Anwendungssystem eines Schulungsanbieters

Hausarbeit im Bereich Datenbanksysteme

Björn Christian Schaefer
Georg-Gröning-Straße 48
28209 Bremen
Email: bcs@tzi.de
Matr.Nr. 2082808
Bremen, 8. März 2009

Inhaltsverzeichnis

1 Informelle Anwendungsbeschreibung

In dieser Arbeit wird ein Datenbankschema für einen Veranstalter von Schulungen im Multimediabereich entworfen. Als reale Vorlage diente dabei eine berufsbildende Schule, die neben dem Schulangebot auch Seminare und Kurse zu unterschiedlichsten Themen anbietet. Es gibt dort ein allgemeines Schulsekretariat, in dem auch Anfragen zum Kursangebot eintreffen, und eine Lehrkraft, die die Organisation des Angebots und der Durchführung (zusatzlich zur normalen Lehrtätigkeit) übernimmt. Diese Situation diente als Ausgangspunkt für die Analyse der Anforderungen, aber das hier entworfene System ist nicht für den tatsächlichen Einsatz geplant.

Das System soll vor allem die Verwaltung der Kunden und Kontakte, der angebotenen Kurse und geschäftsrelevanten Vorgänge wie Buchungen, Rechnungsstellung usw. abdecken. Bisher werden die relevante Daten von Hand in verschiedenen Excel-Tabellen gepflegt, was gravierende Nachteile mit sich bringt, weil Geschäftsabläufe ohne Unterstützung durch Automatismen oder Fehler- und Konsistenzprüfungen abgewickelt werden müssen. Die schwerwiegendsten Probleme sind:

- Die Synchronisierung der verschiedenen Datenbestände ist aufwändig und fehleranfällig
- Daten sind oft inkonsistent, redundant oder unvollständig
- Zusammenarbeit mehrerer Personen bei der Verwaltung ist schwierig
- Das Anbieten neuer Kurse erfordert die Duplizierung aller Verwaltungstabellen
- Wichtige Fragen wie z.B. “Wer interessiert sich für einen Kurs im Bereich XY?” oder “Welchen Interessenten wurde bereits Informationsmaterialzugeschickt?” oder “Welche Rechnungen sind noch offen?” können nicht automatisch beantwortet werden

Allgemein gesagt ist das Customer Relationship Management (CRM) untragbar aufwändig und fehleranfällig. Die Ziele der Umstellung auf ein datenbankgestütztes System sind also, die vielschichtigen Probleme der manuellen Datenpflege und -auswertung zu beseitigen:

- Konsolidierung der Daten: Es soll ein einziger vollständiger, zentraler und möglichst redundanzfreier Datenbestand geschaffen werden.
- Kollaboration: Die Zusammenarbeit mehrerer Personen bei der Verwaltung soll vereinfacht werden. Die Konsolidierung des Datenbestandes ist dafür eine Voraussetzung. Mit einem zentralen Datenbestand können dann Wege gefunden werden, diesen für mehrere MitarbeiterInnen zugänglich zu machen, z.B. eine Webanwendung im Intra- oder Internet.
- Ablaufüberwachung/Qualitätssicherung: Das System soll wichtige Geschäftsabläufe abbilden können und für automatische Auswertungen zugänglich machen. Vor allem die Ausführung und Überwachung von Routineabläufen soll durch die Umstellung auf ein datenbankgestütztes System deutlich vereinfacht und verbessert werden. Jeder Beteiligte soll jederzeit den aktuellen Stand von Vorgängen wie z.B. dem Versand von Informationsmaterial, Anmeldebestätigungen oder Rechnungen abfragen können.

Der Entwurf muss für die Erreichung dieser Ziele viele unterschiedliche geschäftsrelevanten Informationen umfassen, die man grob in die Hauptgruppen Kurse, Kontakte, Buchungen und Geschäftsvorfälle unterteilen kann. Weil die Eigenheiten dieser Informationsgruppen den Entwurf direkt beeinflussen, werden sie in den folgenden Abschnitten genauer erklärt.

1.1 Kurse

Bei den Kursen muss zwischen Konzept und konkretem Kurs unterschieden werden: Das Konzept ist die abstrakte Definition, der Kurs die konkrete Realisierung des Unterrichts.

1.1.1 Kurskonzepte

Ein Kurskonzept definiert eine Bezeichnung, eine Freitextbeschreibung, den ungefähren zeitlichen Umfang (wird erst im konkreten Kurs definitv festgelgt) und die Zuordnung zu einem oder mehreren Themengebieten wie z.B. Webdesign, Print-Publishing, Bildbearbeitung oder 3D-Grafik. Bei den Themen kann es sich auch um spezielle Softwarekurse handeln, z.B. zu Dreamweaver oder Photoshop. Ein zweitägiger "Chrashkurs Photoshop" könnte ein solches Konzept sein. Für Themen sollen Verwandschaften definierbar sein, die die erweiterte Suchen nach angrenzenden Themen ermöglichen. Der Entwurf soll später auch um baumförmige Themenhierarchien erweiterbar sein. Die Trennung von Kurs und Konzept ermöglicht es, die selben Inhalte mehrmals unter dem selben Namen aber mit anderen Zeiten, Orten, Dozenten und Teilnehmern zu vermitteln.

Die Dauer reicht von wenigen Stunden bis hin zu mehreren Wochen oder Monaten als Vollzeit- oder berufsbegleitenden (Abend-)Seminaren. Konzepte sollen sich im Laufe der Zeit ändern können (z.B. Schwerpunktverschiebungen bei Themen und in der Beschreibung), ohne dass sich ihre Bezeichnung ändert, und ohne dass diese Änderungen an vergangene Kurse weitergegeben werden (Dokumentationsfunktion).

1.1.2 Konkrete Kurse

Ein Kurs setzt ein Kurskonzept um, indem Termine, Dozenten und Teilnehmer (TN) zusammengeführt werden. Beliebig viele Kurse können das selbe Konzept umsetzen, auch parallel. Z.B. kann der genannte Crashkurs bei großer Nachfrage zeitgleich an zwei verschiedenen Orten mit verschiedenen Dozenten und Teilnehmern stattfinden, oder ein eintägiger Kurs jeden Samstag angeboten werden. Kurse werden entweder in den Veranstaltungsräumen des Anbieters durchgeführt oder finden als sogenannte "Inhouse-Schulungen" in den Räumen des Kunden (normalerweise Unternehmen) statt. Die Teilnehmer sind normalerweise namentlich bekannte Personen; allerdings kann es durchaus sein, dass z.B. ein Unternehmen eine Inhouse-Schulung pauschal bucht und weder die Zahl noch die Namen der Teilnehmer je bekannt werden. Konkrete Kurse haben kalkulatorische Kosten für den Veranstalter (Aufwendungen für Dozenten, Räume etc.), die für interne Zwecke gespeichert werden müssen, sowie ggf. minimale und maximale Teilnehmerzahlen sowie einen Grundpreis pro TN. Diese Angaben können bei pauschal vereinbarten Kursen natürlich leer bleiben.

Kurse können (zeitlich) fest geplant sein oder "nach Vereinbarung" stattfinden. Beispielsweise kann das Konzept "Photoshop für Einsteiger" jeden Samstag und zusätzlich nach Vereinbarung angeboten werden. Interessenten können sich für solche Kurse vormerken lassen und werden mit Terminvorschlägen angeschrieben, sobald die Mindestteilnehmerzahl erreicht ist. Für diesen Zweck wird für jeden Kurs auch ein Status vermerkt, der u.a. angeben kann, dass sich der Kurs

"in Planung" befindet (z.B. während die Bedingungen einer Inhouse-Schulung mit dem Kunden verhandelt werden) oder "Vormerkungen möglich" sind, etc.

1.1.3 Kurstermine

Ein Termin ist eine Zeitspanne an einem definierten Ort. Zeitlich betrachtet können Kurse in verschiedensten Varianten gehalten werden: Täglich zu festen Zeiten, in mehrern ganztägigen Blöcken an verschiedenen Orten, an einem Tag in mehreren Blöcken mit unterschiedlichen Dozenten usw. Es gibt also kein festes Raster, an das sich alle Kurstermine halten, deswegen ist bei der Modellierung der Termine größtmögliche Flexibilität gefragt. Es muss sich z.B. problemlos speichern lassen, dass ein Kurs an einem Tag in drei Zeitblöcken an drei unterschiedlichen Orten stattfindet, wobei der zweite Block von vier Dozenten gleichzeitig betreut wird (z.B. Laborpraktikum). Ausserdem muss die Zeitplanung unabhängig von Dozenten möglich sein, d.h. Termine müssen definierbar sein, ohne dass ein Dozent dafür bekannt ist (Dozenten werden gelegentlich erst sehr zeitnah engagiert). Termine sind grundsätzlich auch unabhängig von Kursen; Termine ohne Kurszuordnung sind sozusagen "freie Slots", die z.B. bei Räumen sinnvoll sind, die in festen Zeitrastern belegt werden können, wo also die möglichen Termine schon feststehen.

1.2 Kontakte

1.2.1 Allgemeines

Ein Kontakt ist prinzipiell eine namentlich bekannte Einheit, mit der kommuniziert werden kann. Pro Kontakt sollen beliebig viele konkrete Kontaktinformationen zu den drei möglichen Kontaktwegen Telefon, Email und Anschrift gespeichert werden können. Es kann auch Kontakte ohne weitere Kontaktinformationen geben, vor allem Kursteilnehmer, die bei Gruppenbuchung oder Inhouse-Schulung keine eigene Rechung erhalten und von denen nur der Name bekannt ist.

Jede Kontaktinformation (Telefonnummer, Emailadresse oder Anschrift) trägt zur genaueren Beschreibung einen Zusatz (z.B. "privat"', "abends" o.ä.) und ein Notizfeld für Freitextbemerkungen. Wenn zu einem Kontakweg Kontaktinformationen vorliegen, ist davon immer genau eine als Standard zu markieren. Durch diese Markierung gibt es immer eine eindeutige bevorzugte Kontaktinformation (z.B. für Serienbriefexport). Bei Telefonnummern wird zusätzlich zur Nummer auch der Typ (Mobil, Festnetz, Fax) gespeichert. Anschriften setzen sich aus Straße, Hausnummer, Postleitzahl und Ort zusammen und enthalten implizit den Namen des Kontaktes.

1.2.2 Personen und Unternehmen

Im Entwurf sind zwei spezielle Kontaktarten zu unterscheiden, nämlich (natürliche) Personen und Unternehmen (bzw. im weiteren Sinne juristische Personen). Bei Personen sind zusätzliche Angaben zu Vorname, Anrede (Geschlecht) und Geburtsdatum zu erfassen, bei Firmen die Steuernummer. Daneben sollen auch allgemeine Kontakte verwaltet werden können, Näheres dazu in Abschnitt 1.2.3.

Dozenten sind eine Spezialisierung von Personen. Für alle Dozenten muss bekannt sein, in welchen Themenbereichen sie unterrichten. Das System muss für die Kursplanung ermöglichen, eine Liste möglicher Dozenten (passend zu Kursinhalten) zu liefern und ihren Status zu überwachen (angefragt, Vertrag verschickt, Vertrag abgeschlossen, Rechnung erhalten/bezahlt). Die Zuordnung zu Themen soll dabei die Dozentensuche bei der Planung erleichtern und kein "hartes" Kriterium sein, d.h. ein Dozent kann durchaus Kurse unterrichten, deren Themen nicht in seiner

Themenliste vorkommen. Mit Dozenten werden normalerweise grundsätzliche Stundensätze vereinbart, der ihr (Basis-)Honorar pro Stunde festlegen. Das tatsächliche Honorar pro gehaltenem Kurstermin kann aber auch abweiched verhandelt werden und muss zur Dokumentation und Rechnungsstellung ebenfalls gespeichert werden.

1.2.3 Hierarchien

Zwischen Kontakten sollen sich baumförmige hierarchische Beziehungen definieren lassen, um z.B. Filialen, Abteilungen oder Ansprechpartner einer Firma zu erfassen und einem Hauptkontakt zuordnen zu können. Abbildung 1.1 zeigt ein Beispiel für eine mögliche Kontakthierarchie (Firmenkontakte sind blau hinterlegt, Personen grün, allgemeine Kontakte farblos).

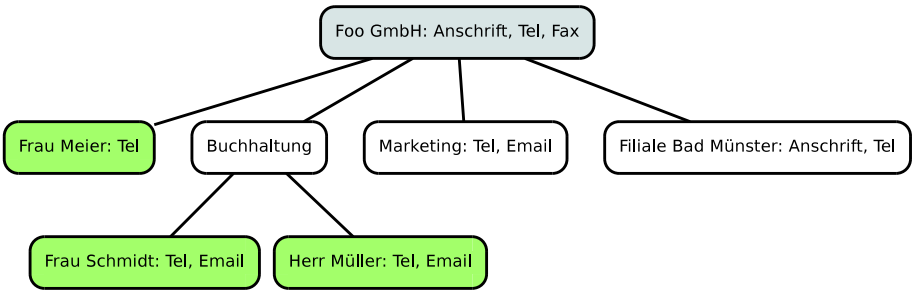


Abbildung 1.1: Beispiel einer Kontakthierarchie

Die gezeigte Foo GmbH ist unter einer allgemeinen Telefon- und Faxnummer erreichbar und hat eine Anschrift. Es gibt eine Filiale, von der uns Anschrift und Telefonnummer bekannt sind. Frau Meier ist eine nicht näher spezifizierte Kontaktperson mit eigener Telefonnummer. Es gibt eine Marketingabteilung mit Telefonnummer und Emailadresse, aber ohne namentlich bekannte Ansprechpartner, und die Buchhaltung, in der zwei Ansprechpartner bekannt sind, die aber keine allgemeinen Kontaktinformationen hat.

Die "Buchhaltung" zeigt, dass Kontakte ohne Kontaktinformationen zur logischen Strukturierung sinnvoll sind. Diese Strukturen müssen es erlauben, effizient zu einem Kontakt den übergeordneten Kontakt, alle direkt untergeordneten Kontakte und alle zusammengehörigen Kontakte einer Hierarchie ausfindig zu machen.

1.2.4 Rollen

Aus Geschäftssicht können Kontakte verschieden Rollen haben, wobei auch Kombinationen denkbar sind. Die Rollen ergeben sich dabei immer implizit aus den Beziehungen zu einem Kontakt und müssen/sollen nicht explizit angegeben werden. Ein Kontakt kann in unterschiedlichen Zusammenhängen unterschiedliche Rollen einnehmen, z.B. kann er ein Mal Teilnehmer, ein anderes Mal Kunde und in einem Dritten Fall Ansprechpartner sein. Das System muss Rollenzugehörigkeiten feststellen können. Die wichtigsten Rollen sind:

- Kontaktperson ist jede natürliche Person einer Kontakthierarchie
- Auftraggeber ist der Kontakt, der eine Buchung veranlasst hat. Also z.B. eine namentlich bekannte Ansprechperson eines Unternehmens.
- Ein Kunde ist immer die Spitze der Kontakhierarchie, die eine Buchung beauftragt hat. Im Beispiel also das Unternehmen, dessen Kontaktperson die Buchung beauftragt hat.
- Ein Teilnehmer ist eine Person, die mindestens ein Mal für einen Kurs gebucht ist.

1.3 Kursbuchungen

Kursbuchungen bringen Kurse, Teilnehmer und Kunden (= Auftraggeber, Rechnungsempfänger) zusammen. Dabei sind verschiedene Varianten möglich die sich vor allem darin unterscheiden, wie viele Teilnehmer (TN) über eine Buchung abgerechnet werden, ob der Auftraggeber selbst auch TN ist und ob die TN namentlich bekannt sind:

- Selbstzahler: Eine Person bucht für sich selbst einen Platz in einem Kurs und zahlt selbst die Rechnung.
- Fremdzahler: Ein TN wird von einem anderen Kontakt bezahlt (z.B. von Arbeitsagentur bei SGB-III geförderten Maßnahmen)
- Gruppenbuchung: Mehrere TN werden über eine Rechnung bezahlt. Der Zahler kann einer dieser TN sein, das muss aber nicht der Fall sein.
- Anonyme Teilnehmer: Der Auftraggeber bucht und zahlt eine Anzahl von Plätzen in einem Kurs, die TN sind aber (noch) nicht namentlich bekannt. Vor allem bei Firmenschulungen üblich.
- Pauschalbuchung: Der Auftraggeber bucht einen kompletten Kurs; die Zahl und Namen der TN können bekannt sein oder auch nicht.
- Mehrfachbuchung: Der selbe Kontakt darf jeden Kurs mehrfach buchen. Dadurch erhält er mehrere getrennte Rechnungen. Anwendungsfall: Fördermaßnahme der Arbeitsagentur, die mehrere Teilnehmer in einem Kurs finanziert und für jeden eine separate Rechnung benötigt.

Dabei kann es auch Mischformen geben, bei denen es im selben Kurs Einzelteilnehmer mit eigener Rechnung gibt und Teilnehmergruppen, die gemeinsam über einen Kunden abgerechnet werden. Es kann aber niemals eine Buchung ohne beauftragenden Kontakt und ohne gebuchten Kurs geben. Bei allen Varianten müssen flexibel verhandelbare Konditionen pro Kunde und Buchung möglich sein, und die Daten von anfangs anonymen Teilnehmern müssen jederzeit nacherfasst werden können. Die letztendlich vereinbarten Konditionen (Preis, TN-Anzahl) und die verwendete Rechungnsanschrift müssen dauerhaft für Dokumentationszwecke gespeichert werden.

1.4 Geschäftsvorfälle

Ein sehr wichtiger Teil jedes Unternehmens sind die dynamischen Geschäftsprozesse, die während der Leistungserbringung ablaufen. Ein Beispiel sind die Schritte, Tätigkeiten und Zwischenzustände, die den Prozess von der ersten Anfrage eines Interessenten über die Abwicklung seiner Kursanmeldung bis zum abschließenden Versand des Zeugnisses und der Überwachung der Rechnungsbegleichung ausmachen, und die ungefähr so aussehen könnten: Die Kontaktdaten des Interessenten müssen aufgenommen werden; ein Anmeldeformular muss zugeschickt werden; die Anmeldung muss ausgeführt werden; eine Anmeldebestätigung muss verschickt werden; die Rechnung muss verschickt werden; nach Abschluss des Kurses muss ein Zeugnis ausgestellt werden; Zahlungseingänge müssen vermerkt werden.

Das System soll Geschäftsprozesse dieser Art unterstützen und überwachen können. Die bisher besprochenen Daten zu Kontakten, Kursen und Buchungen sind dabei die Grundlage dieser Prozesse. Es fehlen bisher noch die Angaben darüber, welche Teiltätigkeiten es gibt und welche wann erledigt wurden.

Das ist Prozesswissen und damit direkt abhängig von den betrieblichen Vorgängen. Diese Vorgänge änden sich in der Regel im Laufe der Zeit: Neue Aufgaben und Abläufe kommen dazu,

alte werden verändert oder entfallen völlig. So könnte z.B. die Rechnungsstellung an ein externes Inkassounternehmen abgegeben werden, wodurch sich die entsprechenden Geshäftsabläufe stark ändern würden. Solche Änderungen sollen möglichst ohne Anpassungen des Datenschemas möglich sein. Daraus folgt, dass das Wissen um die Geschäftsabläufe in den Inhalten und nicht in der Struktur des Schemas abgelegt werden muss. Die Art der möglichen Vorfälle und die Geschäftslogik dahinter dürfen deswegen nicht in das DB-Schema eingehen, sondern müssen selbst als Daten behandelt werden. Der Entwurf soll es ermöglichen, die Definitionen von Ereignissen dieser Art zu speichern, so dass z.B. die Buchhaltung selbst neue Ereignistypen definieren kann, wenn das nötig werden sollte, z.B. "Kurs beworben" um die Effektivität von Werbemaßnahmen auswerten zu können.

Für die Modellierung solcher Vorgänge bieten sich erweiterte Ereignis-Prozess-Ketten (eEPK) an. Ereignisse sind darin definiert als "das Eingetretensein eines definierten Zustandes" ([KNS92], in Anlehnung an DIN 69900) und sind damit ideale Kanditaten für die Speicherung zur Überwachung von Geschäftsprozessen. Mit dieser Definition kann man also aus Ereignissen direkt den Zustand eines Geschäftsprozesses ablesen, und genau das ist ja eine Anforderung an den Entwurf. Denn wenn bekannt ist, welche Ereignisse einen Prozess ausmachen und welche davon bereits eingetreten sind, folgt damit automatisch das Wissen um den aktuellen Zustand eines Prozesses.

Das einzige "Wissen" das strukturell in den Datenbankentwurf eingeht ist also die Tatsache, dass es Ereignisse gibt die Entitäten der Datenbank betreffen können. Alles Wissen über die eigentlichen, konkreten Ereignisse liegt als Datenbankinhalt vor und nicht in der Struktur. Eine Änderung der Geschäftsprozesse berührt daher nicht die Struktur der DB, sondern nur deren Inhalt (und natürlich das Anwendungssystem).

Das System soll damit in der Lage sein, in diesem Zusammenhang Fragen der folgenden Art beantworten zu können: [1]

- Wurde an alle Interessenten Infomaterial geschickt?
- Wurden für alle Buchungen Bestätigungen geschickt?
- Wurden alle Rechnungen für Kurs XY verschickt? Wann? Welche sind bezahlt?
- Sind (global) noch Rechnungen offen? Welche?
- Anfragen zu Kursen, Buchungen, ...
- Versand von Informationsmaterial, Rechnungen, Zeugnissen, ...
- Eingang von Anmeldungen, Honorarrechnungen, ...
- Zahlungsein- und -ausgänge

Es muss also eine möglichst generische Moglichkeit gefunden werden, Ereignisse zu speichern. Welche Eriegnisse das genau sind, muss dabei dynamisch definierbar sein, darf sich also möglichst nicht in der Struktur des Modells niederschlagen.

Ein Beispiel: Um auszuwerten welche Zertifikate nach Kursabschluss schon verschickt wurden, kann dafür ein Entitytyp geschaffen werden, der dieses Wissen abbildet. Oder es wird ein Ereignis vom Typ "Zertifikat verschickt" gespeichert, das sich auf einen bestimmten Teilnehmer

[1] Das erreicht dieser Entwurf natürlich nur Beispielhaft und in Ansätzen, Schwerpunkt der Arbeit ist ja nicht die Modellierung von Geschäftsprozessen.

eines bestimmten Kurses bezieht. Statt eines Entitytypen für jeden Ereignistyp gibt es also eine einzige Liste generischer Ereignisse, die durch Typ ("Zertifikat verschick") und typspezifische Bezüge auf andere Entitytypen ("An wen?" "Für welchen Kurs?") das konkrete Prozesswissen der allgemeinen EPK speichern. Wenn jetzt noch alle Ereignisse immer das Datum des Eingetretenseins beinhalten, lässt sich mit diesem Wissen der Ablauf jedes realen Geschäftsprozesses nachvollziehen.

Wenn man das vollständig und realitätstauglich umsetzen wollte, müssten als Voraussetzung für den Entwurf alle Geschäftsprozesse detailliert untersucht werden, um alle möglichen und nötigen Bezugstypen zu erfassen. Das Beispiel für einen Standardablauf einer Kursteilnahme ist in der folgenden Tabelle in mögliche Ereignistypen aufgeschlüsselt; "betrifft" enthält dabei die Entititypen, auf die sich ein konkretes Ereignis bezieht.

Ereignis	betrifft
Infomaterial verschickt	Kurs, Kontakt
Anmeldung eingegangen	Kurs, Kontakt
Anmeldung vollzogen	Buchung
Anmeldebestätigung verschickt	Buchung
Rechnung verschickt	Buchung
Zahlungseingang	Buchung
Rechung vollst. bez.	Buchung
Zertifikat an TN verschickt	Buchung, Kontakt
Zertifikat an Kunde verschickt	Buchung

Für diese Arbeit können die Geschäftsprozesse nicht detilliert analysiert werden. Es wird daher davon ausgegangen, dass sich Geschäftsvorfälle nur auf Kontakte, Buchungen und Kurse beziehen können, dabei aber auf beliebige Kombinationen daraus und auch mehrfach auf einen Bezugstyp. Damit sollten die wichtigsten Anwendungsfälle abgedeckt sein.

Auserdem wird es eine Möglichkeit geben, Geschäftsvorfälle zur Wiedervorlage zu vermerken und als "endgültig abgeschlossen" zu markieren. Damit ist das Schema dan sogar in der Lage, "ToDo"-Listen und ähnliche Dinge zu verwalten, indem passende Vorfälle definiert werden.

2 Konzeptioneller Entwurf als ER-Schema

Das konzeptionelle Schema steht in der klassischen drei-Ebenen-Architektur von Datenbanksystemen zwischen der physischen Sicht, die die letztendlich vom DBMS gespeicherten Tabellen, Indices usw. umfasst, und den externen Sichten, die die Informationsbedarfe spezieller Anwendungen abdecken und Teilmengen oder Auswertungen der gespeicherten Daten enthalten können. Das konzeptionelle Schema enthält also alle geschäftsrelevanten Informatioen und ihre Beziehungen untereinander ohne Berücksichtigung der Eigenheiten der späteren Anwendungen oder des DBMS.

Weil es in dieser Hausarbeit vor allem um die Prinzipien geht, ist der Entwurf an vielen Stellen nicht vollständig in dem Sinne, dass bei der tatsächlichen Realisierung noch zusätzliche Attribute und/oder Entitytypen verwendet werden würden. Diese hier aufzuzählen würde aber nichts Wesentliches hinzufügen. Beispielsweise gibt es mehrere Attribute, die nur Elemente einer bestimmten Wertemenge annehmen können - ein Kursstatus könnte z.B. nur die Werte “in Planung”, “laufend” und “abgeschlossen” annehmen. Solche Auswahllisten würden oft als eigene Entitytypen wie in Abbildung 2.1 modelliert werden, die die möglichen Werte enthalten. Dadurch kann vor allem die Bedienung erleichtert und Fehleingaben vermieden werden, weil Tippfehler und unterschiedliche Schreibweisen ausgeschlossen werden können. Ausserdem macht ein definierter Werteberich automatische Auswertungen einfacher oder überhaupt erst möglich. Darüber hinaus kann eine Trennung von Bezeichnung und ID erfolgen, wodurch spätere Änderungen der Bezeichnung und z.B. auch Übersetzungen in andere Sprachen sehr vereinfacht werden.

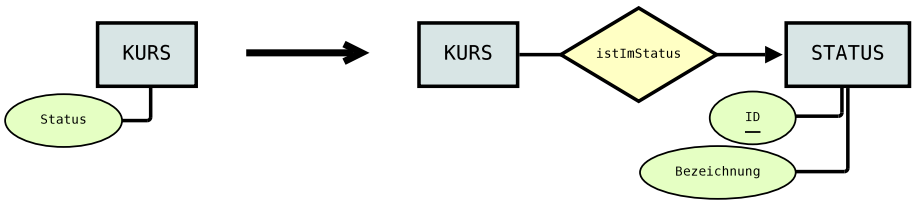


Abbildung 2.1: Detaillierte Modellierung fest definierter Wertelisten

Um das Schema nicht ausufern zu lassen, wurden diese Detailmodellierungen wie gesagt ausgelassen, man kann sie sich aber leicht nach dem gezeigten Schema dazudenken. Kandidaten dafür sind im Gesamtschema insbesondere PERSON.Anrede, KONTAKTWEG.Zusatz, TELEFON.Typ, KURS.Status. Als Sonderfall kann auch ANSCHRIFT.PLZ betrachtet werden, wobei die PLZ als Schlüssel erhalten bleibt und der zugehörige Ortsname die Rolle der Bezeichnung einnehmen würde.

2.1 Grafische Darstellung

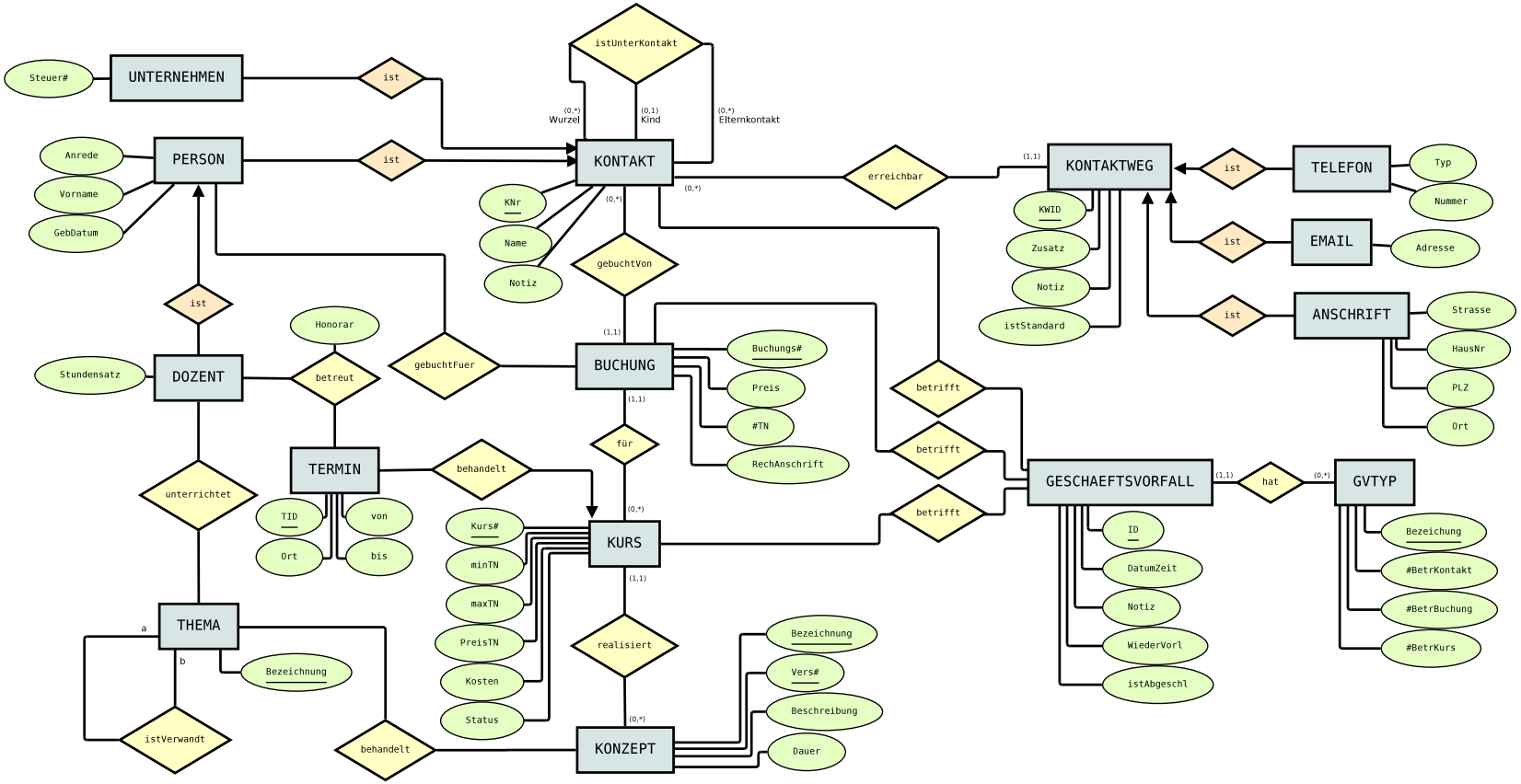


Abbildung 2.2: Konzeptioneller Entwurf als ER-Diagramm

2.2 Verwendete Elemente

Bevor der Entwurf im Detail beschrieben wird, sollen hier kurz die verwendeten grafischen Elemente vorgestellt werden.

Entitytypen, Attribute und Schlüssel

Die Entitytypen sind wie in Abb. 2.3 als blau hinterlegte Rechtecke dargestellt und ihre Namen sind komplett groß geschrieben. Das ermöglicht auch in textueller Notation die Unterscheidung von Relationships. Attributnamen werden mit Linien verbunden in grünen Ovalen dargestellt und können Rauten "#" anthalten, die als "Nummer" oder "Anzahl" zu lesen sind. Schlüsselattribute werden unterstrichen (Beispiel 2.3 enthält nur eins, es können aber auch mehrere sein).

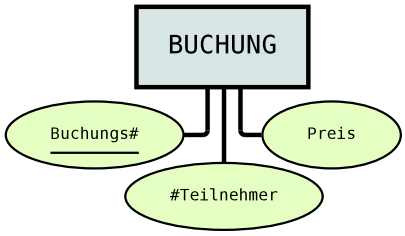
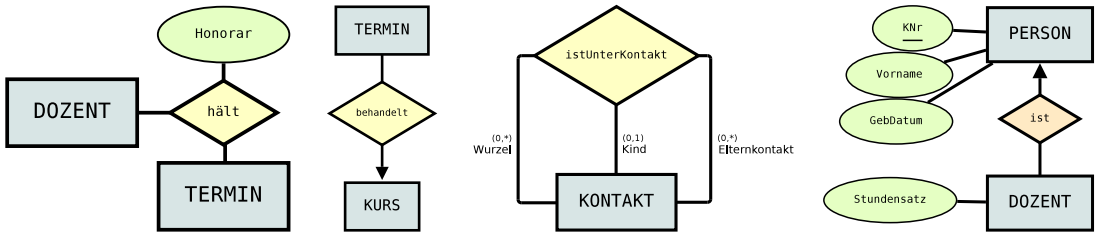


Abbildung 2.3: Entitytyp mit Attributen

Entities e sind Elemente eines konkreten Zustandes ihres Entitytyps E (Tupel in der DB): $e \in \sigma(E)$

Zu einem Attribut A gehört neben dem Entitytyp E formal auch immer ein Datentyp D, der hier zur besseren Übersichtlichkeit weggelassen ist. Formal definieren Attribute eine Funktion $A : E \rightarrow D$, die Entitytypen Datentpen zuordnen, konkret z.B. $\#Teilnehmer : KursBuchung \rightarrow Integer$. Das Attribut "#Teilnehmer" ist also eine Funktion, die Kursbuchungen Ganzzahlen zuordnet. Die Interpretation in der konkreten Datenbank ist eine totale Funktion aus der Menge der tatsächlichen Entities in den Wertebereich des jeweiligen Datentyps: $\sigma(A) : \sigma(E) \rightarrow \mid D \mid$

Relationships



(a) zweistellig mit Attribut (b) funktional (c) mehrstellig, rekursiv, mit Rollennamen und Kardinalitäten (d) Vererbungsbeziehung

Abbildung 2.4: Beispiel-Relationships

Relationships verbinden Entitytypen und kommen in verschiedenen kombinierbaren Varianten vor: Sie können zwei- oder mehrstellig sein, Attribute tragen und Kardinalitäten und Rollennamen definieren, vgl. Abbildung 2.4. Grafisch werden sie durch Rauten mit Beziehungsnamen

notiert, die die beteiligten Entitytypen über Linien miteinander verbinden. Die Namen von Relationships beginnen in der Regel mit einem Kleinbuchstaben und werden gegebenenfalls in CamelCase geschrieben. In welcher Richtung eine Relationship zu lesen ist, also ob im Beispiel 2.4(a) ein Dozent einen Termin hält oder umgekehrt, ist nicht in der grafischen Notation enthalten. Das wird definitiv nur in der textuellen Notation deutlich, wo eine Reihenfolge angegeben ist. Im Beispiel wäre das `hält(DOZENT, TERMIN, Honorar)` was als "DOZENT hält TERMIN" zu lesen ist.

Beziehungsattribute Relationships können selbst auch Attribute tragen wie in Abbildung 2.4(a). Dieses Beispiel drückt aus, dass ein Dozent einen (Kurs-)Termin hält, also unterrichtet, und pro gehaltenem Termin ein Honorar gespeichert wird.

Kardinalitäten Kardinalitäten oder Vielfachheiten geben an, wie oft Entitytypen an einer Relationship beteiligt sein können. Sie werden wie in Abb. 2.4(c) duch Angabe der minimalen und maximalen Beteiligungsanzahl in der Form "(min, max)" direkt an der zugehörigen Verbindungslinie notiert, und zwar auf der Seite des Entitytyps, dessen Beteiligungszahl sie definieren. Dabei sind positive Zahlenwerte mit Null zugelassen und für max auch "*", das für "beliebig viele" steht. In Abb. 2.4(c) kann ein KONTAKT also maximal ein Mal als Kind in die Beziehung eingehen, aber beliebig oft als Wurzel oder Elternkontakt beteiligt sein. Wenn keine weiteren Angaben zur Kardinalität gemacht werden, gilt (0,*) als Standard, was keine Beschränkungen bedeutet.

Funktionale Beziehungen (FB) Partielle funktionale Beziehungen sind durch eine Pfeilspitze an einem Ende der Verbindungslinie gekennzeichnet wie in Abbildung 2.4(b). Solche Relationships definieren die Beziehung zwischen zwei Entitytypen als eine partielle Funktion in Pfeilrichtung, d.h. eine Entity vom Typ TERMIN steht mit *höchstens* einer Entity vom Typ KURS (auf den die Pfeilspitze zeigt) in Beziehung. Es wird also die Kardinalität (0,1) für TERMIN und (0,*) für KURS impliziert.

Totale funktionale Beziehungen kommen ebenfalls vor. Sie ordnen jedem A *genau ein* B zu und werden durch explizite Angabe der Kardinalität (1,1) bei A ausgedrückt.

Rekursive Relationships und Rollennamen Entitytypen können wie in Abbildung 2.4(c) auch rekursiv zu sich selbst in Beziehung stehen. Dabei können zur Unterscheidung Rollennamen angegeben werden, die die einzelnen Beteiligungen genauer spezifizieren und die Navigation über Namen ermöglichen.

Vererbung: IST-Relationships Vererbungsbeziehungen wie in Beispielabbildung 2.4(d) tragen immer den Namen "ist", der somit ein Schlüsselwort mit besonderer Semantik und kein einfacher Bezeichner ist, und eine Pfeilspitze zum allgemeineren Typ. Im Beispiel ist ein Dozent eine Spezialisierung von Person. Spezialisierungen erben alle Attribute ihrer Elterntypen und können zusätzliche Attribute tragen. Jeder Dozent trägt somit zusätzlich zum Stundensatz auch z.B. Name und Geburtsdatum wie eine Person, auch wenn diese Attribute nicht explizit angegeben sind. Insbesondere werden auch Schlüssel vererbt, d.h. wenn eine Person über die Kontaktnummer "KNr" identifiziert wird, dann gilt das auch für Dozenten.

2.3 Beschreibung des konzeptionellen Schemas

Neben der textuellen Notation aller Entitytypen und Relationships des ER-Schemas werden auch die wichtigsten Gründe für bestimmte Modellierungsentscheidungen erklärt, vor allem die Umsetzung spezieller Anforderungen der informellen Anwendungsbeschreibung. Auf Offensichtliches wie die Bedeutung des Attributs "Vorname" einer Person wird dabei aber nicht eingegangen.

2.3.1 Entitytypen

Bei der textuellen Notation stellen unterstrichene Attribute wieder Schlüsselattribute dar.

E1: `KONTAKT(KNr, Name, Notiz)`
Als Schlüssel wird eine Kontaktnummer vergeben, weil Firmen- oder Personennamen im Allgemeinen nicht eindeutig sind. Bei Personen ist mit Name der Nachname gemeint und bei Unternehmen die Firmenbezeichung. Freitextnotizen ermöglichen nähere Beschreibungen. Aufbau von Hierarchien wird über rekursive Relationships ermöglicht, die zu jedem Kontakt einen Eltern- und Wurzelkontakt enthalten können.

E2: `UNTERNEHMEN(Steuer#)`
Spezialisierung von KONTAKT, die juristische Personen abbildet und vor allem zur Unterscheidung zwischen Personen und Firmen dient. Kann später Grundlage zur Erfassung von Lieferanten oder Dienstleistern werden, z.B. wenn Catering durch Fremdfirmen durchgeführt wird.

E3: `PERSON(Anrede, Vorname, GebDatum)`
Spezialisierung von KONTAKT. In der Anrede ist auch das Geschlecht enthalten. Nur Personen können Kursteilnehmer oder Dozenten sein, deswegen wichtiger Entitytyp zur Differenzierung.

E4: `DOZENT(Stundensatz)`
Spezialisierung von PERSON. Trägt den Basis-Stundensatz und stellt Verbindung zu unterrichteten Themen her. Erbt transitiv auch von KONTAKT. Ob eine Person Dozent ist oder nicht könnte auch als boolesches Attribut zusammen mit dem Stundensatz in PERSON gespeichert werden - dann hätten aber alle nicht-Dozenten (und das ist die große Mehrheit) ein nutzloses Attribut, deswegen wird weiter spezialisiert. Ausserdem gehen Dozenten spezielle weitere Beziehungen ein, für die sie als Entitytyp vorliegen müssen.

E5: `KONTAKTWEG(KWID, Zusatz, Notiz, istStandard)`
KONTAKTWEG repräsentiert eine allgemeine Kontaktinformation und ist als eigener Entitytyp modelliert, weil Kontakte diese Angaben mehrfach oder gar nicht haben können und alle spezielleren Kontaktinformationen TELEFON, EMAIL und ANSCHRIFT gewisse Attribute gemeinsam haben. Die Spezialisierungen sind eine vollständige, überlappungsfreie Überdeckung, d.h. jede Kontaktinformation ist entweder Telefon, Email oder Anschrift. Der "Zusatz" gibt nähere Informationen über die Art bzw. die Erreichbarkeit eines Eintrags, z.B. "tagsüber", "abends", "privat" (vlg. Wertelisten-Modellierung S. 10). Die Markierung in "istStandard" vermeidet Mehrdeutigkeiten z.B. bei automatischen Serienbriefen.

E6: `TELEFON(Typ, Nummer)`
Spezialisierung von KONTAKTWEG. "Typ" unterscheidet Festnetz, Handy und Fax.

E7: `EMAIL(Adresse)`
Spezialisierung von KONTAKTWEG.

E8: `ANSCHRIFT(Strasse, HausNr, PLZ, Ort)`
Spezialisierung von KONTAKTWEG. Zu einer vollständigen Anschrift fehlt noch der Name des Kontaktes, zu dem sie gehört. Der Name wird nicht gespeichert, damit Namensänderungen ausschließlich beim Kontakt ausgeführt werden müssen, und um Redundanz zu vermeiden. Außerdem entspricht das der Definition eines Kontaktes als namentlich bekannter Einheit - ein anderer Name bedingt einen anderen Kontakt. Zu einem Kontakt gehörige Anschriften mit abweichenden Namen können als untergeordnete Kontakte realisiert werden.
PLZ und Ort sind hier aus Übersichtsgründen als Attribute von Anschrift modelliert; bei einer realen Umsetzung würden sie einen eigenen Entitytyp bilden (vlg. Wertelisten-Modellierung S. 10), weil der Ort funktional abhängig von der PLZ ist. In der Anschrift würde dann die PLZ als Fremdschlüssel erscheinen. Dadurch würden Redundanzen und Anomalien wie z.B. verschiedene Schreibweisen (Tippfehler!) des selben Ortes vermieden werden.

E9: `KONZEPT(Bezeichnung, Ver#, Beschreibung, Dauer)`
Bezeichnung und Version bilden einen zusammengesetzten Schlüssel. Dadurch können kleine Änderungen an einem Konzept vorgenommen werden, die keinen neuen Namen rechtfertigen, aber auch nicht an vergangene Kurse weitergereicht werden dürfen (Dokumentationsfunktion).

E10: `KURS(Kurs#, minTN, maxTN, PreisTN, Kosten, Status)`
Bei konkreten Kursen ergeben räumliche und kalkulatorische Bedingungen eine minimale und maximale Teilnehmerzahl. Auch die kalkulierten Kosten werden festgehalten; der Endpreis pro TN basiert darauf, enthält aber auch Gewinnspannen etc, und dient als Angebotsgrundlage. Der endgültige TN-Preis wird dann pro BUCHUNG gespeichert.

E11: `BUCHUNG(Buchungs#, Preis, #TN, RechAnschrift)`
Die Rechnungsanschrift wird zur Vereinfachung (und weil diese Adresse nicht weiter ausgewertet/verwendet wird) als einfaches Attribut angegeben. Realisierung als Refernz zum Entitityp "Anschrift" denkbar, verkompliziert jedoch alles unnötig: Eine Anschrift ist ohne zugehörigen Kontakt unvollständig (Name fehlt). Eine Rechnungsanschrift mit abweichendem Namen würde dadurch einen neuen Kontakt erfordern. Zur Erfüllung der Dokumentationsfunktion müsste dann bei Anschriftsänderungen immer geprüft werden, ob es sich um eine verwendete Rechnungsanschrift handelt, die nicht mehr geändert werden darf.
Nachteil der verwendeten Modellierung ist mögliche Redundanz, wenn viele Buchungen unter der selbe Rechnungsadresse laufen. Wenn sich das in der Praxis als relevanter Fall herausstellen sollte, kann ein Entitytyp "Rechnungsadresse" eingeführt werden, der von BUCHUNG referenziert wird. Für Entities dieses Typs würde dann ein Änderungsverbot nach Abschluss des Gesamtvorgangs gelten.

E12: `TERMIN(TID, von, bis, Ort)`

Definiert eine Zeitspanne ("von" und "bis" jeweils mit Datum, deckt so auch z.B. mehr- und ganztägige Exkursionen ab) und einen Ort, zu dem Dozenten eine Veranstaltung eines Kurses abhalten. Der Ort kann ein Raum im Gebäude des Anbieters oder beim Kunden sein oder extern liegen und würde deswegen eigentlich komplexer modelliert werden, zur Veranschaulichung genügt diese Variante aber. TERMIN ist eigener Entitytyp, weil Kurse an vielen Terminen stattfinden können und jeder Termin prinzipiell von mehreren Dozenten gleichzeitig betreut werden kann (Laborübung, Exkursion). Generischer Schlüssel obwohl {von, bis, Ort} eindeutig wäre, damit bei Referenzen nicht die eigentliche Information redundant gespeichert wird.
Kompliziertere Varianten für regelmäßige Termine wären denkbar (z.B. mit Angaben zu Intervallen, Wiederholungen u.ä.), aber diese Modellierung deckt alle möglichen Fälle auf einfache Art ab. Zu jedem Kurs wird dann eine Liste mit solchen Terminen gespeichert. Das ist zwar in vielen Fällen speicherintensiver als die komplexeren Varianten, dafür ist das Schema aber sehr einfach und unmissverständlich und kann ohne weiteres alle Sonderfälle abdecken (Benutzerfreundlichkeit z.B. bei der Eingabe von regelmäßigen Terminen muss dann die jeweilige Anwendung gewährleisten).

E13: `THEMA(Bezeichnung)`
Stellt einen Katalog von definierten Themen bereit, auf die sich Dozenten und Kurskonzepte beziehen können. Vor allem als Suchbegriffe nützlich (Vermeidung von verschiedenen Schreibweisen für die selbe Sache). Über rekursive Relationships sind Verwandschaften und Hierarchien analog zu Kontakthierarchien möglich (vgl. Abschnitt 1.2.3 S. 6).

E14: `GESCHAEFTSVORFALL(ID, DatumZeit, Notiz, WiederVorl, istAbgeschl)`
Generische ID als Schlüssel, weil selbst Menge aller Attribute nicht unbedingt eindeutig ist (Gleichzeitigkeit kann vorkommen) und sich ihre Werte durchaus ändern können. Mit `DatumZeit` ist kein Datentyp gemeint, sondern die Tatsache, dass das Attribut Datum und Uhrzeit des Vorfalls enthält. Die "Wiedervorlage" enthält den Zeitpunkt, an dem der Vorfall dem Benutzer wieder vorgelegt wird.

E15: `GVTYP(Bezeichnung, BetrKontakt, BetrBuchung, BetrKurs, BetrKonzept)`
Definiert die möglichen Typen von Geschäftsvorfällen (in der Art der Wertelisten-Modellierung, vgl. S. 10), und auf wie viele Entities der drei Möglichen Bezugs-Entitytypen KONTAKT, BUCHUNG und KURS sich ein Geschäftsvorfall jeweils bezieht. Ermöglicht völlig flexible Definition beliebiger GV-Typen, die sich auf beliebige Kombinationen von KURS, KONTAKT und BUCHUNG-Entities beziehen können.

2.3.2 Relationships

Wenn im ER-Schema Rollennamen definiert wurden, werden diese zur eindeutigen Identifizierung vor den eigentlichen Entitytypen notiert. Kardinalitäten ausser (0,*) werden explizit (in eckigen Klammern) hinter dem Entitytyp angegeben, den sie einschränken.

R1: `istUnterKontakt(Kind:KONTAKT[0,1], Elternkontakt:KONTAKT, Wurzel:KONTAKT)`
Bei baumförmigen Hierarchien (vgl. Abbildung 1.1 auf Seite 6) kann ein Kontakt höchstens ein Mal als Unterkontakt eines anderen auftreten, deswegen (0,1)-Kardinalität der Kind-Rolle. Er hat dann auf jeden Fall einen Elternkontakt und einen Wurzelkontakt (die aber gleich sein können). Ein Kontakt kann Elternkontakt und/oder Wurzel beliebig vieler Kontakte sein. Das explizite Speichern der Wurzel ermöglicht effizientes Auffinden aller zu

einer Hierarchie gehörenden Kontakte (sonst müsste man bei jedem Kontakt erst bis zur Wurzel aufsteigen um herauszufinden, ob er zur Hierarchie gehört).
Alternativ könnte man das z.B. durch zwei rekursive funktionalen Beziehungen modellieren: `hatWurzel( Kind:KONTAKT[0,1], Wurzel:KONTAKT)` und `istKindVon( Kind:KONTAKT[0,1], Elternkontakt:KONTAKT)`. Diese müsste man dann aber mit weiteren Integritätsbedingungen absichern (*wenn* ein Kontakt Kindkontakt ist, *muss* er auch eine Wurzel haben usw.), die in der dreistelligen Relationship bereits enthalten sind (implizit dadurch, dass Einträge immer drei Kontakte umfassen und explizit durch die Kardinalitäten).

R2: `erreichbar(KONTAKT, KONTAKTWEG[1,1])`
Liste von Kontaktinformationen (Spezialisierungen von Kontaktweg), über die ein Kontakt verfügbar ist; totale funktionale Beziehung, jede Kontaktinformation gehört zu genau einem Kontakt.

R3: `gebuchtVon(BUCHUNG[1,1], KONTAK)`
Totale funktionale Beziehung: Jede Buchung wird genau einem Kontakt zugeordnet. Heisst für die Praxis, dass eine Buchung nicht vor dem betreffenden Kontakt angelegt werden kann.

R4: `für(BUCHUNG[1,1], KURS)`
Totale funktionale Beziehung: Jede BUCHUNG gilt "für" genau einen Kurs. Stellt Eindeutigkeit zwischen Teilnehmern, Kursen und Zahlern her.

R5: `gebuchtFuer(PERSON, BUCHUNG)`
Ohne Einschränkung der Vielfachheiten, eine Person kann Teil beliebig vieler Buchungen sein und eine Buchung kann beliebig viele Personen umfassen, insbesondere auch überhaupt keine (bei anonymen Teilnehmern).

R6: `realisiert(KURS[1,1], KONZEPT)`
Totale funktionale Beziehung: Jeder Kurs realisiert genau ein Konzept (ein Konzept kann aber von beliebig vielen Kursen realisiert werden).

R7: `behandelt(KONZEPT, THEMA)`
Keine Einschränkungen: Ein Konzept kann beliebige Themen behandeln, auch gar kein spezielles (die Inhalte ergeben sich dann nur aus der Konzeptbeschreibung). So können aus Konzepten auch erst später spezielle Themen als Schlagworte in den Katalog aufgenommen werden. Die behandelten Themen sind ja als Such- und Klassifikationshilfe gedacht, nicht als harte Systematik.

R8: `unterrichtet(DOZENT, THEMA)`
Ohne Einschränkungen: Als Planungshilfe hat ein Dozent eine Liste mit Unterrichtsthemen, die aber auch leer sein kann.

R9: `behandelt(TERMIN[0,1], KURS)`
Partielle funktionale Beziehung: Ein Termin "behandelt" höchstens einen Kurs, d.h. zu einer Zeit an einem Ort wird nur höchstens ein einziger Kurs durchgeführt. Name auch so gewählt, dass die Lesrichtung der Funktions(Pfeil)richtung entspricht.

R10: `betreut(DOZENT, TERMIN, Honorar)`
Das Honorar wird für jeden einzelnen Termin festgehalten, den ein Dozent hält. So lassen sich Ausfälle, Verschiebungen, Vertretungen etc. flexibel handhaben und abrechnen. Ein Dozent kann dabei beliebig viele Termine betreuen, und jeder Termin kann von beliebig vielen Dozenten betreut werden. Weil Termine auch "freie Slots", also nicht durch Kurse belegt sein können, muss zu einem Termin nicht zwingend ein Dozent vorhanden sein.

R11: `istVerwandt(a:THEMA, b:THEMA)`
Beispiel für rekursive Beziehung ohne Einschränkung der Vielfachheit. Themen können untereinander beliebig als verwandt gekennzeichnet werden. Generische Rollennamen a und b zur eindeutigen Navigation. Drückt aus, dass die zwei Themen a und b verwandt sind, um später auch nach "ähnlichen" Themen suchen zu können. Wenn später auch Themenhierarchien nötig werden (z.B. für Kategoriezuordnungen o.ä.) kann das nach dem Schema der Kontakthierarchien mit einer separaten dreistelligen Relationship umgesetzt werden.

R12: `betrifft(GESCHAEFTSVORFALL, KONTAKT)`
R13: `betrifft(GESCHAEFTSVORFALL, BUCHUNG)`
R14: `betrifft(GESCHAEFTSVORFALL, KURS)`
Relationships, die die Beziehung zwischen Geschäftsvorfällen und den Entities herstellen, auf die sich die GV beziehen. Standardmultiplizität; auf welche Entitytypen und wie viele Entities sich ein konkreter GV bezieht, ist in seinem Typ definiert und muss ggf. über Integritätsbedingungen abgesichert werden.

R15: `hat(GESCHAEFTSVORFALL[1,1], GVTYP)`
Totale funktionale Beziehung, jeder Geschäftsvorfall hat genau einen Typ.

R16: `ist(TELEFON, KONTAKTWEG)`
R17: `ist(EMAIL, KONTAKTWEG)`
R18: `ist(ANSCHRIFT, KONTAKTWEG)`
Telefon, Email und Anschrift erben von ihrer Generalisierung KONTAKTWEG.

R19: `ist(UNTERNEHMEN, KONTAKT)`
R20: `ist(PERSON, KONTAKT)`
R21: `ist(DOZENT, PERSON)`
Unternehmen und Personen sind spezielle Kontakte. Dozenten sind (transitiv) ebenfalls Kontakte.

3 Integritätsbedingungen

Integritätsbedingungen (IB) sind Anforderungen an gültige Datenbankzustände, sie schränken also die Menge der möglichen gültigen Zustände ein. Man kann sie auch umgekehrt als Zusicherungen verstehen, die in allen gültigen DB-Zuständen erfüllt sind. Sie werden u.A. gebraucht, damit das System keine Zustände zulässt, die im modellierten Weltausschnitt nicht möglich sind. Im eingangs dargestellten ER-Diagramm sind bereits zwei wichtige Klassen von Integritätsbedingungen enthalten, nämlich Schlüsselattribute und Kardinalitäten (auch implizit durch funktionale Beziehungen). Andere Integritätsbedingungen ergeben sich aus der informellen Anwendungsbeschreibung und müssen noch explizit formuliert werden.

3.1 Schlüsselattribute

Alle gültigen DB-Zustände müssen die Bedingung erfüllen, dass sich unterschiedliche Tupel in den Werten ihrer Schlüsselattribute unterscheiden oder umgekehrt formuliert: Es kann nicht zwei verschiedene Tupel geben, die die selben Werte in ihren Schlüsselattributen tragen. Formal lässt sich das so fassen:

$\forall e1, e2 \in \sigma(E) : [e1 \neq e2] \Rightarrow [\exists i \in \{1, \ldots, k\} : \sigma(k_i)(e1) \neq \sigma(k_i)(e2)]$

In relationaler Algebra lässt sich diese Bedingung zum Beispiel dadurch ausdrücken, dass die Zahl der Elemente bei der Projektion auf die Menge der Schlüsselattribute genau der Anzahl der Elemente der ursprünglichen Relation entsprechen muss. Denn Duplikate fallen bei der Projektion weg; wenn also die Schlüsselbedingung verletzt wäre, hätte das Ergebnis der Projektion weniger Elemente (die Betragsstriche sollen für die Anzahl der Elemente einer Menge stehen)

$|\pi_{k_1,\ldots,k_n}(R)| = |R|$ mit $\{k_1, \ldots, k_n\}$ Menge der Schlüsselattribute von R

3.2 Referentielle Integrität

Darunter ist zu verstehen, dass Fremdschlüssel nur Werte annehmen dürfen, die in der Zielrelation, auf die sie sich beziehen, auch tatsächlich vorkommen. Das ist intuitiv verständlich, denn man kann sich nur auf etwas beziehen, das auch existiert. Zum Beispiel dürfen Dozenten nicht für die Betreuung von nicht-existenten Terminen eingetragen werden und umgekehrt. In RA ausgedrückt lässt sich das Beispiel so fassen, dass die Projektion auf den Fremdschlüssel eine Teilmenge der Projektion auf dieses Attribut in der Zielrelation ist:

$\pi_{KNr}(betreut) \subseteq \pi_{KNr}(DOZENT)$

3.3 Kardinalitäten

Entities dürfen nur in den angegebenen Vielfachheiten an Beziehungen teilnehmen. Formal ist diese IB für binäre Beziehungen `R(E1[l,h],E2)` (aus [GK08])

- ganz allgemein:
  $\forall e_1 \in \sigma(E1) : l \leq \left|\{(e_1, e_2) | e_2 \in \sigma(E2), (e_1, e_2) \in \sigma(R)\}\right| \leq h$

- für partielle funktionale Beziehungen (l=0, h=1):
  $\forall e_1 \in \sigma(E1) : \left|\{(e_1, e_2) | e_2 \in \sigma(E2), (e_1, e_2) \in \sigma(R)\}\right| \leq 1$
- für totale funktionale Beziehungen (l=h=1):
  $\forall e_1 \in \sigma(E1) : \left|\{(e_1, e_2) | e_2 \in \sigma(E2), (e_1, e_2) \in \sigma(R)\}\right| = 1$

IB des vorliegenden Entwurfs, die sich nach diesem Schema aus den Kardinalitäten ergeben, sind z.B.:

- Eine Buchung enthält immer genau einen KONTAKT und genau einen KURS.
- Ein Kurs referenziert immer genau ein Konzept.
- Ein Kontaktweg referenziert immer genau einen Kontakt
- Ein Termin befasst sich mit höchstens einem Kurs
- ...

3.4 Andere

Die IB durch (Fremd-)Schlüssel und Kardinalitäten sichern zu, dass jederzeit fehlerfrei über die Daten navigiert werden kann (alle Schlüssel sind eindeutig, alle Referenzen gültig). Neben diesen "technischen" IB gibt es aber noch sehr viel mehr "inhaltliche" IB, die durch die Logik des modellierten Weltausschnitts bzw. der Geschäftslogik vorgegeben werden. Einige werden im Folgenden beispielhaft formuliert - für eine möglichst vollständige Menge von IB müsste eine viel genauere Analyse der Geschäftslogik erfolgen.

- Der Startzeitpunkt jedes Termins muss vor seinem Ende liegen:
  $\forall t \in \sigma(Termin) : t.von \leq t.bis$
- An einem Ort können nicht zwei Termine gleichzeitig (incl. Überlappung) stattfinden. Anders ausgedrückt: Für jedes Paar von unterschiedlichen Terminen, die am selben Ort stattfinden, muss ein Termin spätestens dann enden, wenn der andere anfängt (wenn einer genau dann startet, wenn der andere endet, ist alles in Ordnung).
  $\forall t_1, t_2 \in \sigma(TERMIN) | [t_1 \neq t_2] \wedge [(t_1.Ort = t_2.Ort] : t_2.von < t_1.bis \rightarrow t_2.bis \leq t_1.von$
- Ein Dozent kann nicht zwei gleichzeitige oder überlappende Termine betreuen.
  $\forall d \in \sigma(DOZENT), t_1, t_2 \in \sigma(TERMIN) | [t_1 \neq t_2] \wedge [(t_1, d), (t_2, d) \in \sigma(betreut)] : t_2.von < t_1.bis \rightarrow t_2.bis \leq t_1.von$
- Der Studensatz eines Dozenten und das Honorar, für das ein Dozent einen Termin betreut, dürfen nicht negativ sein:
  $\forall d \in \sigma(DOZENT) : d.Stundensatz > 0$
  $\forall b \in \sigma(betreut) : b.Honorar > 0$
- Die maximale Teilnehmerzahl eines Kurses muss mindestens so groß sein wie die minimale, und beide dürfen nicht negativ sein:
  $\forall k \in \sigma(KURS) : [k.minTN \geq 0] \wedge [k.maxTN \geq k.minTN]$

- Ein Konzept darf keine negative Dauer definieren:
  $\forall k \in \sigma(KONZEPT) : k.Dauer \geq 0$

- Ein Kontakt darf nicht Firma und Person gleichzeitig sein (lässt sich umgekehrt äquivalent über alle Unternehmen statt Personen formulieren):
  $\forall p \in \sigma(PERSON) : \nexists u \in \sigma(UNTERNEHMEN) : u.KNr = p.KNr$

  In RA lässt sich das sehr einfach über die Schnittmenge der Kontaktnummern ausdrücken:
  $\pi_{KNr}(PERSON) \cap \pi_{KNr}(UNTERNEHMEN) = \emptyset$
  Oder noch einfacher über den natürlichen Verbund (der hier ja über die KNr hergestellt wird):
  $PERSON * UNTERNEHMEN = \emptyset$

- Ein Wurzelkontakt muss eine Person oder Firma sein (weil Kunden als Wurzelkontakte von Buchungen definiert sind und Wurzelkontakte deswegen geschäftsfähige Entitäten sein müssen, mit denen z.B. Verträge abgeschlossen werden können). $\forall w \in \sigma(Wurzel)(istUnterkontakt) :$
  $[\exists p \in \sigma(Person) : p.KNr = w.KNr)] \vee [\exists u \in \sigma(Unternehmen) : u.KNr = w.KNr)]$

3.5 Transaktionen

Die Grundoperationen auf einzelnen Tupeln (Anlegen, Ändern, Löschen) sind atomare Operationen, d.h. ihre Ausführung ist nur als Ganzes möglich, sie werden entweder vollständig oder gar nicht erfolgreich ausgeführt. Viele Benutzeroperationen umfassen allerdings Sequenzen von Grundoperationen, die alle erfolgreich ausgeführt werden müssen. Dabei kann es vor allem in Mehrbenutzersystemen (was ja der Normalfall ist) zu unterschiedlichsten Problemen kommen, wenn sich die Anweisungssequenzen mehrerer Benutzer überschneiden oder durch Fehler einzelnen Anweisungen nicht ausgeführt werden.

Diese Probleme können hier aus Platzgründen nicht ausführlicher beschrieben werden, vgl. dazu [Kle06, Kap. 10]. Als einfaches Beispiel kann man sich eine Überweisung von einem Girokonto A auf ein zweites B vorstellen: Von Konto A muss der Betrag abgezogen werden, und auf Konto B muss er gutgeschrieben werden. Das sind zwei atomare Operationen, aber nur die Ausführung beider Operationen (oder keiner von beiden) liefert wieder einen gültigen DB-Zustand. Offensichtlich darf auf keinen Fall nur eine Operation ausgeführt werden.

Um diese Probleme zu lösen bieten viele DBMS sogenannte Transaktionen (TA). Darunter versteht man die Möglichkeit, mehrere atomare Operationen praktisch zu einer einzigen, größeren Operation zusammenzufassen, die dann atomar ausgeführt wird. Transaktionen sollten das ACID-Prinzip umsetzen:

Atomicity - eine TA wird entweder komplett erfolgreich ausgeführt oder gar nicht.
Consistency - es wird von einem DB-Zustand ausgegangen, in dem alle IB erfüllt sind. Nach der TA sind wieder alle IB erfüllt.
Isolation - es wird garantiert, dass die Zwischenzustände während einer TA nicht von anderern TA gelesen oder verändert werden können.
Durability - die Ergebnisse einer TA sind dauerhaft und können nur von einer anderen TA verändert werden.

Auch im Anwendungssystem des Schulungsanbieters gibt es Kandidaten für mögliche Transaktionen. Beim Neuerfassen eines Dozenten muss z.B. immer auch ein Kontakt und eine Person angelegt werden (die ja die allgemeineren Informationen tragen), damit alle IB erfüllt sind.

4 Realisierung als relationales Schema

4.1 Standardverfahren

Das Standardverfahren zur Übersetzung des konzeptionellen DB-Schemas in ein relationales DB-Schema umfasst folgende Schritte:

- Aus jedem Entitytyp wird ein Relationenschema (RS, auch "relationales Schema")
- Aus jeder Relationship wird ein RS, und für jeden "Arm" werden Schlüsselattribute der referenzierten Entitytypen übernommen, die sinnvoll und eindeutig umbenannt werden können.
- Optimierung der entstandenen Relationenschemata durch Zusammenfassen, Streichen, Umbenennen

Der letzte Schritt ist dabei nicht genau definiert, die Kriterien für die Optimierung sind anwendungsabhängig und können unter anderem die Reduzierung von Redundanz, Minimierung des Speicherbedarfs oder die Optimierung für schnelle Abfragen zum Ziel haben. Umbenennungen sind manchmal zur Herstellung von Eindeutigkeit notwendig und bietet sich auch oft an, um vor allem Fremdschlüssel-Attribute aussagekräftiger zu benennen. Natürliche Verbünde operieren allerdings auf gleichbenannten Attributen, was bei einer Umbenennung immer zu beachten ist.

Es ist zu beachten, dass funktionale Beziehungen und Vererbungsbeziehungen dabei wie im Folgenden beschrieben gesondert behandelt werden.

Funktionale Beziehungen Weil funktionale Beziehungen (FB) wie in Abb. 4.1 jedem A immer nur höchstens ein B zuordnen, können sie ohne eigenes Relationenschema als Fremdschlüssel im Relationenschema A umgesetzt werden:

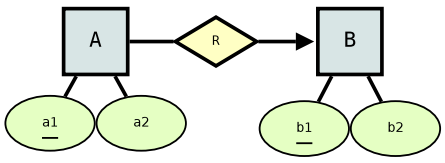


Abbildung 4.1: allgemeine funktionale Beziehung

```
A(a1, a2, b1)
B(b1, b2)
```

Dabei ist in der Praxis zu berücksichtigen, dass bei dieser Umsetzung bei Tupeln aus A, die kein B referenzieren, NULL-Werte im Fremdschlüssel `b1` auftreten (bei partiellen FB; bei totalen FB muss das Fremdschlüsselattribut per Definition einen Wert haben, NULL ist unzulässig). Das ist erlaubt, kann aber unerwünscht sein. Will man NULL-Werte ganz vermeiden muss die Beziehung über ein eigenes Relationenschema `R(a1, b1)` ausgedrückt werden. Der Schlüssel sorgt dabei dafür, dass ein A höchstens ein B referenzieren kann.

Vererbungsbeziehungen werden ebenfalls nach einem festen Schema umgesetzt:

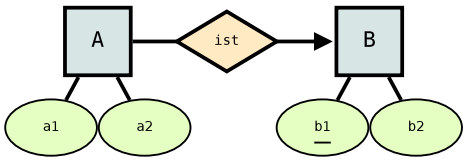


Abbildung 4.2: allgemeine Vererbungsbeziehung

Vererbungsbeziehungen wie in Abb. 4.2 werden umgesetzt, indem das Relationenschema der Spezialisierung das Schlüsselattribut der Generalisierung übernimmt und vererbte Attribute bei der Generalisierung bleiben:

```
A(a1, a2, b1)
B(b1, b2)
```

Dabei muss die interrelationale Integritätsbedingung gelten, dass alle Werte von `b1` in `A` auch in `B` vorkommen müssen: $\pi_{b1}(A) \subseteq \pi_{b1}(B)$

Bemerkung: Es gibt auch andere Möglichkeiten, Vererbungsbeziehungen umzusetzen. Dabei spielen vor allem Überlegungen zum effizienten Zugriff eine Rolle, und ob explizit zwischen vollständigen und teilweisen Überdeckungen mit und ohne Überlappung unterschieden werden soll. Eine Übersicht der Methoden findet sich in [JQL05] und eine ausführliche Beschreibung mit Details zu Überlappung und Überdeckung z.B. in [Ste06, Kap. 3.1.3]

4.2 Das relationale DB-Schema

Durch die Anwendung des Standardverfahrens und die spezielle Behandlung der funktionalen und Vererbungsbeziehungen ergibt sich aus dem konzeptionellen Schema das in diesem Abschnitt beschriebene relationale Datenbankschema (=die Menge aller Relationenschemata; durch die Uneindeutigkeit von Schritt 3 des Standardverfahrens ist dies nur eines von vielen möglichen):

4.2.1 Übersetzung der Entitytypen

RS1: `KONTAKT(Knr, Name, Notiz, WurzelKNr, ElternKNr)`
Die Relationship `istUnterkontaktVon` wird in dieses RS integriert, um effizient Kunden zu identifizieren (das sind Wurzelkontakte von Kontakten, die Buchungen beauftragt haben). Jeder Kontakt trägt über Fremdschlüssel Informationen zu seinem Eltern- und Wurzelkontakt. Dabei definieren wir, dass ein Kontakt ohne Elternkontakt sich selbst in beiden Feldern referenziert (totale funktionale Beziehung) - das ist leicht redundant, vereinfacht aber die Handhabung von Hierarchien- und Kundenbeziehungen (über Wurzel definiert) sehr.

RS2: `UNTERNEHMEN(Steuer#, KNr)`
RS3: `PERSON(Anrede, Vorname, GebDatum, KNr)`
RS4: `DOZENT(Stundensatz, KNr)`
Die Spezialisierungen von `KONTAKT` bekommen wie beschrieben das Schlüsselattribut ihres allgemeineren RS; weil ein Dozent nicht nur ein spezieller Kontakt sondern eine spezielle Person ist muss die zusätzliche IB gelten, dass es zu jedem Dozenten auch eine Person

geben muss: $\pi_{KNr}(DOZENT) \subseteq \pi_{KNr}(PERSON)$

RS5: KONTAKTWEG(KWID, Zusatz, Notiz, istStandard, *KNr)*
Die totale funktionale Beziehung erreichbar(KONTAKT, KONTAKTWEG) wird als Fremdschlüsselattribut KNr umgesetzt.

RS6: TELEFON(Typ, Nummer, KWID)
RS7: EMAIL(Adresse, KWID)
RS8: ANSCHRIFT(Strasse, HausNr, PLZ, Ort, KWID)
Die Spezialisierungen von KONTAKTWEG erhalten wie eingangs beschrieben das Schlüsselattribut des allgemeineren Relationenschemas, hier KWID.

RS9: KONZEPT(Bezeichnung, Vers#, Beschreibung, Dauer)
RS10: KURS(Kurs#, minTN, maxTN, PreisTN, Kosten, Status, *Bezeichnung*, *Vers#*)
Die Schlüsselattribute des realisierten Konzeptes werden als Fremdschlüssel integriert (totale funktionale Beziehung).

RS11: BUCHUNG(Buchungs#, Preis, #TN, RechnAnschrift, *KNr*, *Kurs#*)
Die Relationships gebuchtVon(BUCHUNG, KONTAKT) und für(BUCHUNG, KURS) werden wegen der (1,1)-Kardinalität als Fremdschlüsselattribute KNr und Kurs# in BUCHUNG integriert, die außerdem nicht NULL sein dürfen.

RS12: TERMIN(TID, von, bis, Ort, *Kurs#)*
Die funktionale Beziehung behandelt(TERMIN, KURS) wird durch die Einbindung der Nummer des referenzierten Kurses als Fremdschlüssel Kurs# in TERMIN umgesetzt. Um redundantes Speichern der Termindaten bei Referenzen auf einen Termin zu vermeiden, wird eine generische ID als Schlüssel verwendet statt der Attributemenge {von, bis, Ort}.

RS13: THEMA(Bezeichnung)

RS14: GVTYP(Bezeichnung, #BetrKontakt, #BetrBuchung, #BetrKurs) unverändert übernommen

RS15: GESCHAEFTSVORFALL (GVID, DatumZeit, Notiz, WiederVorl, istAbgeschl, *Bezeichnung*)
Die funktionale Beziehung zu GVTYP wird Fremdschlüssel (mit zusätzlicher IB "not NULL", weil es eine totale FB ist).

4.2.2 Übersetzung der Relationships

RS16: gebuchtFuer(*KNr*, *Buchungs#*)
Realisiert die Zuordnungsliste von Personen zu Kursen, an denen sie teilnehmen.
RS17: betreut(*DozentKNr*, *TerminTID*, Honorar)
Wird nach Standardverfahren mit den Schlüsseln der referenzierten RS und dem Relationship-Attribut Honorar gebildet. Weil ein Dozent einen Termin nur ein mal halten kann, ist diese Attributmenge Schlüssel.
RS18: unterrichtet(*DozentKNr*, *ThemaBezeichnung*)

Dei (0,*)-Kardinalität auf beiden Seiten ergibt ein RS, das nur aus den Schlüsselattributen der referenzierten RS besteht.

RS19: `istVerwandt(`*Thema1, Thema2*`)`

Wie bei `unterrichtet`.

RS20: `behandelt(`*KonzeptBezeichnung, Vers#, ThemaBezeichnung*`)`

Weil in beiden beteiligten RS ein Schlüsselattribut "Bezeichnung" vorkommt, *müssen* die Fremdschlüsselattribute des RS zur Relationship `behandelt(KONZEPT, THEMA)` eindeutig umbenannt werden. Beide Schlüsselattribute von THEMA werden referenziert.

RS21: `betrifftKONTAKT(`*GVID, KontaktKNr*`)`

RS22: `betrifftBUCHUNG(`*GVID, Buchungs#*`)`

RS23: `betrifftKURS(`*GVID, Kurs#*`)`

Die RS wurden eindeutig umbenannt.

5 Sichten

Sichten (engl. Views) definieren über benannte Anfragen sozusagen dynamische, anwendungsspezifische Schemata, die auf den tatsächlichen Schemata beruhen und diese z.B. durch Auswertung, Umgruppieren oder Zusammefassung auf die Informationsbedarfe verschiedener Anwendungen anpassen. Sie können damit niemals mehr Informationen enthalten als die Schemata, auf denen sie beruhen, aber sehr wohl Informationen bereitstellen, die erst durch Auswertung/Kombination der Basisinformationen gewonnen werden können.

Sichten sind besonders dann nützlich, wenn regelmäßig benötigte Informationen erst durch Auswertunge oder Verknüpfung diverser Relationen gewonnen werden müssen. Als Sicht definiert, kann man diese Informationen unter einem Namen abrufen und prinzipiell wie "normale" RS behandeln [1].

Hier sollen einige konkrete Beispiele für das entworfenen Schema gegeben werden [2]. Den Namen von Sichten ist zur deutlichen Unterscheidung von den RS ein "V" für "View" vorangestellt.

Personen Eine Liste aller natürlichen Personen könnte nützlich sein, die die Informationen aus PERSON und KONTAKT zusammenführt und als eine Relation liefert.

```
CREATE VIEW "VPersonen" AS
   SELECT *
   FROM "PERSON" NATURAL JOIN "KONTAKT";
```

Der natürliche Verbund sorgt dafür, dass in der Ergebnisrelation Name und Notiz aus KONTAKT ihrer jeweiligen PERSON zugeordnet werden (über das gemeinsame Attribut KNr), und dass keine KNr enthalten sind, die nicht in PERSON enthalten sind. Der natürliche Verbund ist dabei nur eine Kurzschreibweise für

```
SELECT *
   FROM "PERSON" p, "KONTAKT" k
   WHERE p."KNr" = k."KNr";
```

Die Reihenfolge der Attribute in der Ergebnisrelation ergibt sich durch "Hintereinanderhängen" der Attribute der Einzelnen RS. Wenn man die Reihenfolge ändern möchte, um z.B. erst Anrede, dann Vor- und dann Nachnamen zu erhalten, kann man die Attribute und ihre Reihenfolge explizit im SELECT-Teil der Anfrage festlegen [3]:

```
SELECT p."KNr", p."Anrede", p."Vorname", k."Name", k."Notiz"
FROM "PERSON" p NATURAL JOIN "KONTAKT" k;
```

[1] zumindest beim Lesen von Informationen; Datenänderungen in Sichten sind nur möglich, wenn das DBMS ein eindeutiges umgekehrtes Mapping der View-Attribute auf Attribute der zugrunde liegenden RS herstellen kann . Manche DBMS, z.B. Oracle, bieten sog. "Instead of Trigger" an, mit denen Update-Prozeduren von Hand definiert werden können, wenn kein eindeutiges Mapping möglich ist. Vgl. zu diesem Thema z.B. `http://download.oracle.com/docs/cd/B19306_01/appdev.102/b14251/adfns_triggers.htm#i1025919`

[2] Eventuelle Abweichungen in der SQL-Syntax vom Vorlesungsstandard, z.B. Anführungszeichen um RS-Namen, kommen daher, dass die Anfragen in einer realen PostgreSQL-DB entwickelt wurden

[3] in realen Anwendungen wird das sogar empfohlen, damit sich Ergebnisse von Anfragen oder Views nicht versehentlich ändern, wenn z.B. Attribute in RS dazukommen

Kursteilnehmer Die Liste der Personen kann man für eine Liste aller Kursteilnehmern einsetzen (aus der dann mit einer Standardabfrage z.B. die Teilnehmerliste eines bestimmten Kurses erzeugt werden kann):

```
CREATE VIEW "VKursteilnehmer" AS
   SELECT p."KNr", p."Vorname", p."Name", k."Bezeichnung", k."Vers#"
   FROM "VPersonen" p, "gebuchtFuer" f, "BUCHUNG" b, "KURS" k
   WHERE
      p."KNr" = f."KNr" AND f."Buchungs#" = b."Buchungs#"
      AND b."Kurs#" = k."Kurs#";
```

Zwischen `gebuchtFuer` und `BUCHUNG` darf kein `NATURAL JOIN` erfolgen, weil beide RS ein Attribut "KNr" haben, das aber verschiedene Bedeutungen hat (die Verknüpfung darf nur über die Buchungsnummer erfolgen). Deswegen sind alle Vergleichskriterien explizit aufgeführt.

Kurskatalog Ein Kurskatalog, der die Kurse im Status "buchbar" oder "nach Vereinbarung" ("nV") enthält, z.B. für die Veröffentlichung auf der Homepage des Anbieters[4]:

```
CREATE VIEW "VKurskatalog" AS
 SELECT ku."Bezeichnung", ko."Beschreibung", ko."Dauer", ku."PreisTN", ku."Kurs#", ku."Status"
   FROM "KURS" ku
NATURAL JOIN "KONZEPT" ko
  WHERE ku."Status" = 'buchbar' OR ku."Status" = 'nV';
```

BuchungsKunden Eine Liste aller Buchungen mit ihren zugehörigen Kunden ist als Baustein für weitere Abfragen sinnvoll. Dafür müssen zuerst alle Auftraggeber selektiert werden:

```
SELECT *
FROM "BUCHUNG" b natural join "KONTAKT" k;
```

Mit diesen Auftraggebern als Unteranfrage lassen sich zu allen Buchungen die Kunden finden. Dabei müssen in der Unteranfrage einige Attribute umbenannt werden, damit die Eindeutigkeit erhalten bleibt (hier mit Präfix "ag" für "Auftraggeber"). In der Ergebnisrelation wir noch der Name des Kunden zu "Kunde" umbenannt.

```
CREATE VIEW "VBuchungsKunden" AS
   SELECT
      auftraggeber."Buchungs#",
      kunde."KNr", kunde."Name" AS "Kunde",
      auftraggeber."agName", auftraggeber."agKNr"
   FROM
      "KONTAKT" kunde,
      ( SELECT
         ag."Name" AS "agName",
         ag."KNr" AS "agKNr",
         ag."WurzelKNr" AS "agWurzelKNr",
         b."Buchungs#"
        FROM "BUCHUNG" b NATURAL JOIN "KONTAKT" ag
      ) auftraggeber
   WHERE kunde."KNr" = auftraggeber."agWurzelKNr";
```

[4] im Realeinsatz würde man für Stringvergleiche "LIKE" verwenden bzw. den Status wie auf S. 10 als Liste modellieren

Kunden Als Beispiel für eine Sicht, die eine andere Sicht und Aggregatfunktionen verwendet, soll eine Kundenliste dienen, die zu jedem Kunden die Anzahl der Buchungen, Teilnehmer und den Gesamtumsatz enthält.

```
CREATE VIEW "VKunden" AS
   SELECT
      bk."KNr", bk."Kunde",
      count(bk."Buchungs#") AS "Buchungen",
      sum(b."#TN") AS "Teilnehmer",
      sum(b."Preis") AS "Umsatz"
   FROM "VBuchungsKunden" bk, "BUCHUNG" b
   WHERE bk."agKNr" = b."KNr"
   GROUP BY bk."Kunde", bk."KNr";
```

Buchungen, für die noch keine Rechnung verschickt wurde Mit dem Konzept der Geschäftsvorfälle lassen sich die in der informellen Anwendungsbeschreibung gestellten Fragen nach dem aktuellen Zustand von Abläufen beantworten. Als Beispiel wird hier eine Sicht definiert, die die Buchungen enthält, zu denen noch keine Rechnung verschickt wurde. Diese sind dadurch zu erkennen, dass es für sie keinen Geschäftsvorfall "Rechnung verschickt" gibt.

```
CREATE VIEW "VBuchungenOhneRechnung" AS
   SELECT b."Buchungs#", b."Preis", b."#TN", b."RechnAnschrift", b."KNr", b."Kurs#"
   FROM "BUCHUNG" b
   WHERE NOT (
      b."Buchungs#" IN (
         SELECT "BUCHUNG"."Buchungs#"
         FROM "BUCHUNG" NATURAL JOIN "betrifftBUCHUNG" NATURAL JOIN "GESCHAEFTSVORFALL" gv
         WHERE gv."Bezeichnung" = 'Rechnung verschickt'
      )
   );
```

Geschäftsvorfälle sind als positive Information definiert ("Eingetretensein eines definierten Zustandes"). In der DB ist also z.B. nur gespeichert, für welche Buchungen bereits Rechnungen verschickt wurden. Diese Buchungen werden von der Unterabfrage in der Sicht geliefert. Die gesuchten Buchungen sind dann diejenigen, die *nicht* in dieser Relation enthalten sind.

Nach diesem Muster lassen sich Überwachungs-Anfragen für beliebige Typen von Geschäftsvorfällen anlegen, z.B. zur Kontrolle des Versands von Informationsmaterial, Zeugnissen, Zahlung von Dozentenhonoraren etc. Hier wird die Flexibilität dieses Modellierungsansatzes deutlich, denn neue Geschäftsvorfall-Typen und dazugehörige Überwachungsinstrumente lassen sich sehr einfach durch neue Einträge in `GVTYP` und passende Views realisieren - eine Anpassung des Datenbankschemas ist nicht nötig.

6 Standardanfragen

Dies sind Beispiele für häufig gestellte Anfragen bestimmter Anwendungsfälle an die Datenbank. Sie beziehen sich häufig auf spezifische Sichten, sind aber (in diesem Anwendungssystem) nicht darauf beschränkt. Es ist auch denkbar, einzelnen Anwendungen nur den Zugriff auf bestimmte Sichten zu gewähren, um z.B. die Daten der Buchhaltung gegen andere Abteilungen abzuschotten und geheim zu halten.

Privatkunden vs. Unternehmenskunden Will man nur Kunden eines bestimmten Typs, kann man das z.B. durch Verknüpfung der Liste aller Kunden aus der Sicht "VKunden" mit dem RS des gewünschten Kundentyps erreichen. Für Privatkunden wäre das dann:

SQL: `select * from "VKunden" natural join "PERSON";`
RA: $VKunden * PERSON$

Und für Unternehmenskunden:

SQL: `select * from "VKunden" natural join "UNTERNEHMEN";`
RA: $VKunden * UNTERNEHMEN$

Alle Kontakte einer Hierarchie Durch die Speicherung des Wurzelkontaktes zu jedem Kontakt ist das sehr einfach durch Selektion aller Kontakte möglich, die als Wurzel den Wurzelkontakt der gewünschten Hierarchie haben; wenn die gewünschte Wurzel z.B. die KNr "1" hat:

SQL: `SELECT * from "KONTAKT" where "WurzelKNr" = 1;`
RA: $\sigma_{WurzelKNr=1}(KONTAKT)$

Das liefert auch den Wurzelkontakt selbst, da dieser nach Vereinbarung sich selbst referenziert.

Teilnehmerliste eines Kurses Aus der Sicht aller Kursteilnehmer VKursteilnehmer kann man über die Kursnummer Teilnehmerlisten für einzelne Kurse selektieren, z.B. für Kurs Nummer 5:

SQL: `select * from "VKursteilnehmer" where "Kurs#" = 5;`
RA: $\sigma_{Kurs\#=5}(VKursteilnehmer)$

Dozenten für Fachgebiete Dozenten für bestimmte Themen lassen sich leicht finden; hier ist wieder kein NATURAL JOIN möglich, weil die korrespondierenden Attribute verschiedene Namen haben. Folgende Anfrage findet z.B. alle Dozenten, die das Thema "Photoshop" unterrichten können:

SQL:

```
SELECT *
from "DOZENT" d, "unterrichtet" u
where d."KNr" = u."DozentKNr"
and u."ThemaBezeichnung" = 'Photoshop';
```

RA: $\sigma_{ThemaBezeichnung=Photoshop}(\sigma_{DOZENT.KNr=unterrichtet.DozentKNr}(DOZENT \times unterrichtet))$

Telefonliste Eine Telefonliste mit allen Standard-Telefonnummern aller bekannten Personen, die die Sicht VPersonen benutzt, um bequem an Name und Vorname zu kommen (weil "istStandard" als Typ boolean realisiert wird, kann es in der where-Klausel ohne Vergleichsoperator verwendet werden):

SQL:

```
select p."Name", p."Vorname", t."Typ", k."Zusatz", t."Nummer", k."Notiz"
from "VPersonen" p, "KONTAKTWEG" k, "TELEFON" t
where k."istStandard" and p."KNr" = k."KNr" and k."KWID" = t."KWID"
order by p."Name";
```

RA: $\pi_{Name,Vorname,Typ,Zusatz,Nummer,Notiz}\langle\sigma_{KONTAKTWEG.istStandard=true}(VPersonen*KONTAKTWEG*TELEFON)\rangle$

Geschäftsvorfälle zur Wiedervolage Damit sind alle GV gemeint, deren Wiedervorlage-Zeitpunkt in der Zukunft liegt. Dabei liefert die Funktion `now()` immer die aktuelle Zeit. (Im Realeinsatz kann eine andere Definition sinnvoll sein, z.B. Wiedervorlage innerhalb von 48 Stunden um "jetzt" herum und noch nicht abgeschlossen o.Ä.. zur Demonstration soll dieses Beispiel genügen)

SQL:

```
SELECT *
   FROM "GESCHAEFTSVORFALL" gv
   WHERE gv."WiederVorl" >= now();
```

RA: $\sigma_{WiederVorl>=now()}(GESCHAEFTSVORFALL)$

Fazit Nach diesen Mustern lassen sich sich beliebig viele verschiedene Abfragen zusammenstellen; die wichtigsten Klassen von Anwendungsfällen dürften in diesen Beispielen und vor allem auch in den eingangs beschriebenen Sichten deutlich geworden sein. Mit diesen Auswertungsmöglichkeiten sollte für das Schulungsunternehmen die Verwaltung ihrer Kurse, Teilnehmer und Kontakte wesentlich effizienter als früher möglich sein.

Darüber hinaus erschließt die Modellierung der Geschäftsvorfälle die Möglichkeit, auch diese dynamischen Prozesse zu überwachen. Das war im ursprünglichen System überhaupt nicht automatisiert möglich. Dabei bleibt das System sehr flexibel und erweiterbar, indem die Arten möglicher Geschäftsvorfälle nicht durch das DB-Schema ausgedrückt werden, sondern selbst als Daten behandelt werden. Obwohl dieser Entwurf sicher nicht mit einem vollständigen Customer Relationship Management System vergleichbar ist, deckt es doch durch seine Flexibilität einen sehr großen Bereich von Informationsbedarfen und Verwaltungsaufgaben ab.

Literaturverzeichnis

[GK08] Gogolla, Martin und Mirco Kuhlmann: *Vorlesungsmaterialien zur Vorlesung Datenbanksysteme im Wintersemester 2008/2009*, 2008. http://db.informatik.uni-bremen.de/teaching/courses/ws2008_dbs, [Online, Abruf 06.03.2009].

[JQL05] Jarke, Matthias, Christoph Quix und Dominik Lübbers: *Mapping of isA-relation*. 2005. http://www-i5.informatik.rwth-aachen.de/lehrstuhl/lehre/EDB05/tutorials/MappingOfIsASummary.pdf, Lecture notes to the course "Introduction to Database Systems (Einführung in Datenbanken)", winter term 2005/2006, RWTH Aachen.

[Kle06] Kleuker, Stephan: *Grundkurs Datenbankentwicklung*. Friedr. Vieweg & Sohn Verlag, Wiesbaden, 1. Auflage, Februar 2006.

[KNS92] Keller, G., M. Nüttgens und A. W. Scheer: *Semantische Prozeßmodellierung auf der Grundlage „Ereignisgesteuerter Prozeßketten (EPK)"*. Veröffentlichungen des Instituts für Wirtschaftsinformatik(IWi), Universität des Saarlandes, 89, 1992. (http://www.iwi.uni-sb.de/iwi-hefte/heft089.pdf).

[Ste06] Steiner, René: *Grundkurs Relationale Datenbanken*. Friedr. Vieweg & Sohn Verlag, Wiesbaden, 6., überarbeitete und erweiterte Auflage, 2006, ISBN 978-3-8348-0163-0 (PRINT) 978-3-8348-9076-4 (ONLINE).