Kapitel 10: Datenmodellierung mit ERM und UML

Ziel der Datenmodellierung (des konzeptionellen Datenbankentwurfs):

Modellierung von Sachverhalten der realen Welt mittels weniger Grundkonzepte
mit dem Zweck, eine computergestützte Datenbank aufzubauen.
Dazu sind notwendig:

- eine *Modellierungssprache* (ein "semantisches" Datenmodell)
- eine *Entwurfsmethodologie*
- computergestützte *Entwurfswerkzeuge* (für große Entwürfe)

Rolle der Datenmodellierung bei der Entwicklung von Informationssystemen:

Informationssysteme werden in mehreren Stufen entwickelt:

1) Analyse der Anwendungswelt (Informationsbedarf, zu automatisierende Abläufe)
2) Datenmodellierung (sowie Funktions- und Prozeßmodellierung)
3) Abbildung des Datenmodells auf ein (relationales) Datenbankschema

Notwendigkeit einer systematischen Datenmodellierung (Entwurfstheorie):

Relationale Datenbankschemata können "gut" oder "schlecht" entworfen sein.
Es sollte Konstruktionsrichtlinien für (komplexere) Datenbankschemata geben.

Beispiel eines schlechten bzw. debattierbaren Entwurfs:

Bestellungen (Datum, KNr, PNr, Bez, Preis, Gewicht, Menge, Summe, Status)
Produkte (PNr, Bez, Lagerort, Vorrat)
Kunden (KNr, Name, Adresse, Rabatt, Saldo)

→ Probleme: Datenredundanz, potentielle Inkonsistenz,
Nichtdarstellbarkeit bestimmter Informationen

Beispiel:
Wie wäre das Schema der Bestellungs-DB aufzubauen, wenn mit einer Bestellung mehrere Produkte auf einmal bestellt werden können, wenn diese u.U. über mehrere Lieferungen verteilt geliefert werden, entsprechend mehrere Teilzahlungen eingehen, etc. ?

10.1 Datenmodellierung mit dem Entity-Relationship-Modell (ERM)

Entities und Entity-Mengen (Entitäten, Objekte, Gegenstände)

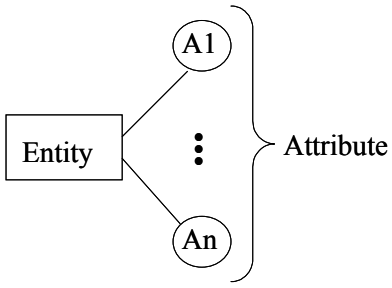
Ein Entity ist ein eindeutig identifizierbarer Gegenstand der realen Welt oder unserer Vorstellung (eine Person, ein Kunde, ein Buch, ein Bankkonto, etc.).
Gleichartige Entities werden zu einem *Entity-Typ* zusammengefaßt (z.B. alle Bücher, alle Bankkonten). Alle Entities desselben Entity-Typs werden durch eine einheitliche Menge von *Attributen* (Eigenschaften, Merkmalen) beschrieben, die jeweils einem Entity einen Wert eines bestimmten Wertebereichs (Domains) zuordnen.
Im Gegensatz zum Relationenmodell sind die Wertebereiche von Attributen nicht auf primitive Datentypen beschränkt, sondern können beispielsweise mengenwertig sein (z.B. Informatikkenntnisse als Attribut von Studenten oder Angestellten).

Beispiele:
- ISBN, Titel, Autoren, Verlag, Erscheinungsjahr als Attribute von Büchern
- Kontonummer, Kontoinhaber, Kontostand, Zinssatz als Attribute von Bankkonten
- Matrikelnummer, Name, Adresse, Semester, Prüfungen als Attribute von Studenten

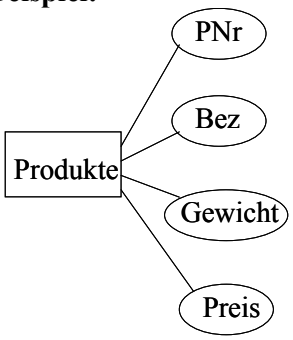
Eine Menge von Entities desselben Typs heißt Entity-Menge (Objektmenge, Objektklasse). Es kann mehrere, voneinander verschiedene Entity-Mengen desselben Entity-Typs geben (z.B. Studenten der Informatik und Studenten der Elektrotechnik, Bücher der Informatikbibliothek und Bücher der Universitätsbibliothek). Entity-Mengen müssen nicht disjunkt sein (z.B. Ärzte und Patienten, beide vom Entity-Typ Personen).

Bemerkungen:
- Bei vielen Anwendungen gibt es pro Entity-Typ genau eine relevante Entity-Menge. In diesen Fällen wird in der Regel nicht explizit zwischen Entity-Typ und Entity-Menge unterschieden.
- Der (Analyse-) Schritt von der Betrachtung einzelner Entities zur Betrachtung von Entity-Mengen wird auch als Abstraktion oder Klassifikation bezeichnet. Bei der Informationsmodellierung ist nur diese (Schema-) Ebene von Bedeutung.

Graphische Darstellung:



Beispiel:



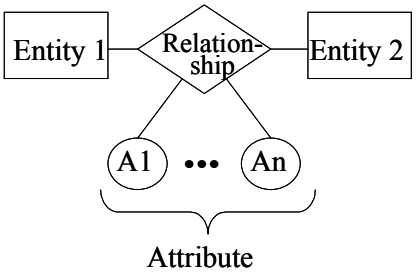
Eine Attributmenge K einer Entity-Menge E heißt *Schlüsselkandidat*, wenn zu jedem Zeitpunkt für je zwei verschiedene Entities e1, e2 ∈ E gelten muß, daß sich e1 und e2 bezüglich K unterscheiden, und es keine echte Teilmenge von K gibt, die diese Eigenschaft hat (z.B. Kontonummer bei Bankkonten einer bestimmten Bank, Kontonummer und Name der Bank bei allen Bankkonten, Personalnummer bei Angestellten).
Der *Primärschlüssel* einer Entity-Menge ist ein explizit ausgewählter Schlüsselkandidat.

Relationships und Relationship-Mengen (Beziehungen, Assoziationen)

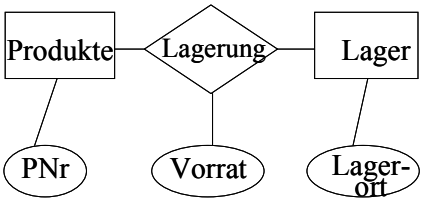
Eine *Relationship-Menge* R zwischen Entity-Mengen E1, ..., En ist eine Teilmenge des kartesischen Produkts E1 × ... × En (meist ist n=2). Ein Element <e1, ..., en> einer Relationship-Menge bildet eine Relationship zwischen den Entities e1, ..., en.
Alle möglichen Relationship-Mengen zwischen denselben Entity-Typen bilden zusammen einen Relationship-Typ.
Die Elemente einer Relationship-Menge können durch Attribute zusätzlich charakterisiert werden (z.B. die Vorratsmenge, die von einem Produkt in einem Lager vorhanden ist, oder die Note, die ein Student in einem Prüfungsfach erreicht hat). Die Primärschlüssel der an einer Relationship-Menge beteiligten Entity-Mengen gelten als implizite Attribute der Relationship-Menge.

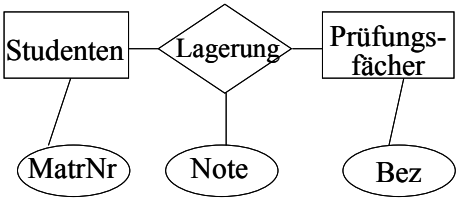
Bemerkungen:
- Bei vielen Anwendungen gibt es pro Relationship-Typ genau eine relevante Relationship-Menge. In diesen Fällen wird in der Regel nicht explizit zwischen Relationship-Typ und Relationship-Menge unterschieden.
- Die Modellierung von Relationships zwischen Entities wird auch als (spezielle Form der) Aggregation bezeichnet.

Graphische Darstellung (ERM-Diagramm):



Beispiele:



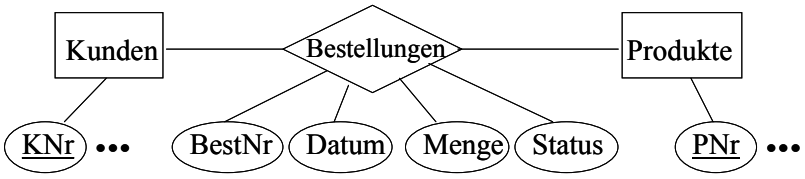


Eine Teilmenge K der Vereinigung $\bigcup_{i=1}^{n} Ki$ der Primärschlüssel K1, ..., Kn von E1, ..., En heißt *Schlüsselkandidat* der Relationship-Menge R, wenn sich zu jedem Zeitpunkt je zwei verschiedene Elemente r=<e1, ..., en> und r'=<e1', ..., en'> von R bezüglich K unterscheiden müssen und es keine echte Teilmenge von K gibt, die diese Eigenschaft hat.
Der *Primärschlüssel* einer Relationship-Menge ist ein explizit ausgewählter Schlüsselkandidat.

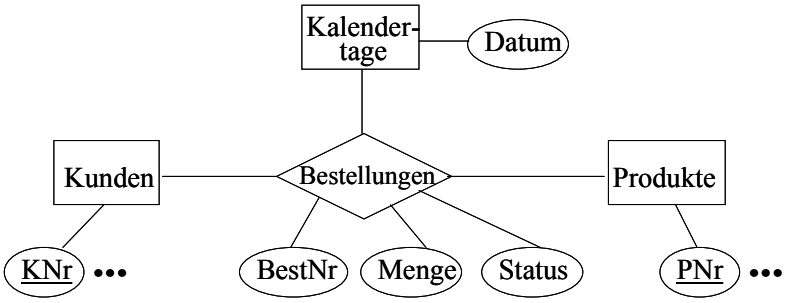
Beispiel:
PNr ist ein Schlüsselkandidat der Relationship-Menge Lagerung, falls ein Produkt immer nur an einem Ort gelagert wird. Andernfalls ist {PNr, Lagerort} ein Schlüsselkandidat von Lagerung.

Modellierung eines Sachverhalts als Entity oder als Relationship?

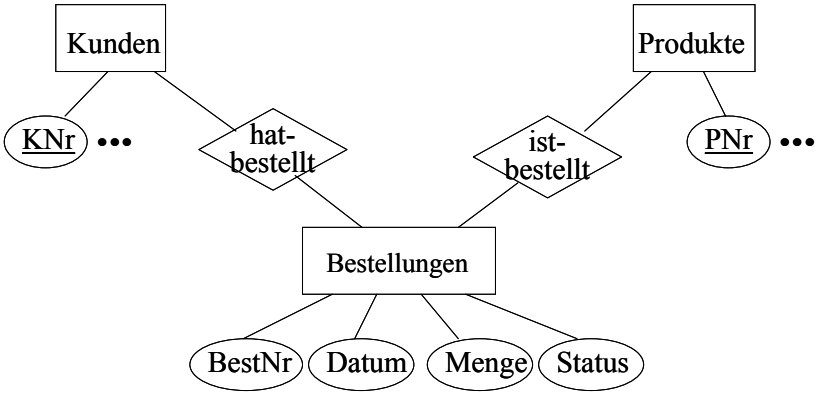
Variante 1:



Variante 2:



Variante 3:



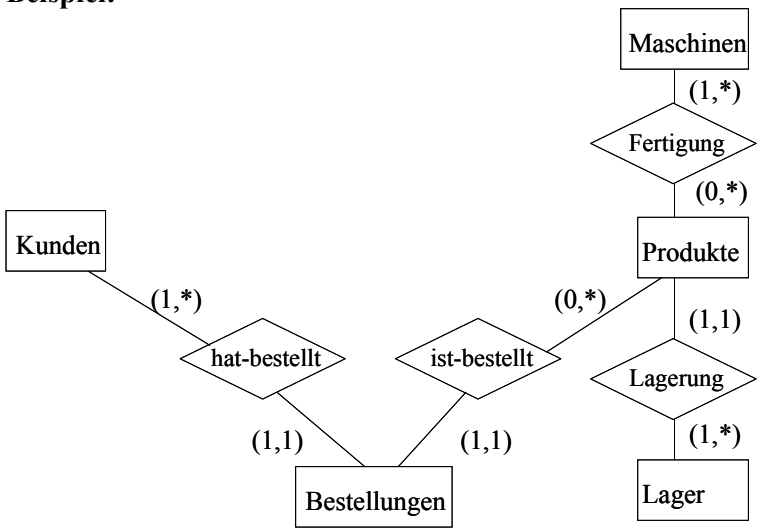
Variante 1 ist inadäquat bzw. inkorrekt, da es mehrere Bestellungen mit demselben Wert für <KNr, PNr> geben kann.
Variante 2 gibt dem Kalender eine unangemessen prominente Position.
Variante 3 ist hier wohl die angemessenste Modellierung.

Kardinalitätsbedingungen für (Entities in) Relationship-Mengen

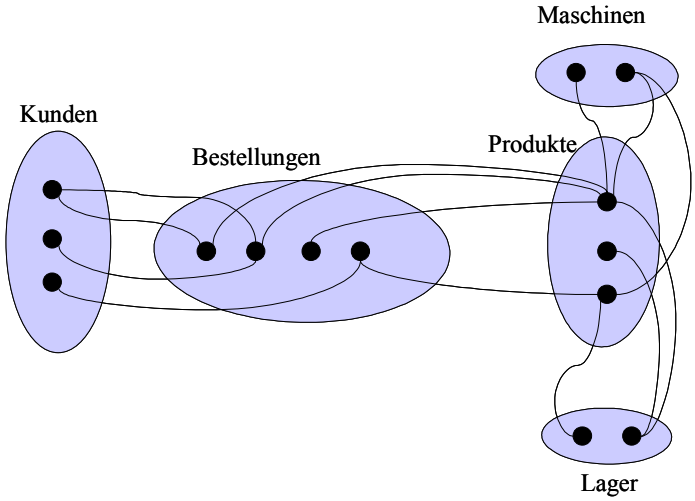
Eine Kardinalitätsbedingung für die Rolle (Bindung, Assoziation) einer Entity-Menge E in einer Relationship-Menge R spezifiziert, wie häufig ein Entity von E in R mindestens vorkommen muß und/oder wie häufig es höchstens vorkommen darf.

Notation:
Die Kante zwischen E und R in einem ERM-Diagramm wird mit (x,y) annotiert, wobei x, y natürliche Zahlen sind und für y auch das Symbol "*" (für "don't care") stehen darf.
x bezeichnet die Untergrenze für die Anzahl der Beziehungen eines Entities, y die Obergrenze. Falls für y das Symbol "*" angegeben ist, ist keine Obergrenze spezifiziert; in diesem Fall wird statt "*" gelegentlich auch "N" geschrieben.

Beispiel:

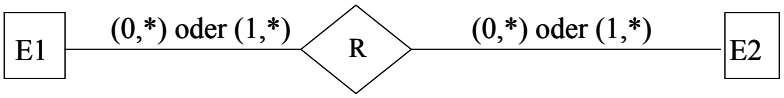


Beispielausprägung:

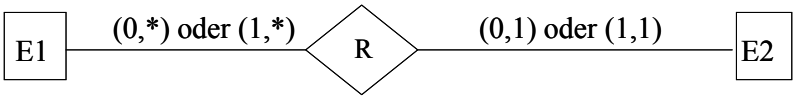


Verschiedene Arten von Relationships je nach Art der Kardinalitätsbedingung

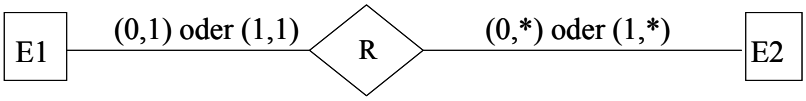
M:N-Relationship (Viele-zu-viele-Beziehung, komplexe Beziehung)



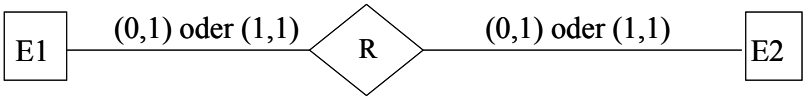
1:N-Relationship (Eins-zu-viele-Beziehung, hierarchische Beziehung)



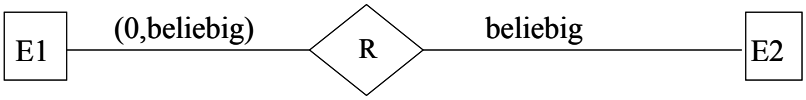
N:1-Relationship (Viele-zu-eins-Beziehung, hierarchische Beziehung)



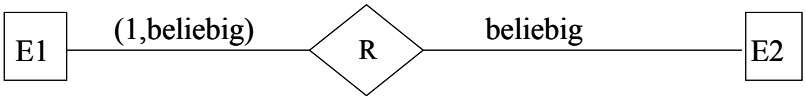
1:1-Relationship (Eins-zu-eins-Beziehung, injektive Beziehung)



Optionale Rolle eines Entities E1 in einer Relationship R



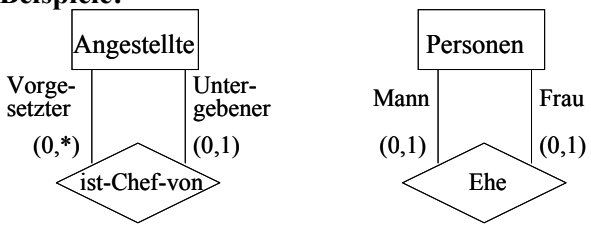
Verpflichtende Rolle eines Entities E1 in einer Relationship R



Rekursive Relationship-Mengen

Eine Relationship-Menge zwischen einer Entity-Menge E und sich selbst heißt *rekursiv*. Um Kardinalitätsbedingungen richtig ausdrücken zu können, werden bei rekursiven Relationship-Mengen die verschiedenen "Rollen" von E benannt. Im ERM-Diagramm werden die entsprechenden Kanten mit den Rollennamen annotiert.

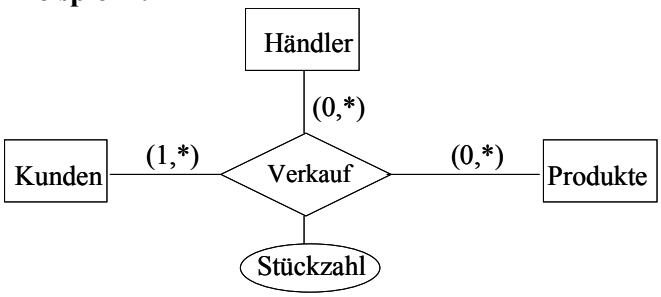
Beispiele:



Relationship-Mengen zwischen mehr als zwei Entity-Mengen

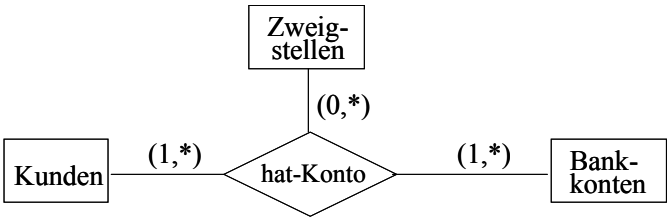
Relationship-Mengen sind nicht notwendigerweise binär. In vielen Anwendungen kommen ternäre Relationship-Mengen vor. Einige davon lassen sich in mehrere binäre Relationship-Mengen zerlegen, bei anderen ist eine solche Zerlegung nicht möglich.

Beispiel 1:

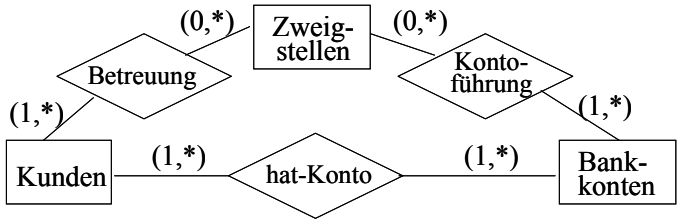


Verkauf ist nicht in mehrere binäre Relationship-Mengen zerlegbar.

Beispiel 2:



Die Relationship-Menge hat-Konto ist zerlegbar in drei binäre Relationship-Mengen.
Achtung: Die Zerlegung verändert aber die Semantik der Relationships in subtiler Weise. Die Betreuung wäre jetzt unabhängig davon, ob der Kunde ein Konto bei einer Zweigstelle hat.

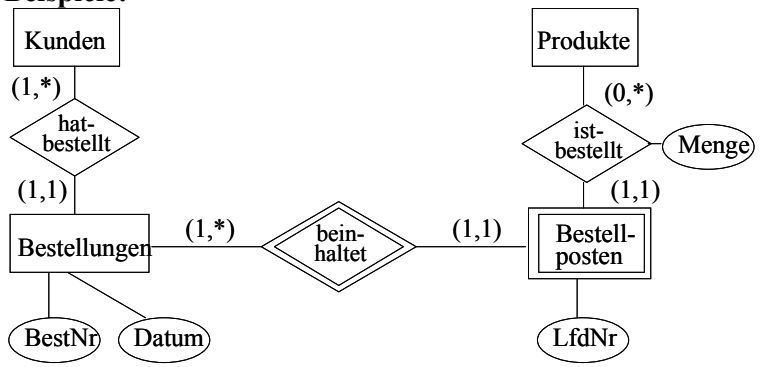


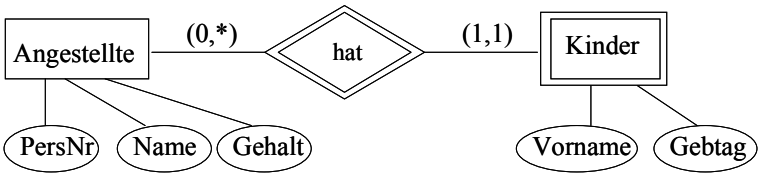
Schwache Entity-Mengen und Schwache Relationships

Eine schwache Entity-Menge S ist eine Entity-Menge, die in einer hierarchischen N:1-Relationship R zu einer anderen (nichtschwachen) Entity-Menge E steht, so daß der Primärschlüssel von S den Primärschlüssel von E enthält. R heißt dann auch schwache Relationship, wobei S eine verpflichtende Rolle in R hat.
Ein Entity einer schwachen Entity-Menge ist nur in Verbindung mit dem entsprechenden nichtschwachen Entity für die modellierte Anwendungswelt interessant.

Darstellung:
Die schwache Entity-Menge S und die schwache Relationship R werden in ERM-Diagrammen speziell markiert. Der Primärschlüssel von E wird bei S nicht explizit aufgeführt.

Beispiele:



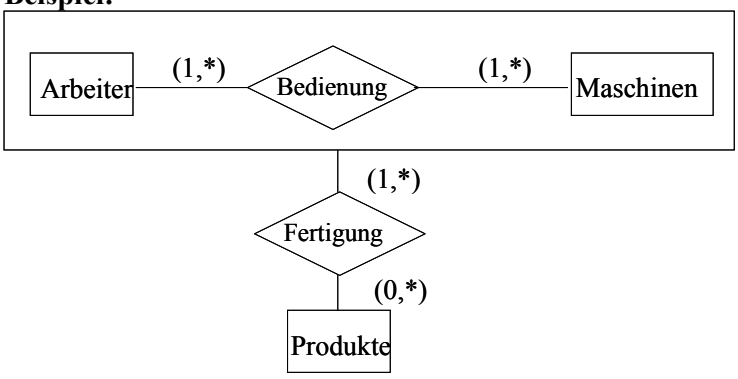


Aggregation

Mehrere Entities mit ihren Relationships können selbst wiederum zu einem *zusammengesetzten Entity* (komplexen Objekt) zusammengefaßt werden. Die entsprechende Entity-Menge heißt aggregierte Entity-Menge.

Bemerkung: Aggregation ist in vielen „semantischen“ Datenmodellen nur in eingeschränkter Form vorgesehen, wird aber z.B. von der ERM-Erweiterung SERM (Structured ERM) unterstützt.

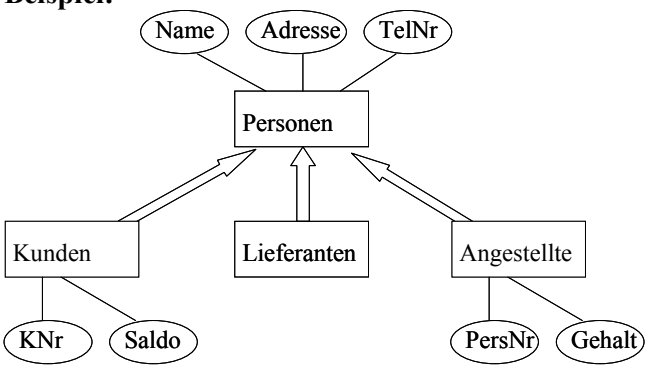
Beispiel:



ISA-Relationships (Generalisierungsbeziehungen)

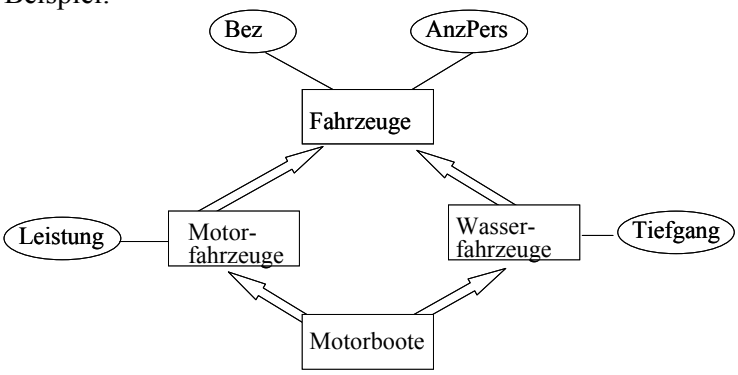
Eine binäre 1:1-Relationship-Menge R zwischen zwei Entity-Mengen E und F heißt *ISA-Relationship-Menge,* wenn die Attributmenge von F eine Obermenge der Attributmenge von E ist und die Teilmengenbeziehung $F \subseteq E$ gilt.
Man sagt dann "F is a E" und bezeichnet F als *Spezialisierung* (Subklasse) von E und E als *Generalisierung* (Superklasse) von F. Ferner bezeichnet man den Entity-Typ von F als einen Subtyp des Entity-Typs von E und den Entity-Typ von E als einen Supertyp des Entity-Typs von F.

Beispiel:



Mit der Generalisierung lassen sich ganze Hierarchien aufbauen, die häufig die morphologische Gliederung eines Anwendungsfelds modellieren.
Beispiele:
1) Wirbeltiere - Säugetiere - Katzen - Tiger - Sibirische Tiger
2) Bankkonten - Sparkonten - Jugendsparkonten
3) Finanzinstrumente - Derivative Instrumente - Optionen - Aktienindexoptionen - DAX-Optionen

Generalisierungshierarchien müssen nicht notwendigerweise Bäume sein.
Beispiel:



Verschiedene Arten von Generalisierungsbeziehungen

Seien F1, ..., Fn Spezialisierungen der Entity-Menge E.
Die Generalisierungsbeziehungen zwischen F1, ..., Fn und E heißen *disjunkt*, wenn für je zwei verschiedene Spezialisierungen Fi, Fj gilt: $Fi \cap Fj = \varnothing$.

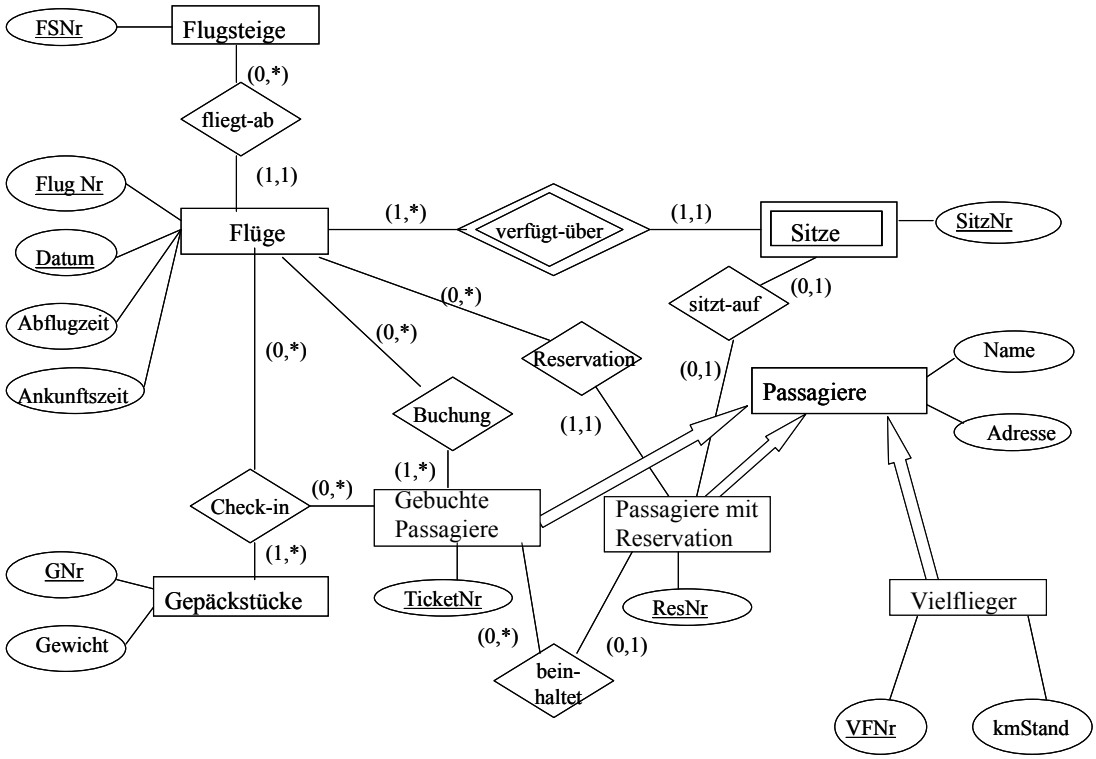
Sie heißen *vollständig-überdeckend*, wenn gilt: $\bigcup_{i=1}^{n} Fi = E$.

10.2 Entwurfsmethodologie (grob)

1) Abstecken des Problemrahmens (Analyse des Informationsbedarfs)
2) Spezifikation von Entity-Mengen mit ihren Attributen
3) Festlegung von Primärschlüssel für die Entity-Mengen
4) Bildung von Generalisierungsbeziehungen
5) Spezifikation der relevanten Relationship-Mengen
6) Spezifikation von Kardinalitätsbedingungen und ggf. weiteren Integritätsbedingungen

Beispiel:

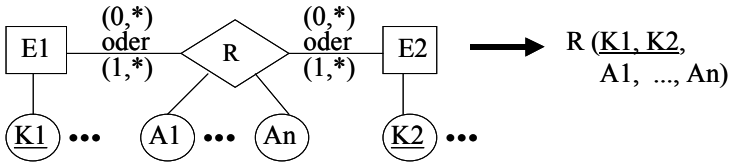
Es sollen die Informationszusammenhänge für ein Flugbuchungssystem einer Fluggesellschaft modelliert werden.
Flüge werden durch eine Flugnummer identifiziert, die für Flüge am selben Tag eindeutig ist.
Passagiere können einen Flug reservieren, was durch eine Reservationsnummer bestätigt wird. Eine Reservation wird zu einer festen Buchung, indem man ein Ticket käuft.
Bei der Reservation oder später können Passagiere auch eine Sitzplatzreservierung vornehmen.
Für Teilnehmer des Vielfliegerprogramms ist die gesamte mit der Fluggesellschaft geflogene Kilometerzahl von Bedeutung.
Flüge fliegen von einem bestimmten Flugsteig ab. Passagiere müssen vor dem Abflug eine Check-in-Prozedur durchlaufen. Dabei können sie auch Gepäckstücke aufgeben.



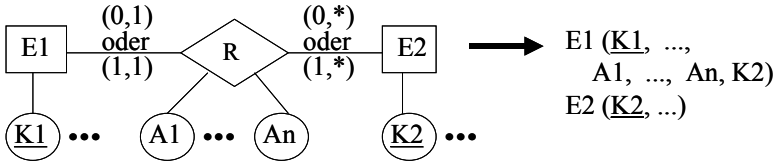
10.3 Abbildung eines ERM-Entwurfs auf Relationen

Regel 1: Jede Entity-Menge bildet eine Relation

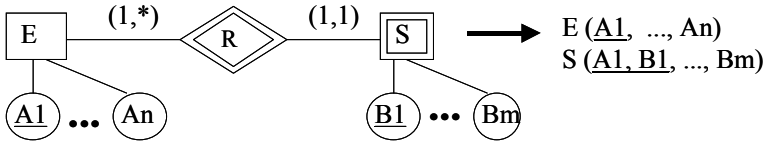
Regel 2: M:N-Relationships bilden eigene Relationen



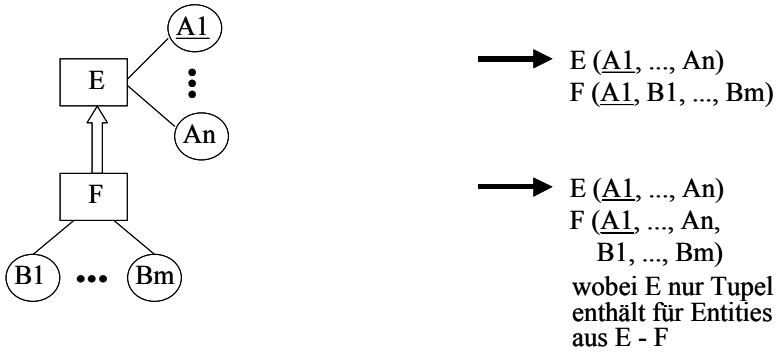
Regel 3: N:1-, 1:N und 1:1-Relationships können in eine "Entity-Relation" integriert werden



Regel 4: Eine schwache Entity-Menge bildet eine eigene Relation



Regel 5: Generalisierung und Spezialisierung bilden jeweils eine eigene Relation



Flugbuchungs-Beispiel:

Flüge (FlugNr, Datum, Abflugzeit, Ankunftszeit, FSNr)
Flugsteige (FSNr)
Sitze (FlugNr, Datum, SitzNr)
PassagiereMitReservation (ResNr, Name, Adresse, FlugNr, Datum, SitzNr, TicketNr)
GebuchtePassagiere (TicketNr, Name, Adresse)
Buchungen (TicketNr, FlugNr, Datum)
Vielflieger (VFNr, Name, Adresse, kmStand)
Gepäckstücke (GNr, Gewicht)
Check-In (FlugNr, Datum, TicketNr, GNr)

10.4 Datenmodellierung mit UML

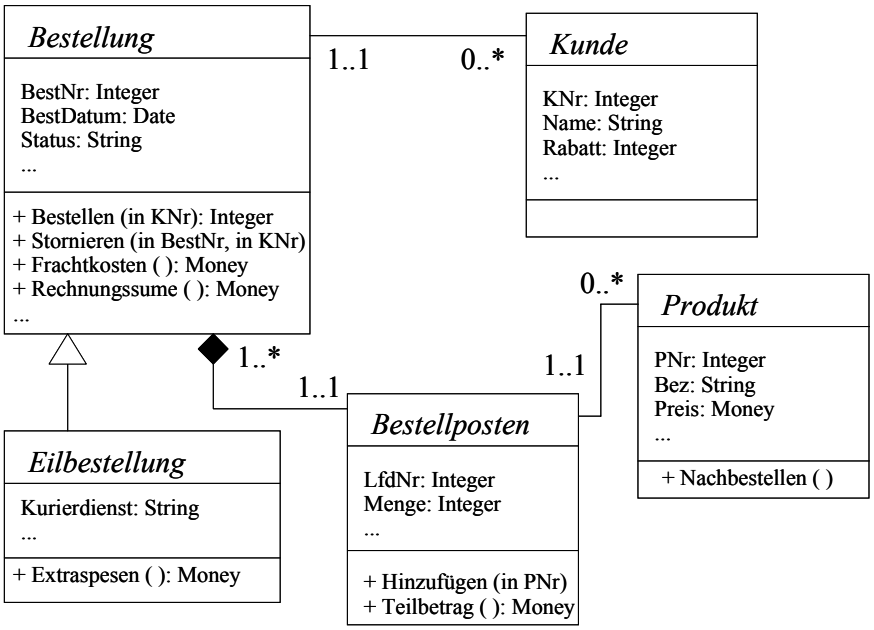
Datenbankentwürfe, die mit einer „semantischen“ Datenmodellierungsmethode, z.B. als Entity-Relationship-Diagramm, erstellt worden sind, lassen sich auch in Form von *Objektklassendiagrammen* darstellen (vgl. auch Kapitel 8 über ODMG). Für die Darstellung hat sich der Industriestandard *UML (Unified Modeling Language)* durchgesetzt. Vorteile gegenüber ERM-ähnlichen Darstellungen liegen angeblich darin, daß

1) Objektaggregationen, d.h. die Bildung komplexer Objekte, zu gekapselten Objekten und
2) anwendungsspezifische Funktionen, d.h. Methoden auf Objekten,

in einfacher Weise direkt in einem *Klassendiagramm (Class Diagram)* dargestellt werden können.

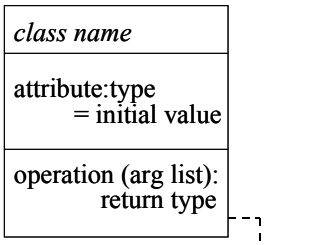
Darüber hinaus wird häufig proklamiert, daß die Fokussierung auf Objekte (und das “Zurückdrängen" von Relationships) den Entwurfsprozeß selbst besser strukturiert und zu einer übersichtlicheren Darstellung führt.

Beispiel eines Klassendiagramms nach UML:



Klassendiagramme basieren auf den folgenden Piktogrammen, deren Bedeutung jeweils nur informal definiert ist. Es gibt z.Zt. keine formal-präzise Beschreibung der Semantik aller UML-Konstrukte.

Klassen und Objekte

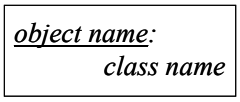


Klasse (Objekttyp, Komponente)

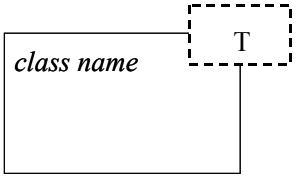
Attribute

Methoden mit in-, out- und inout-Parametern
+: public, -: private, #: protected

Konsistenzbedingungen, Vor- und Nachbedingungen (constraints) in Textform

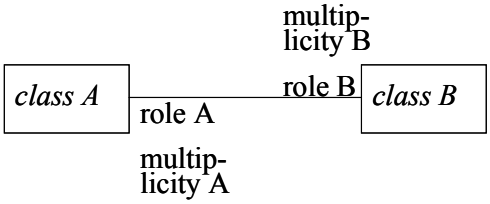


Objekt (Instanz einer Klasse)

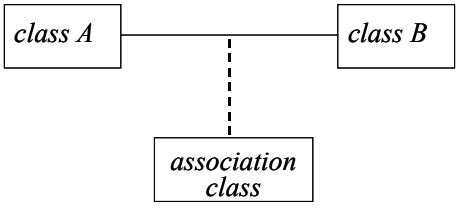


Parametrisierte Klasse (Template Class)
z.B. List <T>

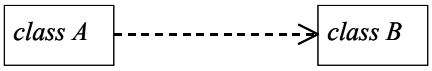
Assoziationen, Aggregationen, Kompositionen



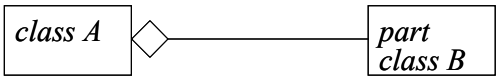
Assoziation / Relationship (assocations)
zwischen Klassen mit Rollennamen
und Kardinalitäten (min..max) und
ggf. der Option ordered



Beziehungsklasse (assocation class):
entspricht den ERM-Relationships
mit eigenen Attributen



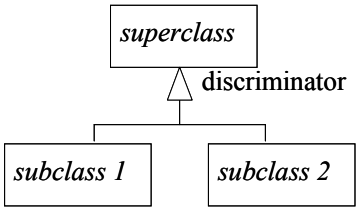
Abhängigkeit (dependeny)
der Klasse A von Klasse B
(z.B. Server-API-Klasse B und
Client-Klasse A)



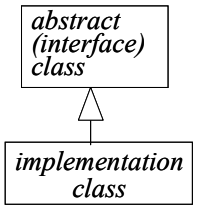
Aggregation (aggregation)
mit shared objects
(Sonderfall der Assoziation)

Komposition (composition)
mit non-shared objects
(Sonderfall der Assoziation)

Generalisierungen / Spezialisierungen



Generalisierungshierarchie
(generalization)

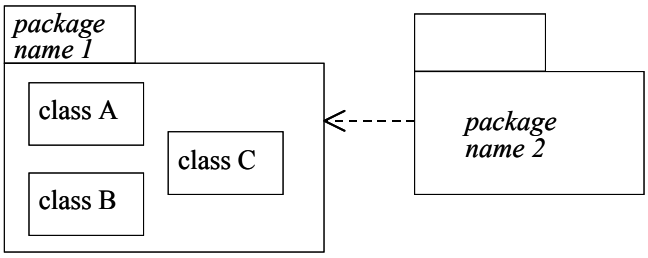


Beziehung zwischen Schnittstelle (ADT)
und Implementierung
(Sonderfall der Generalisierung /
Spezialisierung)

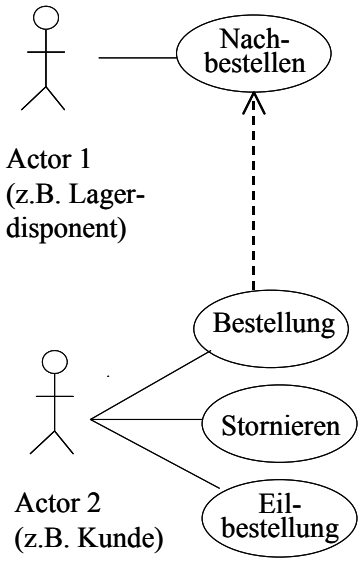
UML beinhaltet eine Reihe weiterer Diagrammarten zur Darstellung des gesamten Anwendungsentwurfs (also nicht nur der Datenaspekte). Dazu zählen:

- *Paketdiagramme (Package Diagrams)* als vergröberte Sicht auf große Klassendiagramme,
- *Anwendungsfalldiagramme (Use-Case Diagrams)* zur Grobdarstellung der Beziehung zwischen Benutzern und Funktionen in bestimmten Anwendungsabläufen,
- *Interaktionsdiagramme (Sequence Diagrams)* zur Darstellung der Methodenaufrufe zwischen Objektklassen,
- *Zustandsübergangsdiagramme (State-Transition Diagrams)* zur feineren Darstellung des Verhaltens von Objekten in bestimmten Anwendungsabläufen,
- *Aktivitätsdiagrame (Activity Diagrams)* zur Darstellung des Kontroll- und Datenflusses zwischen Objekten.

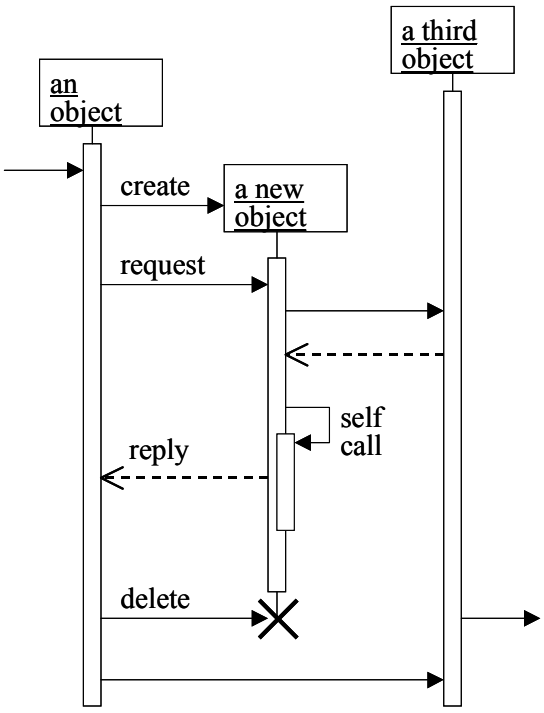
Insbesondere Zustandsübergangsdiagramme sind für die Spezifikation der “Dynamik" eines Informationssystems, also des Verhaltens der Datenobjekte im Kontext von *Geschäftsprozessen (Business Processes)* von Bedeutung (siehe Kapitel 11).



Paketdiagramm
(package diagram)



Anwendungsfalldiagramm
(use-case diagram):
Zusammenhang zwischen
Benutzern bzw. Rollen
und Funktionen / Ereignissen



Interaktionsdiagramm
(sequence diagram):
Nachrichtenfluss zwischen
Objekten (bzw. Threads)

Insgesamt dient UML als Beschreibungsmittel für den gesamten Entwurfs- und Entwicklungsprozeß von der informellen Kommunikation zwischen Anwendern, Fachkonzeptexperten und Informatikern in den frühen Phasen der Anforderungsanalyse und des Grobentwurfs bis hin zur Dokumentation der Implementierung. Dies erklärt die etwas barocke Vielfalt der Diagrammtypen und auch den erkennbaren Mangel an formaler Durchdringung. An der Beschreibung einer präzisen Semantik für die im Feindesign und der Implementierung wichtigen UML-Konstrukte wird aktiv gearbeitet.

10.5 Objektorientierte Analyse- und Entwurfsmethodologie

Für die Anforderungsanalyse und den Anwendungsentwurf wird - basierend auf der objektorientierten Darstellungsmethode (insbesondere den Klassendiagrammen) - häufig auch eine bestimmte *Vorgehensmethodologie* verfochten. Diese stellt - mehr oder weniger grobe - Richtlinien zusammen, in welchen Schritten die Analyse bzw. der Entwurf vorangetrieben werden sollte. Die wohl bekannteste unter diesen Schulen ist die *Object Modeling Technique (OMT)* von Rumbaugh, deren diagrammbasierte Notation zusammen mit den Notationen von Booch und Jacobson zum Industriestandard UML geführt hat.

OMT propagiert die folgenden Analyse- bzw. Entwurfsschritte, die zur schrittweisen Verfeinerung ineinander verzahnt iterierbar sind:

1) *strukturelle Modellierung* der Datenobjekte:
   Objektklassen mit ihren Assoziationen, insbesondere Aggregationen
2) *Dynamikmodellierung*:
   Zustandsübergangsdiagramme zur Darstellung des dynamischen Verhaltens von Objekten bzw. Geschäftsprozessen
3) *Funktionsmodellierung*:
   Beschreibung der Objektmethoden, ihrer gegenseitigen Aufrufe und des Datenflusses zwischen ihnen

Diese Grobstruktur kann weiter detailliert werden, bleibt jedoch letzten Endes nur ein Leitfaden und darf keinesfalls als Patentlösung für die Praxis angesehen werden.

Werkzeuge

Es gibt etliche Softwarewerkzeuge (z.B. Erwin, Rational Rose, etc.), mit denen ERM- oder UML-artige Modelle graphisch erstellt werden können; diese Werkzeuge bilden solche Modelle automatisch auf relationale oder objektorientierte Schemata oder Interface-Beschreibungen ab und können u.a. SQL-Anweisungen für Create Table inkl. Constraints generieren.

Ergänzende Literatur zu Kapitel 10:

Teorey, T.J., Database Modelling and Design, Morgan Kaufmann, 1999

Batini, C., Ceri, S., Navathe, S.B., Conceptual Database Design, Benjamin/Cummings, 1992

Muller, R.J., Database Design for Smarties: Using UML for Data Modeling, Morgan Kaufmann, 1999

Fowler, M., Scott, K., UML Distilled, Addison-Wesley, 2000

Maciaszek, L.A., Requirements Analysis and System Design: Developing Information Systems with UML, Addison-Wesley, 2001

Object Management Group (OMG), Unified Modeling Language (UML) Documentation, http://www.omg.org/technology/uml/index.htm oder http://www.rational.com/uml/index.jsp

Scheer, A.-W., ARIS - Business Process Modeling, 2nd Edition, Springer-Verlag, 1999

Balzert, H., Lehrbuch der Software-Technik, Band 1: Software-Entwicklung, Spektrum-Verlag, 1996