar sind.

Interessante Situation

- Und das ist auch richtig, denn konzeptuelles Modellieren soll Technologieunabhängig sein.
- Zweitens könnten wir sagen: „*Wenn es nicht anders geht, dann werden wir auf technischer Ebene einfach nicht erzwingen, dass jedes Gebäude mindestens einen Raum hat. Wir wissen, dass die Benutzer Raum-Datensätze dann später erstellen werden. Wir akzeptieren einfach, dass einen temporären Zustand der Datenbank gibt, wo diese gegenseitige Abhängigkeit verletzt wird. Es wäre ja nur für eine kurze Zeit, zwischen dem Moment, wo der Gebäudedatensatz und dem Moment wo der erste Raumdatensatz erstellt wird. We just accept that there may be a temporary state of the DB where one of the mutual dependency constraints is violated. Das spielt keine Rolle.*“
- Die dritte Lösung ist, dass wir das tatsächlich so implementieren, wie es spezifiziert ist.
- Die meisten DBMSe unterstützen nämlich *Transactionen*, also Gruppen von Befehlen die als eine unteilbare (EN: *atomic*) Einheit ausgeführt werden und entweder zusammen erfolgreich sind oder zusammen fehlschlagen.
- Es gibt keinen Zwischenzustand, weil Transactionen unteilbar sind.
- Wir könnten also erzwingen, dass beim Erstellen eines Gebäudes auch der erste Raum im Gebäue gleich miterstellt wird.

Interessante Situation

- Zweitens könnten wir sagen: *„Wenn es nicht anders geht, dann werden wir auf technischer Ebene einfach nicht erzwingen, dass jedes Gebäude mindestens einen Raum hat. Wir wissen, dass die Benutzer Raum-Datensätze dann später erstellen werden. Wir akzeptieren einfach, dass einen temporären Zustand der Datenbank gibt, wo diese gegenseitige Abhängigkeit verletzt wird. Es wäre ja nur für eine kurze Zeit, zwischen dem Moment, wo der Gebäudedatensatz und dem Moment wo der erste Raumdatensatz erstellt wird. We just accept that there may be a temporary state of the DB where one of the mutual dependency constraints is violated. Das spielt keine Rolle.“*
- Die dritte Lösung ist, dass wir das tatsächlich so implementieren, wie es spezifiziert ist.
- Die meisten DBMSe unterstützen nämlich *Transactionen*, also Gruppen von Befehlen die als eine unteilbare (EN: *atomic*) Einheit ausgeführt werden und entweder zusammen erfolgreich sind oder zusammen fehlschlagen.
- Es gibt keinen Zwischenzustand, weil Transactionen unteilbar sind.
- Wir könnten also erzwingen, dass beim Erstellen eines Gebäudes auch der erste Raum im Gebäue gleich miterstellt wird.
- Das wäre eine unteilbare Transaktion.

Interessante Situation

- Die dritte Lösung ist, dass wir das tatsächlich so implementieren, wie es spezifiziert ist.
- Die meisten DBMSe unterstützen nämlich *Transactionen*, also Gruppen von Befehlen die als eine unteilbare (EN: *atomic*) Einheit ausgeführt werden und entweder zusammen erfolgreich sind oder zusammen fehlschlagen.
- Es gibt keinen Zwischenzustand, weil Transactionen unteilbar sind.
- Wir könnten also erzwingen, dass beim Erstellen eines Gebäudes auch der erste Raum im Gebäue gleich miterstellt wird.
- Das wäre eine unteilbare Transaktion.
- Dann können wir die konzeptuelle Einschränkung auf eine logische / technische Einschränkung abgebildet.

Interessante Situation

- Die dritte Lösung ist, dass wir das tatsächlich so implementieren, wie es spezifiziert ist.
- Die meisten DBMSe unterstützen nämlich *Transactionen*, also Gruppen von Befehlen die als eine unteilbare (EN: *atomic*) Einheit ausgeführt werden und entweder zusammen erfolgreich sind oder zusammen fehlschlagen.
- Es gibt keinen Zwischenzustand, weil Transactionen unteilbar sind.
- Wir könnten also erzwingen, dass beim Erstellen eines Gebäudes auch der erste Raum im Gebäue gleich miterstellt wird.
- Das wäre eine unteilbare Transaktion.
- Dann können wir die konzeptuelle Einschränkung auf eine logische / technische Einschränkung abgebildet.
- Wir würden dafür mit komplizierteren Operationen bezahlen, weil wir dann eben wirklich Transaktionen brauchen.

Interessante Situation

- Die dritte Lösung ist, dass wir das tatsächlich so implementieren, wie es spezifiziert ist.
- Die meisten DBMSe unterstützen nämlich *Transactionen*, also Gruppen von Befehlen die als eine unteilbare (EN: *atomic*) Einheit ausgeführt werden und entweder zusammen erfolgreich sind oder zusammen fehlschlagen.
- Es gibt keinen Zwischenzustand, weil Transactionen unteilbar sind.
- Wir könnten also erzwingen, dass beim Erstellen eines Gebäudes auch der erste Raum im Gebäue gleich miterstellt wird.
- Das wäre eine unteilbare Transaktion.
- Dann können wir die konzeptuelle Einschränkung auf eine logische / technische Einschränkung abgebildet.
- Wir würden dafür mit komplizierteren Operationen bezahlen, weil wir dann eben wirklich Transaktionen brauchen.
- PostgreSQL erlaubt es uns, das Überprüfen von Einschränkungen auf das Ende von Transaktionen zu verschieben[46,60].

Interessante Situation

- Die dritte Lösung ist, dass wir das tatsächlich so implementieren, wie es spezifiziert ist.
- Die meisten DBMSe unterstützen nämlich *Transactionen*, also Gruppen von Befehlen die als eine unteilbare (EN: *atomic*) Einheit ausgeführt werden und entweder zusammen erfolgreich sind oder zusammen fehlschlagen.
- Es gibt keinen Zwischenzustand, weil Transactionen unteilbar sind.
- Wir könnten also erzwingen, dass beim Erstellen eines Gebäudes auch der erste Raum im Gebäue gleich miterstellt wird.
- Das wäre eine unteilbare Transaktion.
- Dann können wir die konzeptuelle Einschränkung auf eine logische / technische Einschränkung abgebildet.
- Wir würden dafür mit komplizierteren Operationen bezahlen, weil wir dann eben wirklich Transaktionen brauchen.
- PostgreSQL erlaubt es uns, das Überprüfen von Einschränkungen auf das Ende von Transaktionen zu verschieben[46,60].
- Wir werden später lernen, dass wir sogar eine PostgreSQL-spezifische SQL-Erweiterung benutzen können, um dieses Problem zu lösen.

Interessante Situation

- Die meisten DBMSe unterstützen nämlich *Transactionen*, also Gruppen von Befehlen die als eine unteilbare (EN: *atomic*) Einheit ausgeführt werden und entweder zusammen erfolgreich sind oder zusammen fehlschlagen.
- Es gibt keinen Zwischenzustand, weil Transactionen unteilbar sind.
- Wir könnten also erzwingen, dass beim Erstellen eines Gebäudes auch der erste Raum im Gebäue gleich miterstellt wird.
- Das wäre eine unteilbare Transaktion.
- Dann können wir die konzeptuelle Einschränkung auf eine logische / technische Einschränkung abgebildet.
- Wir würden dafür mit komplizierteren Operationen bezahlen, weil wir dann eben wirklich Transaktionen brauchen.
- PostgreSQL erlaubt es uns, das Überprüfen von Einschränkungen auf das Ende von Transaktionen zu verschieben[46,60].
- Wir werden später lernen, dass wir sogar eine PostgreSQL-spezifische SQL-Erweiterung benutzen können, um dieses Problem zu lösen.
- Egal.

Interessante Situation

- Es gibt keinen Zwischenzustand, weil Transactionen unteilbar sind.
- Wir könnten also erzwingen, dass beim Erstellen eines Gebäudes auch der erste Raum im Gebäue gleich miterstellt wird.
- Das wäre eine unteilbare Transaktion.
- Dann können wir die konzeptuelle Einschränkung auf eine logische / technische Einschränkung abgebildet.
- Wir würden dafür mit komplizierteren Operationen bezahlen, weil wir dann eben wirklich Transaktionen brauchen.
- PostgreSQL erlaubt es uns, das Überprüfen von Einschränkungen auf das Ende von Transaktionen zu verschieben[46,60].
- Wir werden später lernen, dass wir sogar eine PostgreSQL-spezifische SQL-Erweiterung benutzen können, um dieses Problem zu lösen.
- Egal.
- Der Punkt ist, dass konzeptuelle Beziehungen die schwierig technisch realisierbar erscheinen existieren können.

Interessante Situation

- Das wäre eine unteilbare Transaktion.
- Dann können wir die konzeptuelle Einschränkung auf eine logische / technische Einschränkung abgebildet.
- Wir würden dafür mit komplizierteren Operationen bezahlen, weil wir dann eben wirklich Transaktionen brauchen.
- PostgreSQL erlaubt es uns, das Überprüfen von Einschränkungen auf das Ende von Transaktionen zu verschieben[46,60].
- Wir werden später lernen, dass wir sogar eine PostgreSQL-spezifische SQL-Erweiterung benutzen können, um dieses Problem zu lösen.
- Egal.
- Der Punkt ist, dass konzeptuelle Beziehungen die schwierig technisch realisierbar erscheinen existieren können.
- Wir werden später sehen, dass wir alle (binären) Beziehungsmodelle, die wir mit der Krähenfußnotation darstellen können, auch in einem relationalen DBMS implementiert werden können.

Interessante Situation

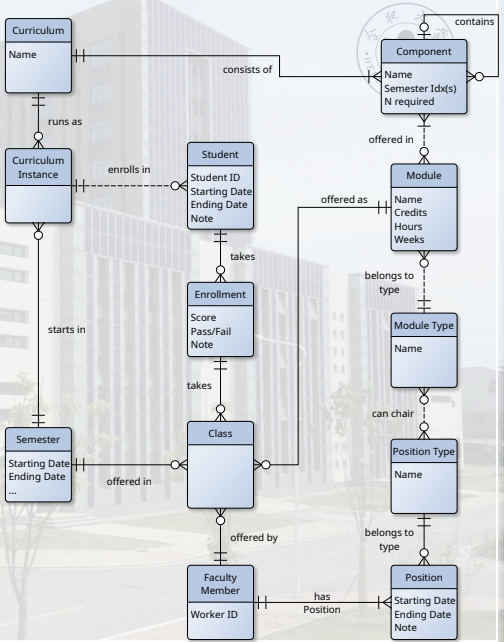
- Dann können wir die konzeptuelle Einschränkung auf eine logische / technische Einschränkung abgebildet.
- Wir würden dafür mit komplizierteren Operationen bezahlen, weil wir dann eben wirklich Transaktionen brauchen.
- PostgreSQL erlaubt es uns, das Überprüfen von Einschränkungen auf das Ende von Transaktionen zu verschieben[46,60].
- Wir werden später lernen, dass wir sogar eine PostgreSQL-spezifische SQL-Erweiterung benutzen können, um dieses Problem zu lösen.
- Egal.
- Der Punkt ist, dass konzeptuelle Beziehungen die schwierig technisch realisierbar erscheinen existieren können.
- Wir werden später sehen, dass wir alle (binären) Beziehungsmodelle, die wir mit der Krähenfußnotation darstellen können, auch in einem relationalen DBMS implementiert werden können.
- Ob das immer sinnvoll ist (es kann kompliziert werden) oder nicht, das soll uns her nicht interessieren.

Bigger Picture, Part 2

- Wir re-designen nun die Interaktionen zwischen Studenten, Studiengängen, Mitarbeitern, und Modulen.

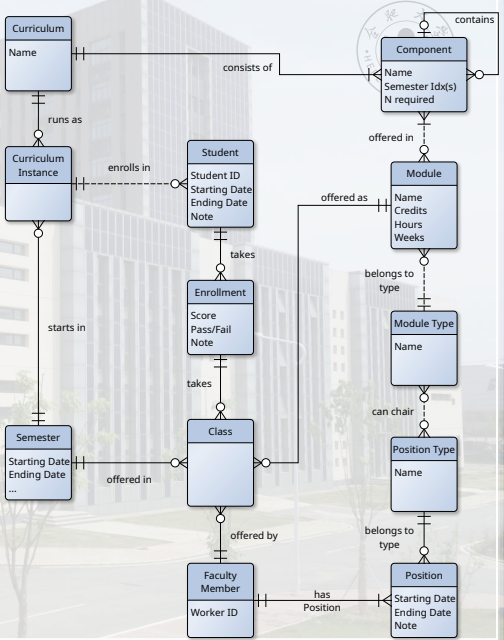
Bigger Picture, Part 2

- Wir re-designen nun die Interaktionen zwischen Studenten, Studiengängen, Mitarbeitern, und Modulen.
- Bevor wir unsere ursprünglichen Pläne für diese Systeme entwickelten, hatten wir mit den Stakeholders in der Uni diskutiert.



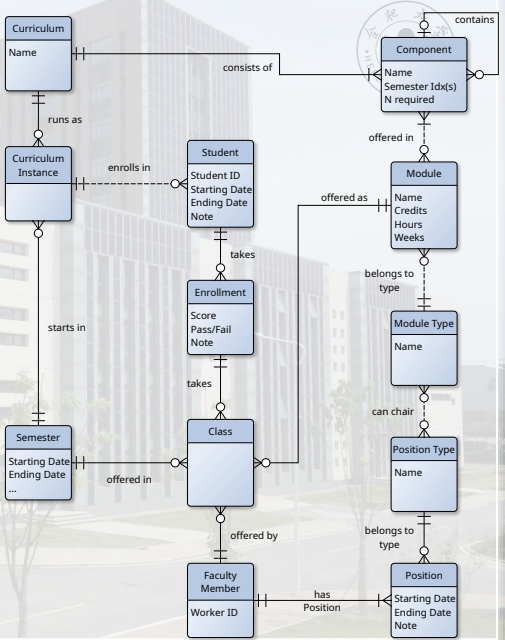
Bigger Picture, Part 2

- Wir re-designen nun die Interaktionen zwischen Studenten, Studiengängen, Mitarbeitern, und Modulen.
- Bevor wir unsere ursprünglichen Pläne für diese Systeme entwickelten, hatten wir mit den Stakeholders in der Uni diskutiert.
- Wir hatten gelernt, dass einige der Fähigkeiten der Lehrer an ihre Position gebunden sind, z. B. welche Module sie unterrichten dürfen, und andere an ihre Person, z. B. ob sie Masterarbeiten beaufsichtigen können.



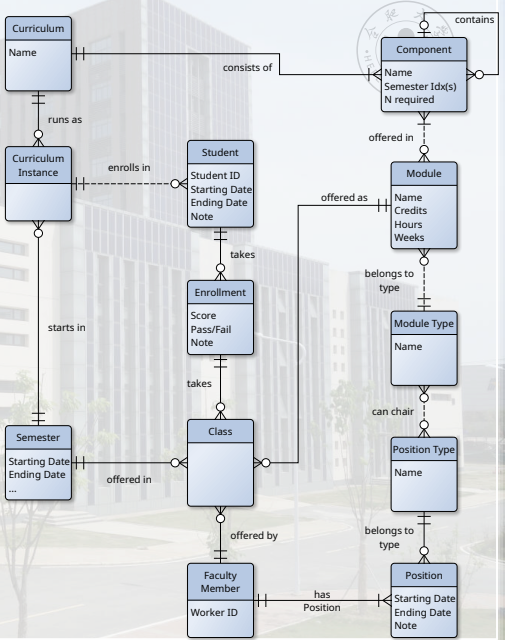
Bigger Picture, Part 2

- Wir re-designen nun die Interaktionen zwischen Studenten, Studiengängen, Mitarbeitern, und Modulen.
- Bevor wir unsere ursprünglichen Pläne für diese Systeme entwickelten, hatten wir mit den Stakeholders in der Uni diskutiert.
- Wir hatten gelernt, dass einige der Fähigkeiten der Lehrer an ihre Position gebunden sind, z. B. welche Module sie unterrichten dürfen, und andere an ihre Person, z. B. ob sie Masterarbeiten beaufsichtigen können.
- So hatten wir das ja auch modelliert.



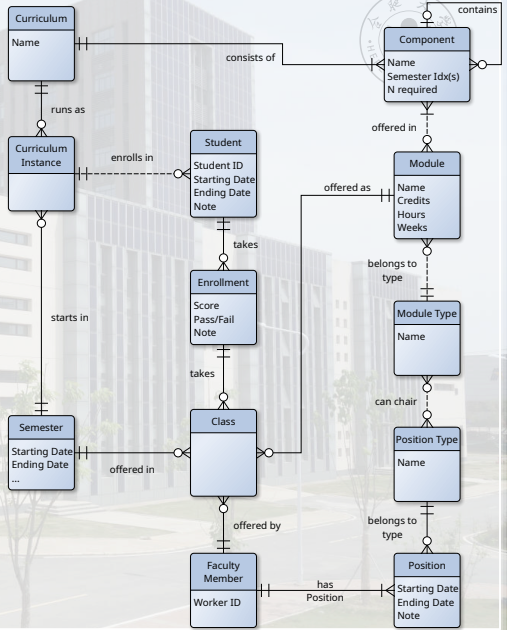
Bigger Picture, Part 2

- Wir re-designen nun die Interaktionen zwischen Studenten, Studiengängen, Mitarbeitern, und Modulen.
- Bevor wir unsere ursprünglichen Pläne für diese Systeme entwickelten, hatten wir mit den Stakeholders in der Uni diskutiert.
- Wir hatten gelernt, dass einige der Fähigkeiten der Lehrer an ihre Position gebunden sind, z. B. welche Module sie unterrichten dürfen, und andere an ihre Person, z. B. ob sie Masterarbeiten beaufsichtigen können.
- So hatten wir das ja auch modelliert.
- Es gibt auch zwei verschiedene Arten, wie Lehrer und Studenten interagieren können.



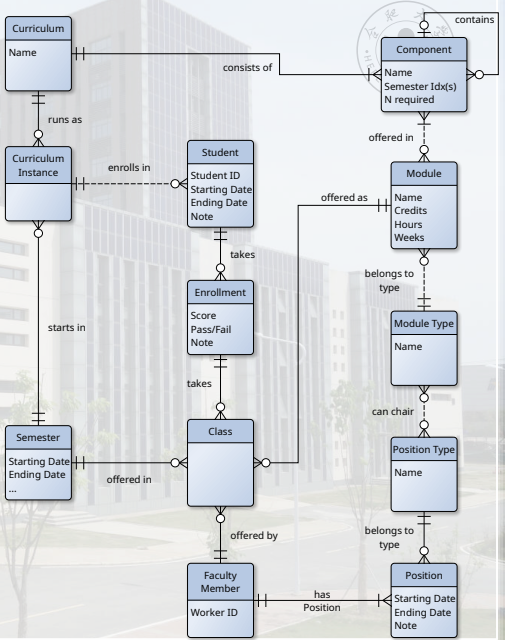
Bigger Picture, Part 2

- Bevor wir unsere ursprünglichen Pläne für diese Systeme entwickelten, hatten wir mit den Stakeholders in der Uni diskutiert.
- Wir hatten gelernt, dass einige der Fähigkeiten der Lehrer an ihre Position gebunden sind, z. B. welche Module sie unterrichten dürfen, und andere an ihre Person, z. B. ob sie Masterarbeiten beaufsichtigen können.
- So hatten wir das ja auch modelliert.
- Es gibt auch zwei verschiedene Arten, wie Lehrer und Studenten interagieren können:
- Studenten können sich in Klassen von Professoren einschreiben und Professoren können MSc- und BSc-Arbeiten betreuen.



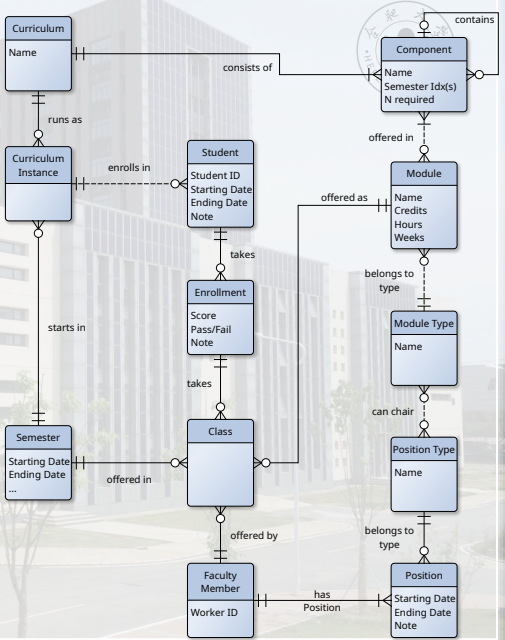
Bigger Picture, Part 2

- Wir hatten gelernt, dass einige der Fähigkeiten der Lehrer an ihre Position gebunden sind, z. B. welche Module sie unterrichten dürfen, und andere an ihre Person, z. B. ob sie Masterarbeiten beaufsichtigen können.
- So hatten wir das ja auch modelliert.
- Es gibt auch zwei verschiedene Arten, wie Lehrer und Studenten interagieren können:
- Studenten können sich in Klassen von Professoren einschreiben und Professoren können MSc- und BSc-Arbeiten betreuen.
- Wir hätten also zwei Mengen von Interaktionen die von zwei verschiedenen Formen von Qualifikationen auf Seiten der Lehrer abhängen.



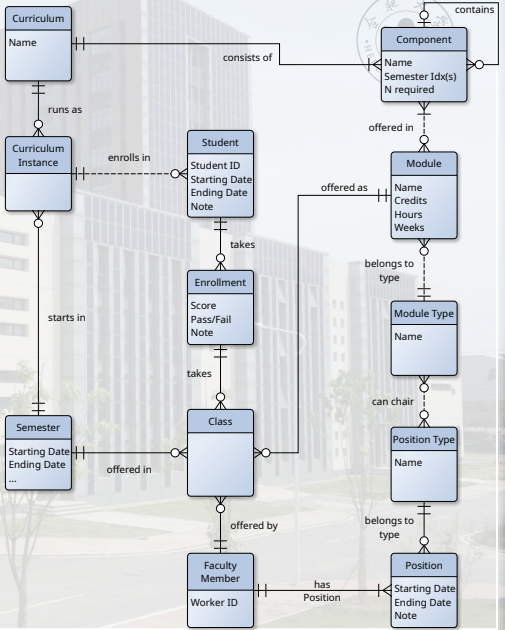
Bigger Picture, Part 2

- Wir hatten gelernt, dass einige der Fähigkeiten der Lehrer an ihre Position gebunden sind, z. B. welche Module sie unterrichten dürfen, und andere an ihre Person, z. B. ob sie Masterarbeiten beaufsichtigen können.
- So hatten wir das ja auch modelliert.
- Es gibt auch zwei verschiedene Arten, wie Lehrer und Studenten interagieren können:
- Studenten können sich in Klassen von Professoren einschreiben und Professoren können MSc- und BSc-Arbeiten betreuen.
- Wir hätten also zwei Mengen von Interaktionen die von zwei verschiedenen Formen von Qualifikationen auf Seiten der Lehrer abhängen.
- Solche Situation sind immer schlecht.



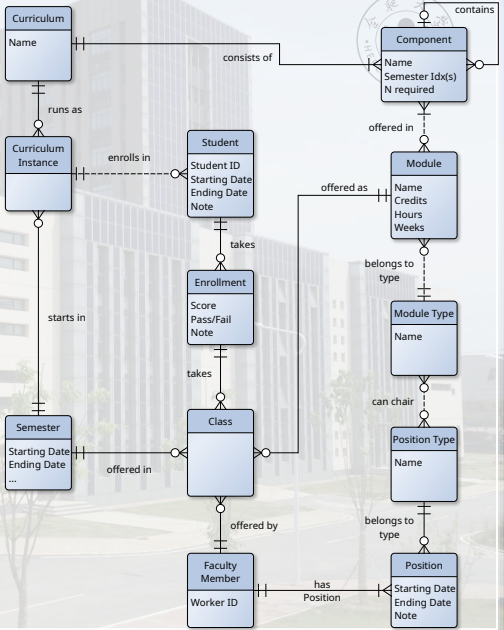
Bigger Picture, Part 2

- So hatten wir das ja auch modelliert.
- Es gibt auch zwei verschiedene Arten, wie Lehrer und Studenten interagieren können:
- Studenten können sich in Klassen von Professoren einschreiben und Professoren können MSc- und BSc-Arbeiten betreuen.
- Wir hätten also zwei Mengen von Interaktionen die von zwei verschiedenen Formen von Qualifikationen auf Seiten der Lehrer abhängen.
- Solche Situation sind immer schlecht.
- Sie machen unsere Modelle kompliziert.



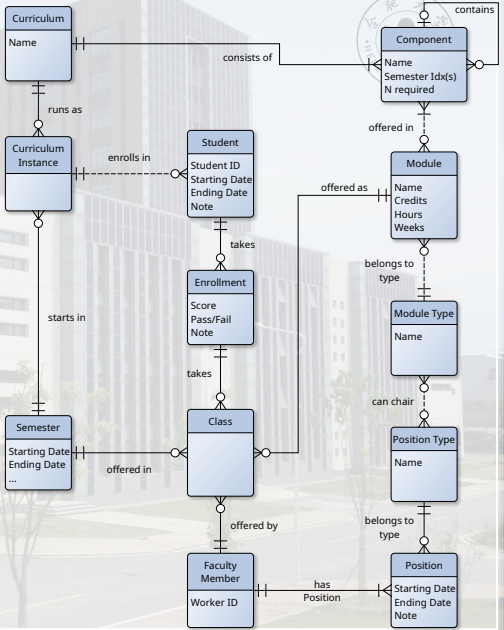
Bigger Picture, Part 2

- Es gibt auch zwei verschiedene Arten, wie Lehrer und Studenten interagieren können:
- Studenten können sich in Klassen von Professoren einschreiben und Professoren können MSc- und BSc-Arbeiten betreuen.
- Wir hätten also zwei Mengen von Interaktionen die von zwei verschiedenen Formen von Qualifikationen auf Seiten der Lehrer abhängen.
- Solche Situation sind immer schlecht.
- Sie machen unsere Modelle kompliziert.
- Sie riechen förmlich nach Redundanz.



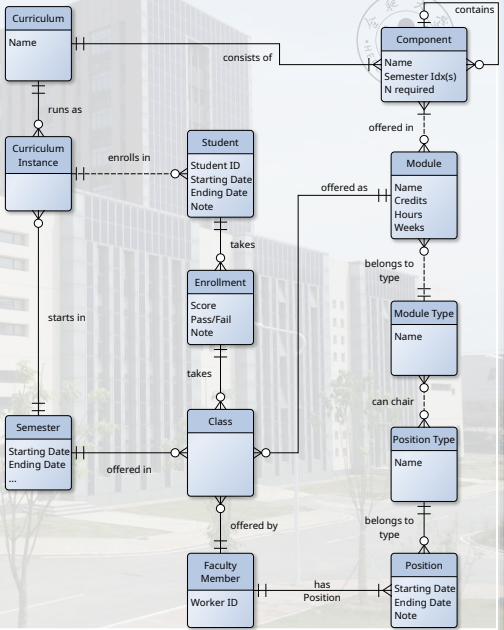
Bigger Picture, Part 2

- Studenten können sich in Klassen von Professoren einschreiben und Professoren können MSc- und BSc-Arbeiten betreuen.
- Wir hätten also zwei Mengen von Interaktionen die von zwei verschiedenen Formen von Qualifikationen auf Seiten der Lehrer abhängen.
- Solche Situation sind immer schlecht.
- Sie machen unsere Modelle kompliziert.
- Sie riechen förmlich nach Redundanz.
- Wir haben also die Idee, beide Interaktionen zu vereinigen.



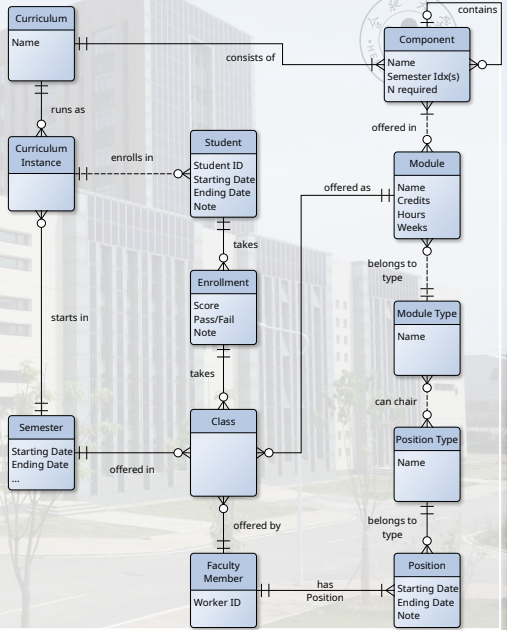
Bigger Picture, Part 2

- Wir hätten also zwei Mengen von Interaktionen die von zwei verschiedenen Formen von Qualifikationen auf Seiten der Lehrer abhängen.
- Solche Situation sind immer schlecht.
- Sie machen unsere Modelle kompliziert.
- Sie riechen förmlich nach Redundanz.
- Wir haben also die Idee, beide Interaktionen zu vereinigen.
- Es ist klar, dass nicht alle Lehre alle Arten von Modulen unterrichten können.



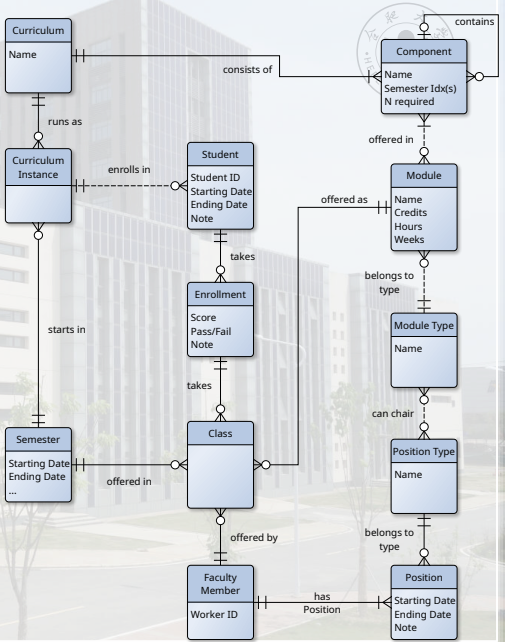
Bigger Picture, Part 2

- Solche Situation sind immer schlecht.
- Sie machen unsere Modelle kompliziert.
- Sie riechen förmlich nach Redundanz.
- Wir haben also die Idee, beide Interaktionen zu vereinigen.
- Es ist klar, dass nicht alle Lehre alle Arten von Modulen unterrichten können.
- Vielleicht erlaubt unsere Universität nur Professoren, die Kernmodule eines Studiengangs zu unterichten, während jüngere Vorlesende Wahlfächer machen dürfen.



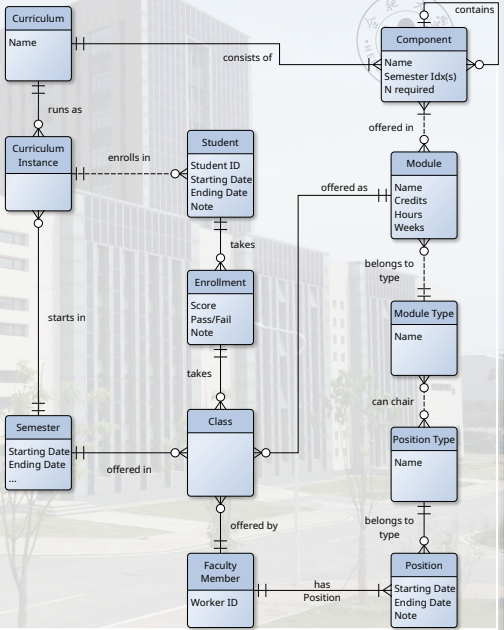
Bigger Picture, Part 2

- Sie machen unsere Modelle kompliziert.
- Sie riechen förmlich nach Redundanz.
- Wir haben also die Idee, beide Interaktionen zu vereinigen.
- Es ist klar, dass nicht alle Lehre alle Arten von Modulen unterrichten können.
- Vielleicht erlaubt unsere Universität nur Professoren, die Kernmodule eines Studiengangs zu unterichten, während jüngere Vorlesende Wahlfächer machen dürfen.
- Das kann man darstellen, in dem man Positions-Typen und Modul-Typen in Beziehung setzt.



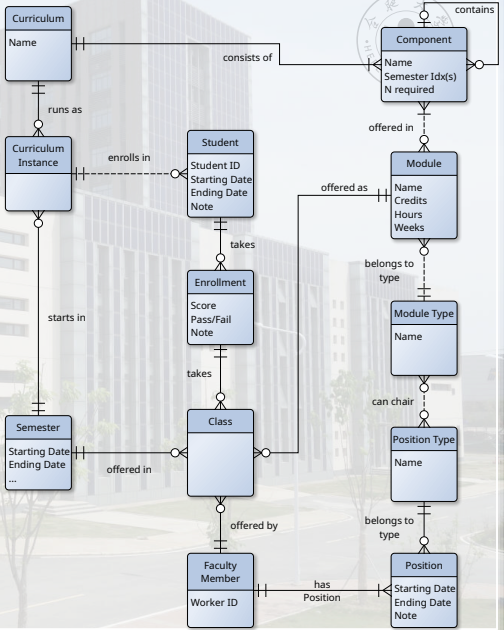
Bigger Picture, Part 2

- Sie riechen förmlich nach Redundanz.
- Wir haben also die Idee, beide Interaktionen zu vereinigen.
- Es ist klar, dass nicht alle Lehre alle Arten von Modulen unterrichten können.
- Vielleicht erlaubt unsere Universität nur Professoren, die Kernmodule eines Studiengangs zu unterichten, während jüngere Vorlesende Wahlfächer machen dürfen.
- Das kann man darstellen, in dem man Positions-Typen und Modul-Typen in Beziehung setzt.
- Es kann aber auch sein, dass manche Module bestimmte Zertifikate voraussetzen, vielleicht Chemnie-Sicherheits-Zertifikate oder so etwas.



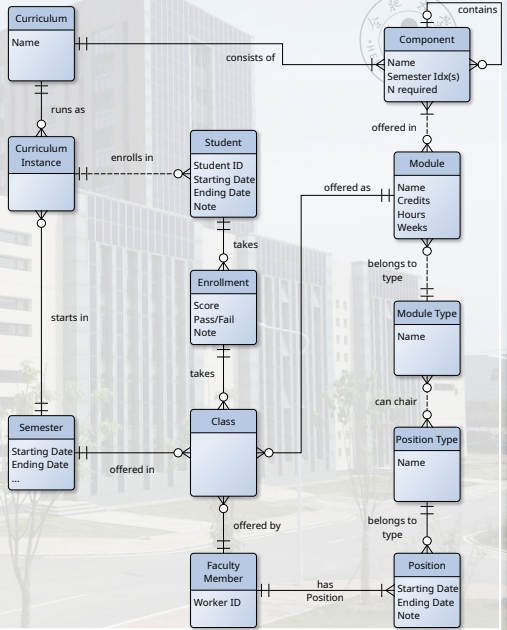
Bigger Picture, Part 2

- Es ist klar, dass nicht alle Lehre alle Arten von Modulen unterrichten können.
- Vielleicht erlaubt unsere Universität nur Professoren, die Kernmodule eines Studiengangs zu unterichten, während jüngere Vorlesende Wahlfächer machen dürfen.
- Das kann man darstellen, in dem man Positions-Typen und Modul-Typen in Beziehung setzt.
- Es kann aber auch sein, dass manche Module bestimmte Zertifikate voraussetzen, vielleicht Chemnie-Sicherheits-Zertifikate oder so etwas.
- Das passt dann nicht so gut zu unserem Modell, weil „Chemie-Sicherheits-Zertifiziert“ ist keine Position.



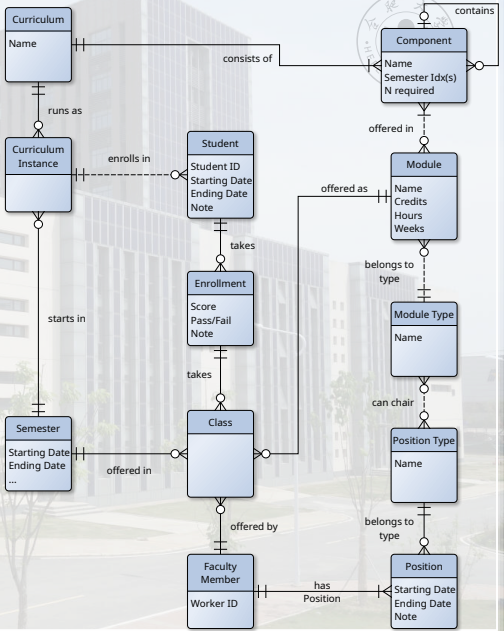
Bigger Picture, Part 2

- Vielleicht erlaubt unsere Universität nur Professoren, die Kernmodule eines Studiengangs zu unterichten, während jüngere Vorlesende Wahlfächer machen dürfen.
- Das kann man darstellen, in dem man Positions-Typen und Modul-Typen in Beziehung setzt.
- Es kann aber auch sein, dass manche Module bestimmte Zertifikate voraussetzen, vielleicht Chemnie-Sicherheits-Zertifikate oder so etwas.
- Das passt dann nicht so gut zu unserem Modell, weil „Chemie-Sicherheits-Zertifiziert“ ist keine Position.
- Das selbe gilt für „Master-Betreuer“.



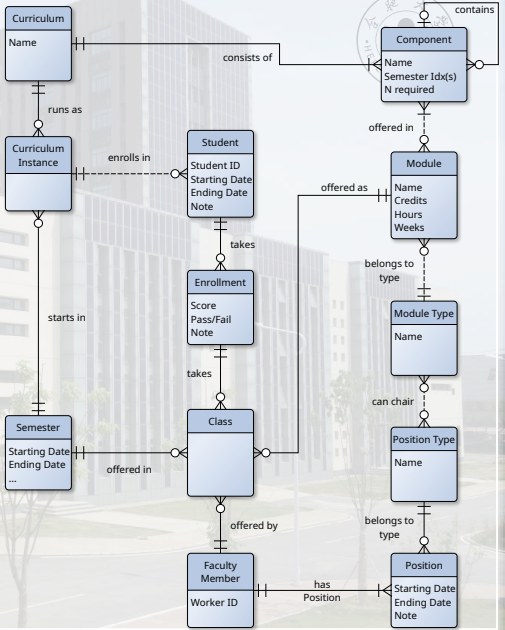
Bigger Picture, Part 2

- Vielleicht erlaubt unsere Universität nur Professoren, die Kernmodule eines Studiengangs zu unterichten, während jüngere Vorlesende Wahlfächer machen dürfen.
- Das kann man darstellen, in dem man Positions-Typen und Modul-Typen in Beziehung setzt.
- Es kann aber auch sein, dass manche Module bestimmte Zertifikate voraussetzen, vielleicht Chemnie-Sicherheits-Zertifikate oder so etwas.
- Das passt dann nicht so gut zu unserem Modell, weil „Chemie-Sicherheits-Zertifiziert“ ist keine Position.
- Das selbe gilt für „Master-Betreuer“.
- Warum eigentlich nicht?



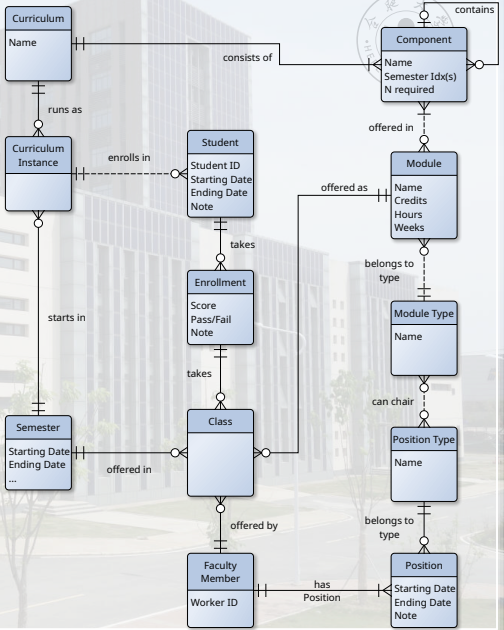
Bigger Picture, Part 2

- Das kann man darstellen, in dem man Positions-Typen und Modul-Typen in Beziehung setzt.
- Es kann aber auch sein, dass manche Module bestimmte Zertifikate voraussetzen, vielleicht Chemnie-Sicherheits-Zertifikate oder so etwas.
- Das passt dann nicht so gut zu unserem Modell, weil „Chemie-Sicherheits-Zertifiziert“ ist keine Position.
- Das selbe gilt für „Master-Betreuer“.
- Warum eigentlich nicht?
- Wir wäre es damit, wenn wir erlauben, dass eine Person mehrere Positionen gleichzeitig haben kann.



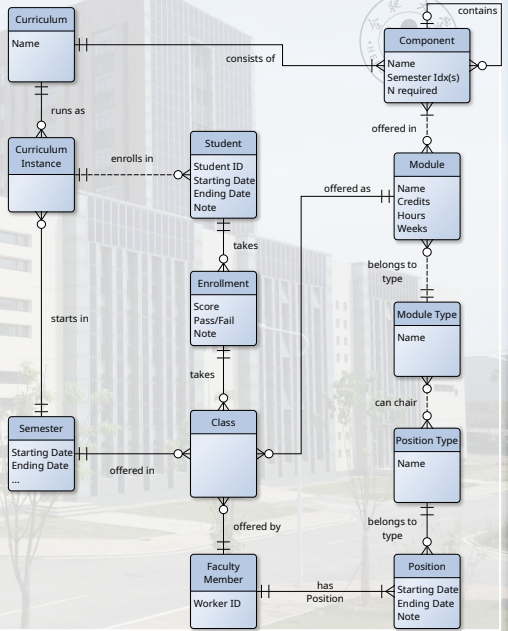
Bigger Picture, Part 2

- Es kann aber auch sein, dass manche Module bestimmte Zertifikate voraussetzen, vielleicht Chemnie-Sicherheits-Zertifikate oder so etwas.
- Das passt dann nicht so gut zu unserem Modell, weil „Chemie-Sicherheits-Zertifiziert“ ist keine Position.
- Das selbe gilt für „Master-Betreuer“.
- Warum eigentlich nicht?
- Wir wäre es damit, wenn wir erlauben, dass eine Person mehrere Positionen gleichzeitig haben kann.
- Wir haben ja schon die traditionellen laufbahnbezogenen Positionen modelliert.



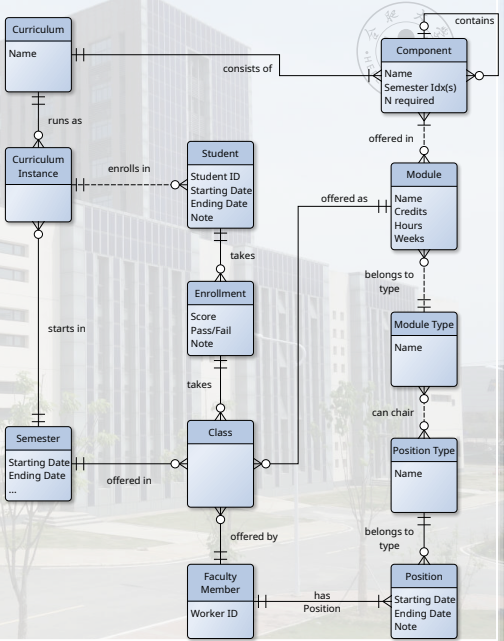
Bigger Picture, Part 2

- Das passt dann nicht so gut zu unserem Modell, weil „Chemie-Sicherheits-Zertifiziert“ ist keine Position.
- Das selbe gilt für „Master-Betreuer“.
- Warum eigentlich nicht?
- Wir wäre es damit, wenn wir erlauben, dass eine Person mehrere Positionen gleichzeitig haben kann.
- Wir haben ja schon die traditionellen laufbahnbezogenen Positionen modelliert.
- Zusätzlich könnten wir Positionstypen wie „Chemie-Sicherheits-Zertifiziert“ und „Master-Betreuer“ und was auch immer wir brauchen einführen.



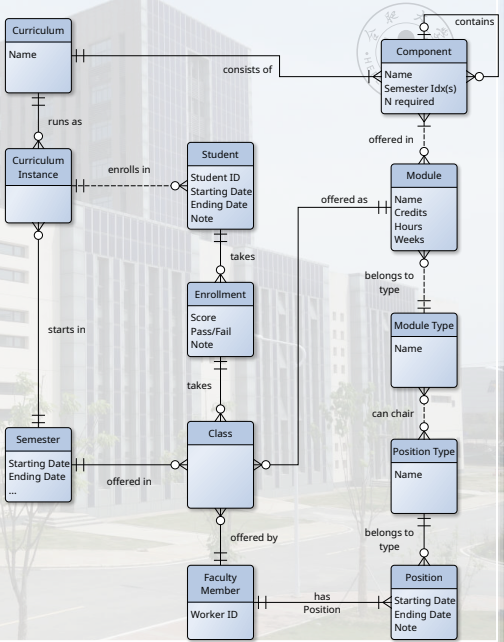
Bigger Picture, Part 2

- Das selbe gilt für „Master-Betreuer“.
- Warum eigentlich nicht?
- Wir wäre es damit, wenn wir erlauben, dass eine Person mehrere Positionen gleichzeitig haben kann.
- Wir haben ja schon die traditionellen laufbahnbezogenen Positionen modelliert.
- Zusätzlich könnten wir Positionstypen wie „Chemie-Sicherheits-Zertifiziert“ und „Master-Betreuer“ und was auch immer wir brauchen einführen.
- Das Positionen durch Start- und Enddatum limitert sind ist kein Problem.



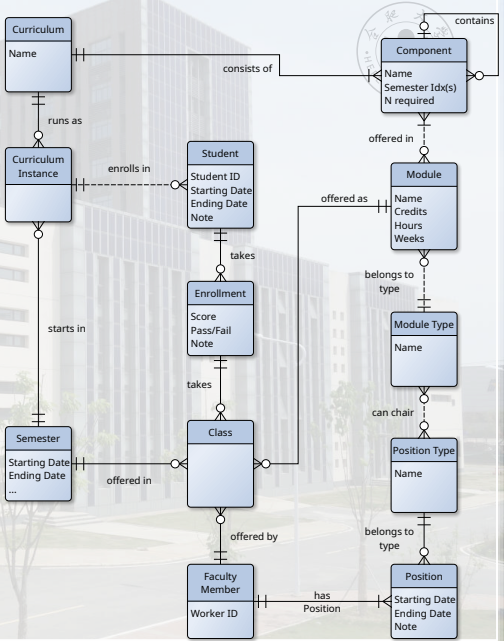
Bigger Picture, Part 2

- Warum eigentlich nicht?
- Wir wäre es damit, wenn wir erlauben, dass eine Person mehrere Positionen gleichzeitig haben kann.
- Wir haben ja schon die traditionellen laufbahnbezogenen Positionen modelliert.
- Zusätzlich könnten wir Positionstypen wie „Chemie-Sicherheits-Zertifiziert“ und „Master-Betreuer“ und was auch immer wir brauchen einführen.
- Das Positionen durch Start- und Enddatum limitert sind ist kein Problem.
- Das ist genau das Feature, was wir sowieso wollen.



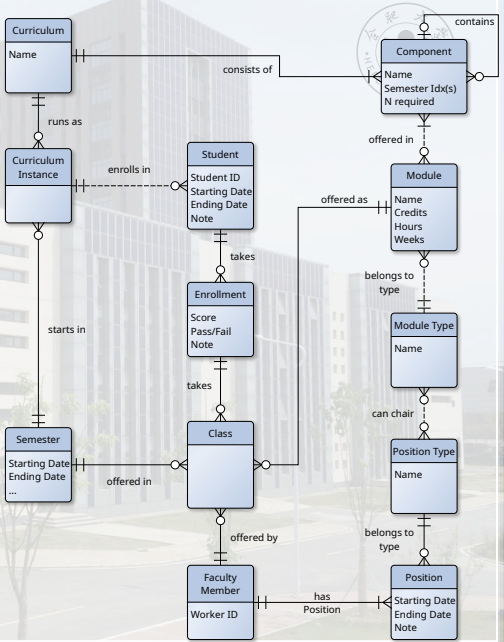
Bigger Picture, Part 2

- Wir haben ja schon die traditionellen laufbahnbezogenen Positionen modelliert.
- Zusätzlich könnten wir Positionstypen wie „Chemie-Sicherheits-Zertifiziert“ und „Master-Betreuer“ und was auch immer wir brauchen einführen.
- Das Positionen durch Start- und Enddatum limitert sind ist kein Problem.
- Das ist genau das Feature, was wir sowieso wollen.
- Wenn wir unsere *Position* und *Position Type* Entitätstypen so verwenden, dann könnten wir auch andere Funktionen Dekan, Vizedekan für Lehre, Teamleiter, usw. ebenfalls mit abdecken.



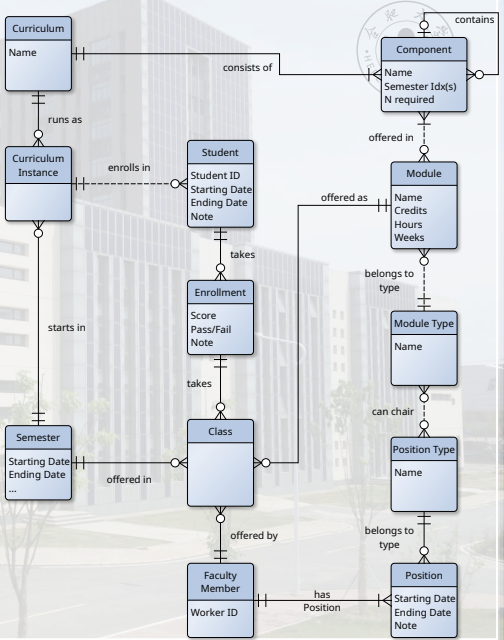
Bigger Picture, Part 2

- Zusätzlich könnten wir Positionstypen wie „Chemie-Sicherheits-Zertifiziert“ und „Master-Betreuer“ und was auch immer wir brauchen einführen.
- Das Positionen durch Start- und Enddatum limitert sind ist kein Problem.
- Das ist genau das Feature, was wir sowieso wollen.
- Wenn wir unsere *Position* und *Position Type* Entitätstypen so verwenden, dann könnten wir auch andere Funktionen Dekan, Vizedekan für Lehre, Teamleiter, usw. ebenfalls mit abdecken.
- Diese Positionen könnten dann auch verwendet werden, um zu entscheiden, welche Daten eine Person ändern kann.



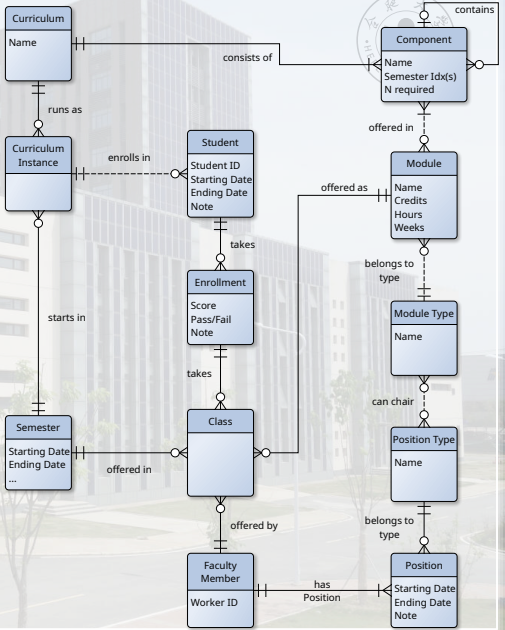
Bigger Picture, Part 2

- Das Positionen durch Start- und Enddatum limitert sind ist kein Problem.
- Das ist genau das Feature, was wir sowieso wollen.
- Wenn wir unsere *Position* und *Position Type* Entitätstypen so verwenden, dann könnten wir auch andere Funktionen Dekan, Vizedekan für Lehre, Teamleiter, usw. ebenfalls mit abdecken.
- Diese Positionen könnten dann auch verwendet werden, um zu entscheiden, welche Daten eine Person ändern kann.
- Zurück zur Beziehung zwischen Lehre und Position.



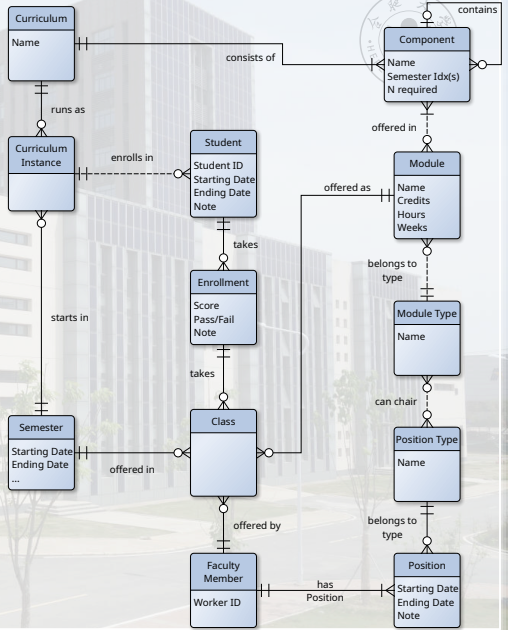
Bigger Picture, Part 2

- Das ist genau das Feature, was wir sowieso wollen.
- Wenn wir unsere *Position* und *Position Type* Entitätstypen so verwenden, dann könnten wir auch andere Funktionen Dekan, Vizedekan für Lehre, Teamleiter, usw. ebenfalls mit abdecken.
- Diese Positionen könnten dann auch verwendet werden, um zu entscheiden, welche Daten eine Person ändern kann.
- Zurück zur Beziehung zwischen Lehre und Position.
- Jede *Module Type*-Entität kann nun mit einer oder mehreren *Position Type* Entitäten verbunden sein, die ein Lehrer haben muss, um sie zu unterrichten.



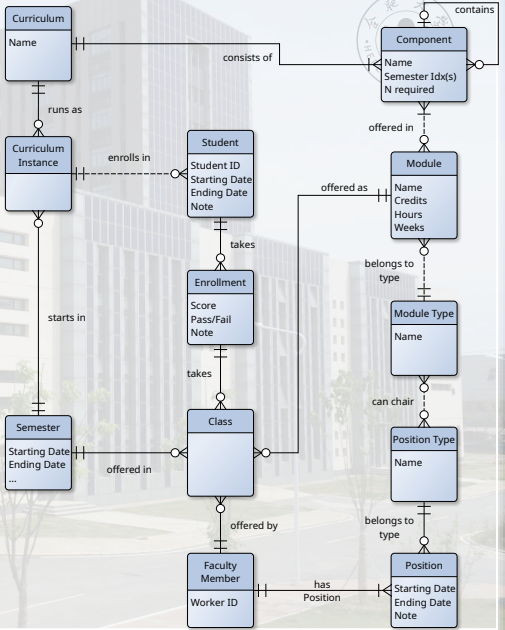
Bigger Picture, Part 2

- Wenn wir unsere *Position* und *Position Type* Entitätstypen so verwenden, dann könnten wir auch andere Funktionen Dekan, Vizedekan für Lehre, Teamleiter, usw. ebenfalls mit abdecken.
- Diese Positionen könnten dann auch verwendet werden, um zu entscheiden, welche Daten eine Person ändern kann.
- Zurück zur Beziehung zwischen Lehre und Position.
- Jede *Module Type*-Entität kann nun mit einer oder mehreren *Position Type* Entitäten verbunden sein, die ein Lehrer haben muss, um sie zu unterrichten.
- Jeder Positionstyp kann die Voraussetzung zum Lehren beliebig vieler Modultypen sein.



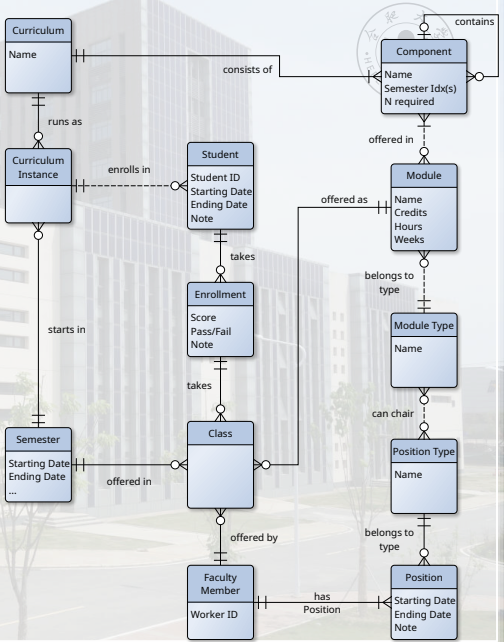
Bigger Picture, Part 2

- Diese Positionen könnten dann auch verwendet werden, um zu entscheiden, welche Daten eine Person ändern kann.
- Zurück zur Beziehung zwischen Lehre und Position.
- Jede *Module Type*-Entität kann nun mit einer oder mehreren *Position Type* Entitäten verbunden sein, die ein Lehrer haben muss, um sie zu unterrichten.
- Jeder Positionstyp kann die Voraussetzung zum Lehren beliebig vieler Modultypen sein.
- Dieser Ansatz funktioniert, weil MSc- und BSc-Projekte im Grunde nur Module sind, die zu besondere Modultypen gehören.



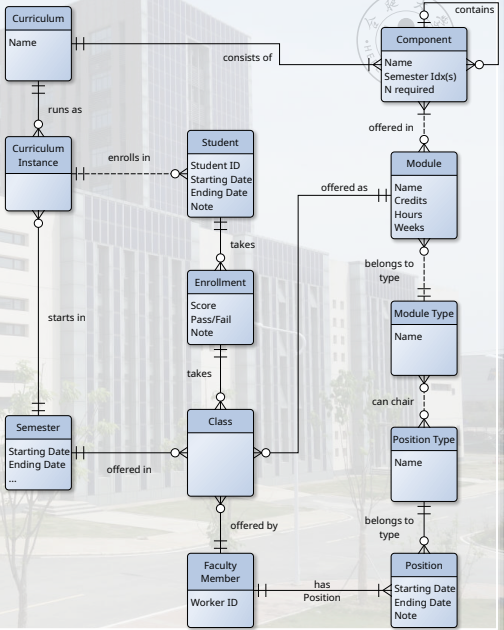
Bigger Picture, Part 2

- Zurück zur Beziehung zwischen Lehre und Position.
- Jede *Module Type*-Entität kann nun mit einer oder mehreren *Position Type* Entitäten verbunden sein, die ein Lehrer haben muss, um sie zu unterrichten.
- Jeder Positionstyp kann die Voraussetzung zum Lehren beliebig vieler Modultypen sein.
- Dieser Ansatz funktioniert, weil MSc- und BSc-Projekte im Grunde nur Module sind, die zu besondere Modultypen gehören.
- Sie sind automatisch durch dieses System mit abgedeckt.



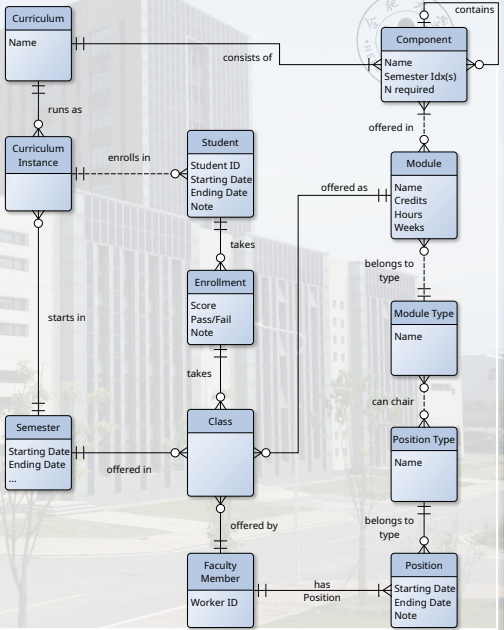
Bigger Picture, Part 2

- Jede *Module Type*-Entität kann nun mit einer oder mehreren *Position Type* Entitäten verbunden sein, die ein Lehrer haben muss, um sie zu unterrichten.
- Jeder Positionstyp kann die Voraussetzung zum Lehren beliebig vieler Modultypen sein.
- Dieser Ansatz funktioniert, weil MSc- und BSc-Projekte im Grunde nur Module sind, die zu besondere Modultypen gehören.
- Sie sind automatisch durch dieses System mit abgedeckt.
- Eine Fakultät unserer Universität kann verschiedene Studiengänge (EN: *curricula*) anbieten, z. B. einen Master of Computer Science oder einen Bachelor of Engineering in Computer Science.



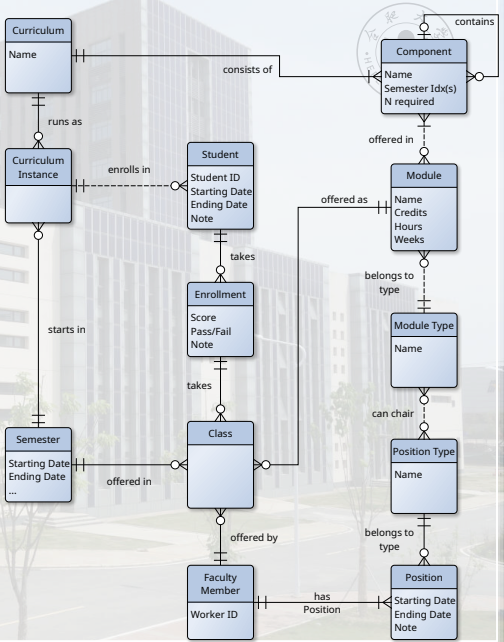
Bigger Picture, Part 2

- Jede *Module Type*-Entität kann nun mit einer oder mehreren *Position Type* Entitäten verbunden sein, die ein Lehrer haben muss, um sie zu unterrichten.
- Jeder Positionstyp kann die Voraussetzung zum Lehren beliebig vieler Modultypen sein.
- Dieser Ansatz funktioniert, weil MSc- und BSc-Projekte im Grunde nur Module sind, die zu besondere Modultypen gehören.
- Sie sind automatisch durch dieses System mit abgedeckt.
- Eine Fakultät unserer Universität kann verschiedene Studiengänge (EN: *curricula*) anbieten, z. B. einen Master of Computer Science oder einen Bachelor of Engineering in Computer Science.



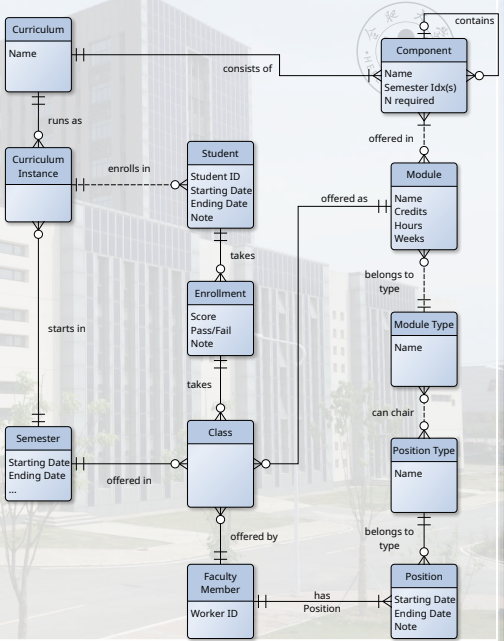
Bigger Picture, Part 2

- Jeder Positionstyp kann die Voraussetzung zum Lehren beliebig vieler Modultypen sein.
- Dieser Ansatz funktioniert, weil MSc- und BSc-Projekte im Grunde nur Module sind, die zu besondere Modultypen gehören.
- Sie sind automatisch durch dieses System mit abgedeckt.
- Eine Fakultät unserer Universität kann verschiedene Studiengänge (EN: *curricula*) anbieten, z. B. einen Master of Computer Science oder einen Bachelor of Engineering in Computer Science.
- Ursprünglich nahmen wir an, dass jeder Studiengang aus verschiedenen Modulen besteht und jedes Modul in einem bestimmten Semester des Studiengangs stattfindet.



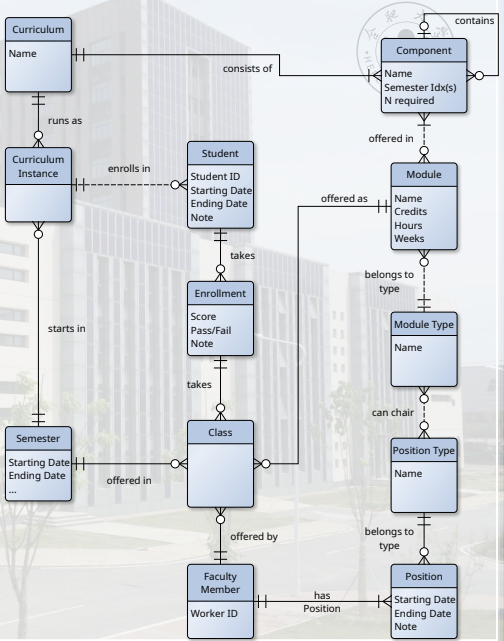
Bigger Picture, Part 2

- Sie sind automatisch durch dieses System mit abgedeckt.
- Eine Fakultät unserer Universität kann verschiedene Studiengänge (EN: *curricula*) anbieten, z. B. einen Master of Computer Science oder einen Bachelor of Engineering in Computer Science.
- Ursprünglich nahmen wir an, dass jeder Studiengang aus verschiedenen Modulen besteht und jedes Modul in einem bestimmten Semester des Studiengangs stattfindet.
- Wir stellen fest, dass das zu zwei Problemen führt:
- Erstens haben wir manchmal Wahlmodule, also Situationen, wo ein Student eine oder zwei Module aus einer Menge von drei oder vier Modulen auswählen muss.



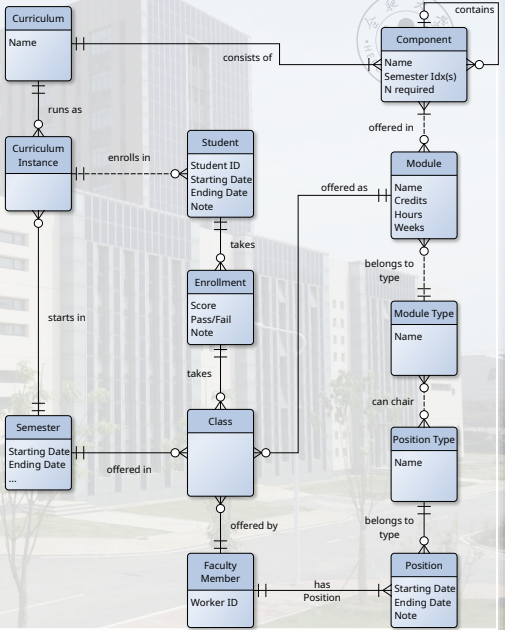
Bigger Picture, Part 2

- Sie sind automatisch durch dieses System mit abgedeckt.
- Eine Fakultät unserer Universität kann verschiedene Studiengänge (EN: *curricula*) anbieten, z. B. einen Master of Computer Science oder einen Bachelor of Engineering in Computer Science.
- Ursprünglich nahmen wir an, dass jeder Studiengang aus verschiedenen Modulen besteht und jedes Modul in einem bestimmten Semester des Studiengangs stattfindet.
- Wir stellen fest, dass das zu zwei Problemen führt:
- Erstens haben wir manchmal Wahlmodule, also Situationen, wo ein Student eine oder zwei Module aus einer Menge von drei oder vier Modulen auswählen muss.



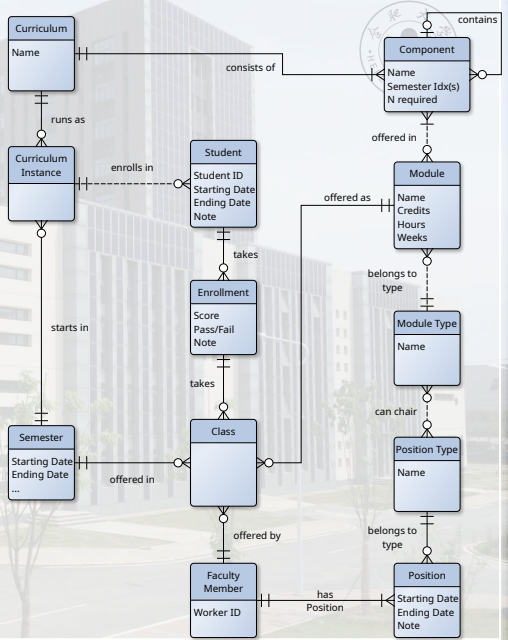
Bigger Picture, Part 2

- Eine Fakultät unserer Universität kann verschiedene Studiengänge (EN: *curricula*) anbieten, z. B. einen Master of Computer Science oder einen Bachelor of Engineering in Computer Science.
- Ursprünglich nahmen wir an, dass jeder Studiengang aus verschiedenen Modulen besteht und jedes Modul in einem bestimmten Semester des Studiengangs stattfindet.
- Wir stellen fest, dass das zu zwei Problemen führt:
- Erstens haben wir manchmal Wahlmodule, also Situationen, wo ein Student eine oder zwei Module aus einer Menge von drei oder vier Modulen auswählen muss.
- Zweitens kann es für einen Studiengang verschiedene Vertiefungsrichtungen geben.



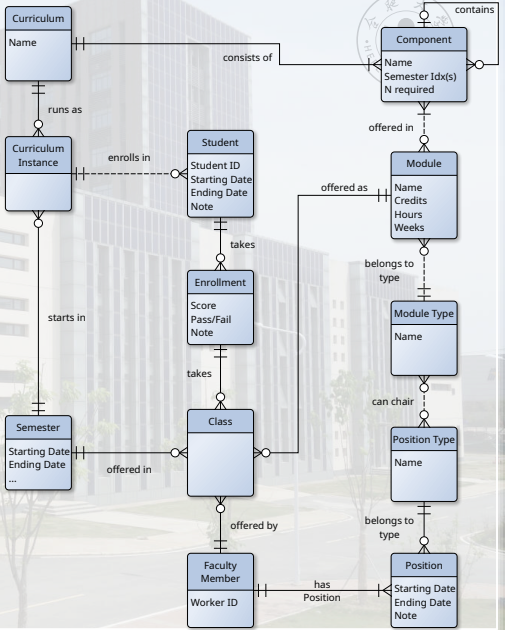
Bigger Picture, Part 2

- Wir stellen fest, dass das zu zwei Problemen führt:
- Erstens haben wir manchmal Wahlmodule, also Situationen, wo ein Student eine oder zwei Module aus einer Menge von drei oder vier Modulen auswählen muss.
- Zweitens kann es für einen Studiengang verschiedene Vertiefungsrichtungen geben.
- Ein Master in Computer Science könnte z. B. die Vertiefungsrichtungen Artificial Intelligence (AI), Datenbanken und Computer Security anbieten.
- Jede Vertifungsrichtung kann (rekursiv) eigene andere Pflicht- und Wahlfächer anbieten.



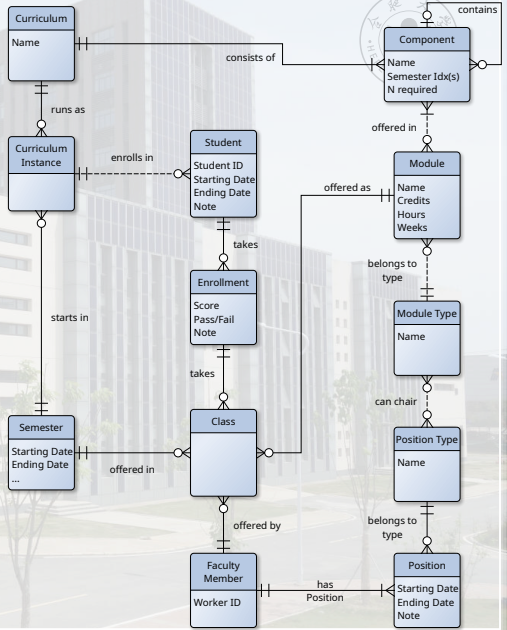
Bigger Picture, Part 2

- Wir stellen fest, dass das zu zwei Problemen führt:
- Erstens haben wir manchmal Wahlmodule, also Situationen, wo ein Student eine oder zwei Module aus einer Menge von drei oder vier Modulen auswählen muss.
- Zweitens kann es für einen Studiengang verschiedene Vertiefungsrichtungen geben.
- Ein Master in Computer Science könnte z. B. die Vertiefungsrichtungen Artificial Intelligence (AI), Datenbanken und Computer Security anbieten.
- Jede Vertifungsrichtung kann (rekursiv) eigene andere Pflicht- und Wahlfächer anbieten.
- Wir versuchen das zu modellieren, in dem wir den Entitätstyp *Component* einführen.



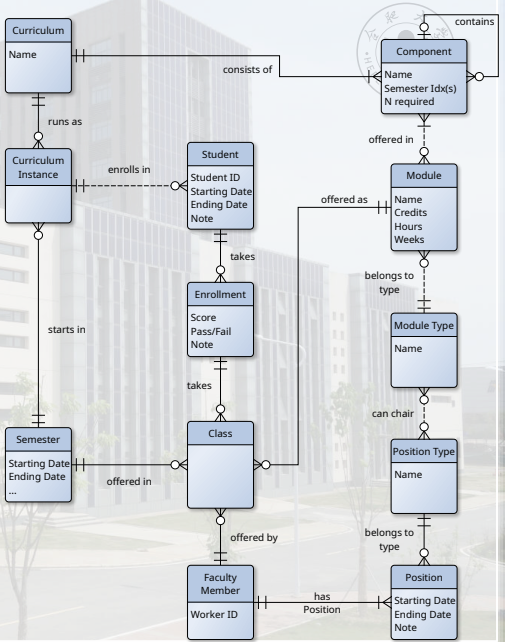
Bigger Picture, Part 2

- Zweitens kann es für einen Studiengang verschiedene Vertiefungsrichtungen geben.
- Ein Master in Computer Science könnte z. B. die Vertiefungsrichtungen Artificial Intelligence (AI), Datenbanken und Computer Security anbieten.
- Jede Vertifungsrichtung kann (rekursiv) eigene andere Pflicht- und Wahlfächer anbieten.
- Wir versuchen das zu modellieren, in dem wir den Entitätstyp *Component* einführen.
- Ein Studiengang besteht aus mindestens einer Komponente (EN: *component*) und jede Komponente gehört zu genau einem Studiengang.



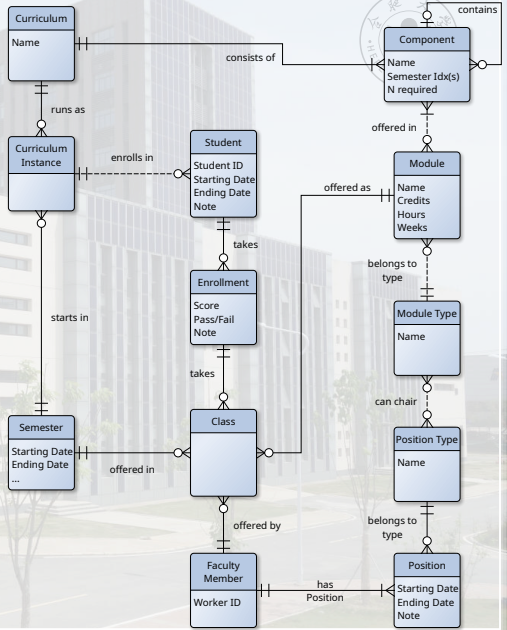
Bigger Picture, Part 2

- Zweitens kann es für einen Studiengang verschiedene Vertiefungsrichtungen geben.
- Ein Master in Computer Science könnte z. B. die Vertiefungsrichtungen Artificial Intelligence (AI), Datenbanken und Computer Security anbieten.
- Jede Vertifungsrichtung kann (rekursiv) eigene andere Pflicht- und Wahlfächer anbieten.
- Wir versuchen das zu modellieren, in dem wir den Entitätstyp *Component* einführen.
- Ein Studiengang besteht aus mindestens einer Komponente (EN: *component*) und jede Komponente gehört zu genau einem Studiengang.
- Eine Komponente kann ist Unter-Komponente von höchstens einer (= von einer oder von keiner) anderen Komponente.



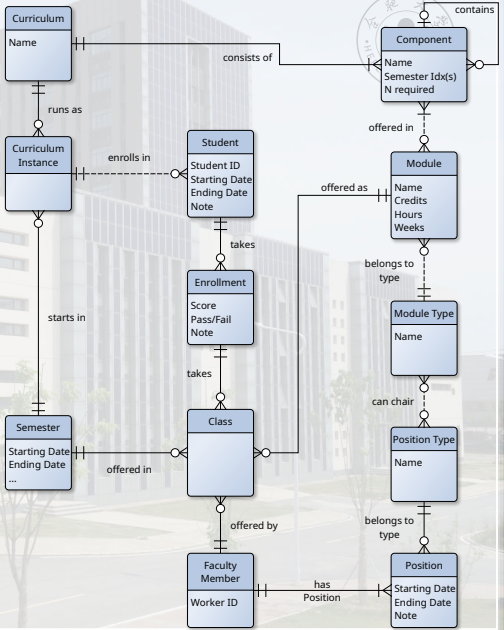
Bigger Picture, Part 2

- Ein Master in Computer Science könnte z. B. die Vertiefungsrichtungen Artificial Intelligence (AI), Datenbanken und Computer Security anbieten.
- Jede Vertifungsrichtung kann (rekursiv) eigene andere Pflicht- und Wahlfächer anbieten.
- Wir versuchen das zu modellieren, in dem wir den Entitätstyp *Component* einführen.
- Ein Studiengang besteht aus mindestens einer Komponente (EN: *component*) und jede Komponente gehört zu genau einem Studiengang.
- Eine Komponente kann ist Unter-Komponente von höchstens einer (= von einer oder von keiner) anderen Komponente.
- Jede Komponente kann eine beliebige Anzahl von Unter-Komponenten haben.



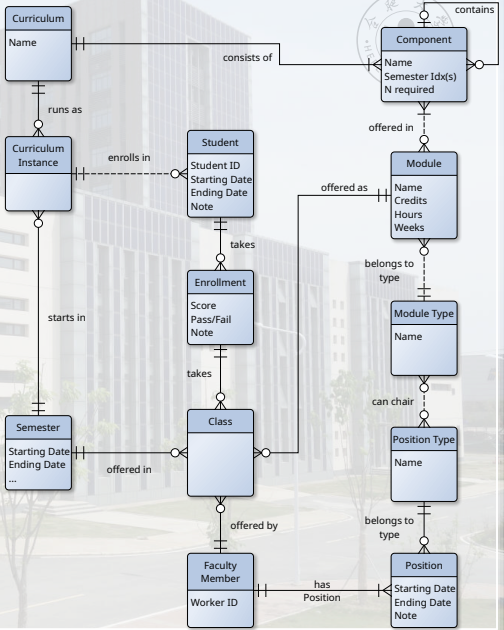
Bigger Picture, Part 2

- Jede Vertifungsrichtung kann (rekursiv) eigene andere Pflicht- und Wahlfächer anbieten.
- Wir versuchen das zu modellieren, in dem wir den Entitätstyp *Component* einführen.
- Ein Studiengang besteht aus mindestens einer Komponente (EN: *component*) und jede Komponente gehört zu genau einem Studiengang.
- Eine Komponente kann ist Unter-Komponente von höchstens einer (= von einer oder von keiner) anderen Komponente.
- Jede Komponente kann eine beliebige Anzahl von Unter-Komponenten haben.
- Jede Komponente hat einen Namen und eine Menge von Semester-Indizies in denen sie stattfindet (z. B. im 7. und 8. Semester).



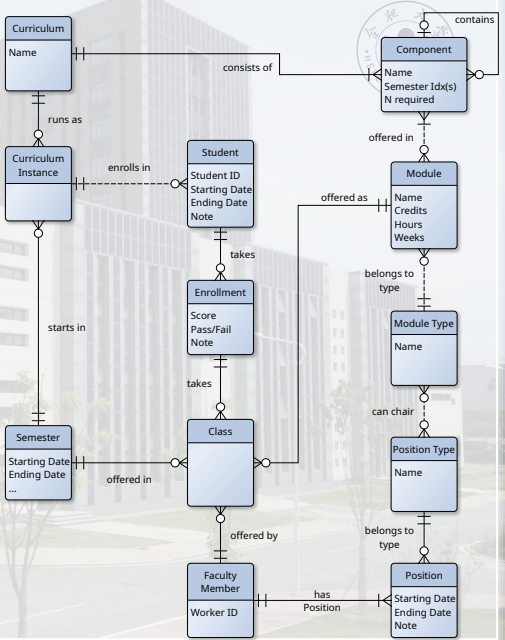
Bigger Picture, Part 2

- Wir versuchen das zu modellieren, in dem wir den Entitätstyp *Component* einführen.
- Ein Studiengang besteht aus mindestens einer Komponente (EN: *component*) und jede Komponente gehört zu genau einem Studiengang.
- Eine Komponente kann ist Unter-Komponente von höchstens einer (= von einer oder von keiner) anderen Komponente.
- Jede Komponente kann eine beliebige Anzahl von Unter-Komponenten haben.
- Jede Komponente hat einen Namen und eine Menge von Semester-Indizies in denen sie stattfindet (z. B. im 7. und 8. Semester).
- Ein Modul kann nun in einer oder mehreren solcher Komponenten angeboten werden.



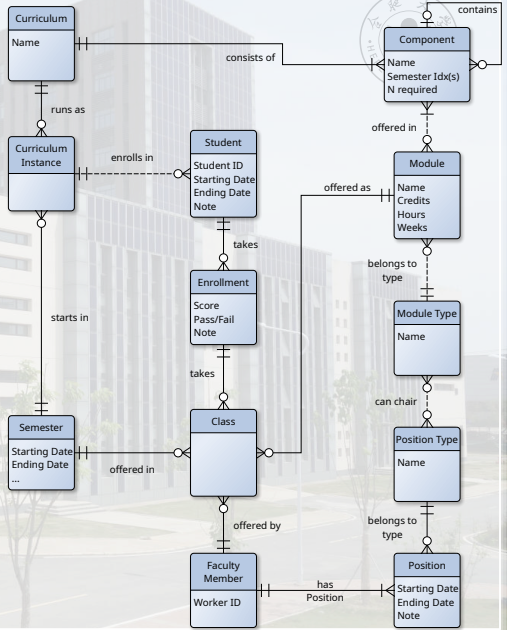
Bigger Picture, Part 2

- Ein Studiengang besteht aus mindestens einer Komponente (EN: *component*) und jede Komponente gehört zu genau einem Studiengang.
- Eine Komponente kann ist Unter-Komponente von höchstens einer (= von einer oder von keiner) anderen Komponente.
- Jede Komponente kann eine beliebige Anzahl von Unter-Komponenten haben.
- Jede Komponente hat einen Namen und eine Menge von Semester-Indizies in denen sie stattfindet (z. B. im 7. und 8. Semester).
- Ein Modul kann nun in einer oder mehreren solcher Komponenten angeboten werden.
- Jede Komponente kann ein oder mehrere Module anbieten.



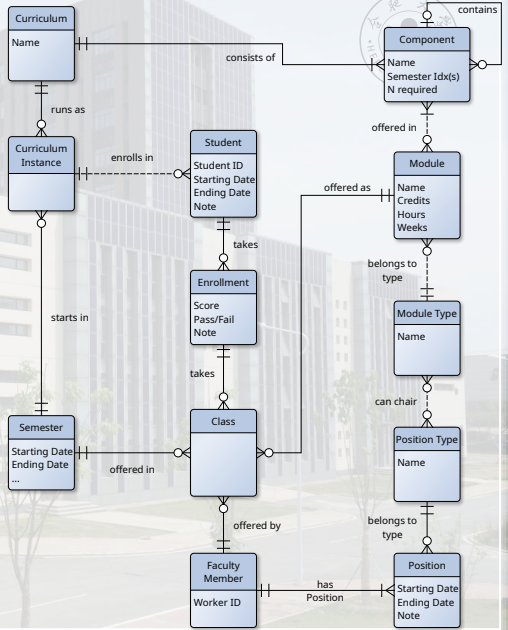
Bigger Picture, Part 2

- Jede Komponente kann eine beliebige Anzahl von Unter-Komponenten haben.
- Jede Komponente hat einen Namen und eine Menge von Semester-Indizies in denen sie stattfindet (z. B. im 7. und 8. Semester).
- Ein Modul kann nun in einer oder mehreren solcher Komponenten angeboten werden.
- Jede Komponente kann ein oder mehrere Module anbieten.
- Jede Komponente kann eine Zahl definierten, die festlegt wie viele ihrer angebotenen Module/Komponenten ein Student abschließen muss.



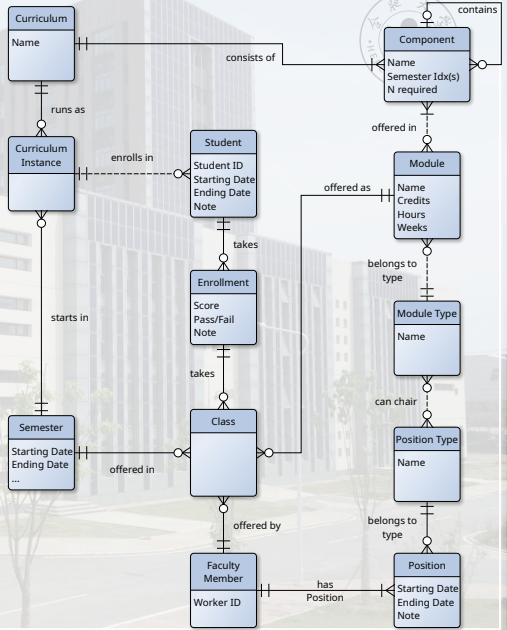
Bigger Picture, Part 2

- Jede Komponente kann eine beliebige Anzahl von Unter-Komponenten haben.
- Jede Komponente hat einen Namen und eine Menge von Semester-Indizies in denen sie stattfindet (z. B. im 7. und 8. Semester).
- Ein Modul kann nun in einer oder mehreren solcher Komponenten angeboten werden.
- Jede Komponente kann ein oder mehrere Module anbieten.
- Jede Komponente kann eine Zahl definierten, die festlegt wie viele ihrer angebotenen Module/Komponenten ein Student abschließen muss.
- Wir könnten nun sagen das. . .



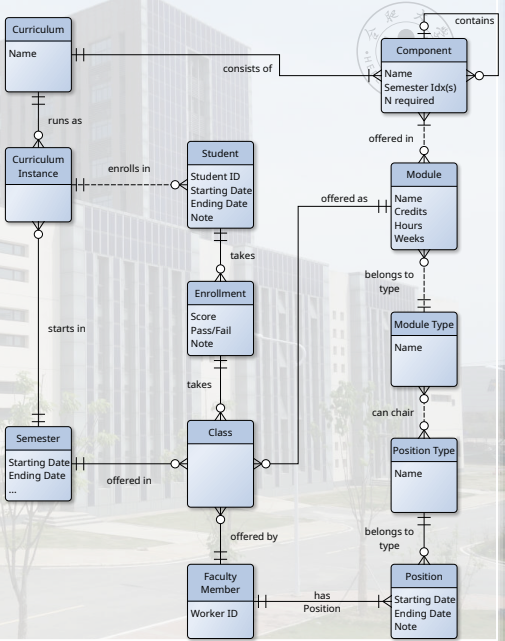
Bigger Picture, Part 2

- Jede Komponente hat einen Namen und eine Menge von Semester-Indizies in denen sie stattfindet (z. B. im 7. und 8. Semester).
- Ein Modul kann nun in einer oder mehreren solcher Komponenten angeboten werden.
- Jede Komponente kann ein oder mehrere Module anbieten.
- Jede Komponente kann eine Zahl definierten, die festlegt wie viele ihrer angebotenen Module/Komponenten ein Student abschließen muss.
- Wir könnten nun sagen das: *„Unter den Komponenten des Master of Computer Science Studiengangs gibt es die Komponente ‘Vertiefungsrichtung’.“*



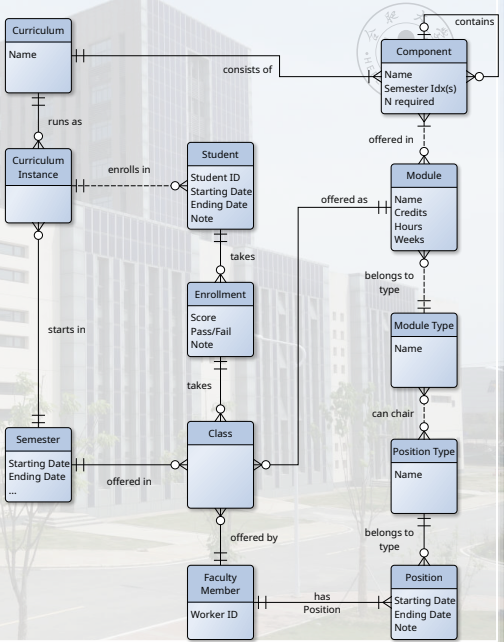
Bigger Picture, Part 2

- Ein Modul kann nun in einer oder mehreren solcher Komponenten angeboten werden.
- Jede Komponente kann ein oder mehrere Module anbieten.
- Jede Komponente kann eine Zahl definierten, die festlegt wie viele ihrer angebotenen Module/Komponenten ein Student abschließen muss.
- Wir könnten nun sagen das: *„Unter den Komponenten des Master of Computer Science Studiengangs gibt es die Komponente ‘Vertiefungsrichtung’. Diese beinhaltet wiederum die Komponenten ‘AI’, ‘Datenbanken’ und ‘Computer Security’.“*



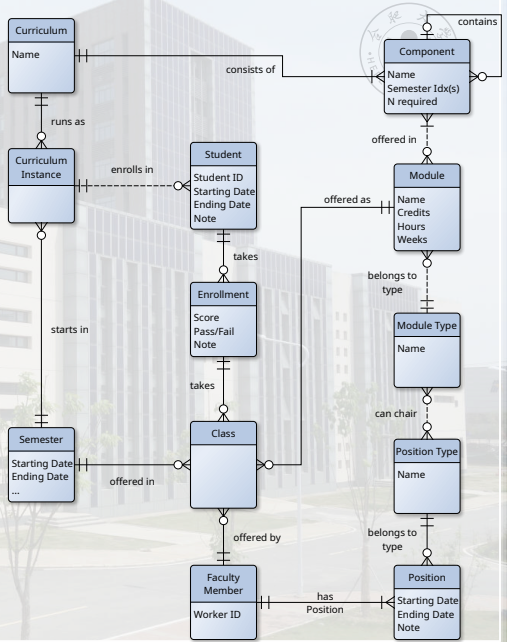
Bigger Picture, Part 2

- Jede Komponente kann ein oder mehrere Module anbieten.
- Jede Komponente kann eine Zahl definierten, die festlegt wie viele ihrer angebotenen Module/Komponenten ein Student abschließen muss.
- Wir könnten nun sagen das: *„Unter den Komponenten des Master of Computer Science Studiengangs gibt es die Komponente ‘Vertiefungsrichtung’. Diese beinhaltet wiederum die Komponenten ‘AI’, ‘Datenbanken’ und ‘Computer Security’. Ein Student muss genau eine solche Komponente abschließen.“*



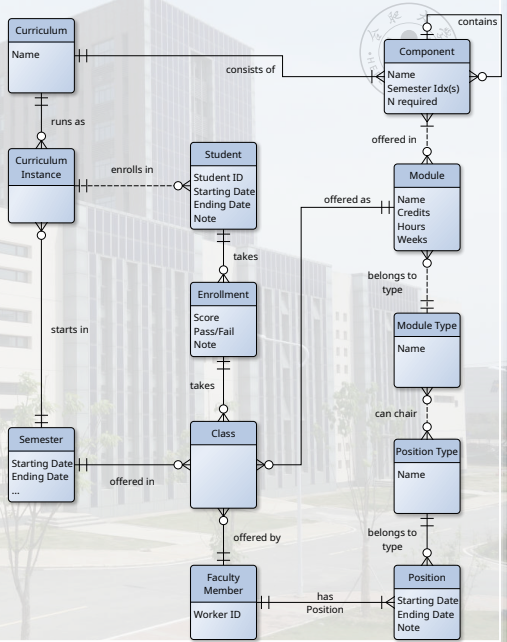
Bigger Picture, Part 2

- Jede Komponente kann eine Zahl definierten, die festlegt wie viele ihrer angebotenen Module/Komponenten ein Student abschließen muss.
- Wir könnten nun sagen das: *„Unter den Komponenten des Master of Computer Science Studiengangs gibt es die Komponente ‘Vertiefungsrichtung’. Diese beinhaltet wiederum die Komponenten ‘AI’, ‘Datenbanken’ und ‘Computer Security’. Ein Student muss genau eine solche Komponente abschließen. In der Vertiefungsrichtung ‘Datenbanken’ wird das Pflichtfach ‘Databases’ angeboten.“*



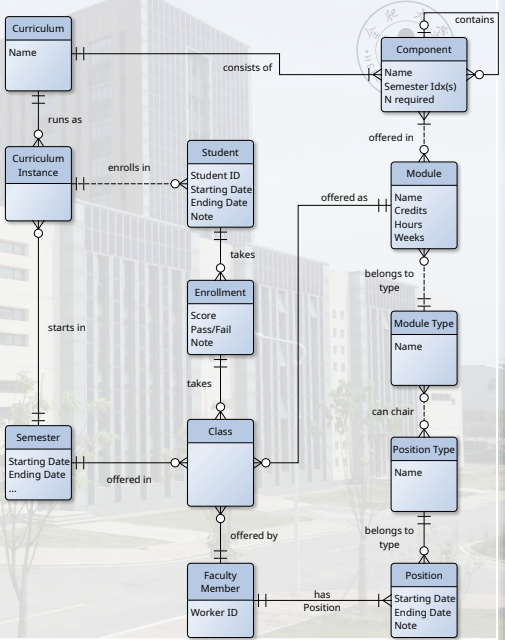
Bigger Picture, Part 2

- Jede Komponente kann eine Zahl definierten, die festlegt wie viele ihrer angebotenen Module/Komponenten ein Student abschließen muss.
- Wir könnten nun sagen das: *„Unter den Komponenten des Master of Computer Science Studiengangs gibt es die Komponente ‘Vertiefungsrichtung’. Diese beinhaltet wiederum die Komponenten ‘AI’, ‘Datenbanken’ und ‘Computer Security’. Ein Student muss genau eine solche Komponente abschließen. In der Vertiefungsrichtung ‘Datenbanken’ wird das Pflichtfach ‘Databases’ angeboten. Ebenso die Komponente ‘Wahlfächer’.“*



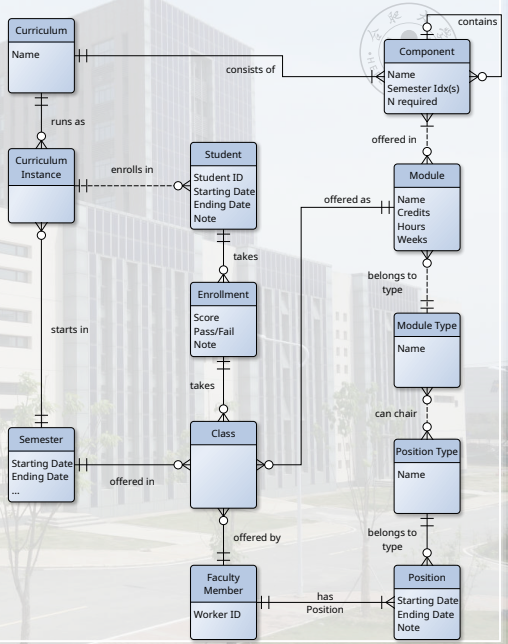
Bigger Picture, Part 2

- Jede Komponente kann eine Zahl definierten, die festlegt wie viele ihrer angebotenen Module/Komponenten ein Student abschließen muss.
- Wir könnten nun sagen das: *„Unter den Komponenten des Master of Computer Science Studiengangs gibt es die Komponente 'Vertiefungsrichtung'. Diese beinhaltet wiederum die Komponenten 'AI', 'Datenbanken' und 'Computer Security'. Ein Student muss genau eine solche Komponente abschließen. In der Vertiefungsrichtung 'Datenbanken' wird das Pflichtfach 'Databases' angeboten. Ebenso die Komponente 'Wahlfächer'. Beide müssen von Studenten abgeschlossen werden.“*



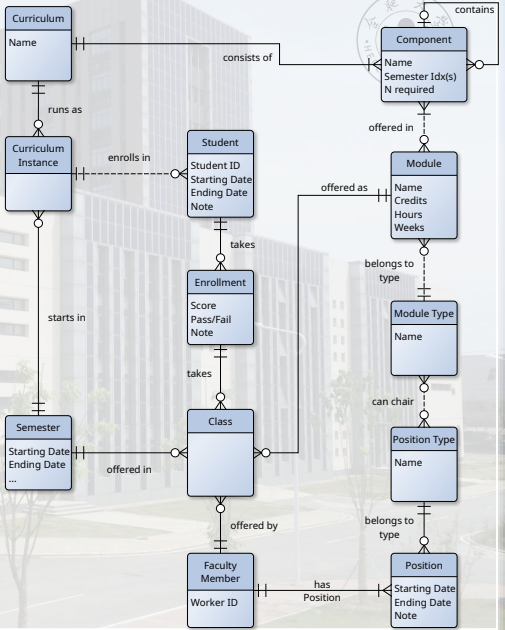
Bigger Picture, Part 2

- Wir könnten nun sagen das: *„Unter den Komponenten des Master of Computer Science Studiengangs gibt es die Komponente ‘Vertiefungsrichtung’. Diese beinhaltet wiederum die Komponenten ‘AI’, ‘Datenbanken’ und ‘Computer Security’. Ein Student muss genau eine solche Komponente abschließen. In der Vertiefungsrichtung ‘Datenbanken’ wird das Pflichtfach ‘Databases’ angeboten. Ebenso die Komponente ‘Wahlfächer’. Beide müssen von Studenten abgeschlossen werden. Die Komponente ‘Wahlfächer’ bietet die Module ‘PostgreSQL,’ ‘Systems Security’ und ‘Datenbankdesign’, von denen zwei abgeschlossen werden müssen.“*



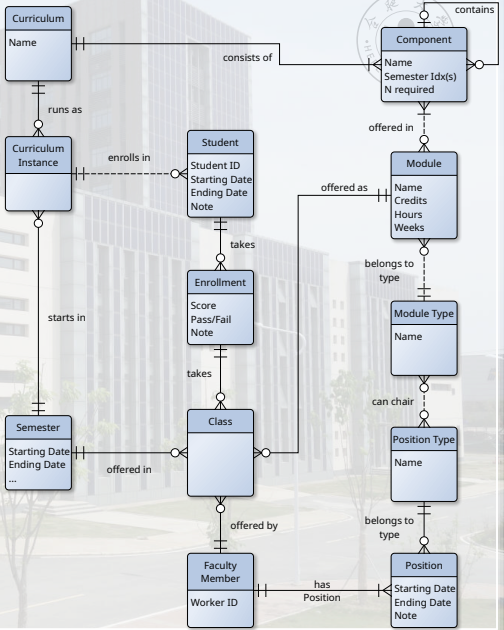
Bigger Picture, Part 2

- *„Unter den Komponenten des Master of Computer Science Studiengangs gibt es die Komponente ‘Vertiefungsrichtung’. Diese beinhaltet wiederum die Komponenten ‘AI’, ‘Datenbanken’ und ‘Computer Security’. Ein Student muss genau eine solche Komponente abschließen. In der Vertiefungsrichtung ‘Datenbanken’ wird das Pflichtfach ‘Databases’ angeboten. Ebenso die Komponente ‘Wahlfächer’. Beide müssen von Studenten abgeschlossen werden. Die Komponente ‘Wahlfächer’ bietet die Module ‘PostgreSQL,’ ‘Systems Security’ und ‘Datenbankdesign’, von denen zwei abgeschlossen werden müssen.“*
- OK, das sieht nicht sehr schön aus, aber es erlaubt uns, komplizierte Situationen in der Datenbank darzustellen.



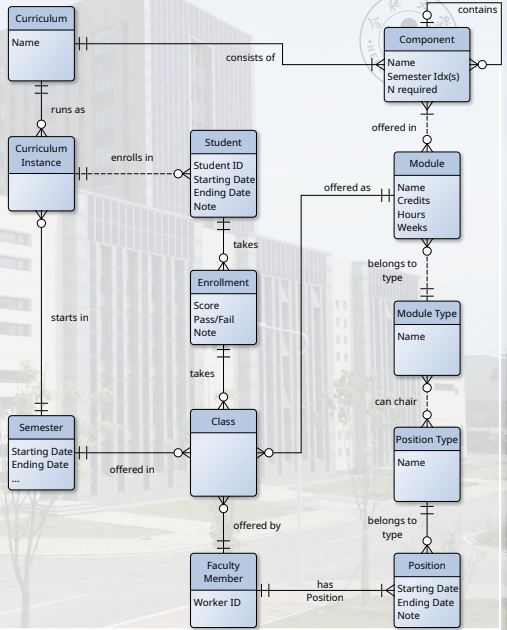
Bigger Picture, Part 2

- OK, das sieht nicht sehr schön aus, aber es erlaubt uns, komplizierte Situationen in der Datenbank darzustellen.
- Das bringt uns auch zum Entitätstyp *Module*.



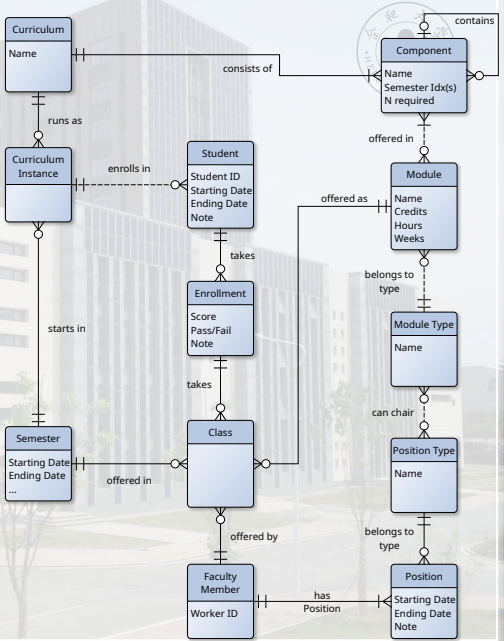
Bigger Picture, Part 2

- OK, das sieht nicht sehr schön aus, aber es erlaubt uns, komplizierte Situationen in der Datenbank darzustellen.
- Das bringt uns auch zum Entitätstyp *Module*.
- Module müssen von mindestens einer Komponente angeboten werden, denn sonst sind sie nutzlos.



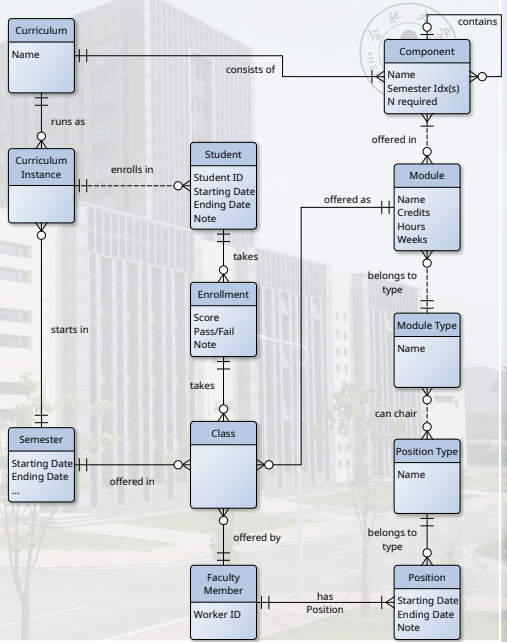
Bigger Picture, Part 2

- OK, das sieht nicht sehr schön aus, aber es erlaubt uns, komplizierte Situationen in der Datenbank darzustellen.
- Das bringt uns auch zum Entitätstyp *Module*.
- Module müssen von mindestens einer Komponente angeboten werden, denn sonst sind sie nutzlos.
- Sie gehören auch zu einem Modultyp, der sie zurück zu den Positions-bezogenen Qualifikationen der Lehrer verlinkt.



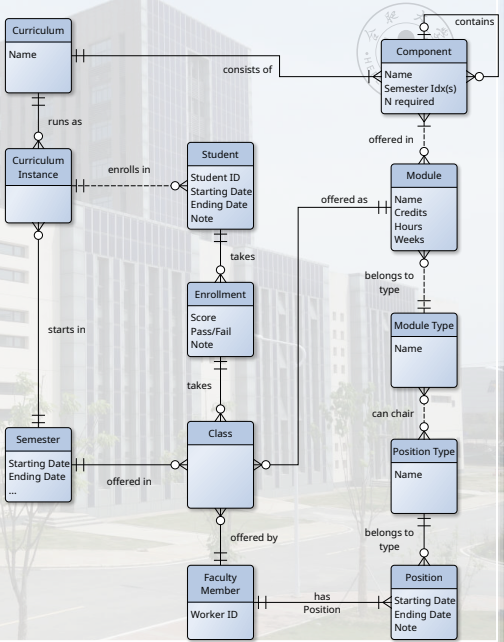
Bigger Picture, Part 2

- OK, das sieht nicht sehr schön aus, aber es erlaubt uns, komplizierte Situationen in der Datenbank darzustellen.
- Das bringt uns auch zum Entitätstyp *Module*.
- Module müssen von mindestens einer Komponente angeboten werden, denn sonst sind sie nutzlos.
- Sie gehören auch zu einem Modultyp, der sie zurück zu den Positions-bezogenen Qualifikationen der Lehrer verlinkt.
- Jedes Modul hat einen Namen, eine Anzahl Credits und eine Anzahl von Lehrstunden.



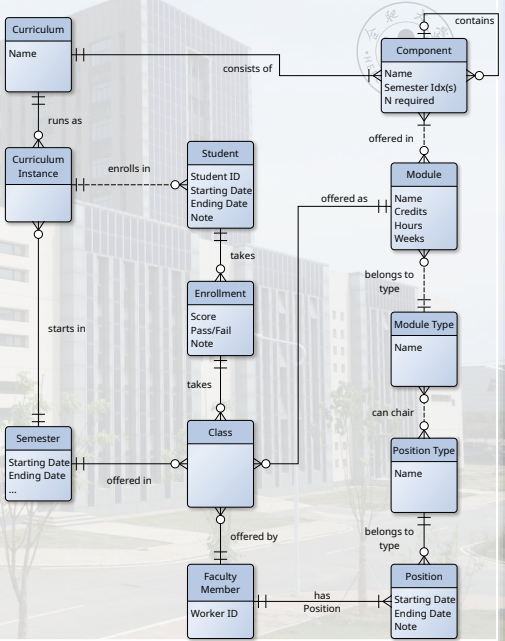
Bigger Picture, Part 2

- OK, das sieht nicht sehr schön aus, aber es erlaubt uns, komplizierte Situationen in der Datenbank darzustellen.
- Das bringt uns auch zum Entitätstyp *Module*.
- Module müssen von mindestens einer Komponente angeboten werden, denn sonst sind sie nutzlos.
- Sie gehören auch zu einem Modultyp, der sie zurück zu den Positions-bezogenen Qualifikationen der Lehrer verlinkt.
- Jedes Modul hat einen Namen, eine Anzahl Credits und eine Anzahl von Lehrstunden.
- Manche Module, wie Praktika und externe praktische Trainingseinheiten haben stattdessen Wochen als dauer.



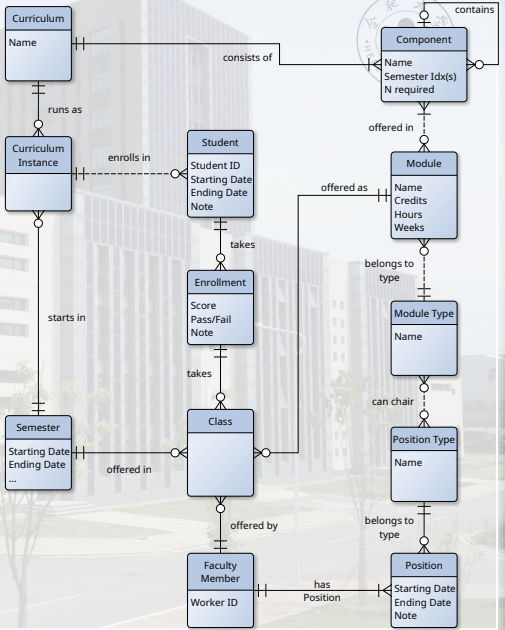
Bigger Picture, Part 2

- Das bringt uns auch zum Entitätstyp *Module*.
- Module müssen von mindestens einer Komponente angeboten werden, denn sonst sind sie nutzlos.
- Sie gehören auch zu einem Modultyp, der sie zurück zu den Positions-bezogenen Qualifikationen der Lehrer verlinkt.
- Jedes Modul hat einen Namen, eine Anzahl Credits und eine Anzahl von Lehrstunden.
- Manche Module, wie Praktika und externe praktische Trainingseinheiten haben stattdessen Wochen als dauer.
- Module haben auch ein Inhaltsvereichnis, eine Zusammenfassung, und andere Informationen, die wir hier weglassen.



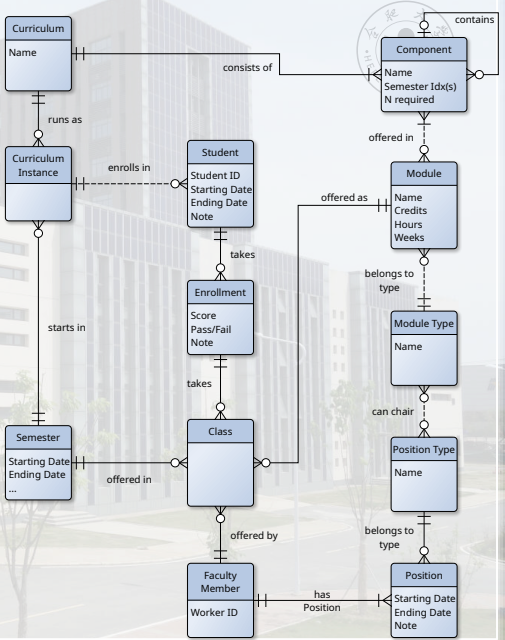
Bigger Picture, Part 2

- Module müssen von mindestens einer Komponente angeboten werden, denn sonst sind sie nutzlos.
- Sie gehören auch zu einem Modultyp, der sie zurück zu den Positions-bezogenen Qualifikationen der Lehrer verlinkt.
- Jedes Modul hat einen Namen, eine Anzahl Credits und eine Anzahl von Lehrstunden.
- Manche Module, wie Praktika und externe praktische Trainingseinheiten haben stattdessen Wochen als dauer.
- Module haben auch ein Inhaltsvereichnis, eine Zusammenfassung, und andere Informationen, die wir hier weglassen.
- Module bilden das Grundgerüst vom Lehrinhalt, der angeboten wird.



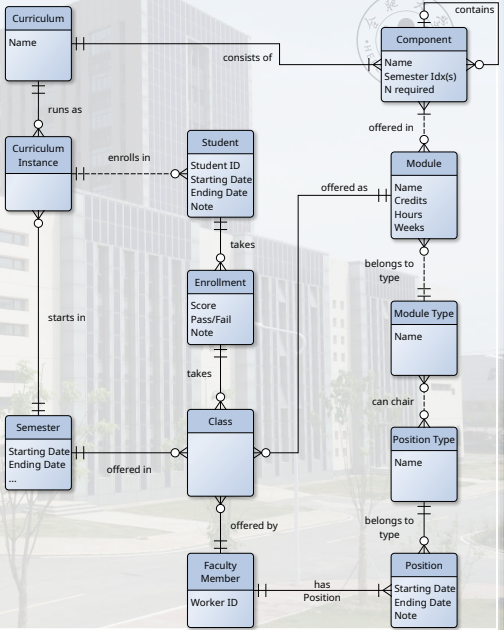
Bigger Picture, Part 2

- Sie gehören auch zu einem Modultyp, der sie zurück zu den Positions-bezogenen Qualifikationen der Lehrer verlinkt.
- Jedes Modul hat einen Namen, eine Anzahl Credits und eine Anzahl von Lehrstunden.
- Manche Module, wie Praktika und externe praktische Trainingseinheiten haben stattdessen Wochen als dauer.
- Module haben auch ein Inhaltsvereichnis, eine Zusammenfassung, und andere Informationen, die wir hier weglassen.
- Module bilden das Grundgerüst vom Lehrinhalt, der angeboten wird.
- Sie gehören zu bestimmten Semestern, gezählt vom Beginn eines Studiengangs.



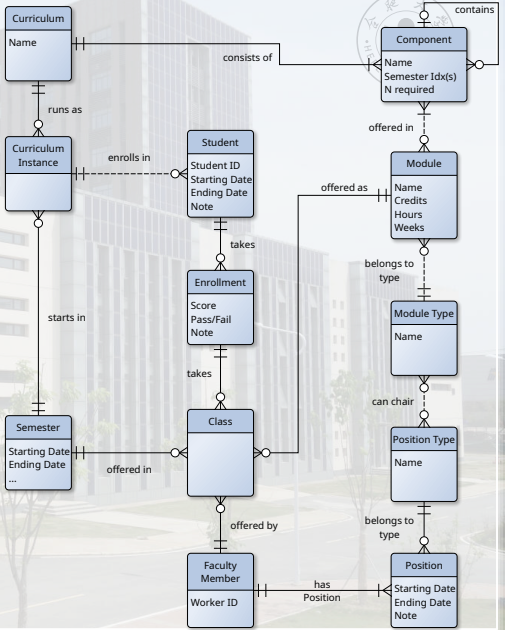
Bigger Picture, Part 2

- Jedes Modul hat einen Namen, eine Anzahl Credits und eine Anzahl von Lehrstunden.
- Manche Module, wie Praktika und externe praktische Trainingseinheiten haben stattdessen Wochen als dauer.
- Module haben auch ein Inhaltsvereichnis, eine Zusammenfassung, und andere Informationen, die wir hier weglassen.
- Module bilden das Grundgerüst vom Lehrinhalt, der angeboten wird.
- Sie gehören zu bestimmten Semestern, gezählt vom Beginn eines Studiengangs.
- Eine Fakultät kann einen bestimmten Studiengang instantiieren und eine Entität vom Typ *Curriculum Instance* erstellen.



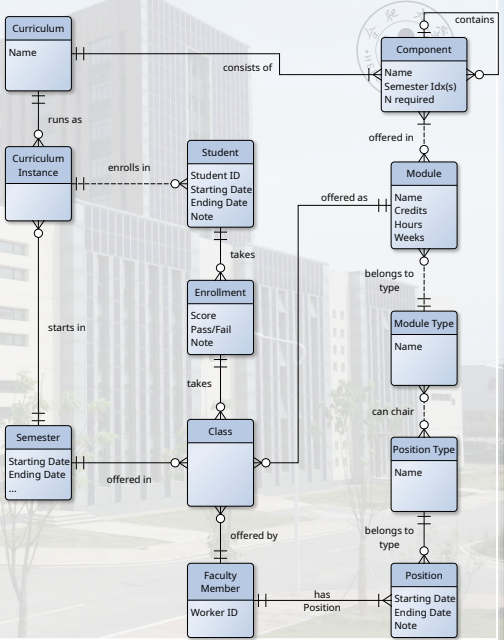
Bigger Picture, Part 2

- Module haben auch ein Inhaltsvereichnis, eine Zusammenfassung, und andere Informationen, die wir hier weglassen.
- Module bilden das Grundgerüst vom Lehrinhalt, der angeboten wird.
- Sie gehören zu bestimmten Semestern, gezählt vom Beginn eines Studiengangs.
- Eine Fakultät kann einen bestimmten Studiengang instantiieren und eine Entität vom Typ *Curriculum Instance* erstellen.
- Diese Instanz ist immer mit einem Startsemester verbunden, z. B. dem Winter Semester 2025 oder vielleicht dem maybe dem Summer Semester 2026.



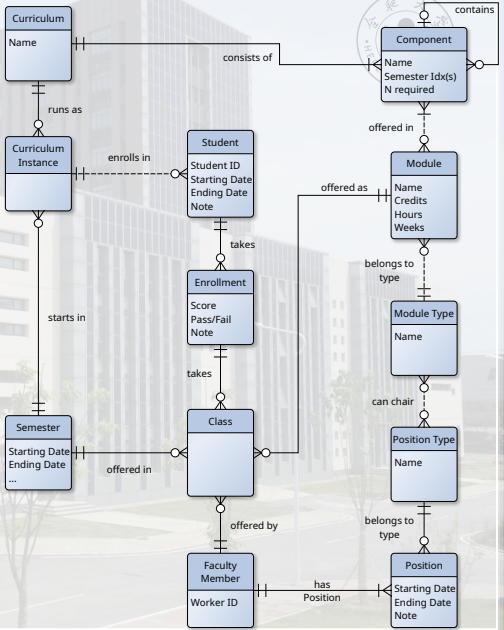
Bigger Picture, Part 2

- Module bilden das Grundgerüst vom Lehrinhalt, der angeboten wird.
- Sie gehören zu bestimmten Semestern, gezählt vom Beginn eines Studiengangs.
- Eine Fakultät kann einen bestimmten Studiengang instantiieren und eine Entität vom Typ *Curriculum Instance* erstellen.
- Diese Instanz ist immer mit einem Startsemester verbunden, z. B. dem Winter Semester 2025 oder vielleicht dem maybe dem Summer Semester 2026.
- Wir behalten den Entitätstyp *Semester* aus diesem Grund.



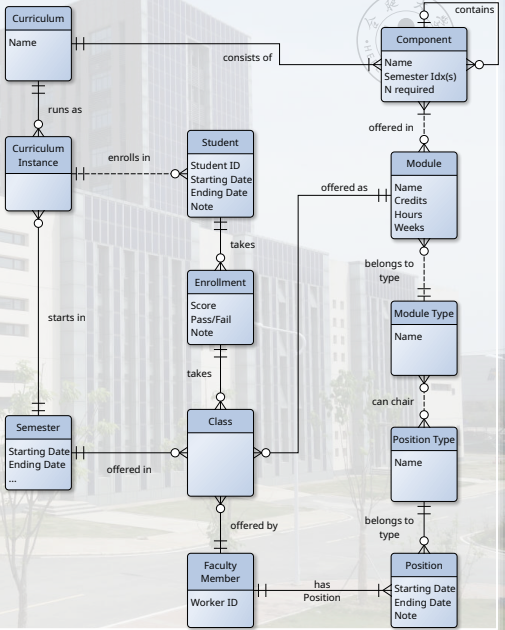
Bigger Picture, Part 2

- Sie gehören zu bestimmten Semestern, gezählt vom Beginn eines Studiengangs.
- Eine Fakultät kann einen bestimmten Studiengang instantiieren und eine Entität vom Typ *Curriculum Instance* erstellen.
- Diese Instanz ist immer mit einem Startsemester verbunden, z. B. dem Winter Semester 2025 oder vielleicht dem maybe dem Summer Semester 2026.
- Wir behalten den Entitätstyp *Semester* aus diesem Grund.
- Er beinhaltet alle notwendigen Informationen eines Semesters, z. B. die Zeitperiode, wie von der Uni definiert.



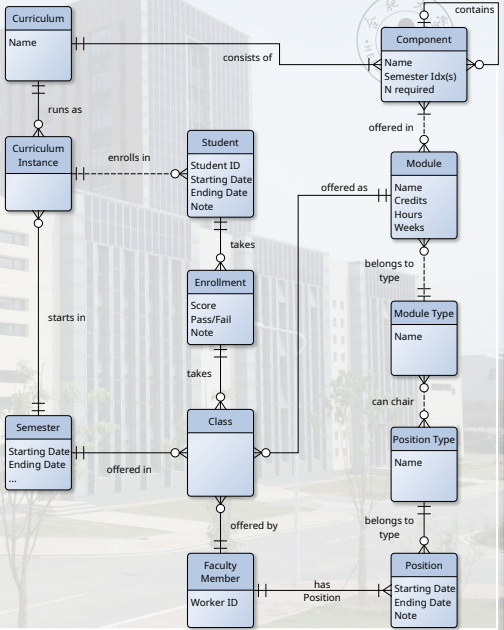
Bigger Picture, Part 2

- Eine Fakultät kann einen bestimmten Studiengang instantiieren und eine Entität vom Typ *Curriculum Instance* erstellen.
- Diese Instanz ist immer mit einem Startsemester verbunden, z. B. dem Winter Semester 2025 oder vielleicht dem maybe dem Summer Semester 2026.
- Wir behalten den Entitätstyp *Semester* aus diesem Grund.
- Er beinhaltet alle notwendigen Informationen eines Semesters, z. B. die Zeitperiode, wie von der Uni definiert.
- In jedem Semester kann ein Mitarbeiter keinen, einen, oder mehrere Kurse (EN: *classes*) anbieten.



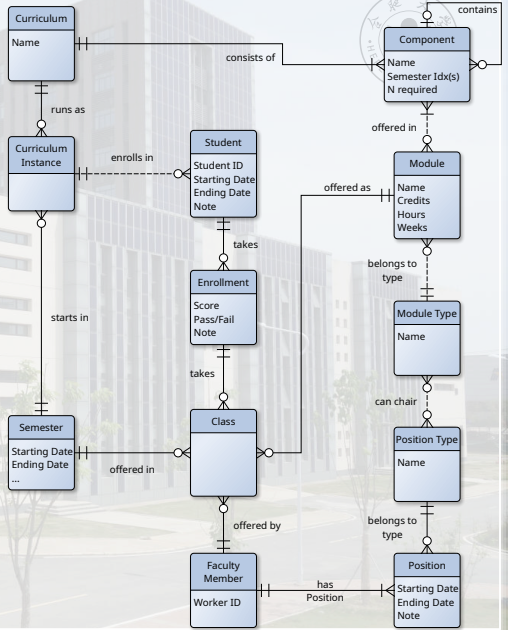
Bigger Picture, Part 2

- Eine Fakultät kann einen bestimmten Studiengang instantiieren und eine Entität vom Typ *Curriculum Instance* erstellen.
- Diese Instanz ist immer mit einem Startsemester verbunden, z. B. dem Winter Semester 2025 oder vielleicht dem maybe dem Summer Semester 2026.
- Wir behalten den Entitätstyp *Semester* aus diesem Grund.
- Er beinhaltet alle notwendigen Informationen eines Semesters, z. B. die Zeitperiode, wie von der Uni definiert.
- In jedem Semester kann ein Mitarbeiter keinen, einen, oder mehrere Kurse (EN: *classes*) anbieten.
- Jeder Kurs ist eine Instanz eines Moduls.



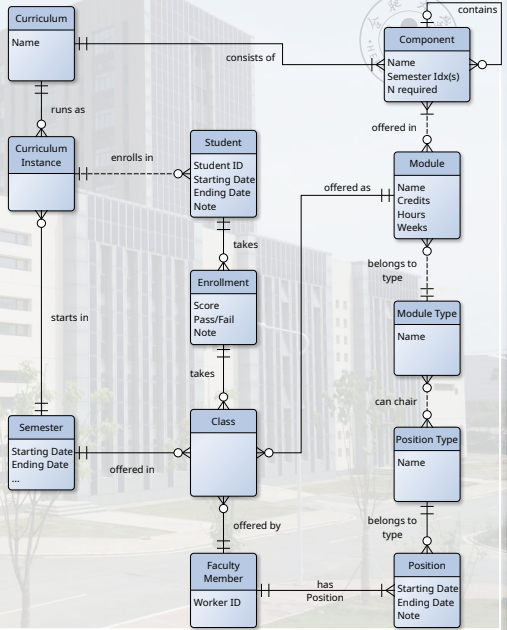
Bigger Picture, Part 2

- Diese Instanz ist immer mit einem Startsemester verbunden, z. B. dem Winter Semester 2025 oder vielleicht dem maybe dem Summer Semester 2026.
- Wir behalten den Entitätstyp *Semester* aus diesem Grund.
- Er beinhaltet alle notwendigen Informationen eines Semesters, z. B. die Zeitperiode, wie von der Uni definiert.
- In jedem Semester kann ein Mitarbeiter keinen, einen, oder mehrere Kurse (EN: *classes*) anbieten.
- Jeder Kurs ist eine Instanz eines Moduls.
- Jedes Modul kann beliebig oft als Kurs angeboten werden, vielleicht sogar mehrfach im selben Semester.



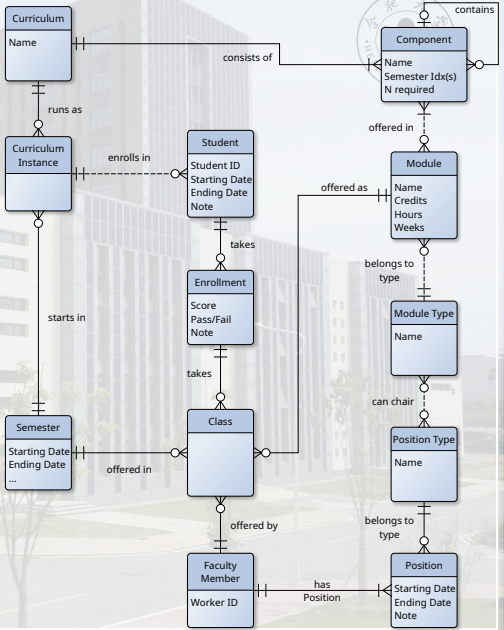
Bigger Picture, Part 2

- Wir behalten den Entitätstyp *Semester* aus diesem Grund.
- Er beinhaltet alle notwendigen Informationen eines Semesters, z. B. die Zeitperiode, wie von der Uni definiert.
- In jedem Semester kann ein Mitarbeiter keinen, einen, oder mehrere Kurse (EN: *classes*) anbieten.
- Jeder Kurs ist eine Instanz eines Moduls.
- Jedes Modul kann beliebig oft als Kurs angeboten werden, vielleicht sogar mehrfach im selben Semester.
- Wir behandeln auch die Master- und Bachelor-Projekte als Module.



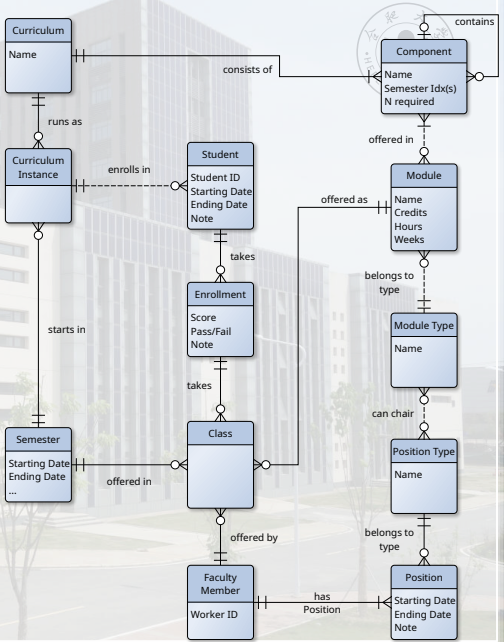
Bigger Picture, Part 2

- Er beinhaltet alle notwendigen Informationen eines Semesters, z. B. die Zeitperiode, wie von der Uni definiert.
- In jedem Semester kann ein Mitarbeiter keinen, einen, oder mehrere Kurse (EN: *classes*) anbieten.
- Jeder Kurs ist eine Instanz eines Moduls.
- Jedes Modul kann beliebig oft als Kurs angeboten werden, vielleicht sogar mehrfach im selben Semester.
- Wir behandeln auch die Master- und Bachelor-Projekte als Module.
- Sie beginnen in einem Semester, aber es sagt ja niemand, das sie da auch enden müssen.



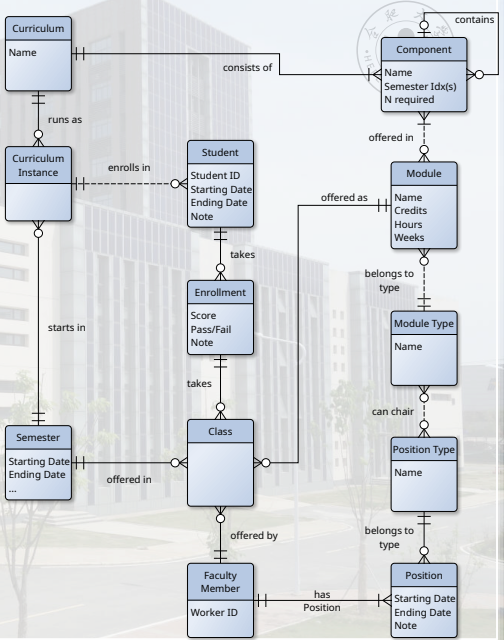
Bigger Picture, Part 2

- In jedem Semester kann ein Mitarbeiter keinen, einen, oder mehrere Kurse (EN: *classes*) anbieten.
- Jeder Kurs ist eine Instanz eines Moduls.
- Jedes Modul kann beliebig oft als Kurs angeboten werden, vielleicht sogar mehrfach im selben Semester.
- Wir behandeln auch die Master- und Bachelor-Projekte als Module.
- Sie beginnen in einem Semester, aber es sagt ja niemand, das sie da auch enden müssen.
- Unser System kann nun leicht prüfen, ob ein Lehrer die notwendigen Qualifikationen hat, um einen Kurs anzubieten.



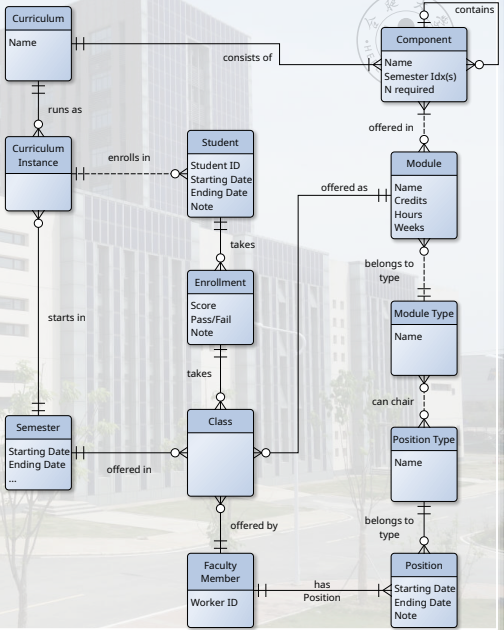
Bigger Picture, Part 2

- Jeder Kurs ist eine Instanz eines Moduls.
- Jedes Modul kann beliebig oft als Kurs angeboten werden, vielleicht sogar mehrfach im selben Semester.
- Wir behandeln auch die Master- und Bachelor-Projekte als Module.
- Sie beginnen in einem Semester, aber es sagt ja niemand, das sie da auch enden müssen.
- Unser System kann nun leicht prüfen, ob ein Lehrer die notwendigen Qualifikationen hat, um einen Kurs anzubieten.
- Die Kurse werden natürlich von der Administration der Fakultäten eingegeben.



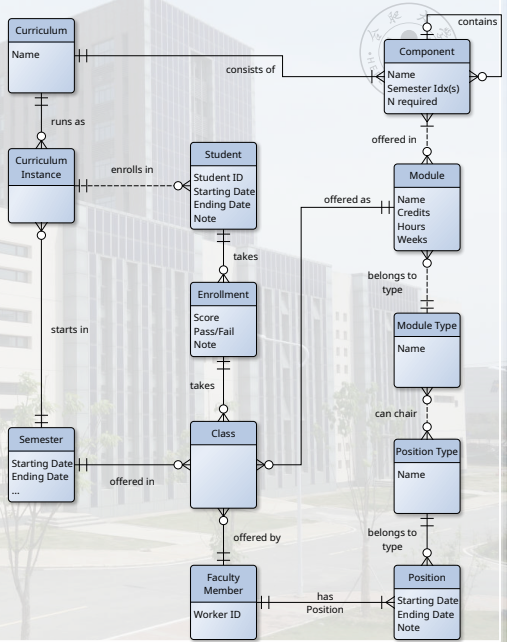
Bigger Picture, Part 2

- Jedes Modul kann beliebig oft als Kurs angeboten werden, vielleicht sogar mehrfach im selben Semester.
- Wir behandeln auch die Master- und Bachelor-Projekte als Module.
- Sie beginnen in einem Semester, aber es sagt ja niemand, das sie da auch enden müssen.
- Unser System kann nun leicht prüfen, ob ein Lehrer die notwendigen Qualifikationen hat, um einen Kurs anzubieten.
- Die Kurse werden natürlich von der Administration der Fakultäten eingegeben.
- Ein Student kann sich in einen Kurs einschreiben.



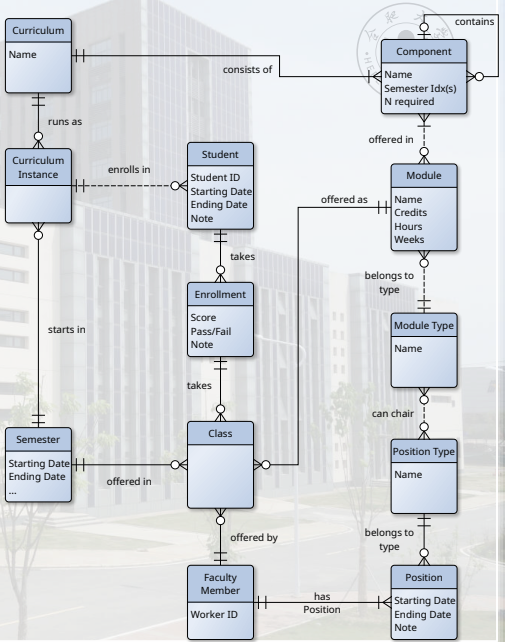
Bigger Picture, Part 2

- Wir behandeln auch die Master- und Bachelor-Projekte als Module.
- Sie beginnen in einem Semester, aber es sagt ja niemand, das sie da auch enden müssen.
- Unser System kann nun leicht prüfen, ob ein Lehrer die notwendigen Qualifikationen hat, um einen Kurs anzubieten.
- Die Kurse werden natürlich von der Administration der Fakultäten eingegeben.
- Ein Student kann sich in einen Kurs einschreiben.
- Kurse und Einschreibungen (EN: *enrollments*) sind schwachte Entitäten.



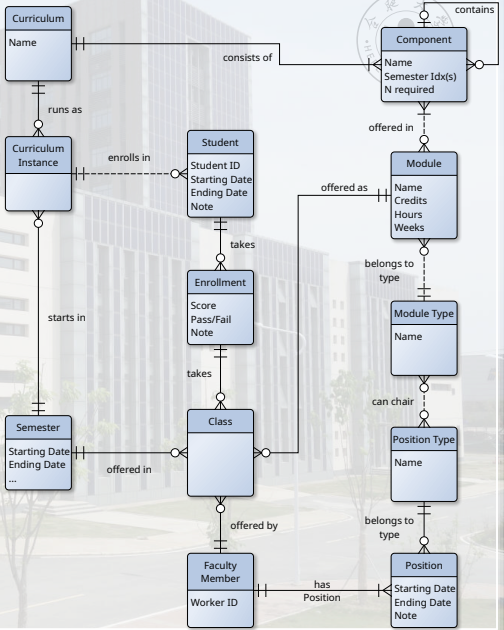
Bigger Picture, Part 2

- Sie beginnen in einem Semester, aber es sagt ja niemand, das sie da auch enden müssen.
- Unser System kann nun leicht prüfen, ob ein Lehrer die notwendigen Qualifikationen hat, um einen Kurs anzubieten.
- Die Kurse werden natürlich von der Administration der Fakultäten eingegeben.
- Ein Student kann sich in einen Kurs einschreiben.
- Kurse und Einschreibungen (EN: *enrollments*) sind schwachte Entitäten.
- Jede Einschreibung ist mit genau einem Studenten verbunden.



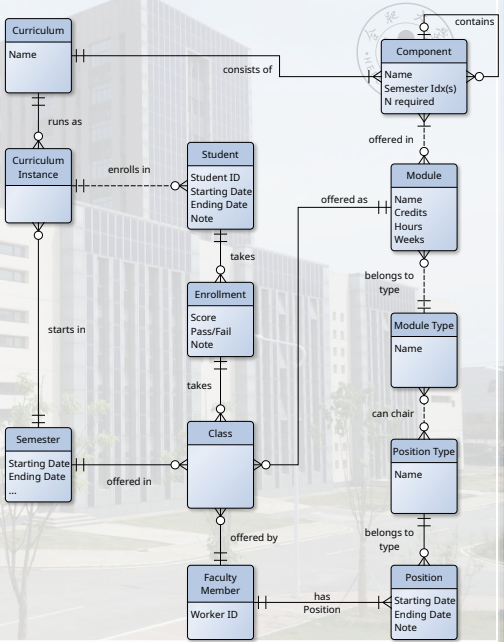
Bigger Picture, Part 2

- Unser System kann nun leicht prüfen, ob ein Lehrer die notwendigen Qualifikationen hat, um einen Kurs anzubieten.
- Die Kurse werden natürlich von der Administration der Fakultäten eingegeben.
- Ein Student kann sich in einen Kurs einschreiben.
- Kurse und Einschreibungen (EN: *enrollments*) sind schwachte Entitäten.
- Jede Einschreibung ist mit genau einem Studenten verbunden.
- Ein Student kann mit beliebig vielen Einschreibungen verbunden sein.



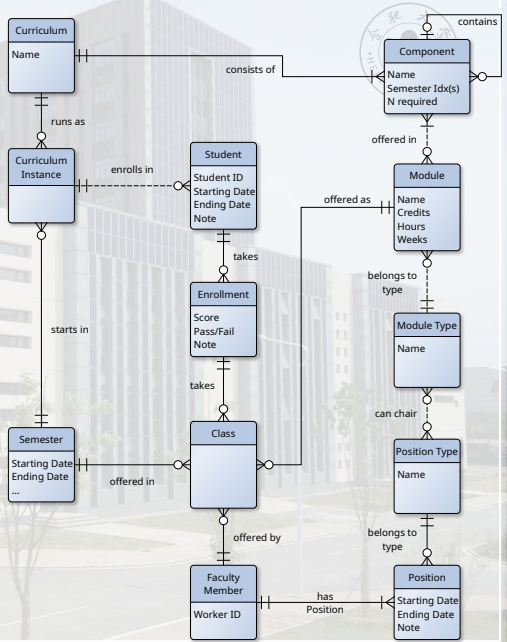
Bigger Picture, Part 2

- Die Kurse werden natürlich von der Administration der Fakultäten eingegeben.
- Ein Student kann sich in einen Kurs einschreiben.
- Kurse und Einschreibungen (EN: *enrollments*) sind schwachte Entitäten.
- Jede Einschreibung ist mit genau einem Studenten verbunden.
- Ein Student kann mit beliebig vielen Einschreibungen verbunden sein.
- Der Einschreibungsdatensatz wird später um Informationen wie die Abschlussnote, ob der Student bestanden hat, und ggf. weitere Erklärungen ergänzt.



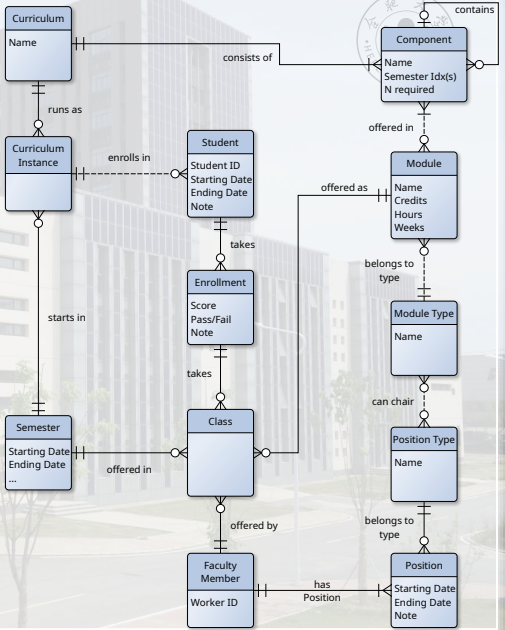
Bigger Picture, Part 2

- Ein Student kann sich in einen Kurs einschreiben.
- Kurse und Einschreibungen (EN: *enrollments*) sind schwachte Entitäten.
- Jede Einschreibung ist mit genau einem Studenten verbunden.
- Ein Student kann mit beliebig vielen Einschreibungen verbunden sein.
- Der Einschreibungsdatensatz wird später um Informationen wie die Abschlussnote, ob der Student bestanden hat, und ggf. weitere Erklärungen ergänzt.
- Jeder Student ist in genau einen Studiengang eingeschrieben.



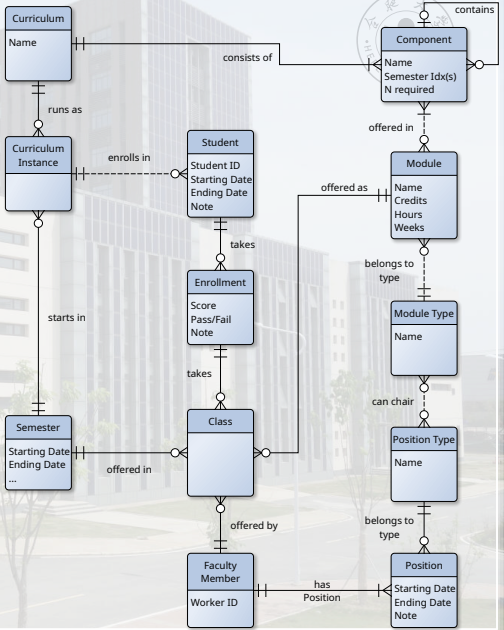
Bigger Picture, Part 2

- Kurse und Einschreibungen (EN: *enrollments*) sind schwachte Entitäten.
- Jede Einschreibung ist mit genau einem Studenten verbunden.
- Ein Student kann mit beliebig vielen Einschreibungen verbunden sein.
- Der Einschreibungsdatensatz wird später um Informationen wie die Abschlussnote, ob der Student bestanden hat, und ggf. weitere Erklärungen ergänzt.
- Jeder Student ist in genau einen Studiengang eingeschrieben.
- Beliebig viele Studenten können in einen Studiengang eingeschrieben sein.



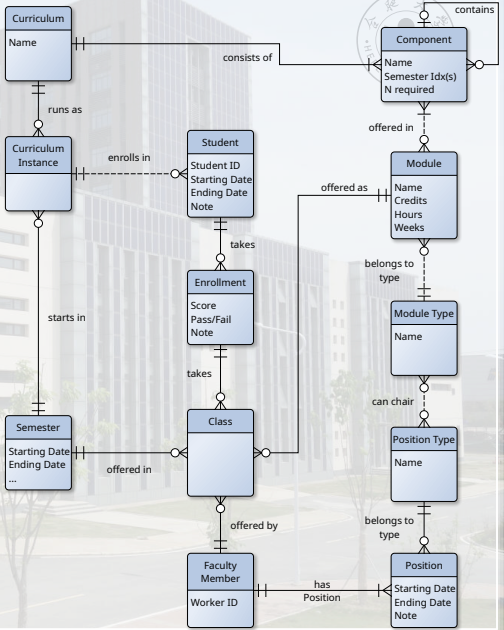
Bigger Picture, Part 2

- Ein Student kann mit beliebig vielen Einschreibungen verbunden sein.
- Der Einschreibungsdatensatz wird später um Informationen wie die Abschlussnote, ob der Student bestanden hat, und ggf. weitere Erklärungen ergänzt.
- Jeder Student ist in genau einen Studiengang eingeschrieben.
- Beliebig viele Studenten können in einen Studiengang eingeschrieben sein.
- Ok, vielleicht werden wir das begrenzen . . . jetzt nehmen wir erstmal an, dass die Administration der Fakultäten die Studenten in die Datenbank einpflegt und weiß was sie macht.



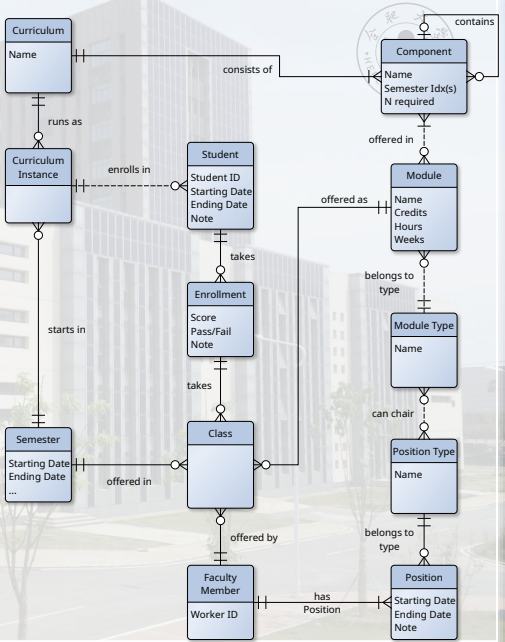
Bigger Picture, Part 2

- Der Einschreibungsdatensatz wird später um Informationen wie die Abschlussnote, ob der Student bestanden hat, und ggf. weitere Erklärungen ergänzt.
- Jeder Student ist in genau einen Studiengang eingeschrieben.
- Beliebig viele Studenten können in einen Studiengang eingeschrieben sein.
- Ok, vielleicht werden wir das begrenzen . . . jetzt nehmen wir erstmal an, dass die Administration der Fakultäten die Studenten in die Datenbank einpflegt und weiß was sie macht.
- Unsere Plattform kann nun Studenten automatisch in Kurse einschreiben, die sie nehmen müssen.



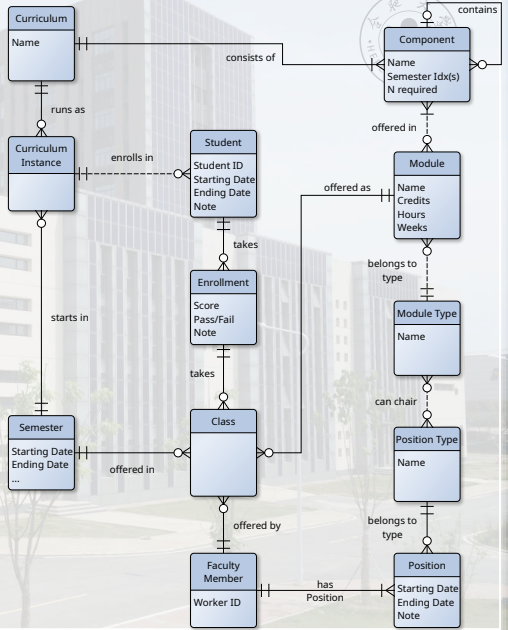
Bigger Picture, Part 2

- Jeder Student ist in genau einen Studiengang eingeschrieben.
- Beliebig viele Studenten können in einen Studiengang eingeschrieben sein.
- Ok, vielleicht werden wir das begrenzen . . . jetzt nehmen wir erstmal an, dass die Administration der Fakultäten die Studenten in die Datenbank einpflegt und weiß was sie macht.
- Unsere Plattform kann nun Studenten automatisch in Kurse einschreiben, die sie nehmen müssen.
- Es kann ja den Studiengang sehen, zu dme die Studenten gehören, weiß welche Module Pflicht sind, und kann die Studenten dann automatisch eintragen, wenn es keine Auswahlmöglichkeit gibt.



Bigger Picture, Part 2

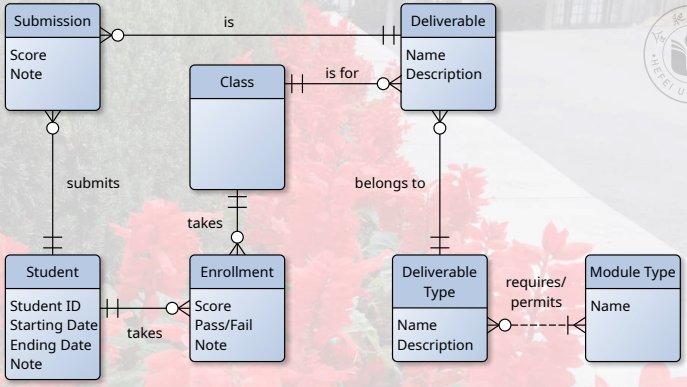
- Beliebig viele Studenten können in einen Studiengang eingeschrieben sein.
- Ok, vielleicht werden wir das begrenzen ... jetzt nehmen wir erstmal an, dass die Administration der Fakultäten die Studenten in die Datenbank einpflegt und weiß was sie macht.
- Unsere Plattform kann nun Studenten automatisch in Kurse einschreiben, die sie nehmen müssen.
- Es kann ja den Studiengang sehen, zu dme die Studenten gehören, weiß welche Module Pflicht sind, und kann die Studenten dann automatisch eintragen, wenn es keine Auswahlmöglichkeit gibt.
- Basierend auf den angebotenen Kursen kann es ihnen auch Wahlmöglichkeiten über das Web-Portal anbieten.



Deliverable System

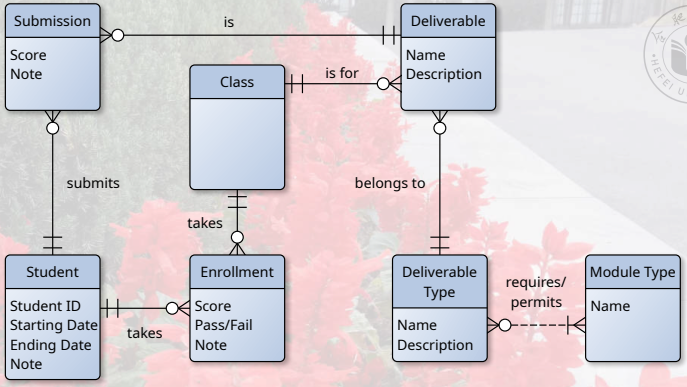
- Ein weiteres Subsystem unserer Plattform kümmert sich um die zu liefernden Leistungen (EN: *deliverables*).

Deliverable System



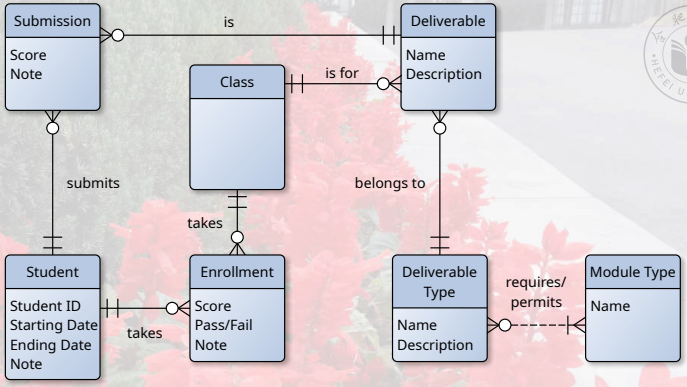
- Ein weiteres Subsystem unserer Plattform kümmert sich um die zu liefernden Leistungen (EN: *deliverables*).

Deliverable System



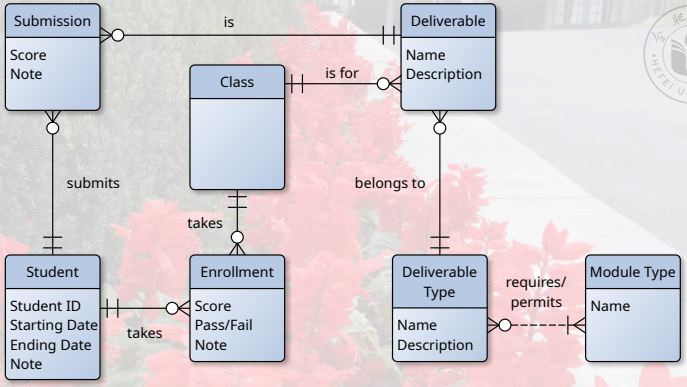
- Ein weiteres Subsystem unserer Plattform kümmert sich um die zu liefernden Leistungen (EN: *deliverables*).
- Es gibt verschiedene Arten von Deliverables, Z. B. schriftliche Prüfungen, mündliche Prüfungen, Midterm-Prüfungen, Hausaufgaben, Praktikumsberichte, und Abschlussarbeiten.

Deliverable System



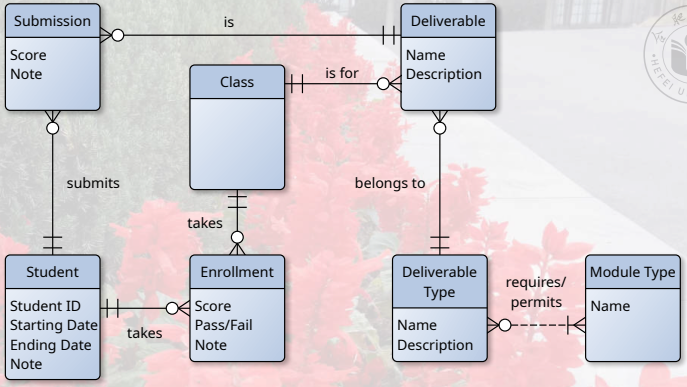
- Ein weiteres Subsystem unserer Plattform kümmert sich um die zu liefernden Leistungen (EN: *deliverables*).
- Es gibt verschiedene Arten von Deliverables, Z. B. schriftliche Prüfungen, mündliche Prüfungen, Midterm-Prüfungen, Hausaufgaben, Praktikumsberichte, und Abschlussarbeiten.
- Jeder Modultyp kann eine beliebige Anzahl von Deliverable-Typen *erfordern*.

Deliverable System



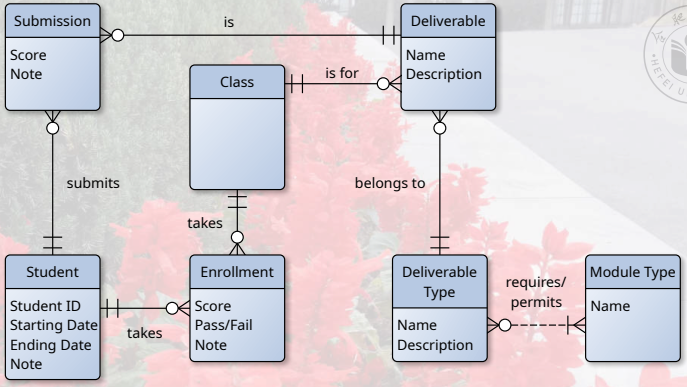
- Es gibt verschiedene Arten von Deliverables, Z. B. schriftliche Prüfungen, mündliche Prüfungen, Midterm-Prüfungen, Hausaufgaben, Praktikumsberichte, und Abschlussarbeiten.
- Jeder Modultyp kann eine beliebige Anzahl von Deliverable-Typen *erfordern*.
- Jede Modultyp kann eine beliebige Anzahl von Deliverable-Typen *erlauben*.

Deliverable System



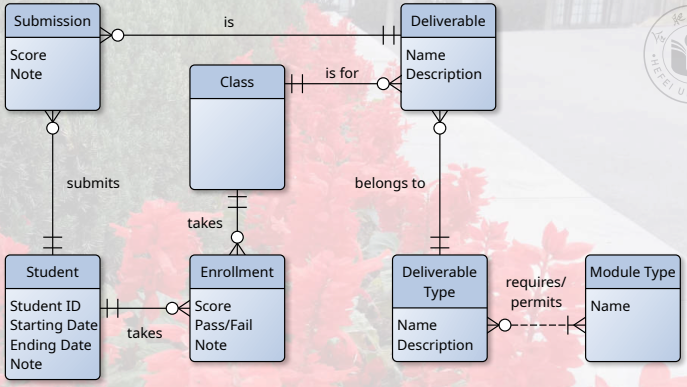
- Es gibt verschiedene Arten von Deliverables, Z. B. schriftliche Prüfungen, mündliche Prüfungen, Midterm-Prüfungen, Hausaufgaben, Praktikumsberichte, und Abschlussarbeiten.
- Jeder Modultyp kann eine beliebige Anzahl von Deliverable-Typen *erfordern*.
- Jede Modultyp kann eine beliebige Anzahl von Deliverable-Typen *erlauben*.
- Um Platz zu sparen, haben wir diese beiden Beziehungen als eine modelliert.

Deliverable System



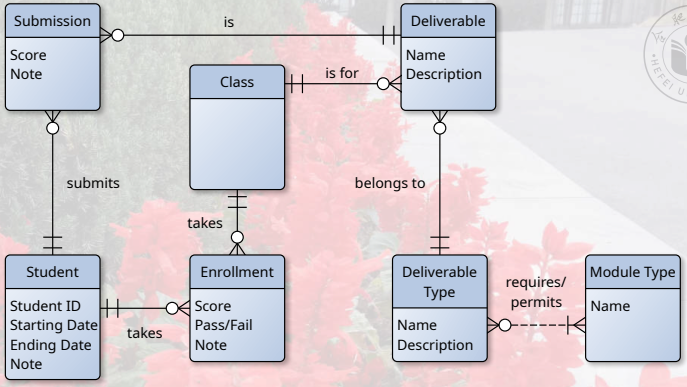
- Jeder Modultyp kann eine beliebige Anzahl von Deliverable-Typen *erfordern*.
- Jede Modultyp kann eine beliebige Anzahl von Deliverable-Typen *erlauben*.
- Um Platz zu sparen, haben wir diese beiden Beziehungen als eine modelliert.
- Wir haben bereits festgelegt, das Kurse zu Modulen und Module zu Modultypen gehören.

Deliverable System



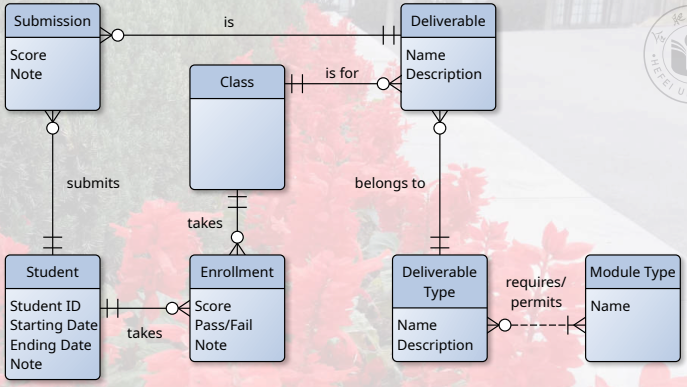
- Jede Modultyp kann eine beliebige Anzahl von Deliverable-Typen *erlauben*.
- Um Platz zu sparen, haben wir diese beiden Beziehungen als eine modelliert.
- Wir haben bereits festgelegt, das Kurse zu Modulen und Module zu Modultypen gehören.
- Aus den Beziehungen von Deliverable-Typen und Modultypen können wir ableiten, welche Deliverable-Typen für einen Kurs zulässig/erfordert sind.

Deliverable System



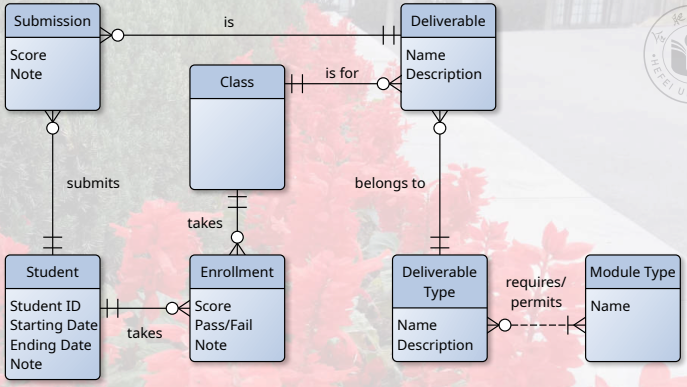
- Um Platz zu sparen, haben wir diese beiden Beziehungen als eine modelliert.
- Wir haben bereits festgelegt, das Kurse zu Modulen und Module zu Modultypen gehören.
- Aus den Beziehungen von Deliverable-Typen und Modultypen können wir ableiten, welche Deliverable-Typen für einen Kurs zulässig/erfordert sind.
- Für ein Master's-Projekt könnte eine MSc-These ein erfordertes Deliverable sein.

Deliverable System



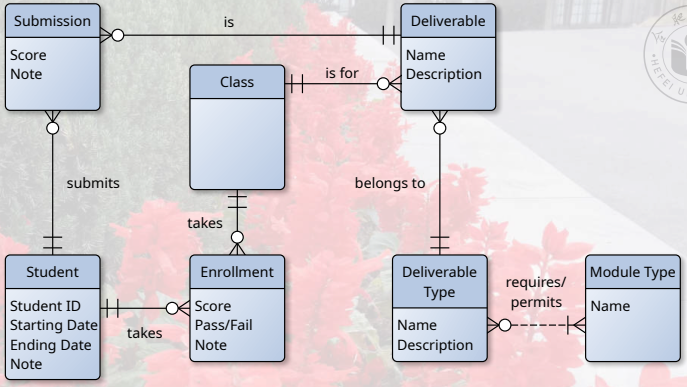
- Aus den Beziehungen von Deliverable-Typen und Modultypen können wir ableiten, welche Deliverable-Typen für einen Kurs zulässig/erfordert sind.
- Für ein Master's-Projekt könnte eine MSc-These ein erfordertes Deliverable sein.
- Für einen Kurs „Datenbanken“ könnte ein schriftliches Examen erforderlich und ein Midterm-Exam und Hausaufgaben erlaubt sein.

Deliverable System



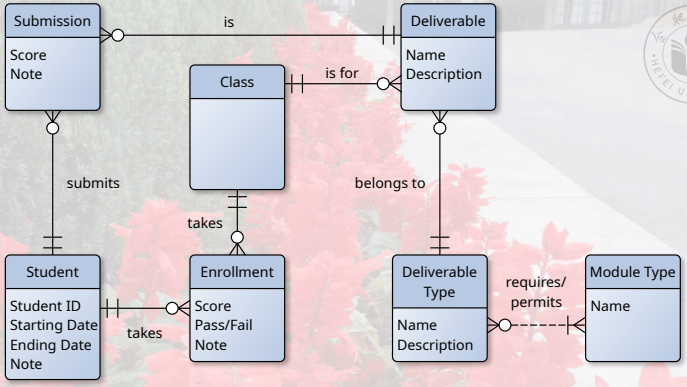
- Für ein Master's-Projekt könnte eine MSc-These ein erfordertes Deliverable sein.
- Für einen Kurs „Datenbanken“ könnte ein schriftliches Examen erforderlich und ein Midterm-Exam und Hausaufgaben erlaubt sein.
- Der entsprechende Lehrer wird dann eine (schwache) *Deliverable* Entität erstellen, die einen Namen und eine Beschreibung hat.

Deliverable System



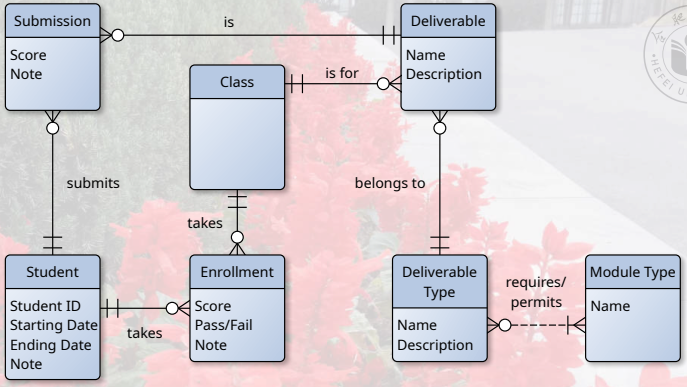
- Für ein Master's-Projekt könnte eine MSc-These ein erfordertes Deliverable sein.
- Für einen Kurs „Datenbanken“ könnte ein schriftliches Examen erforderlich und ein Midterm-Exam und Hausaufgaben erlaubt sein.
- Der entsprechende Lehrer wird dann eine (schwache) *Deliverable* Entität erstellen, die einen Namen und eine Beschreibung hat.
- Jeder Student wird dann eine entsprechende Einreichung für das Deliverable machen.

Deliverable System



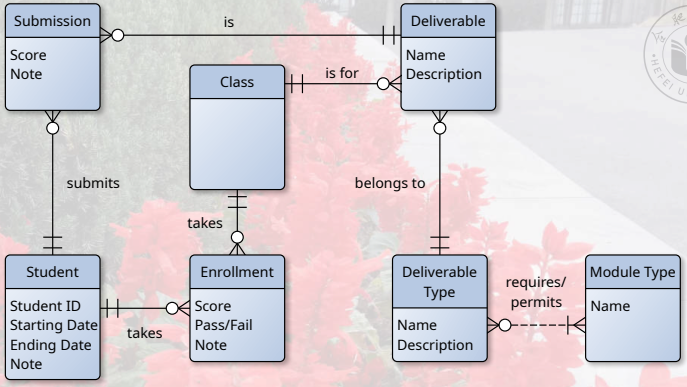
- Für einen Kurs „Datenbanken“ könnte ein schriftliches Examen erforderlich und ein Midterm-Exam und Hausaufgaben erlaubt sein.
- Der entsprechende Lehrer wird dann eine (schwache) *Deliverable* Entität erstellen, die einen Namen und eine Beschreibung hat.
- Jeder Student wird dann eine entsprechende Einreichung für das Deliverable machen.
- Diese Einreichungen werden nicht über unsere Plattform laufen.

Deliverable System



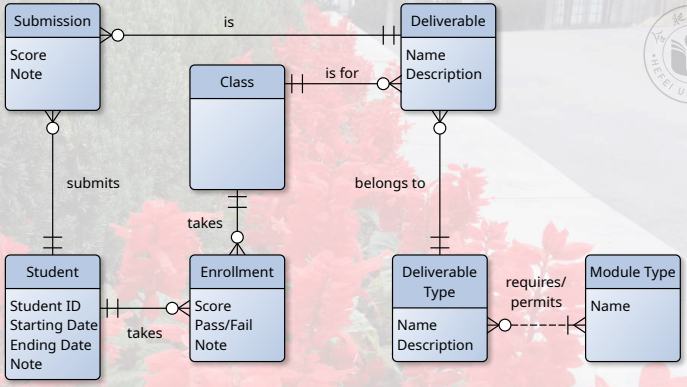
- Der entsprechende Lehrer wird dann eine (schwache) *Deliverable* Entität erstellen, die einen Namen und eine Beschreibung hat.
- Jeder Student wird dann eine entsprechende Einreichung für das Deliverable machen.
- Diese Einreichungen werden nicht über unsere Plattform laufen.
- Sie werden von den Studenten dem Lehrer übergeben.

Deliverable System



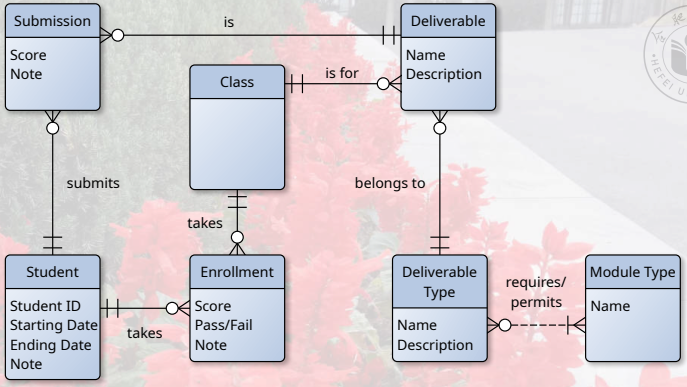
- Jeder Student wird dann eine entsprechende Einreichung für das Deliverable machen.
- Diese Einreichungen werden nicht über unsere Plattform laufen.
- Sie werden von den Studenten dem Lehrer übergeben.
- Der Lehrer bewertet sie und erstellt ein entsprechende (schwache) *Submission* Entitäten – eine pro Student – im System.

Deliverable System



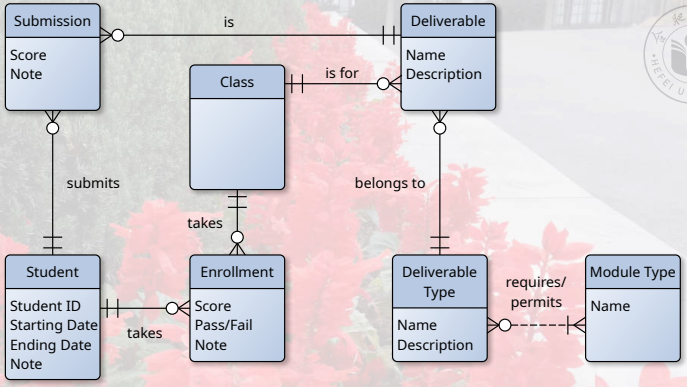
- Diese Einreichungen werden nicht über unsere Plattform laufen.
- Sie werden von den Studenten dem Lehrer übergeben.
- Der Lehrer bewertet sie und erstellt ein entsprechende (schwache) *Submission* Entitäten – eine pro Student – im System.
- Diese schwache Entität enthält das Ergebnis des Studenten und vielleicht ein Kommentar vom Lehrer.

Deliverable System



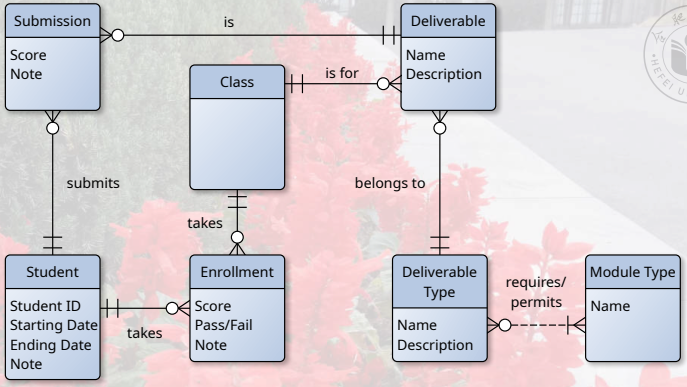
- Sie werden von den Studenten dem Lehrer übergeben.
- Der Lehrer bewertet sie und erstellt ein entsprechende (schwache) *Submission* Entitäten – eine pro Student – im System.
- Diese schwache Entität enthält das Ergebnis des Studenten und vielleicht ein Kommentar vom Lehrer.
- So kann der Lehrer Prüfungsergebnise, Midterm-Ergebnisse, usw. hochladen.

Deliverable System



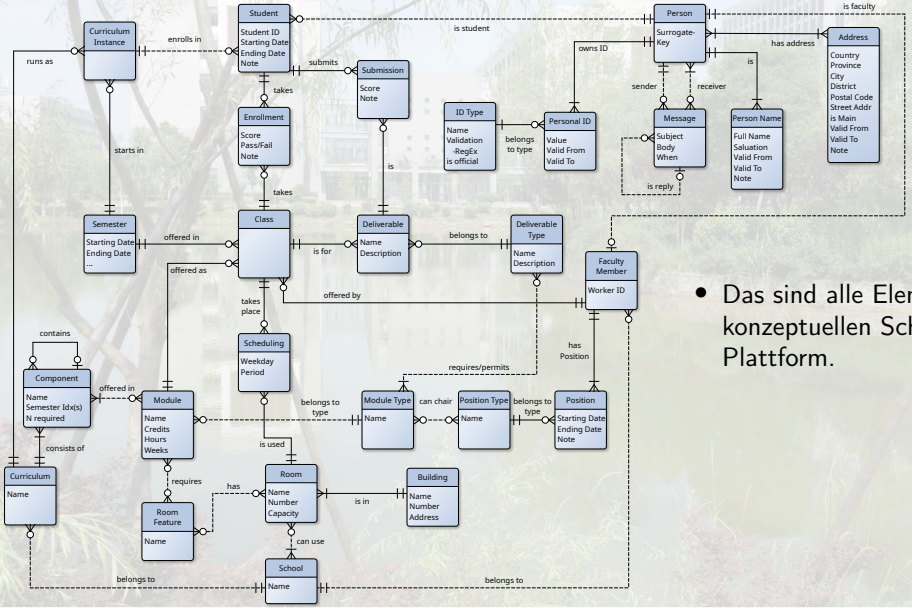
- Der Lehrer bewertet sie und erstellt ein entsprechende (schwache) *Submission* Entitäten – eine pro Student – im System.
- Diese schwache Entität enthält das Ergebnis des Studenten und vielleicht ein Kommentar vom Lehrer.
- So kann der Lehrer Prüfungsergebnise, Midterm-Ergebnisse, usw. hochladen.
- Die Studenten sehen diese dann im Onlinesystem.

Deliverable System



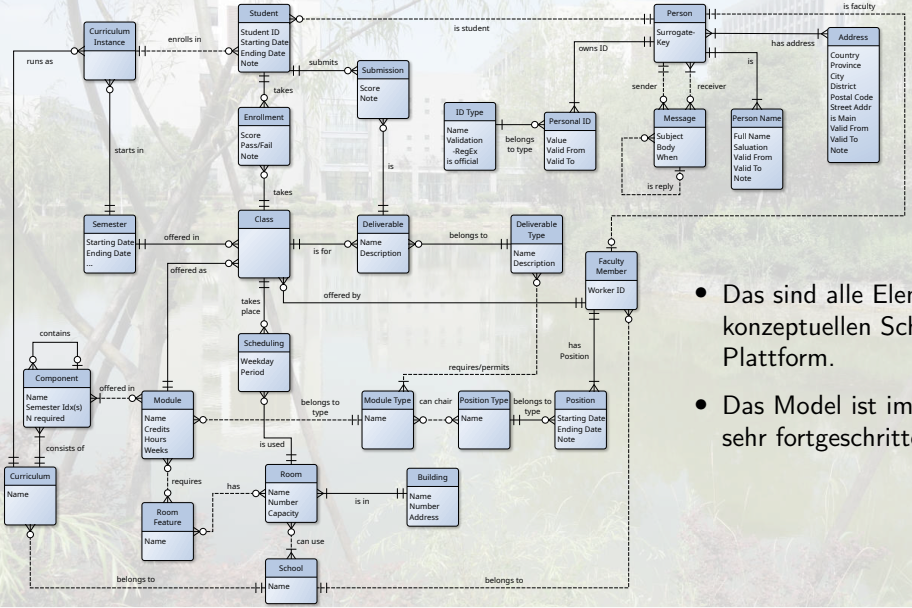
- Diese schwache Entität enthält das Ergebnis des Studenten und vielleicht ein Kommentar vom Lehrer.
- So kann der Lehrer Prüfungsergebnise, Midterm-Ergebnisse, usw. hochladen.
- Die Studenten sehen diese dann im Onlinesystem.
- Wir haben hier nur ein Attribut *Score* hingeschrieben, aber wir können uns auch weitere Attribute wie *Pass/Fail* vorstellen, die hier Sinn ergeben.

Lehre-Management-System



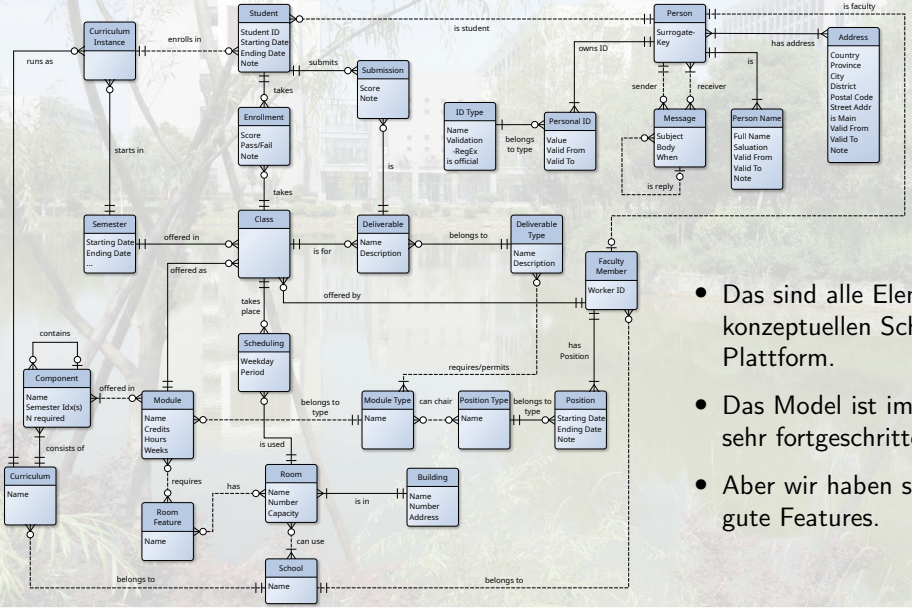
- Das sind alle Elemente des konzeptuellen Schemas unserer Plattform.

Lehre-Management-System



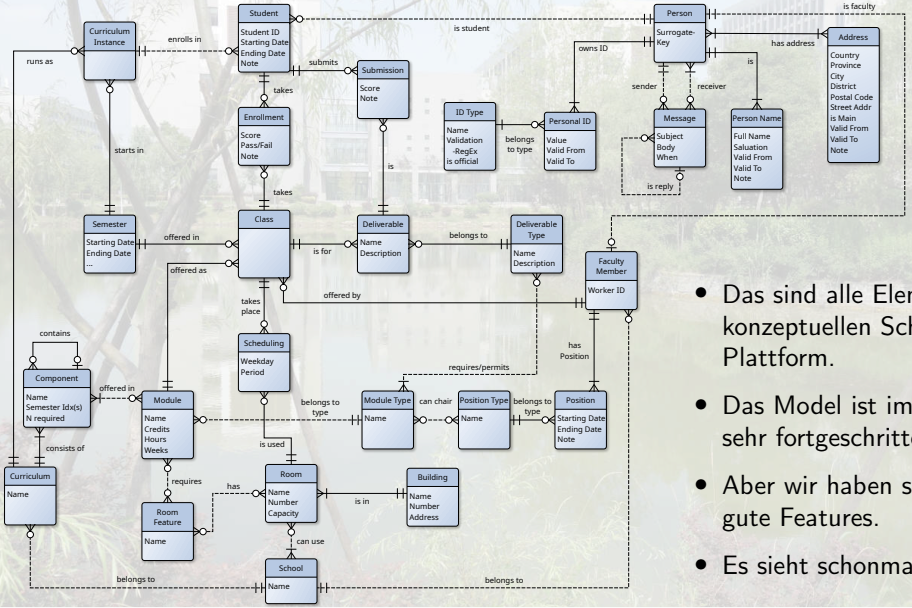
- Das sind alle Elemente des konzeptuellen Schemas unserer Plattform.
- Das Model ist immer noch nicht sehr fortgeschritten.

Lehre-Management-System



- Das sind alle Elemente des konzeptuellen Schemas unserer Plattform.
- Das Model ist immer noch nicht sehr fortgeschritten.
- Aber wir haben schon ein paar gute Features.

Lehre-Management-System



- Das sind alle Elemente des konzeptuellen Schemas unserer Plattform.
- Das Model ist immer noch nicht sehr fortgeschritten.
- Aber wir haben schon ein paar gute Features.
- Es sieht schonmal benutzbar aus.

Nur Einfügen

Nur Einfügen

- Wir haben das vorher nicht explitiz gesagt, aber unser Model für eine Lehre-Plattform folgt dem Prinzip einer „Nur Einfügen Datenbank“ (EN: *Insert-Only Database*)[52].

Nur Einfügen

- Wir haben das vorher nicht explitiz gesagt, aber unser Model für eine Lehre-Plattform folgt dem Prinzip einer „Nur Einfügen Datenbank“ (EN: *Insert-Only Database*)[52]:

Gute Praxis

In vielen Anwendungen wo historische Informationen bewahrt werden müssen, dürfen die Daten in der Datenbank niemals verändert oder gelöscht werden.

Nur Einfügen

- Wir haben das vorher nicht explitiz gesagt, aber unser Model für eine Lehre-Plattform folgt dem Prinzip einer „Nur Einfügen Datenbank“ (EN: *Insert-Only Database*)[52]:

Gute Praxis

In vielen Anwendungen wo historische Informationen bewahrt werden müssen, dürfen die Daten in der Datenbank niemals verändert oder gelöscht werden. Änderungen in der realen Welt werden stattdessen durch Anfügen von Daten an die Datenbank reflektiert.

Nur Einfügen

- Wir haben das vorher nicht explitiz gesagt, aber unser Model für eine Lehre-Plattform folgt dem Prinzip einer „Nur Einfügen Datenbank“ (EN: *Insert-Only Database*)[52]:

Gute Praxis

In vielen Anwendungen wo historische Informationen bewahrt werden müssen, dürfen die Daten in der Datenbank niemals verändert oder gelöscht werden. Änderungen in der realen Welt werden stattdessen durch Anfügen von Daten an die Datenbank reflektiert.

- Aus der SQL-Perspektive würde man sagen, dass die Operationen `UPDATE` und `DELETE` dann nicht verwendet werden dürfen.

Nur Einfügen

- Wir haben das vorher nicht explitiz gesagt, aber unser Model für eine Lehre-Plattform folgt dem Prinzip einer „Nur Einfügen Datenbank“ (EN: *Insert-Only Database*)[52]:

Gute Praxis

In vielen Anwendungen wo historische Informationen bewahrt werden müssen, dürfen die Daten in der Datenbank niemals verändert oder gelöscht werden. Änderungen in der realen Welt werden stattdessen durch Anfügen von Daten an die Datenbank reflektiert.

- Aus der SQL-Perspektive würde man sagen, dass die Operationen `UPDATE` und `DELETE` dann nicht verwendet werden dürfen.
- Stattdessen werden nur neue Datensätze mit `INSERT INTO` angehängt.

Nur Einfügen

- Wir haben das vorher nicht explitiz gesagt, aber unser Model für eine Lehre-Plattform folgt dem Prinzip einer „Nur Einfügen Datenbank“ (EN: *Insert-Only Database*)[52]:

Gute Praxis

In vielen Anwendungen wo historische Informationen bewahrt werden müssen, dürfen die Daten in der Datenbank niemals verändert oder gelöscht werden. Änderungen in der realen Welt werden stattdessen durch Anfügen von Daten an die Datenbank reflektiert.

- Aus der SQL-Perspektive würde man sagen, dass die Operationen `UPDATE` und `DELETE` dann nicht verwendet werden dürfen.
- Stattdessen werden nur neue Datensätze mit `INSERT INTO` angehängt.
- Und in unserer Situation wollen wir natürlich alle Daten über die Vergangenheit bewahren.

Nur Einfügen

- Wir haben das vorher nicht explitiz gesagt, aber unser Model für eine Lehre-Plattform folgt dem Prinzip einer „Nur Einfügen Datenbank“ (EN: *Insert-Only Database*)[52]:

Gute Praxis

In vielen Anwendungen wo historische Informationen bewahrt werden müssen, dürfen die Daten in der Datenbank niemals verändert oder gelöscht werden. Änderungen in der realen Welt werden stattdessen durch Anfügen von Daten an die Datenbank reflektiert.

- Aus der SQL-Perspektive würde man sagen, dass die Operationen `UPDATE` und `DELETE` dann nicht verwendet werden dürfen.
- Stattdessen werden nur neue Datensätze mit `INSERT INTO` angehängt.
- Und in unserer Situation wollen wir natürlich alle Daten über die Vergangenheit bewahren.
- Ein Studiengang besteht z. B. aus Modulen.

Nur Einfügen

Gute Praxis

In vielen Anwendungen wo historische Informationen bewahrt werden müssen, dürfen die Daten in der Datenbank niemals verändert oder gelöscht werden. Änderungen in der realen Welt werden stattdessen durch Anfügen von Daten an die Datenbank reflektiert.

- Aus der SQL-Perspektive würde man sagen, dass die Operationen `UPDATE` und `DELETE` dann nicht verwendet werden dürfen.
- Stattdessen werden nur neue Datensätze mit `INSERT INTO` angehängt.
- Und in unserer Situation wollen wir natürlich alle Daten über die Vergangenheit bewahren.
- Ein Studiengang besteht z. B. aus Modulen.
- Wir hätten das so modellieren können, das jedes Modul genau einem Lehrer zugewiesen ist.

Nur Einfügen

Gute Praxis

In vielen Anwendungen wo historische Informationen bewahrt werden müssen, dürfen die Daten in der Datenbank niemals verändert oder gelöscht werden. Änderungen in der realen Welt werden stattdessen durch Anfügen von Daten an die Datenbank reflektiert.

- Aus der SQL-Perspektive würde man sagen, dass die Operationen `UPDATE` und `DELETE` dann nicht verwendet werden dürfen.
- Stattdessen werden nur neue Datensätze mit `INSERT INTO` angehängt.
- Und in unserer Situation wollen wir natürlich alle Daten über die Vergangenheit bewahren.
- Ein Studiengang besteht z. B. aus Modulen.
- Wir hätten das so modellieren können, das jedes Modul genau einem Lehrer zugewiesen ist.
- Wenn dieser Lehrer seinen Arbeitsplatz wechselt und ein anderer Lehrer übernimmt, dann könnten wir die Zuweisung des Moduls einfach ändern.

Nur Einfügen

Gute Praxis

In vielen Anwendungen wo historische Informationen bewahrt werden müssen, dürfen die Daten in der Datenbank niemals verändert oder gelöscht werden. Änderungen in der realen Welt werden stattdessen durch Anfügen von Daten an die Datenbank reflektiert.

- Aus der SQL-Perspektive würde man sagen, dass die Operationen `UPDATE` und `DELETE` dann nicht verwendet werden dürfen.
- Stattdessen werden nur neue Datensätze mit `INSERT INTO` angehängt.
- Und in unserer Situation wollen wir natürlich alle Daten über die Vergangenheit bewahren.
- Ein Studiengang besteht z. B. aus Modulen.
- Wir hätten das so modellieren können, das jedes Modul genau einem Lehrer zugewiesen ist.
- Wenn dieser Lehrer seinen Arbeitsplatz wechselt und ein anderer Lehrer übernimmt, dann könnten wir die Zuweisung des Moduls einfach ändern.
- Das wäre viel einfacher als unser aktuelles Modell.

Nur Einfügen

- Aus der SQL-Perspektive würde man sagen, dass die Operationen `UPDATE` und `DELETE` dann nicht verwendet werden dürfen.
- Stattdessen werden nur neue Datensätze mit `INSERT INTO` angehängt.
- Und in unserer Situation wollen wir natürlich alle Daten über die Vergangenheit bewahren.
- Ein Studiengang besteht z. B. aus Modulen.
- Wir hätten das so modellieren können, das jedes Modul genau einem Lehrer zugewiesen ist.
- Wenn dieser Lehrer seinen Arbeitsplatz wechselt und ein anderer Lehrer übernimmt, dann könnten wir die Zuweisung des Moduls einfach ändern.
- Das wäre viel einfacher als unser aktuelles Modell.
- Aber es hätte einen entscheidenten Nachteil.

Nur Einfügen

- Aus der SQL-Perspektive würde man sagen, dass die Operationen `UPDATE` und `DELETE` dann nicht verwendet werden dürfen.
- Stattdessen werden nur neue Datensätze mit `INSERT INTO` angehängt.
- Und in unserer Situation wollen wir natürlich alle Daten über die Vergangenheit bewahren.
- Ein Studiengang besteht z. B. aus Modulen.
- Wir hätten das so modellieren können, das jedes Modul genau einem Lehrer zugewiesen ist.
- Wenn dieser Lehrer seinen Arbeitsplatz wechselt und ein anderer Lehrer übernimmt, dann könnten wir die Zuweisung des Moduls einfach ändern.
- Das wäre viel einfacher als unser aktuelles Modell.
- Aber es hätte einen entscheidenten Nachteil:
- Es wäre nicht mehr möglich, festzustellen, welcher Lehrer das Modul früher unterrichtet hat.

Nur Einfügen

- Aus der SQL-Perspektive würde man sagen, dass die Operationen `UPDATE` und `DELETE` dann nicht verwendet werden dürfen.
- Stattdessen werden nur neue Datensätze mit `INSERT INTO` angehängt.
- Und in unserer Situation wollen wir natürlich alle Daten über die Vergangenheit bewahren.
- Ein Studiengang besteht z. B. aus Modulen.
- Wir hätten das so modellieren können, das jedes Modul genau einem Lehrer zugewiesen ist.
- Wenn dieser Lehrer seinen Arbeitsplatz wechselt und ein anderer Lehrer übernimmt, dann könnten wir die Zuweisung des Moduls einfach ändern.
- Das wäre viel einfacher als unser aktuelles Modell.
- Aber es hätte einen entscheidenten Nachteil:
- Es wäre nicht mehr möglich, festzustellen, welcher Lehrer das Modul früher unterrichtet hat.
- Deshalb sind wir einen anderen Weg gegangen.

Nur Einfügen

- Aus der SQL-Perspektive würde man sagen, dass die Operationen `UPDATE` und `DELETE` dann nicht verwendet werden dürfen.
- Stattdessen werden nur neue Datensätze mit `INSERT INTO` angehängt.
- Und in unserer Situation wollen wir natürlich alle Daten über die Vergangenheit bewahren.
- Ein Studiengang besteht z. B. aus Modulen.
- Wir hätten das so modellieren können, das jedes Modul genau einem Lehrer zugewiesen ist.
- Wenn dieser Lehrer seinen Arbeitsplatz wechselt und ein anderer Lehrer übernimmt, dann könnten wir die Zuweisung des Moduls einfach ändern.
- Das wäre viel einfacher als unser aktuelles Modell.
- Aber es hätte einen entscheidenten Nachteil:
- Es wäre nicht mehr möglich, festzustellen, welcher Lehrer das Modul früher unterrichtet hat.
- Deshalb sind wir einen anderen Weg gegangen:
- Anstatt Module direkt Lehrern zuzuweisen, erstellen wir Instanzen von Modulen (die *Class* Entities), welche dann an Semester und Lehrer gebunden sind.

Nur Einfügen

- Aus der SQL-Perspektive würde man sagen, dass die Operationen `UPDATE` und `DELETE` dann nicht verwendet werden dürfen.
- Stattdessen werden nur neue Datensätze mit `INSERT INTO` angehängt.
- Und in unserer Situation wollen wir natürlich alle Daten über die Vergangenheit bewahren.
- Ein Studiengang besteht z. B. aus Modulen.
- Wir hätten das so modellieren können, das jedes Modul genau einem Lehrer zugewiesen ist.
- Wenn dieser Lehrer seinen Arbeitsplatz wechselt und ein anderer Lehrer übernimmt, dann könnten wir die Zuweisung des Moduls einfach ändern.
- Das wäre viel einfacher als unser aktuelles Modell.
- Aber es hätte einen entscheidenten Nachteil:
- Es wäre nicht mehr möglich, festzustellen, welcher Lehrer das Modul früher unterrichtet hat.
- Deshalb sind wir einen anderen Weg gegangen:
- Anstatt Module direkt Lehrern zuzuweisen, erstellen wir Instanzen von Modulen (die *Class* Entities), welche dann an Semester und Lehrer gebunden sind.
- Diese Zuweisungen ändern sich nie.

Nur Einfügen

- Stattdessen werden nur neue Datensätze mit `INSERT INTO` angehängt.
- Und in unserer Situation wollen wir natürlich alle Daten über die Vergangenheit bewahren.
- Ein Studiengang besteht z. B. aus Modulen.
- Wir hätten das so modellieren können, das jedes Modul genau einem Lehrer zugewiesen ist.
- Wenn dieser Lehrer seinen Arbeitsplatz wechselt und ein anderer Lehrer übernimmt, dann könnten wir die Zuweisung des Moduls einfach ändern.
- Das wäre viel einfacher als unser aktuelles Modell.
- Aber es hätte einen entscheidenten Nachteil:
- Es wäre nicht mehr möglich, festzustellen, welcher Lehrer das Modul früher unterrichtet hat.
- Deshalb sind wir einen anderen Weg gegangen:
- Anstatt Module direkt Lehrern zuzuweisen, erstellen wir Instanzen von Modulen (die *Class* Entities), welche dann an Semester und Lehrer gebunden sind.
- Diese Zuweisungen ändern sich nie.
- Jedes Semester werden neue Class-Datensätze eingefügt.

Nur Einfügen

- Und in unserer Situation wollen wir natürlich alle Daten über die Vergangenheit bewahren.
- Ein Studiengang besteht z. B. aus Modulen.
- Wir hätten das so modellieren können, das jedes Modul genau einem Lehrer zugewiesen ist.
- Wenn dieser Lehrer seinen Arbeitsplatz wechselt und ein anderer Lehrer übernimmt, dann könnten wir die Zuweisung des Moduls einfach ändern.
- Das wäre viel einfacher als unser aktuelles Modell.
- Aber es hätte einen entscheidenten Nachteil:
- Es wäre nicht mehr möglich, festzustellen, welcher Lehrer das Modul früher unterrichtet hat.
- Deshalb sind wir einen anderen Weg gegangen:
- Anstatt Module direkt Lehrern zuzuweisen, erstellen wir Instanzen von Modulen (die *Class* Entities), welche dann an Semester und Lehrer gebunden sind.
- Diese Zuweisungen ändern sich nie.
- Jedes Semester werden neue Class-Datensätze eingefügt.
- Wir wissen immer, wer welchen Kurs wann unterrichtet hat.

Nur Einfügen

- Wir hätten das so modellieren können, das jedes Modul genau einem Lehrer zugewiesen ist.
- Wenn dieser Lehrer seinen Arbeitsplatz wechselt und ein anderer Lehrer übernimmt, dann könnten wir die Zuweisung des Moduls einfach ändern.
- Das wäre viel einfacher als unser aktuelles Modell.
- Aber es hätte einen entscheidenten Nachteil:
- Es wäre nicht mehr möglich, festzustellen, welcher Lehrer das Modul früher unterrichtet hat.
- Deshalb sind wir einen anderen Weg gegangen:
- Anstatt Module direkt Lehrern zuzuweisen, erstellen wir Instanzen von Modulen (die *Class* Entities), welche dann an Semester und Lehrer gebunden sind.
- Diese Zuweisungen ändern sich nie.
- Jedes Semester werden neue Class-Datensätze eingefügt.
- Wir wissen immer, wer welchen Kurs wann unterrichtet hat.
- Unsere Datenbank wird schrittweise wachsen und Änderungen entsprechen neuen Datensätzen und nicht Veränderungen existierender Datensätze.

Nur Einfügen

- Wenn dieser Lehrer seinen Arbeitsplatz wechselt und ein anderer Lehrer übernimmt, dann könnten wir die Zuweisung des Moduls einfach ändern.
- Das wäre viel einfacher als unser aktuelles Modell.
- Aber es hätte einen entscheidenten Nachteil:
- Es wäre nicht mehr möglich, festzustellen, welcher Lehrer das Modul früher unterrichtet hat.
- Deshalb sind wir einen anderen Weg gegangen:
- Anstatt Module direkt Lehrern zuzuweisen, erstellen wir Instanzen von Modulen (die *Class* Entities), welche dann an Semester und Lehrer gebunden sind.
- Diese Zuweisungen ändern sich nie.
- Jedes Semester werden neue Class-Datensätze eingefügt.
- Wir wissen immer, wer welchen Kurs wann unterrichtet hat.
- Unsere Datenbank wird schrittweise wachsen und Änderungen entsprechen neuen Datensätzen und nicht Veränderungen existierender Datensätze.
- Wenn ein Modul von einem neuen Professor übernommen wird, dann gibt es neue Class-Datensätze.

Nur Einfügen

- Das wäre viel einfacher als unser aktuelles Modell.
- Aber es hätte einen entscheidenten Nachteil:
- Es wäre nicht mehr möglich, festzustellen, welcher Lehrer das Modul früher unterrichtet hat.
- Deshalb sind wir einen anderen Weg gegangen:
- Anstatt Module direkt Lehrern zuzuweisen, erstellen wir Instanzen von Modulen (die *Class* Entities), welche dann an Semester und Lehrer gebunden sind.
- Diese Zuweisungen ändern sich nie.
- Jedes Semester werden neue Class-Datensätze eingefügt.
- Wir wissen immer, wer welchen Kurs wann unterrichtet hat.
- Unsere Datenbank wird schrittweise wachsen und Änderungen entsprechen neuen Datensätzen und nicht Veränderungen existierender Datensätze.
- Wenn ein Modul von einem neuen Professor übernommen wird, dann gibt es neue Class-Datensätze.
- Aus diesem Grund hat der *Address*-Entitätstyp auch die Attribute *Valid From* und *Valid To*.

Nur Einfügen

- Es wäre nicht mehr möglich, festzustellen, welcher Lehrer das Modul früher unterrichtet hat.
- Deshalb sind wir einen anderen Weg gegangen:
- Anstatt Module direkt Lehrern zuzuweisen, erstellen wir Instanzen von Modulen (die *Class* Entities), welche dann an Semester und Lehrer gebunden sind.
- Diese Zuweisungen ändern sich nie.
- Jedes Semester werden neue Class-Datensätze eingefügt.
- Wir wissen immer, wer welchen Kurs wann unterrichtet hat.
- Unsere Datenbank wird schrittweise wachsen und Änderungen entsprechen neuen Datensätzen und nicht Veränderungen existierender Datensätze.
- Wenn ein Modul von einem neuen Professor übernommen wird, dann gibt es neue Class-Datensätze.
- Aus diesem Grund hat der *Address*-Entitätstyp auch die Attribute *Valid From* und *Valid To*.
- Wenn sich die Addresse eines Studenten oder Mitarbeiters ändert, dann erstellen wir einen neuen Datensatz, setzen *Valid From* auf jetzt, und *Valid To* auf `NULL`.

Nur Einfügen

- Deshalb sind wir einen anderen Weg gegangen:
- Anstatt Module direkt Lehrern zuzuweisen, erstellen wir Instanzen von Modulen (die *Class* Entities), welche dann an Semester und Lehrer gebunden sind.
- Diese Zuweisungen ändern sich nie.
- Jedes Semester werden neue Class-Datensätze eingefügt.
- Wir wissen immer, wer welchen Kurs wann unterrichtet hat.
- Unsere Datenbank wird schrittweise wachsen und Änderungen entsprechen neuen Datensätzen und nicht Veränderungen existierender Datensätze.
- Wenn ein Modul von einem neuen Professor übernommen wird, dann gibt es neue Class-Datensätze.
- Aus diesem Grund hat der *Address*-Entitätstyp auch die Attribute *Valid From* und *Valid To*.
- Wenn sich die Addresse eines Studenten oder Mitarbeiters ändert, dann erstellen wir einen neuen Datensatz, setzen *Valid From* auf jetzt, und *Valid To* auf `NULL`.
- Wir können dann gestern als *Valid To*-Wert für die alte Addresse speichern.

Nur Einfügen

- Deshalb sind wir einen anderen Weg gegangen:
- Anstatt Module direkt Lehrern zuzuweisen, erstellen wir Instanzen von Modulen (die *Class* Entities), welche dann an Semester und Lehrer gebunden sind.
- Diese Zuweisungen ändern sich nie.
- Jedes Semester werden neue Class-Datensätze eingefügt.
- Wir wissen immer, wer welchen Kurs wann unterrichtet hat.
- Unsere Datenbank wird schrittweise wachsen und Änderungen entsprechen neuen Datensätzen und nicht Veränderungen existierender Datensätze.
- Wenn ein Modul von einem neuen Professor übernommen wird, dann gibt es neue Class-Datensätze.
- Aus diesem Grund hat der *Address*-Entitätstyp auch die Attribute *Valid From* und *Valid To*.
- Wenn sich die Addresse eines Studenten oder Mitarbeiters ändert, dann erstellen wir einen neuen Datensatz, setzen *Valid From* auf jetzt, und *Valid To* auf `NULL`.
- Wir können dann gestern als *Valid To*-Wert für die alte Addresse speichern.
- Genauso funktioniert das für die *Personal ID* und *Person Name* Entitätstypen.

Nur Einfügen

- Diese Zuweisungen ändern sich nie.
- Jedes Semester werden neue Class-Datensätze eingefügt.
- Wir wissen immer, wer welchen Kurs wann unterrichtet hat.
- Unsere Datenbank wird schrittweise wachsen und Änderungen entsprechen neuen Datensätzen und nicht Veränderungen existierender Datensätze.
- Wenn ein Modul von einem neuen Professor übernommen wird, dann gibt es neue Class-Datensätze.
- Aus diesem Grund hat der *Address*-Entitätstyp auch die Attribute *Valid From* und *Valid To*.
- Wenn sich die Addresse eines Studenten oder Mitarbeiters ändert, dann erstellen wir einen neuen Datensatz, setzen *Valid From* auf jetzt, und *Valid To* auf `NULL`.
- Wir können dann gestern als *Valid To*-Wert für die alte Addresse speichern.
- Genauso funktioniert das für die *Personal ID* und *Person Name* Entitätstypen.
- Unsere Administratoren können daher immer sehen, welche Adresse, ID, oder Namen eine Person zu einem beliebigen Zeitpunkt hatte.

Schwachstellen

Schwachstellen

- Das konzeptuelle Modell unseres Systems hat auch noch ein paar Schwachstellen.

Schwachstellen

- Das konzeptuelle Modell unseres Systems hat auch noch ein paar Schwachstellen.
- Wahrscheinlich gibt es mehr oganisatorische Ebenen in einer Uni als nur Fakultäten.

Schwachstellen

- Das konzeptuelle Modell unseres Systems hat auch noch ein paar Schwachstellen.
- Wahrscheinlich gibt es mehr oganisatorische Ebenen in einer Uni als nur Fakultäten.
- Eine Fakultät kann sicherlich in Departments und Departments in Teams unterteilt werden.

Schwachstellen

- Das konzeptuelle Modell unseres Systems hat auch noch ein paar Schwachstellen.
- Wahrscheinlich gibt es mehr oganisatorische Ebenen in einer Uni als nur Fakultäten.
- Eine Fakultät kann sicherlich in Departments und Departments in Teams unterteilt werden.
- Außerdem sollten wir wohl Zeitstempel und Notizen an die meisten Datensätze mit anhängen, um nachvollziehen zu können, wann sie erstellt wurden.

Schwachstellen

- Das konzeptuelle Modell unseres Systems hat auch noch ein paar Schwachstellen.
- Wahrscheinlich gibt es mehr oganisatorische Ebenen in einer Uni als nur Fakultäten.
- Eine Fakultät kann sicherlich in Departments und Departments in Teams unterteilt werden.
- Außerdem sollten wir wohl Zeitstempel und Notizen an die meisten Datensätze mit anhängen, um nachvollziehen zu können, wann sie erstellt wurden.
- Diese Technik nennt man timestamping – und wir haben das nicht mit modelliert.

Schwachstellen

- Das konzeptuelle Modell unseres Systems hat auch noch ein paar Schwachstellen.
- Wahrscheinlich gibt es mehr oganisatorische Ebenen in einer Uni als nur Fakultäten.
- Eine Fakultät kann sicherlich in Departments und Departments in Teams unterteilt werden.
- Außerdem sollten wir wohl Zeitstempel und Notizen an die meisten Datensätze mit anhängen, um nachvollziehen zu können, wann sie erstellt wurden.
- Diese Technik nennt man timestamping – und wir haben das nicht mit modelliert.
- Wir haben auch das Rechtemanagement nicht mit modelliert.

Schwachstellen

- Das konzeptuelle Modell unseres Systems hat auch noch ein paar Schwachstellen.
- Wahrscheinlich gibt es mehr oganisatorische Ebenen in einer Uni als nur Fakultäten.
- Eine Fakultät kann sicherlich in Departments und Departments in Teams unterteilt werden.
- Außerdem sollten wir wohl Zeitstempel und Notizen an die meisten Datensätze mit anhängen, um nachvollziehen zu können, wann sie erstellt wurden.
- Diese Technik nennt man timestamping – und wir haben das nicht mit modelliert.
- Wir haben auch das Rechtemanagement nicht mit modelliert.
- Wir brauchen wahrscheinlich eine Tabelle, um zu managen, wer Studenten in einen Studiengang einpflegen kann, wer neue Module erstellen darf, usw.

Schwachstellen

- Das konzeptuelle Modell unseres Systems hat auch noch ein paar Schwachstellen.
- Wahrscheinlich gibt es mehr oganisatorische Ebenen in einer Uni als nur Fakultäten.
- Eine Fakultät kann sicherlich in Departments und Departments in Teams unterteilt werden.
- Außerdem sollten wir wohl Zeitstempel und Notizen an die meisten Datensätze mit anhängen, um nachvollziehen zu können, wann sie erstellt wurden.
- Diese Technik nennt man timestamping – und wir haben das nicht mit modelliert.
- Wir haben auch das Rechtemanagement nicht mit modelliert.
- Wir brauchen wahrscheinlich eine Tabelle, um zu managen, wer Studenten in einen Studiengang einpflegen kann, wer neue Module erstellen darf, usw.
- Vielleicht kann man das auch über die *Position*-Entitäten managen, die wir schon designed haben.

Schwachstellen

- Das konzeptuelle Modell unseres Systems hat auch noch ein paar Schwachstellen.
- Wahrscheinlich gibt es mehr oganisatorische Ebenen in einer Uni als nur Fakultäten.
- Eine Fakultät kann sicherlich in Departments und Departments in Teams unterteilt werden.
- Außerdem sollten wir wohl Zeitstempel und Notizen an die meisten Datensätze mit anhängen, um nachvollziehen zu können, wann sie erstellt wurden.
- Diese Technik nennt man timestamping – und wir haben das nicht mit modelliert.
- Wir haben auch das Rechtemanagement nicht mit modelliert.
- Wir brauchen wahrscheinlich eine Tabelle, um zu managen, wer Studenten in einen Studiengang einpflegen kann, wer neue Module erstellen darf, usw.
- Vielleicht kann man das auch über die *Position*-Entitäten managen, die wir schon designed haben.
- Das wäre wohl eine gute Idee.

Schwachstellen

- Das konzeptuelle Modell unseres Systems hat auch noch ein paar Schwachstellen.
- Wahrscheinlich gibt es mehr oganisatorische Ebenen in einer Uni als nur Fakultäten.
- Eine Fakultät kann sicherlich in Departments und Departments in Teams unterteilt werden.
- Außerdem sollten wir wohl Zeitstempel und Notizen an die meisten Datensätze mit anhängen, um nachvollziehen zu können, wann sie erstellt wurden.
- Diese Technik nennt man timestamping – und wir haben das nicht mit modelliert.
- Wir haben auch das Rechtemanagement nicht mit modelliert.
- Wir brauchen wahrscheinlich eine Tabelle, um zu managen, wer Studenten in einen Studiengang einpflegen kann, wer neue Module erstellen darf, usw.
- Vielleicht kann man das auch über die *Position*-Entitäten managen, die wir schon designed haben.
- Das wäre wohl eine gute Idee.
- Vielleicht wollen wir auch eine Tabelle, um einen Audit-Trail[39] zu erzeugen.

Schwachstellen

- Wahrscheinlich gibt es mehr oganisatorische Ebenen in einer Uni als nur Fakultäten.
- Eine Fakultät kann sicherlich in Departments und Departments in Teams unterteilt werden.
- Außerdem sollten wir wohl Zeitstempel und Notizen an die meisten Datensätze mit anhängen, um nachvollziehen zu können, wann sie erstellt wurden.
- Diese Technik nennt man timestamping – und wir haben das nicht mit modelliert.
- Wir haben auch das Rechtemanagement nicht mit modelliert.
- Wir brauchen wahrscheinlich eine Tabelle, um zu managen, wer Studenten in einen Studiengang einpflegen kann, wer neue Module erstellen darf, usw.
- Vielleicht kann man das auch über die *Position*-Entitäten managen, die wir schon designed haben.
- Das wäre wohl eine gute Idee.
- Vielleicht wollen wir auch eine Tabelle, um einen Audit-Trail[39] zu erzeugen.
- Diese Tabelle würde speichern, welcher Benutzer welchen Record zu welcher Zeit erstellt oder verändert hat.

Schwachstellen

- Eine Fakultät kann sicherlich in Departments und Departments in Teams unterteilt werden.
- Außerdem sollten wir wohl Zeitstempel und Notizen an die meisten Datensätze mit anhängen, um nachvollziehen zu können, wann sie erstellt wurden.
- Diese Technik nennt man timestamping – und wir haben das nicht mit modelliert.
- Wir haben auch das Rechtemanagement nicht mit modelliert.
- Wir brauchen wahrscheinlich eine Tabelle, um zu managen, wer Studenten in einen Studiengang einpflegen kann, wer neue Module erstellen darf, usw.
- Vielleicht kann man das auch über die *Position*-Entitäten managen, die wir schon designed haben.
- Das wäre wohl eine gute Idee.
- Vielleicht wollen wir auch eine Tabelle, um einen Audit-Trail[39] zu erzeugen.
- Diese Tabelle würde speichern, welcher Benutzer welchen Record zu welcher Zeit erstellt oder verändert hat.
- In unserer *Address* sind Länder und Provinzen einfache Attribute.

Schwachstellen

- Außerdem sollten wir wohl Zeitstempel und Notizen an die meisten Datensätze mit anhängen, um nachvollziehen zu können, wann sie erstellt wurden.
- Diese Technik nennt man timestamping – und wir haben das nicht mit modelliert.
- Wir haben auch das Rechtemanagement nicht mit modelliert.
- Wir brauchen wahrscheinlich eine Tabelle, um zu managen, wer Studenten in einen Studiengang einpflegen kann, wer neue Module erstellen darf, usw.
- Vielleicht kann man das auch über die *Position*-Entitäten managen, die wir schon designed haben.
- Das wäre wohl eine gute Idee.
- Vielleicht wollen wir auch eine Tabelle, um einen Audit-Trail[39] zu erzeugen.
- Diese Tabelle würde speichern, welcher Benutzer welchen Record zu welcher Zeit erstellt oder verändert hat.
- In unserer *Address* sind Länder und Provinzen einfache Attribute.
- Wir könnten wahrscheinlich einen weiteren Entitätstyp haben, der Länder und Länder-Teile (wie Provinzen) basierend auf dem ISO 3166 Standard[20,37] speichert, vielleicht sogar mit weiteren Informationen wir Telefonnumer-Vorwahl.

Schwachstellen

- Diese Technik nennt man timestamping – und wir haben das nicht mit modelliert.
- Wir haben auch das Rechtemanagement nicht mit modelliert.
- Wir brauchen wahrscheinlich eine Tabelle, um zu managen, wer Studenten in einen Studiengang einpflegen kann, wer neue Module erstellen darf, usw.
- Vielleicht kann man das auch über die *Position*-Entitäten managen, die wir schon designed haben.
- Das wäre wohl eine gute Idee.
- Vielleicht wollen wir auch eine Tabelle, um einen Audit-Trail[39] zu erzeugen.
- Diese Tabelle würde speichern, welcher Benutzer welchen Record zu welcher Zeit erstellt oder verändert hat.
- In unserer *Address* sind Länder und Provinzen einfache Attribute.
- Wir könnten wahrscheinlich einen weiteren Entitätstyp haben, der Länder und Länder-Teile (wie Provinzen) basierend auf dem ISO 3166 Standard[20,37] speichert, vielleicht sogar mit weiteren Informationen wir Telefonnumer-Vorwahl.
- Jede Adresse könnte zu so einer Entität verlinkt sein.

Schwachstellen

- Wir haben auch das Rechtemanagement nicht mit modelliert.
- Wir brauchen wahrscheinlich eine Tabelle, um zu managen, wer Studenten in einen Studiengang einpflegen kann, wer neue Module erstellen darf, usw.
- Vielleicht kann man das auch über die *Position*-Entitäten managen, die wir schon designed haben.
- Das wäre wohl eine gute Idee.
- Vielleicht wollen wir auch eine Tabelle, um einen Audit-Trail[39] zu erzeugen.
- Diese Tabelle würde speichern, welcher Benutzer welchen Record zu welcher Zeit erstellt oder verändert hat.
- In unserer *Address* sind Länder und Provinzen einfache Attribute.
- Wir könnten wahrscheinlich einen weiteren Entitätstyp haben, der Länder und Länder-Teile (wie Provinzen) basierend auf dem ISO 3166 Standard[20,37] speichert, vielleicht sogar mit weiteren Informationen wir Telefonnumer-Vorwahl.
- Jede Adresse könnte zu so einer Entität verlinkt sein.
- Jetzt lassen wir das aber erstmal weg.

Schwachstellen

- Wir brauchen wahrscheinlich eine Tabelle, um zu managen, wer Studenten in einen Studiengang einpflegen kann, wer neue Module erstellen darf, usw.
- Vielleicht kann man das auch über die *Position*-Entitäten managen, die wir schon designed haben.
- Das wäre wohl eine gute Idee.
- Vielleicht wollen wir auch eine Tabelle, um einen Audit-Trail[39] zu erzeugen.
- Diese Tabelle würde speichern, welcher Benutzer welchen Record zu welcher Zeit erstellt oder verändert hat.
- In unserer *Address* sind Länder und Provinzen einfache Attribute.
- Wir könnten wahrscheinlich einen weiteren Entitätstyp haben, der Länder und Länder-Teile (wie Provinzen) basierend auf dem ISO 3166 Standard[20,37] speichert, vielleicht sogar mit weiteren Informationen wir Telefonnumer-Vorwahl.
- Jede Adresse könnte zu so einer Entität verlinkt sein.
- Jetzt lassen wir das aber erstmal weg.
- Das ist hier ja nur ein Kurs und kein echtes Systemdesign.

Schwachstellen

- Vielleicht kann man das auch über die *Position*-Entitäten managen, die wir schon designed haben.
- Das wäre wohl eine gute Idee.
- Vielleicht wollen wir auch eine Tabelle, um einen Audit-Trail[39] zu erzeugen.
- Diese Tabelle würde speichern, welcher Benutzer welchen Record zu welcher Zeit erstellt oder verändert hat.
- In unserer *Address* sind Länder und Provinzen einfache Attribute.
- Wir könnten wahrscheinlich einen weiteren Entitätstyp haben, der Länder und Länder-Teile (wie Provinzen) basierend auf dem ISO 3166 Standard[20,37] speichert, vielleicht sogar mit weiteren Informationen wir Telefonnumer-Vorwahl.
- Jede Adresse könnte zu so einer Entität verlinkt sein.
- Jetzt lassen wir das aber erstmal weg.
- Das ist hier ja nur ein Kurs und kein echtes Systemdesign.
- Irgendwann müssen wir ja aufhören, um die Vorlesung nicht zu lange zu machen.

Schwachstellen

- Das wäre wohl eine gute Idee.
- Vielleicht wollen wir auch eine Tabelle, um einen Audit-Trail[39] zu erzeugen.
- Diese Tabelle würde speichern, welcher Benutzer welchen Record zu welcher Zeit erstellt oder verändert hat.
- In unserer *Address* sind Länder und Provinzen einfache Attribute.
- Wir könnten wahrscheinlich einen weiteren Entitätstyp haben, der Länder und Länder-Teile (wie Provinzen) basierend auf dem ISO 3166 Standard[20,37] speichert, vielleicht sogar mit weiteren Informationen wir Telefonnumer-Vorwahl.
- Jede Adresse könnte zu so einer Entität verlinkt sein.
- Jetzt lassen wir das aber erstmal weg.
- Das ist hier ja nur ein Kurs und kein echtes Systemdesign.
- Irgendwann müssen wir ja aufhören, um die Vorlesung nicht zu lange zu machen.
- Das System sieht jetzt halbwegs vernünftig aus und unser Ziel ist nun ein logisches Modell zu erstellen.

Zusammenfassung

Zusammenfassung (1)

- Wir können nun wirklich fast beliebig komplexe Systeme auf konzeptueller Ebene modellieren.

Zusammenfassung (1)

- Wir können nun wirklich fast beliebig komplexe Systeme auf konzeptueller Ebene modellieren.
- Wir können Entitätstypen, Beziehungstypen, Kardinalitäten, Modalitäten, usw. modellieren.

Zusammenfassung (1)

- Wir können nun wirklich fast beliebig komplexe Systeme auf konzeptueller Ebene modellieren.
- Wir können Entitätstypen, Beziehungstypen, Kardinalitäten, Modalitäten, usw. modellieren.
- Wir können eine kompakte Notation verwenden, mit der wir leicht zehn Entitätstypen in ein ERD packen können, ohne die Lesbarkeit zu beeinträchtigen.

Zusammenfassung (1)

- Wir können nun wirklich fast beliebig komplexe Systeme auf konzeptueller Ebene modellieren.
- Wir können Entitätstypen, Beziehungstypen, Kardinalitäten, Modalitäten, usw. modellieren.
- Wir können eine kompakte Notation verwenden, mit der wir leicht zehn Entitätstypen in ein ERD packen können, ohne die Lesbarkeit zu beeinträchtigen.
- In Sachen konzeptueller Modellierung habe wir jetzt alle Werkzeuge, die wir brauchen.

Zusammenfassung (2)

- An diesem Punkt beenden wir also unsere spannede Reise in die wundersame Welt der konzeptuellen Modellierung.

Zusammenfassung (2)

- An diesem Punkt beenden wir also unsere spannede Reise in die wundersame Welt der konzeptuellen Modellierung.
- Dieser wichtige Schritt in der Datenbankentwicklung ist der Übergang von der Anforderungsanalyse hin zur Systemimplementierung.

Zusammenfassung (2)

- An diesem Punkt beenden wir also unsere spannede Reise in die wundersame Welt der konzeptuellen Modellierung.
- Dieser wichtige Schritt in der Datenbankentwicklung ist der Übergang von der Anforderungsanalyse hin zur Systemimplementierung.
- Wir haben die funktionalen Anforderungen in ein halb-formales Modell überführt.

Zusammenfassung (2)

- An diesem Punkt beenden wir also unsere spannede Reise in die wundersame Welt der konzeptuellen Modellierung.
- Dieser wichtige Schritt in der Datenbankentwicklung ist der Übergang von der Anforderungsanalyse hin zur Systemimplementierung.
- Wir haben die funktionalen Anforderungen in ein halb-formales Modell überführt.
- Wir können es immer noch mit den Stakeholders besprechen und verbessern.

Zusammenfassung (2)

- An diesem Punkt beenden wir also unsere spannede Reise in die wundersame Welt der konzeptuellen Modellierung.
- Dieser wichtige Schritt in der Datenbankentwicklung ist der Übergang von der Anforderungsanalyse hin zur Systemimplementierung.
- Wir haben die funktionalen Anforderungen in ein halb-formales Modell überführt.
- Wir können es immer noch mit den Stakeholders besprechen und verbessern.
- Konzeptuelle Modelle sind abstrakt.

Zusammenfassung (2)

- An diesem Punkt beenden wir also unsere spannede Reise in die wundersame Welt der konzeptuellen Modellierung.
- Dieser wichtige Schritt in der Datenbankentwicklung ist der Übergang von der Anforderungsanalyse hin zur Systemimplementierung.
- Wir haben die funktionalen Anforderungen in ein halb-formales Modell überführt.
- Wir können es immer noch mit den Stakeholders besprechen und verbessern.
- Konzeptuelle Modelle sind abstrakt.
- Sie sind nicht an eine bestimmte Datenbank-Technologie gebunden und brauche auch noch nicht mal dem relationalen Modell zu gehorchen.

Zusammenfassung (2)

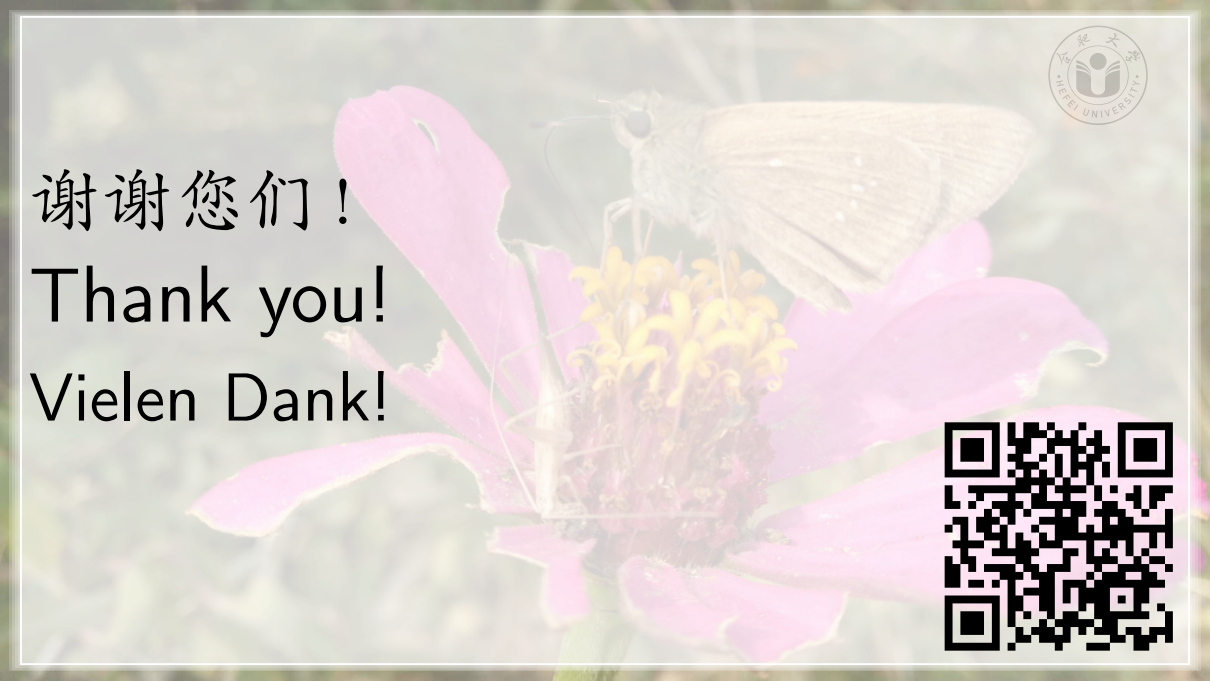
- An diesem Punkt beenden wir also unsere spannede Reise in die wundersame Welt der konzeptuellen Modellierung.
- Dieser wichtige Schritt in der Datenbankentwicklung ist der Übergang von der Anforderungsanalyse hin zur Systemimplementierung.
- Wir haben die funktionalen Anforderungen in ein halb-formales Modell überführt.
- Wir können es immer noch mit den Stakeholders besprechen und verbessern.
- Konzeptuelle Modelle sind abstrakt.
- Sie sind nicht an eine bestimmte Datenbank-Technologie gebunden und brauche auch noch nicht mal dem relationalen Modell zu gehorchen.
- Stattdessen stellen sie den Teil der Welt dar, der für unser System relevant ist – und zwar so klar wie möglich.

Zusammenfassung (2)

- An diesem Punkt beenden wir also unsere spannede Reise in die wundersame Welt der konzeptuellen Modellierung.
- Dieser wichtige Schritt in der Datenbankentwicklung ist der Übergang von der Anforderungsanalyse hin zur Systemimplementierung.
- Wir haben die funktionalen Anforderungen in ein halb-formales Modell überführt.
- Wir können es immer noch mit den Stakeholders besprechen und verbessern.
- Konzeptuelle Modelle sind abstrakt.
- Sie sind nicht an eine bestimmte Datenbank-Technologie gebunden und brauche auch noch nicht mal dem relationalen Modell zu gehorchen.
- Stattdessen stellen sie den Teil der Welt dar, der für unser System relevant ist – und zwar so klar wie möglich.
- Das gibt uns viel Freiheit.

Zusammenfassung (2)

- An diesem Punkt beenden wir also unsere spannede Reise in die wundersame Welt der konzeptuellen Modellierung.
- Dieser wichtige Schritt in der Datenbankentwicklung ist der Übergang von der Anforderungsanalyse hin zur Systemimplementierung.
- Wir haben die funktionalen Anforderungen in ein halb-formales Modell überführt.
- Wir können es immer noch mit den Stakeholders besprechen und verbessern.
- Konzeptuelle Modelle sind abstrakt.
- Sie sind nicht an eine bestimmte Datenbank-Technologie gebunden und brauche auch noch nicht mal dem relationalen Modell zu gehorchen.
- Stattdessen stellen sie den Teil der Welt dar, der für unser System relevant ist – und zwar so klar wie möglich.
- Das gibt uns viel Freiheit.
- Wir können Dinge viel natürlicher modellieren, ohne uns viele Gedanken über die praktische Implementierung zu machen.


谢谢您们！
Thank you!
Vielen Dank!

References I

[1] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume I: Introduction to SQL Queries*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-4-0. See also[2] (siehe S. 279, 289).

[2] Adam Aspin und Karine Aspin. *Query Answers with MariaDB – Volume II: In-Depth Querying*. Tetras Publishing, Okt. 2018. ISBN: 978-1-9996172-5-7. See also[1] (siehe S. 279, 289).

[3] Richard Barker. *Case*Method: Entity Relationship Modelling (Oracle)*. 1. Aufl. Redwood City, CA, USA: Addison Wesley Longman Publishing Co., Inc., Jan. 1990. ISBN: 978-0-201-41696-1 (siehe S. 288).

[4] Daniel J. Barrett. *Efficient Linux at the Command Line*. Sebastopol, CA, USA: O'Reilly Media, Inc., Feb. 2022. ISBN: 978-1-0981-1340-7 (siehe S. 289, 290).

[5] Daniel Bartholomew. *Learning the MariaDB Ecosystem: Enterprise-level Features for Scalability and Availability*. New York, NY, USA: Apress Media, LLC, Okt. 2019. ISBN: 978-1-4842-5514-8 (siehe S. 289).

[6] Tim Berners-Lee. *Re: Qualifiers on Hypertext links...* Geneva, Switzerland: World Wide Web project, European Organization for Nuclear Research (CERN) und Newsgroups: alt.hypertext, 6. Aug. 1991. URL: https://www.w3.org/People/Berners-Lee/1991/08/art-6484.txt (besucht am 2025-02-05) (siehe S. 290).

[7] Alex Berson. *Client/Server Architecture*. 2. Aufl. Computer Communications Series. New York, NY, USA: McGraw-Hill, 29. März 1996. ISBN: 978-0-07-005664-0 (siehe S. 288).

[8] Grady Booch, James Rumbaugh und Ivar Jacobson. *The Unified Modeling Language Reference Manual*. 1. Aufl. Reading, MA, USA: Addison-Wesley Professional, Jan. 1999. ISBN: 978-0-201-57168-4 (siehe S. 5–14, 290).

[9] Silvia Botros und Jeremy Tinley. *High Performance MySQL*. 4. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Nov. 2021. ISBN: 978-1-4920-8051-0 (siehe S. 289).

[10] Ed Bott. *Windows 11 Inside Out*. Hoboken, NJ, USA: Microsoft Press, Pearson Education, Inc., Feb. 2023. ISBN: 978-0-13-769132-6 (siehe S. 289).

[11] Ron Brash und Ganesh Naik. *Bash Cookbook*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2018. ISBN: 978-1-78862-936-2 (siehe S. 288).

References II

[12] Ben Brumm. "A Guide to the Entity Relationship Diagram (ERD)". In: *Database Star*. Armadale, VIC, Australia: Elevated Online Services PTY Ltd., 30. Juli 2019–23. Dez. 2023. URL: https://www.databasestar.com/entity-relationship-diagram (besucht am 2025-03-29) (siehe S. 288).

[13] Jason Cannon. *High Availability for the LAMP Stack*. Shelter Island, NY, USA: Manning Publications, Juni 2022 (siehe S. 289, 290).

[14] Donald D. Chamberlin. "50 Years of Queries". *Communications of the ACM (CACM)* 67(8):110–121, Aug. 2024. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/3649887. URL: https://cacm.acm.org/research/50-years-of-queries (besucht am 2025-01-09) (siehe S. 290).

[15] Peter Pin-Shan Chen. "Entity-Relationship Modeling: Historical Events, Future Trends, and Lessons Learned". In: *Software Pioneers: Contributions to Software Engineering*. Hrsg. von Manfred Broy und Ernst Denert. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, Feb. 2002, S. 296–310. doi:10.1007/978-3-642-59412-0_17. URL: http://bit.csc.lsu.edu/%7Echen/pdf/Chen_Pioneers.pdf (besucht am 2025-03-06) (siehe S. 288).

[16] Peter Pin-Shan Chen. "The Entity-Relationship Model – Toward a Unified View of Data". *ACM Transactions on Database Systems (TODS)* 1(1):9–36, März 1976. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0362-5915. doi:10.1145/320434.320440 (siehe S. 280, 288).

[17] Peter Pin-Shan Chen. "The Entity-Relationship Model: Toward a Unified View of Data". In: *1st International Conference on Very Large Data Bases (VLDB'1975)*. 22.–24. Sep. 1975, Framingham, MA, USA. Hrsg. von Douglas S. Kerr. New York, NY, USA: Association for Computing Machinery (ACM), 1975, S. 173. ISBN: 978-1-4503-3920-9. doi:10.1145/1282480.1282492. See[16] for a more comprehensive introduction. (Siehe S. 288).

[18] David Clinton und Christopher Negus. *Ubuntu Linux Bible*. 10. Aufl. Bible Series. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 10. Nov. 2020. ISBN: 978-1-119-72233-5 (siehe S. 290).

[19] Edgar Frank „Ted" Codd. "A Relational Model of Data for Large Shared Data Banks". *Communications of the ACM (CACM)* 13(6):377–387, Juni 1970. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/362384.362685. URL: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf (besucht am 2025-01-05) (siehe S. 289).

References III

[20] *Country Codes*. Techn. Ber. ISO 3166. ISO 3166 Maintenance Agenc, c/o International Organization for Standardization (ISO): ISO 3166 Maintenance Agenc, c/o International Organization for Standardization (ISO), 2020. URL: https://www.iso.org/iso-3166-country-codes.html (besucht am 2025-04-11) (siehe S. 246–263).

[21] *Database Language SQL*. Techn. Ber. ANSI X3.135-1986. Washington, D.C., USA: American National Standards Institute (ANSI), 1986 (siehe S. 290).

[22] "Date/Time Functions and Operators". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 9.9. URL: https://www.postgresql.org/docs/17/functions-datetime.html (besucht am 2025-05-01) (siehe S. 290).

[23] "Date/Time Types". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. 8.5. URL: https://www.postgresql.org/docs/17/datatype-datetime.html (besucht am 2025-02-28) (siehe S. 290).

[24] Matt David und Blake Barnhill. *How to Teach People SQL*. San Francisco, CA, USA: The Data School, Chart.io, Inc., 10. Dez. 2019–10. Apr. 2023. URL: https://dataschool.com/how-to-teach-people-sql (besucht am 2025-02-27) (siehe S. 290).

[25] *Database Language SQL*. International Standard ISO 9075-1987. Geneva, Switzerland: International Organization for Standardization (ISO), 1987 (siehe S. 290).

[26] Paul Deitel, Harvey Deitel und Abbey Deitel. *Internet & World Wide WebW[: How to Program*. 5. Aufl. Hoboken, NJ, USA: Pearson Education, Inc., Nov. 2011. ISBN: 978-0-13-299045-5 (siehe S. 290).

[27] Russell J.T. Dyer. *Learning MySQL and MariaDB*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2015. ISBN: 978-1-4493-6290-4 (siehe S. 289).

[28] Luca Ferrari und Enrico Pirozzi. *Learn PostgreSQL*. 2. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Okt. 2023. ISBN: 978-1-83763-564-1 (siehe S. 289).

[29] *PDF 32000-1:2008 – Document Management – Portable Document Format – Part 1: PDF 1.7*. 1. Aufl. San Jose, CA, USA: Adobe Systems Incorporated, 1. Juli 2008. URL: https://pdf-lib.js.org/assets/with_large_page_count.pdf (besucht am 2024-12-12) (siehe S. 289).

References IV

[30] Terry Halpin und Tony Morgan. *Information Modeling and Relational Databases*. 3. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juli 2024. ISBN: 978-0-443-23791-1 (siehe S. 289).

[31] Jan L. Harrington. *Relational Database Design and Implementation*. 4. Aufl. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Apr. 2016. ISBN: 978-0-12-849902-3 (siehe S. 289).

[32] Michael Hausenblas. *Learning Modern Linux*. Sebastopol, CA, USA: O'Reilly Media, Inc., Apr. 2022. ISBN: 978-1-0981-0894-6 (siehe S. 289).

[33] Matthew Helmke. *Ubuntu Linux Unleashed 2021 Edition*. 14. Aufl. Reading, MA, USA: Addison-Wesley Professional, Aug. 2020. ISBN: 978-0-13-668539-5 (siehe S. 289, 290).

[34] ."When should a database table use timestamps?" In: *Software Engineering*. Hrsg. von GWed. New York, NY, USA: Stack Exchange Inc., 29. Jan. 2014–16. Sep. 2016. URL: https://softwareengineering.stackexchange.com/questions/225903 (besucht am 2025-02-27) (siehe S. 290).

[35] John Hunt. *A Beginners Guide to Python 3 Programming*. 2. Aufl. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2023. ISBN: 978-3-031-35121-1. doi:10.1007/978-3-031-35122-8 (siehe S. 289).

[36] *Information Technology – Database Languages – SQL – Part 1: Framework (SQL/Framework), Part 1*. International Standard ISO/IEC 9075-1:2023(E), Sixth Edition, (ANSI X3.135). Geneva, Switzerland: International Organization for Standardization (ISO) und International Electrotechnical Commission (IEC), Juni 2023. URL: https://standards.iso.org/ittf/PubliclyAvailableStandards/ISO_IEC_9075-1_2023_ed_6_-_id_76583_Publication_PDF_(en).zip (besucht am 2025-01-08). Consists of several parts, see https://modern-sql.com/standard for information where to obtain them. (Siehe S. 290).

[37] *IP2Location™ ISO 3166-2 Subdivision Code*. Bayan Baru, Pulau Pinang, Malaysia: Hexasoft Development Sdn. Bhd., 16. Apr. 2025. URL: https://www.ip2location.com/free/iso3166-2 (besucht am 2025-04-29) (siehe S. 246–263).

[38] Shannon Kempe und Paul Williams. *A Short History of the ER Diagram and Information Modeling*. Studio City, CA, USA: Dataversity Digital LLC, 25. Sep. 2012. URL: https://www.dataversity.net/a-short-history-of-the-er-diagram-and-information-modeling (besucht am 2025-03-06) (siehe S. 288).

References V

[39] Petr Kozelek. *Audit Trail – Tracing Data Changes in Database*. Toronto, ON, Canada: CodeProject, 30. Aug. 2010. URL: https://www.codeproject.com/Articles/105768/Audit-Trail-Tracing-Data-Changes-in-Database (besucht am 2025-04-09) (siehe S. 246–263).

[40] Jay LaCroix. *Mastering Ubuntu Server*. 4. Aufl. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2022. ISBN: 978-1-80323-424-3 (siehe S. 290).

[41] Kent D. Lee und Steve Hubbard. *Data Structures and Algorithms with Python*. Undergraduate Topics in Computer Science (UTICS). Cham, Switzerland: Springer, 2015. ISBN: 978-3-319-13071-2. doi:10.1007/978-3-319-13072-9 (siehe S. 289).

[42] Gloria Lotha, Aakanksha Gaur, Erik Gregersen, Swati Chopra und William L. Hosch. "Client-Server Architecture". In: *Encyclopaedia Britannica*. Hrsg. von The Editors of Encyclopaedia Britannica. Chicago, IL, USA: Encyclopædia Britannica, Inc., 3. Jan. 2025. URL: https://www.britannica.com/technology/client-server-architecture (besucht am 2025-01-20) (siehe S. 288).

[43] Mark Lutz. *Learning Python*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2025. ISBN: 978-1-0981-7130-8 (siehe S. 289).

[44] *MariaDB Server Documentation*. Milpitas, CA, USA: MariaDB, 2025. URL: https://mariadb.com/kb/en/documentation (besucht am 2025-04-24) (siehe S. 289).

[45] Jim Melton und Alan R. Simon. *SQL: 1999 – Understanding Relational Language Components*. The Morgan Kaufmann Series in Data Management Systems. Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, Juni 2001. ISBN: 978-1-55860-456-8 (siehe S. 290).

[46] Adrien „anayrat" Nayrat. "PostgreSQL: Deferrable Constraints". In: *Select * from Adrien*. Valence, Drôme, Rhône-Alpes, France, 13. Aug. 2016. URL: https://blog.anayrat.info/en/2016/08/13/postgresql-deferrable-constraints (besucht am 2025-04-10) (siehe S. 81–112).

[47] Cameron Newham und Bill Rosenblatt. *Learning the Bash Shell – Unix Shell Programming: Covers Bash 3.0*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., 2005. ISBN: 978-0-596-00965-6 (siehe S. 288).

[48] Regina O. Obe und Leo S. Hsu. *PostgreSQL: Up and Running*. 3. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Okt. 2017. ISBN: 978-1-4919-6336-4 (siehe S. 289).

References VI

[49] *OMG® Unified Modeling Language® (OMG UML®). Version 2.5.1*. OMG Document formal/2017-12-05. Milford, MA, USA: Object Management Group, Inc. (OMG), Dez. 2017. URL: https://www.omg.org/spec/UML/2.5.1/PDF (besucht am 2025-03-30) (siehe S. 5–14, 290).

[50] Robert Orfali, Dan Harkey und Jeri Edwards. *Client/Server Survival Guide*. 3. Aufl. Chichester, West Sussex, England, UK: John Wiley and Sons Ltd., 25. Jan. 1999. ISBN: 978-0-471-31615-2 (siehe S. 288).

[51] Charles C. Palmer. "ER Modeling". In: *COSC 61 Winter 2025: Database Systems*. Hanover, MD, USA: Dartmouth College, 15. Jan. 2025. Kap. 4. URL: http://www.cs.dartmouth.edu/~cs61/Lectures/04%20-%20ER%20Modeling/04%20-%20ER%20Modeling.pdf (besucht am 2025-04-08) (siehe S. 43–49).

[52] Hasso Plattner. "Insert-Only". In: *A Course in In-Memory Data Management: The Inner Mechanics of In-Memory Databases*. 2. Aufl. Berlin/Heidelberg, Germany: Springer-Verlag GmbH Germany, Juni 2014. Kap. Advanced Database Storage Techniques, 3, S. 173–180. ISBN: 978-3-642-55269-4. doi:10.1007/978-3-642-55270-0_26 (siehe S. 221–227).

[53] *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), Feb. 2025. URL: https://www.postgresql.org/docs/17/index.html (besucht am 2025-02-25).

[54] *PostgreSQL Essentials: Leveling Up Your Data Work*. Sebastopol, CA, USA: O'Reilly Media, Inc., März 2024 (siehe S. 289).

[55] Abhishek Ratan, Eric Chou, Pradeeban Kathiravelu und Dr. M.O. Faruque Sarker. *Python Network Programming*. Birmingham, England, UK: Packt Publishing Ltd, Jan. 2019. ISBN: 978-1-78883-546-6 (siehe S. 288).

[56] Federico Razzoli. *Mastering MariaDB*. Birmingham, England, UK: Packt Publishing Ltd, Sep. 2014. ISBN: 978-1-78398-154-0 (siehe S. 289).

[57] Mike Reichardt, Michael Gundall und Hans D. Schotten. "Benchmarking the Operation Times of NoSQL and MySQL Databases for Python Clients". In: *47th Annual Conference of the IEEE Industrial Electronics Society (IECON'2021*. 13.–15. Okt. 2021, Toronto, ON, Canada. Piscataway, NJ, USA: Institute of Electrical and Electronics Engineers (IEEE), 2021, S. 1–8. ISSN: 2577-1647. ISBN: 978-1-6654-3554-3. doi:10.1109/IECON48115.2021.9589382 (siehe S. 289).

References VII

[58] Mark Richards und Neal Ford. *Fundamentals of Software Architecture: An Engineering Approach*. Sebastopol, CA, USA: O'Reilly Media, Inc., Jan. 2020. ISBN: 978-1-4920-4345-4 (siehe S. 288).

[59] Stuart J. Russell und Peter Norvig. *Artificial Intelligence: A Modern Approach (AIMA)*. 4. Aufl. Hoboken, NJ, USA: Pearson Education, Inc. ISBN: 978-1-292-40113-3. URL: https://aima.cs.berkeley.edu (besucht am 2024-06-27) (siehe S. 288).

[60] "SET CONSTRAINTS". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. Part VI. Reference. URL: https://www.postgresql.org/docs/17/sql-set-constraints.html (besucht am 2025-04-10) (siehe S. 81–112).

[61] Yuriy Shamshin. "Conceptual Database Model. Entity Relationship Diagram (ERD)". In: *Databases*. Riga, Latvia: ISMA University of Applied Sciences, Mai 2024. Kap. 04. URL: https://dbs.academy.lv/lection/dbs_LS04EN_erd.pdf (besucht am 2025-03-29) (siehe S. 288).

[62] Ellen Siever, Stephen Figgins, Robert Love und Arnold Robbins. *Linux in a Nutshell*. 6. Aufl. Sebastopol, CA, USA: O'Reilly Media, Inc., Sep. 2009. ISBN: 978-0-596-15448-6 (siehe S. 289).

[63] John Miles Smith und Philip Yen-Tang Chang. "Optimizing the Performance of a Relational Algebra Database Interface". *Communications of the ACM (CACM)* 18(10):568–579, Okt. 1975. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/361020.361025 (siehe S. 289).

[64] "SQL Commands". In: *PostgreSQL Documentation*. 17.4. The PostgreSQL Global Development Group (PGDG), 20. Feb. 2025. Kap. Part VI. Reference. URL: https://www.postgresql.org/docs/17/sql-commands.html (besucht am 2025-02-25) (siehe S. 290).

[65] Ryan K. Stephens und Ronald R. Plew. *Sams Teach Yourself SQL in 21 Days*. 4. Aufl. Sams Tech Yourself. Indianapolis, IN, USA: SAMS Technical Publishing und Hoboken, NJ, USA: Pearson Education, Inc., Okt. 2002. ISBN: 978-0-672-32451-2 (siehe S. 285, 290).

[66] Ryan K. Stephens, Ronald R. Plew, Bryan Morgan und Jeff Perkins. *SQL in 21 Tagen. Die Datenbank-Abfragesprache SQL vollständig erklärt (in 14/21 Tagen)*. 6. Aufl. Burgthann, Bayern, Germany: Markt+Technik Verlag GmbH, Feb. 1998. ISBN: 978-3-8272-2020-2. Translation of[65] (siehe S. 290).

References VIII

[67] Allen Taylor. *Introducing SQL and Relational Databases*. New York, NY, USA: Apress Media, LLC, Sep. 2018. ISBN: 978-1-4842-3841-7 (siehe S. 289, 290).

[68] Alkin Tezuysal und Ibrar Ahmed. *Database Design and Modeling with PostgreSQL and MySQL*. Birmingham, England, UK: Packt Publishing Ltd, Juli 2024. ISBN: 978-1-80323-347-5 (siehe S. 289).

[69] Kristian Torp, Christian S. Jensen und Richard T. Snodgrass. "Effective Timestamping in Databases". 8(3-4):267–288, Feb. 2000. doi:10.1007/S007780050008. URL: https://people.cs.aau.dk/~csj/Thesis/pdf/chapter40.pdf (besucht am 2025-05-01) (siehe S. 290).

[70] Linus Torvalds. "The Linux Edge". *Communications of the ACM (CACM)* 42(4):38–39, Apr. 1999. New York, NY, USA: Association for Computing Machinery (ACM). ISSN: 0001-0782. doi:10.1145/299157.299165 (siehe S. 289).

[71] *UML Distilled: A Brief Guide to the Standard Object Modeling Language*. 3. Aufl. The Addison-Wesley Object Technology Series. Reading, MA, USA: Addison-Wesley Professional, Sep. 2003. ISBN: 978-0-321-19368-1 (siehe S. 5–14, 290).

[72] *UML Notation Guide*. Version 1.1. Santa Clara, CA, USA: Rational Software Corporation, Redmond, WA, USA: Microsoft Corporation, Palo Alto, CA, USA: Hewlett-Packard Company, Redwood Shores, CA, USA: Oracle Corporation, Dallas, TX, USA: Sterling Software, Ottawa, ON, Canada: MCI Systemhouse Corporation, Blue Bell, PA, USA: Unisys Corporation, Blue Bell, PA, USA: ICON Computing, Santa Clara, CA, USA: IntelliCorp, Burlington, MA, USA: i-Logix, Armonk, NY, USA: International Business Machines Corporation (IBM), Kanata, ON, Canada: ObjecTime Limited, Chicago, IL, USA: Platinum Technology Inc., Boston, MA, USA: Ptech Inc., Orlando, FL, USA: Taskon A/S, Paoli, PA, USA: Reich Technologies und Paris, Île-de-France, France: Softeam, 1. Sep. 1997. URL: https://web.cse.msu.edu/~cse870/Materials/uml-notation-guide-9-97.pdf (besucht am 2025-03-30) (siehe S. 5–14, 290).

[73] Sander van Vugt. *Linux Fundamentals*. 2. Aufl. Hoboken, NJ, USA: Pearson IT Certification, Juni 2022. ISBN: 978-0-13-792931-3 (siehe S. 289).

[74] Thomas Weise (汤卫思). *Databases*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2025. URL: https://thomasweise.github.io/databases (besucht am 2025-01-05) (siehe S. 288, 289).

References IX

[75] Thomas Weise (汤卫思). *Programming with Python*. Hefei, Anhui, China (中国安徽省合肥市): Hefei University (合肥大学), School of Artificial Intelligence and Big Data (人工智能与大数据学院), Institute of Applied Optimization (应用优化研究所, IAO), 2024–2025. URL: https://thomasweise.github.io/programmingWithPython (besucht am 2025-01-05) (siehe S. 289).

[76] Matthew West. *Developing High Quality Data Models*. Version: 2.0, Issue: 2.1. London, England, UK: Shell International Limited und European Process Industries STEP Technical Liaison Executive (EPISTLE); Burlington, MA, USA/San Mateo, CA, USA: Morgan Kaufmann Publishers, 8. Dez. 1995–Dez. 2010. ISBN: 978-0-12-375107-2. URL: https://www.researchgate.net/publication/286610894 (besucht am 2025-03-24). Edited by Julian Fowler (siehe S. 288).

[77] *What does PDF mean?* San Jose, CA, USA: Adobe Systems Incorporated, 2024. URL: https://www.adobe.com/acrobat/about-adobe-pdf.html (besucht am 2024-12-12) (siehe S. 289).

[78] *What is a Relational Database?* Armonk, NY, USA: International Business Machines Corporation (IBM), 20. Okt. 2021–12. Dez. 2024. URL: https://www.ibm.com/think/topics/relational-databases (besucht am 2025-01-05) (siehe S. 289).

[79] Ulf Michael „Monty" Widenius, David Axmark und Uppsala, Sweden: MySQL AB. *MySQL Reference Manual – Documentation from the Source*. Sebastopol, CA, USA: O'Reilly Media, Inc., 9. Juli 2002. ISBN: 978-0-596-00265-7 (siehe S. 289).

[80] Kinza Yasar und Craig S. Mullins. *Definition: Database Management System (DBMS)*. Newton, MA, USA: TechTarget, Inc., Juni 2024. URL: https://www.techtarget.com/searchdatamanagement/definition/database-management-system (besucht am 2025-01-11) (siehe S. 288).

[81] Giorgio Zarrelli. *Mastering Bash*. Birmingham, England, UK: Packt Publishing Ltd, Juni 2017. ISBN: 978-1-78439-687-9 (siehe S. 288).

Glossary (in English) I

AI Artificial Intelligence, see, e.g.,[59]

Bash is a the shell used under Ubuntu Linux, i.e., the program that „runs“ in the terminal and interprets your commands, allowing you to start and interact with other programs[11,47,81]. Learn more at https://www.gnu.org/software/bash.

client In a client-server architecture, the client is a device or process that requests a service from the server. It initiates the communication with the server, sends a request, and receives the response with the result of the request. Typical examples for clients are web browsers in the internet as well as clients for database management systems (DBMSes), such as psql.

client-server architecture is a system design where a central server receives requests from one or multiple clients[7,42,50,55,58]. These requests and responses are usually sent over network connections. A typical example for such a system is the World Wide Web (WWW), where web servers host websites and make them available to web browsers, the clients. Another typical example is the structure of DB software, where a central server, the DBMS, offers access to the DB to the different clients. Here, the client can be some terminal software shipping with the DBMS, such as psql, or the different applications that access the DBs.

DB A *database* is an organized collection of structured information or data, typically stored electronically in a computer system. Databases are discussed in our book *Databases*[74].

DBMS A *database management system* is the software layer located between the user or application and the DB. The DBMS allows the user/application to create, read, write, update, delete, and otherwise manipulate the data in the DB[80].

DBS A *database system* is the combination of a DB and a the corresponding DBMS, i.e., basically, an installation of a DBMS on a computer together with one or multiple DBs. DBS = DB + DBMS.

ERD Entity relationship diagrams show the relationships between objects, e.g., between the tables in a DB and how they reference each other[3,12,15–17,38,61,76]

IT information technology

Glossary (in English) II

LAMP Stack A system setup for web applications: Linux, Apache (a web server), MySQL, and the server-side scripting language PHP[13,33].

Linux is the leading open source operating system, i.e., a free alternative for Microsoft Windows[4,32,62,70,73]. We recommend using it for this course, for software development, and for research. Learn more at https://www.linux.org. Its variant Ubuntu is particularly easy to use and install.

MariaDB An open source relational database management system that has forked off from MySQL[1,2,5,27,44,56]. See https://mariadb.org for more information.

Microsoft Windows is a commercial proprietary operating system[10]. It is widely spread, but we recommend using a Linux variant such as Ubuntu for software development and for our course. Learn more at https://www.microsoft.com/windows.

MySQL An open source relational database management system[9,27,57,68,79]. MySQL is famous for its use in the LAMP Stack. See https://www.mysql.com for more information.

PDF The *Portable Document Format*[29,77] is the format in which provide this book. It is the standard format for the exchange of documents in the internet.

PostgreSQL An open source object-relational DBMS[28,48,54,68]. See https://postgresql.org for more information.

psql is the client program used to access the PostgreSQL DBMS server.

Python The Python programming language[35,41,43,75], i.e., what you will learn about in our book[75]. Learn more at https://python.org.

relational database A relational DB is a database that organizes data into rows (tuples, records) and columns (attributes), which collectively form tables (relations) where the data points are related to each other[19,30,31,63,67,74,78].

Glossary (in English) III

server In a client-server architecture, the server is a process that fulfills the requests of the clients. It usually waits for incoming communication carring the requests from the clients. For each request, it takes the necessary actions, performs the required computations, and then sends a response with the result of the request. Typical examples for servers are web servers[13] in the internet as well as DBMSes. It is also common to refer to the computer running the server processes as server as well, i.e., to call it the „server computer“[40].

SQL The *Structured Query Language* is basically a programming language for querying and manipulating relational databases[14,21,24,25,36,45,64–67]. It is understood by many DBMSes. You find the Structured Query Language (SQL) commands supported by PostgreSQL in the reference[64].

terminal A terminal is a text-based window where you can enter commands and execute them[4,18]. Knowing what a terminal is and how to use it is very essential in any programming- or system administration-related task. If you want to open a terminal under Microsoft Windows, you can Druck auf ⊞+R, dann Schreiben von `cmd`, dann Druck auf ↵. Under Ubuntu Linux, Ctrl+Alt+T opens a terminal, which then runs a Bash shell inside.

timestamping is a technique used in DBs to keep track of the creation and/or modification time of data[34,69]. For each table in a relational database to which it is applied, an additional column with datatype `TIMESTAMP`[23] is added for the creation timestamp. This column is usually annotated as `NOT NULL DEFAULT CURRENT_TIMESTAMP`[22], meaning that the DBMS will automatically store the current date and time when a row is created. If data can be changed, then another column for the last update time can be added to the table as well.

Ubuntu is a variant of the open source operating system Linux[18,33]. We recommend that you use this operating system to follow this class, for software development, and for research. Learn more at https://ubuntu.com. If you are in China, you can download it from https://mirrors.ustc.edu.cn/ubuntu-releases.

UML The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of distributed object systems[8,49,71,72]

WWW World Wide Web[6,26]