

2 Repräsentation von Daten

Um Daten automatisiert und elektronisch zu verarbeiten, müssen sie in einem geeigneten Format gespeichert werden. Feste Formate sind aber keine Erfindung der IT-Industrie. Schon vor der Einführung der elektronischen Datenverarbeitung wurden Daten, Informationen und Wissen auf sehr unterschiedliche Weise repräsentiert. Als einige der ältesten Beispiele strukturiert repräsentierter Daten gelten die Arbeitszeittabellen, die beim Pyramidenbau angefertigt wurden [91].

Man unterscheidet heute drei verschiedene Klasse von Daten bzw. Datenformaten. *Strukturierte Daten*, wie sie in Datenbanken vorliegen, besitzen eine durch ein Schema vorgegebene Struktur. Das Schema trägt wesentlich zur Bedeutung der Daten bei. Semistrukturierte Daten sind im Kern auch nach einem Schema strukturiert, können aber von diesem abweichen. Der wichtigste Vertreter *semistrukturierter Daten* sind XML-Dokumente ohne ein begleitendes XML-Schema. Die dritte Klasse sind schließlich *unstrukturierte Daten*, wie man sie in natürlichsprachlichen Texten findet.

Der Fokus der automatischen Datenverarbeitung liegt auf strukturierten Daten. Zur Definition ihrer Struktur wurde eine Vielzahl *verschiedener Datenmodelle* entwickelt, wie zum Beispiel einfache Listen, Tabellen, hierarchische Strukturen oder objekt-orientierte Modelle. In diesem Kapitel beschreiben wir zunächst die beiden wichtigsten und am weitesten verbreiteten Datenmodelle, nämlich das relationale Modell und das XML-Datenmodell. Ebenso erläutern wir semistrukturierte Daten und den Zusammenhang von Daten und Texten. Auf andere Datenmodelle, wie das klassische hierarchische Datenmodell und das objektorientierte Modell, gehen wir nicht näher ein, geben aber entsprechende Hinweise in Abschnitt 2.3. Wir beschreiben zudem, wie Daten zur Integration von einer Repräsentation in eine andere überführt werden können. Das Ziel jeder solchen *Datentransformation* muss die Erhaltung der Semantik der Daten sein, was man insbesondere an den Beziehungen zwischen Entitäten festmachen kann.

Auch zum Bau der Pyramiden wurden Daten in festen Formaten präsentiert

Strukturiert,
semistrukturiert,
unstrukturiert

Datenmodelle

Datentransformation

2 Repräsentation von Daten

Um Daten automatisiert und elektronisch zu verarbeiten, müssen sie in einem geeigneten Format gespeichert werden. Feste Formate sind aber keine Erfindung der IT-Industrie. Schon vor der Einführung der elektronischen Datenverarbeitung wurden Daten, Informationen und Wissen auf sehr unterschiedliche Weise repräsentiert. Als einige der ältesten Beispiele strukturiert repräsentierter Daten gelten die Arbeitszeittabellen, die beim Pyramidenbau angefertigt wurden [91].

Man unterscheidet heute drei verschiedene Klassen von Daten bzw. Datenformaten. *Strukturierte Daten*, wie sie in Datenbanken vorliegen, besitzen eine durch ein Schema vorgegebene Struktur. Das Schema trägt wesentlich zur Bedeutung der Daten bei. *Semistrukturierte Daten* sind im Kern auch nach einem Schema strukturiert, können aber von diesem abweichen. Der wichtigste Vertreter *semistrukturierter Daten* sind XML-Dokumente ohne ein begleitendes XML-Schema. Die dritte Klasse sind schließlich *unstrukturierte Daten*, wie man sie in natürlichsprachlichen Texten findet.

Der Fokus der automatischen Datenverarbeitung liegt auf strukturierten Daten. Zur Definition ihrer Struktur wurde eine Vielzahl *verschiedener Datenmodelle* entwickelt, wie zum Beispiel einfache Listen, Tabellen, hierarchische Strukturen oder objektorientierte Modelle. In diesem Kapitel beschreiben wir zunächst die beiden wichtigsten und am weitesten verbreiteten Datenmodelle, nämlich das relationale Modell und das XML-Datenmodell. Ebenso erläutern wir semistrukturierte Daten und den Zusammenhang von Daten und Texten. Auf andere Datenmodelle, wie das klassische hierarchische Datenmodell und das objektorientierte Modell, gehen wir nicht näher ein, geben aber entsprechende Hinweise in Abschnitt 2.3. Wir beschreiben zudem, wie Daten zur Integration von einer Repräsentation in eine andere überführt werden können. Das Ziel jeder solchen *Datentransformation* muss die Erhaltung der Semantik der Daten sein, was man insbesondere an den Beziehungen zwischen Entitäten festmachen kann.

Auch zum Bau der Pyramiden wurden Daten in festen Formaten präsentiert

Strukturiert,
semistrukturiert,
unstrukturiert

Datenmodelle

Datentransformation

neueren SQL-Standards Einzug gehalten [280]. Grundlagen und Techniken objektorientierter und objektrelationaler Datenbanken werden in [281] detailliert besprochen.

Semistrukturierte Datenmodelle tauchten in der Literatur erstmals mit den Projekten UNQL [37] und TSIMMIS [50] auf. Die dabei entstandenen Arbeiten waren grundlegend für die spätere Entwicklung von XML und insbesondere der Anfragesprachen für XML. Eine sehr gute Übersicht dazu findet man in [36].

Schließlich darf nicht unerwähnt bleiben, dass es weitere Datenmodelle gibt, die nur zur Modellierung der Daten dienen, jedoch nicht zur Speicherung. Weit verbreitete Modelle zum Datenbankentwurf sind das Entity-Relationship-Modell (ER-Modell) [52] und die Unified Modeling Language (UML) [29]. Die Integration dieser Modelle wird in diesem Buch nicht besprochen – wir konzentrieren uns auf Modelle, die direkt zur Datenspeicherung geeignet sind, also das relationale Modell und das XML-Datenmodell.

3 Verteilung, Autonomie und Heterogenität

Nachdem wir im vorigen Kapitel typische Datenmodelle, Formate und Anfragesprachen der zu integrierenden Datenquellen beschrieben haben, wenden wir uns nun der Frage zu, warum Informationsintegration ein so komplexes Thema ist.

Einem Menschen fällt es in der Regel nicht schwer, Informationen aus verschiedenen Quellen zusammenzufügen. Stellen wir uns zwei Zeitungsartikel vor, den einen in gedruckter Form und den anderen in der Onlineausgabe einer Zeitung. In einem aus dem Jahr 2000 stammenden Artikel wird die Höhe des Einkommens deutscher Bundeskanzler von 1950 bis 2000 im Text als ausgeschriebene Zahlen dargestellt, während im zweiten Artikel in tabellarischer Form die höchsten Gehaltsstufen des aktuellen deutschen Beamtentarifs aufgelistet sind. Zu jeder Gehaltsstufe ist beispielhaft eine Person angegeben, deren Gehalt nach dieser Gehaltsstufe bemessen wird. Für eine der Gehaltsstufen ist als Beispiel Gerhard Schröder aufgeführt. Aus diesen beiden Artikeln kann ein menschlicher Leser problemlos die Entwicklung der Gehälter deutscher Bundeskanzler von 2000 bis September 2005 fortsetzen, obwohl dazu eine Reihe von Überlegungen anzustellen sind:

- Die Gehälter in dem Artikel aus dem Jahr 2000 werden in Deutscher Mark angegeben, während aktuelle Gehaltsstufen in Euro festgelegt sind.
- Gerhard Schröder war bis September 2005 deutscher Bundeskanzler.
- Man muss zwischen der Darstellung von Informationen in einer Tabelle (zweiter Artikel) und im Freitext (erster Artikel) »umschalten«.
- Man muss Zahlen in ausgeschriebener Form und in Zifferndarstellung erkennen können.

Hauptproblem bei der Integration: Semantik

Hintergrundwissen ist notwendig

- Für einen groben Vergleich kann man die Begriffe Einkommen und Gehalt synonym verwenden.
- Als Hintergrundwissen sollte noch berücksichtigt werden, dass sich das deutsche Beamtentarifrecht in den letzten Jahren nicht wesentlich gewandelt hat, die Zahlen also miteinander in eine Reihe gesetzt werden können.
- Ebenso ist es wichtig zu wissen, dass Gehälter dieser Stufen für gewöhnlich am Beginn jeder Legislaturperiode neu festgelegt und dann nicht mehr geändert werden.

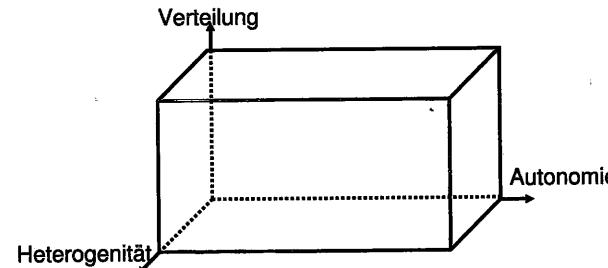
Zusammenführung von Informationen

Damit überbrückt ein Leser spielend eine Reihe von Schwierigkeiten, die sich aus der Tatsache ergeben, dass die beiden Datenquellen *heterogen* sind. Diese Heterogenität fällt einem Menschen meistens nicht auf, machen einem Computerprogramm das Zusammenführen der Informationen – also die Informationsintegration – aber sehr schwer. Im Beispiel finden wir semantische Heterogenität (Deutsche Mark versus Euro), Datenmodellheterogenität (Tabelle versus Freitext), Zugriffsheterogenität (gedruckter Artikel versus Onlineausgabe) und syntaktische Heterogenität (ausgeschriebene Form versus Zifferndarstellung von Zahlen).

Verteilung, Heterogenität und Autonomie

Im Folgenden stellen wir die drei Grundprobleme dar, die die Konstruktion von Systemen zur Informationsintegration zu einer herausfordernden Aufgabe machen: die *physische Verteilung* von Daten, die *Autonomie* der die Daten verwaltenden Datenquellen und die verschiedenen Formen von *Heterogenität* zwischen Datenquellen und dem Integrationssystem. Diese Problemfelder werden auch als die *orthogonalen Dimensionen* der Informationsintegration bezeichnet (siehe Abbildung 3.1).

Abbildung 3.1
Dimensionen der Informationsintegration



Bei der Integration können in jeder der Dimensionen unabhängig voneinander Probleme auftauchen, wobei Autonomie praktisch immer mit Heterogenität Hand in Hand geht (siehe Abschnitt 3.3).

In typischen Projekten treten immer alle drei Probleme gleichzeitig auf; das heißt, Datenquellen sind verteilt und heterogen und werden von autonomen Organisationen betreut.

3.1 Verteilung

Das offensichtlichste Hindernis zur Integration von Daten ist dann gegeben, wenn sie verteilt sind, also auf *unterschiedlichen Systemen* liegen. Wir gehen dabei immer davon aus, dass diese Systeme untereinander vernetzt sind, da sonst der Zugriff durch das Integrationssystem rein technisch gar nicht möglich wäre. Beispielsweise sind Datenquellen, auf die man über eine Webschnittstelle zugreift, verteilt: Der Browser zeigt nur einen Ausschnitt der Daten an, der von einem Webserver zur Verfügung gestellt wird. Die eigentlichen Daten werden aber auf dem Webserver oder einem von diesem angesteuerten Rechner verwaltet – irgendwo auf der Welt.

Verteilung hat zwei Aspekte, nämlich den der physischen und den der logischen Verteilung:

- Daten sind *physisch verteilt*, wenn sie auf physisch getrennten und damit meist auch geografisch entfernten Systemen verwaltet werden.
- Daten sind *logisch verteilt*, wenn es für ein Datum mehrere mögliche Orte zu seiner Speicherung gibt.

Prinzipiell sind diese beiden Eigenschaften voneinander unabhängig. Beispielsweise wird bei der *Replikation* von Datenbanken eine physische Verteilung vorgenommen, die keine logische Verteilung mit sich bringt. Andererseits erzeugt eine zur Optimierung eingeführte *Redundanz* innerhalb eines Schemas bereits logische Verteilung, da semantisch identische Daten plötzlich in mehreren Tabellen repräsentiert sind.

Physische Verteilung

Physische Verteilung bringt für die Integration verschiedene Schwierigkeiten mit sich. Zunächst müssen die *physischen Orte der Daten*, also Rechner, Server und Port, im Netzwerk identifizierbar und im laufenden Betrieb lokalisierbar sein. Dies geschieht für die Anwendung in der Regel vollkommen transparent. Im Internet benutzen Anwendungen Namen (URLs) zur Identifizierung von entfernten Rechnern und Diensten, ohne dass sie noch Kennt-

Zwei Aspekte von Verteilung

Lokalisierung von Daten

**Adressierung
mehrerer Schemata**

nis darüber brauchen, wo sich diese tatsächlich befinden. Die Lokalisierung ist Aufgabe der *Netzwerkebene* eines Rechnersystems, die dazu Protokolle wie TCP/IP benutzt. Die damit verbundene Problematik ist aber nicht Inhalt dieses Buches. Entsprechende Literatur geben wir in Abschnitt 3.5 an.

Als Zweites bringt physische Verteilung immer mit sich, dass Daten in verschiedenen Schemata vorliegen. Herkömmliche Anfragesprachen sind aber nicht in der Lage, Anfragen an *Tabellen in unterschiedlichen Schemata* zu formulieren. Daher ist es notwendig,

- entweder Anfragen über den Gesamtdatenbestand in verschiedene, schemaspezifische Anfragen zu zerlegen und die Ergebnisse in dem Integrationssystem wieder zu vereinen
- oder Sprachen zu entwickeln, die mit mehreren Schemata in einer Anfrage umgehen können.

**Globales Schema
versus Multidaten-
banksprache**

Die erste Variante basiert auf dem Vorhandensein eines *globalen Schemas* und erfordert komplexe Algorithmen, die wir in Kapitel 6 vorstellen werden. Die zweite Lösung, die wir in Abschnitt 5.4 erläutern werden, fasst man unter dem Begriff *Multidatenbanksprachen zusammen*. Der Zugriff auf die Datenquellen erfolgt hier ohne globales Schema.

**Optimierung verteilter
Anfragen**

Das dritte Problem, das mit physischer Verteilung einhergeht, sind die geänderten Anforderungen an die Anfrageoptimierung. In zentralen Datenbanken ist es die Hauptaufgabe des Optimierers, die zur Beantwortung einer Anfrage verursachte Menge von Zugriffen auf den Sekundärspeicher klein zu halten, da diese Zugriffe um Größenordnungen mehr Zeit brauchen als Berechnungen im Hauptspeicher. Bei verteilten Datenbeständen müssen dagegen die *Zugriffe über das Netzwerk* möglichst minimiert werden, da diese nun um Größenordnungen mehr Zeit brauchen als Sekundärspeicherzugriffe. Netzwerkzugriffe haben darüber hinaus spezielle Eigenschaften, wie die verhältnismäßig lange Zeit zum *Aufbau einer Verbindung* und Beschränkungen in der Bandbreite (siehe Abschnitt 6.5).

Logische Verteilung**Daten ohne festen
Platz**

Aus logischer Perspektive bedeutet Verteilung, dass identische Daten im Gesamtsystem an verschiedenen Stellen liegen können. Logische Verteilung liegt damit schon vor, wenn man in einer zentralen Datenbank zwei Tabellen *film1* und *film2* anlegt, ohne dass

klar wäre, wie sich Filme in den beiden Tabellen voneinander unterscheiden sollen. Um in einer solchen Situation alle Filme zu berechnen, muss ein Benutzer zwei Anfragen bzw. eine Anfrage mit einer UNION-Operation ausführen. Ist dagegen klar, dass in Tabelle *film1* alle Filme gespeichert werden, die vor dem Jahr 2000 gedreht wurden, und in *film2* alle danach gedrehten Filme, so liegt keine logische Verteilung vor, sondern nur eine ungeschickte Benennung von Tabellen. Die charakteristische Eigenschaft der logischen Verteilung ist also die *Überlappung* in der Intension verschiedener Datenspeicherorte – was das genau bedeutet, werden wir in Abschnitt 3.3.6 ausführlich erläutern. Dabei ist es unerheblich, ob diese Systeme tatsächlich an verschiedenen physischen Orten stehen – zwei Datenbankinstanzen auf einem Rechner bedingen bereits logische Verteilungsprobleme.

Von zentraler Bedeutung ist in dieser Situation die *Redundanz* im System. Wenn semantisch gleiche Daten an verschiedenen Orten liegen können, bezeichnen wir das System als redundant. Um bei Vorliegen von Redundanz ein *konsistentes Gesamtbild* zu erhalten, muss die Redundanz streng kontrolliert werden (etwa durch Trigger oder Replikationsmechanismen) – es muss erzwungen werden, dass an den verschiedenen Orten auch immer dieselben Daten vorliegen. Bei der Informationsintegration hat man es aber typischerweise mit *unkontrollierter Redundanz* zu tun, da die verschiedenen Datenbestände unabhängig voneinander gepflegt werden. Daraus ergeben sich mehrere Probleme:

- Für einen Benutzer ist nicht klar, in welchen Systemen er welche Daten findet. Das Integrationssystem muss daher Metadaten zu Verfügung stellen, die eine *Datenlokalisierung* ermöglichen, beispielsweise in Form eines Katalogs aller Schemata und deren Beschreibungen oder eines globalen Schemas mit Abbildungen in Quellschemata (siehe Abschnitt 6.2).
- Können Daten in verschiedenen Orten liegen, so werden *Duplikate* existieren, d.h., Objekte, die an beiden Orten gespeichert sind. Diese müssen vom System erkannt werden (siehe Abschnitt 8.2).
- Redundante Daten können *Widersprüche* enthalten, die im Sinne einer homogenen Präsentation aufgelöst werden müssen (siehe Abschnitt 8.3).

Redundanz**Lokalisierung****Duplikate****Widersprüche**

Bewusste Verteilung**Gewollte versus ungewollte Verteilung**

Verteilung ist nicht per se ein Ärgernis, sondern der Einsatz von Verteilung ist oft eine *bewusste Designentscheidung* (siehe Abschnitt 10.1). Datenverteilung ist eine nützliche Hilfe zur Lastverteilung, zur Ausfallsicherheit und zum Schutz vor Datenverlust. Zur *Lastverteilung* werden Daten auf verschiedenen Rechnern repliziert, also logisch und physisch verteilt, die dann wechselseitig Anfragen an die Datenbestände bearbeiten können. Ebenso kann die *Ausfallsicherheit* von Anwendungen durch Datenreplikation erhöht werden, da bei Ausfall eines Rechners eine Bearbeitung der Daten durch die anderen Systeme möglich ist. Schließlich bilden physikalisch weit verteilte Datenbestände einen wirksamen Schutz vor *Datenverlust* durch Katastrophen wie Brand oder Erdbeben.

Kontrollierte oder unkontrollierte Redundanz

All diesen Fällen ist gemeinsam, dass die Verteilung der Daten von *zentraler Stelle* aus kontrolliert wird. Die Datenkonsistenz wird durch aufwändige Mechanismen wie das *2-phase-commit*-Protokoll gesichert [61]. Im Gegensatz dazu ist die Verteilung von Daten in typischen Integrationsprojekten *historisch gewachsen oder organisatorisch bedingt* und daher unkontrolliert redundant.

3.2 Autonomie

Autonomie: Praktisch unvermeidbar

Im Kontext der Informationsintegration bezeichnet Autonomie die Freiheit der Datenquellen, unabhängig über die von ihnen verwalteten Daten, ihre Struktur und die Zugriffsmöglichkeiten auf diese Daten zu entscheiden. Autonome Datenquellen trifft man immer an, wenn unternehmensübergreifend bereits bestehende Systeme integriert werden sollen. Auch innerhalb von Organisationen werden Systeme oft autonom entwickelt, etwa wenn Abteilungen Eigenentwicklungen vorantreiben. Es scheint ein generelles Bestreben von Menschen zu sein, sich *größtmögliche Autonomie* in ihren Entscheidungen zu erhalten, und dieses Bestreben findet man auch bei der Entwicklung von Informationssystemen, selbst wenn es Unternehmensinteressen direkt entgegen läuft. Da, wie wir sehen werden, Autonomie fast immer schwierige Heterogenitätsprobleme erzeugt, ist die *Einschränkung von Autonomie* ein probates Mittel, um Integration zu erleichtern. Gleichzeitig ist dieses Ziel aber auch sehr schwer zu erreichen, wie man an den langwierigen Prozessen typischer Standardisierungsgremien sehen kann. Man muss bei der Informationsintegration daher meistens mit hoher Autonomie und allen damit verbundenen Folgen leben.

Prinzipiell kann man vier Arten von Autonomie unterscheiden: Designautonomie, Schnittstellenautonomie, Zugriffsautonomie und juristische Autonomie.

Verschiedene Arten von Autonomie**Designautonomie**

Eine Datenquelle besitzt Designautonomie, wenn sie frei entscheiden kann, in welcher Art und Weise sie ihre Daten zur Verfügung stellt. Dies umfasst insbesondere das Datenformat und das Datenmodell, das Schema, die syntaktische Darstellung der Daten, die Verwendung von Schlüssel- und Begriffssystemen und die Einheiten von Werten.

Designautonomie ist die Ursache der für die Informationsintegration schwierigsten Arten von Heterogenität, nämlich der strukturellen, semantischen und schematischen Heterogenität (siehe Abschnitt 3.3). Daher ist es verlockend, sie in Projekten einzuschränken, indem man Austauschformate, Schemata oder Zugriffsschnittstellen festlegt. Dies wird oftmals angewandt, wenn Unternehmen ihre Systeme miteinander vernetzen und ein Unternehmen dabei dominierend ist. So schreiben Automobilhersteller ihren Zulieferern oft detailliert vor, wie der Zugriff auf deren IT-Systeme erfolgen soll.

Schnittstellenautonomie

Schnittstellenautonomie bezeichnet die Freiheit jeder Datenquelle, selber zu bestimmen, mit welchen technischen Verfahren auf die von ihr verwalteten Daten zugegriffen werden kann. Beispielsweise kann sie festlegen, welches Protokoll und welche AnfrageSprache benutzt werden muss. Schnittstellenautonomie hängt eng mit Designautonomie zusammen, da die Art der Datenrepräsentation auch die Art des Zugriffs bestimmt oder zumindest stark einschränken kann. Typische Schnittstellen sind heute HTTP-Schnittstellen oder Webformulare (siehe Abschnitt 6.6), Web-Services mit dem Austauschprotokoll SOAP (siehe Abschnitt 10.4), oder JDBC mit der Anfragesprache SQL (siehe Abschnitt 10.1).

Auch die Schnittstellenautonomie kann in Projekten durchaus eingeschränkt werden: So schreibt die Bundesfinanzaufsicht (BaFin) gemäß §24c des Kreditwesengesetzes KWG im Rahmen des so genannten automatisierten Abrufverfahrens jeder deutschen Bank vor, bestimmte Daten über Konten über eine genau definierte Schnittstelle zur Verfügung zu stellen.

Zugriffsautonomie

Zugriffsautonomie ist gegeben, wenn eine Datenquelle frei entscheiden kann, wer auf welche der von ihr verwalteten Daten zugreifen kann. Dies betrifft insbesondere Fragen der Benutzerkonten, der *Authentifizierung* und der *Autorisierung*. Neben der binären Entscheidung »Zugriff oder nicht« umfasst Zugriffsautonomie auch die Vergabe von Lese- und Schreibrechten nur für bestimmte Daten, beispielsweise für alle Elemente einer bestimmten Klasse oder Relation. Die Zugriffsautonomie vieler Systeme ist ein unter dem Schlagwort »Identity Management« derzeit viel diskutiertes Problem, da ein durchschnittlicher Benutzer des Web in kurzer Zeit eine große Menge von unterschiedlichen Benutzerkonten und Passwörtern erzeugt und irgendwie verwalten muss. Verschiedene Konsortien versuchen, Systeme für das einfache und flexible Management dieser Zugangsberechtigungen zu entwickeln, ohne dass die Zugriffsautonomie der beteiligten Systeme eingeschränkt wird, z.B. die Liberty-Allianz¹ oder das Passport-Projekt².

Juristische Autonomie

Mit juristischer Autonomie bezeichnen wir das Recht einer Datenquelle, die Integration ihrer Daten in ein Integrationssystem juristisch zu verbieten, wenn dadurch beispielsweise *Urheberrechte* verletzt werden. Die Durchsetzung der juristischen Autonomie ist keine triviale Aufgabe. Auf Webseiten dargestellte Information kann beispielsweise leicht durch systematisches und periodisches Parsen der HTML-Seiten kopiert und in andere Systeme integriert werden.

Weitere Arten der Autonomie

In der Literatur werden als weitere Arten von Autonomie häufig die *Kommunikationsautonomie* und die *Ausführungsautonomie* genannt [57]. Eine Datenquelle ist autonom bzgl. ihres Kommunikationsverhaltens, wenn sie selber bestimmt, ob und wann sie eine Anfrage beantwortet. Kommunikationsautonomie erweitert damit Zugriffsautonomie um eine temporale Komponente. Ausführungsautonomie ist gegeben, wenn eine Datenquelle selber entscheiden kann, welche Arten von Operationen sie auf Anfrage externer Systeme hin ausführt. Dieser Begriff wird besonders

Kommunikations-
autonomie
Ausführungsautonomie

¹Siehe www.projectliberty.org/.

²Siehe login.passport.com/.

im Zusammenhang mit Problemen des schreibenden Zugriffs auf Datenquellen durch das Integrationssystem genannt und bezeichnet dann die Fähigkeiten bzw. Einwilligung einer Datenquelle, bestimmte Transaktionsprotokolle zu unterstützen. Auch fällt darunter die Entscheidung einer Datenquelle, Anfragen bestimmter Kunden schneller zu beantworten (*golden customers*).

Evolution von Datenquellen

Autonomie bezeichnet nicht nur die Freiheit einer Datenquelle, ihre Daten nach eigenem Gutdünken zum Zugriff freizugeben, sondern auch, diese Entscheidungen jederzeit zu ändern und zum Beispiel einmal erteilte Zugriffsrechte zu entziehen oder das Präsentationsformat der Daten zu ändern. Diese so genannte *Quell-evolution* stellt Integrationssysteme mit mehr als einer Hand voll Datenquellen vor ein schwieriges Problem. Ändert beispielsweise eine Datenquelle im Durchschnitt nur einmal im Jahr etwas an ihrem Schema, bedeutet das für ein Integrationssystem mit 20 Datenquellen ungefähr zwei Änderungen pro Monat. Die Fähigkeit eines Integrationssystems, flexibel und schnell auf Änderungen in Datenquellen reagieren zu können, sollte daher ein wichtiges Kriterium bei der Wahl seiner Methoden zur Integration sein.

Autonomie bei Softwarekomponenten

Die Autonomie von Komponenten ist auch ein wichtiges Forschungsgebiet des *Software Engineering*. In komponentenorientierten Systemen versucht man, die Autonomie der Teilkomponenten sorgfältig gegen die Anforderungen des Gesamtsystems auszustarieren. Das klassische *Black-Box-Modell* beispielsweise gestattet vollständige Designautonomie, bietet aber durch die Festlegung der Zugriffsschnittstelle keine Schnittstellenautonomie. Moderne Softwarearchitekturen gehen zunehmend dazu über, im Sinne einer besseren Beherrschbarkeit des Gesamtsystems und einer leichteren Austauschbarkeit von Komponenten deren Autonomie einzuschränken, wie beispielsweise im EJB-Komponentenmodell.

Black-Box-Modell

Beispiele

Vollständige Autonomie von Datenquellen ist oftmals in Integrationssystemen gegeben, die *Webdatenquellen* integrieren, wie Metasuchmaschinen oder Preisvergleicher. Beispielsweise integriert eine Metasuchmaschine Suchmaschinen, die von ihrer Integri-

Webdatenquellen:
Sehr hoher Grad an Autonomie

Einschränkung von Autonomie

on in der Regel gar nichts wissen und entsprechend auch keine Rücksicht auf Belange des integrierten Systems nehmen.

Andererseits sind nicht in allen Integrationsszenarien nur vollständig autonome Datenquellen vorhanden. Bei der Integration von Systemen innerhalb eines Unternehmens, wie beispielsweise bei der Zusammenführung verschiedener Geldströme verursachender Systeme (Investmentbanking, Kreditgeschäft, Anlagenmanagement etc.) einer Bank zu einem übergeordneten Liquiditätsmanagement, bleiben zwar die ursprünglichen Systeme aus Kosten- und organisatorischen Gründen oft erhalten, verlieren aber einen Teil ihrer Autonomie dahingehend, dass das Integrationssystem vorab über anstehende Änderungen informiert werden muss. Unter Umständen müssen solche Änderungen auch abgesprochen oder sogar genehmigt werden.

3.3 Heterogenität

Zwei Informationssysteme, die nicht die exakt gleichen Methoden, Modelle und Strukturen zum Zugriff auf ihre Daten anbieten, bezeichnen wir als *heterogen*. Heterogenität von Informationssystemen hat viele Facetten, denen wir uns im Folgenden zuwenden. Wir betrachten dabei Heterogenität ausschließlich auf der Ebene der Datenmodelle und -schemata; den ebenfalls wichtigen und schwierigen Integrationsproblemen bezüglich der tatsächlichen Daten ist Kapitel 8 gewidmet. Zunächst beleuchten wir aber den Zusammenhang zwischen Heterogenität und den anderen zwei Dimensionen der Informationsintegration.

Heterogenität und Verteilung

Heterogenität und Verteilung sind orthogonal

Heterogenität und Verteilung von Datenquellen sind prinzipiell orthogonale Konzepte. Es können sowohl zwei homogene Datenquellen physisch verteilt sein als auch zwei auf einem Rechner liegende Systeme heterogen sein. Gleichzeitig ist es aber häufig anzutreffen, dass Systeme, die verteilt werden und damit von unterschiedlichen Personen betreut und weiterentwickelt werden, zum Auseinanderdriften neigen. Insofern erzeugt Verteilung oftmals, aber nicht zwingend, Heterogenität.

Heterogenität und Autonomie

Autonomie erzeugt Heterogenität

In ähnlicher Weise sind Autonomie und Heterogenität zunächst orthogonale Dimensionen, die in der Praxis aber eng zusammenhängen. Generell ist zu beobachten, dass der *Grad der Heterogenität*

genität zwischen Datenquellen mit dem Grad der Autonomie der Quellen zunimmt. Zwei vollkommen unabhängige Datenquellen werden praktisch immer heterogen sein, selbst wenn sie dieselben Arten von Informationen verwalten. Beispielsweise werden sich die Informationssysteme zweier unabhängiger Buchhändler in dem verwendeten Datenbankmanagementsystem, dem Schema, den verwendeten Begriffen, der Menge an Information zu den vertriebenen Büchern, der Zugangskontrolle, der Anfragesprache und so weiter unterscheiden. Ein Händler mag eine relationale Datenbank der Firma Oracle verwenden, eigene Tabellen für neue und antiquarische Bücher benutzen, zusammen mit dem Bücherbestand auch alle Verkäufe und Bestellungen verwalten und Preise in Euro ohne Mehrwertsteuer speichern, während ein anderer Händler ein XML-basiertes System einsetzt, in dem nur der Bestand gehalten wird, keine Unterscheidung zwischen neuen und antiquarischen Büchern getroffen wird, dafür aber zwischen Büchern und Buchreihen, und alle Preise als Brutto gespeichert sind.

Heterogenität ergibt sich in der Praxis aufgrund *unterschiedlicher Anforderungen*, unterschiedlicher Entwickler und unterschiedlicher zeitlicher Entwicklungen. Sie tritt selbst dann auf, wenn zunächst identische Softwaresysteme gekauft werden, da praktisch immer eine Anpassung der Software an die Notwendigkeiten eines Unternehmens vorgenommen wird (engl. *Customizing*).

Heterogenität aufgrund unterschiedlicher Anforderungen

Begrenzung der Heterogenität durch Standards

Homogenität erzwingen

Heterogenität ist das Hauptproblem bei der Informationsintegration. Deswegen wird in Integrationssystemen oftmals versucht, die Autonomie der Quellen einzuschränken und damit in bestimmten Aspekten *Homogenität zu erzwingen*. Dies kann vom Festschreiben des verwendeten Systems über die verwendete Sprache bis zum Festschreiben der exakten Definition aller relevanten Konzepte reichen. Beispielsweise versuchen Branchenorganisationen oftmals, Heterogenität in der Bedeutung und der Repräsentation der branchentypischen Informationen durch die Festlegung von *Standards* zu vermindern. Dies setzt eine Einschränkung der Autonomie der Informationssysteme in dieser Branche voraus, erleichtert aber den Informationsaustausch. Beispiele hierfür sind Austauschformate für Handelswaren wie EBXML und RosettaNet [71], das Datenmodell EXPRESS/STEP für die Automobilbranche [251] oder das Begriffssystem »Gene Ontology« zur Bezeichnung der Funktionen von Genen in der Molekularbiologie (siehe Abschnitt 7.1). Ebenso wie Formate können

Branchenspezifische Standards

Heterogenität zwischen verschiedenen Akteuren

auch Schnittstellen oder Kommunikationsprotokolle vorgeschrieben werden.

Heterogenität zwischen Integrationssystem und den Datenquellen

Heterogenität besteht sowohl zwischen Datenquellen untereinander als auch zwischen dem Integrationssystem und den Datenquellen. Von Interesse ist in den meisten Integrationsarchitekturen nur die letztere Form, da in diesen Datenquellen nicht untereinander kommunizieren³. Heterogenität liegt in diesen Fällen beispielsweise dann vor, wenn das Integrationssystem SQL-Zugriff auf die integrierten Daten gestatten möchte, aber Datenquellen beinhaltet, auf die nur über HTML-Formulare zugegriffen werden kann.

Fehlende Funktionalität

Zur Überbrückung der Heterogenität ist es offensichtlich notwendig, Anfragen zu übersetzen und fehlende Funktionalität im Integrationssystem zu implementieren. Dies ist aber nicht immer bzw. nur mit großem Aufwand möglich. Stellen wir uns ein System zur Integration von Filmdatenbanken vor, das unter anderem die Internet Movie Database (IMDB)⁴) integrieren möchte, selber SQL-Anfragen gestattet und die IMDB, zur Erreichung maximaler Aktualität bei den Antworten, nur über HTML-Formulare benutzen will. Eine Anfrage wie »Alle Filme mit Hans Albers als Darsteller, die vor 1936 gedreht wurden« kann man dann zunächst nicht beantworten, da ein entsprechendes Formular nicht existiert. Es gibt zwei Auswege: Entweder man kopiert die IMDB komplett (was lizenziertlich verboten ist), um Anfragen dann auf einer lokalen Kopie auszuführen. Alternativ kann man eine solche Anfrage beantworten, in dem man zunächst eine Anfrage nach Filmen mit »Hans Albers« per Webformular stellt, die Ergebnisse parst und die Selektion auf das Jahr des Filmes in der Integrationsschicht durchführt.

Überblick

Man kann die folgenden Arten von Heterogenität unterscheiden:

Arten von Heterogenität

Technische Heterogenität umfasst alle Probleme in der technischen Realisierung des Zugriffs auf die Daten der Datenquellen.

³Ausnahmen sind solche Architekturen, die ohne eine ausgezeichnete Integrationskomponente auskommen. Siehe dazu auch Abschnitt 4.3.

⁴Siehe www.imdb.org.

Syntaktische Heterogenität umfasst Probleme der Darstellung von Informationen.

Datenmodellheterogenität bezeichnet Probleme in den zur Präsentation der Daten verwendeten Datenmodellen.

Strukturelle Heterogenität beinhaltet Unterschiede in der strukturellen Repräsentation von Informationen.

Schematische Heterogenität ist ein wichtiger Spezialfall der strukturellen Heterogenität, bei der Unterschiede in den verwendeten Datenmodellelementen vorliegen.

Semantische Heterogenität umfasst Probleme mit der Bedeutung der verwendeten Begriffe und Konzepte.

Datenmodellheterogenität und strukturelle Heterogenität werden oft auch unter dem Begriff *Modellierungsheterogenität* zusammengefasst.

Intuitiv kann man die Unterschiede zwischen den verschiedenen Arten von Heterogenität am einfachsten wie folgt verstehen: Probleme der technischen Heterogenität sind gelöst, wenn das Integrationssystem einer Datenquelle eine Anfrage schicken kann und diese die Anfrage prinzipiell versteht und eine Menge von Daten als Ergebnis produzieren kann – ohne dass sichergestellt wäre, dass die in einer Anfrage benutzten Schemaelemente tatsächlich existieren und auch in beiden Welten dasselbe bedeuten. Probleme der syntaktischen Heterogenität sind bereinigt, wenn alle Informationen, die dasselbe bedeuten, auch gleich dargestellt werden. Probleme der Datenmodellheterogenität sind gelöst, wenn das Integrationssystem und die Datenquelle dasselbe Datenmodell verwenden. Probleme der strukturellen Heterogenität sind gelöst, wenn semantisch identische Konzepte auch strukturell gleich modelliert wurden. Probleme der semantischen Heterogenität schließlich sind überwunden, wenn das Integrationssystem und die Datenquelle unter den verwendeten Namen für Schemaelemente auch tatsächlich dasselbe meinen, gleiche Namen also gleiche Bedeutung mit sich bringen.

Im Folgenden beschreiben wir eine Vielzahl unterschiedlicher Arten von Heterogenität. In tatsächlichen Integrationsprojekten kommen aber in der Regel immer alle Arten gleichzeitig und in *komplexen Mischungen* vor. Der Sinn der in diesem Abschnitt vorgenommenen Klassifikation liegt deshalb nicht darin, eine erschöpfende Methode zur Beschreibung aller Arten von Heterogenität zwischen Datenquellen zu liefern, sondern darin, die Sensibilität des Lesers für die Art und Schwierigkeit der zu erwartenden Probleme zu erhöhen. Die Einteilung der Probleme bildet außer-

*Wann ist
Heterogenität
überwunden?*

*Heterogenitätsarten
kann man nicht
immer klar trennen*

dem die Grundlage der in späteren Kapiteln folgenden Erläuterungen von Verfahren zur Lösung von Heterogenitätskonflikten.

Der Detaillierungsgrad der Darstellung der verschiedenen Heterogenitätsarten variiert mit ihrer Bedeutung für dieses Buch. Der Schwerpunkt liegt auf strukturellen und semantischen Problemen in relationalen Datenbanken, da diese in der Praxis die meisten Probleme bereiten.

3.3.1 Technische Heterogenität

Mit *technischer Heterogenität* fassen wir solche Unterschiede zwischen Informationssystemen zusammen, die sich nicht unmittelbar auf die Daten und ihre Beschreibungen beziehen, sondern auf die Möglichkeiten des Zugriffs auf die Daten. Eine Übersicht wichtiger Konzepte der technischen Heterogenität ist in Tabelle 3.1 zu sehen.

Tabelle 3.1
Technische Ebenen
der Kommunikation
zwischen
Integrationssystem
und Datenquellen

Ebene	Mögliche Ausprägungen
Anfragemöglichkeit	Anfragesprache, parametrisierte Funktionen, Formulare (engl. <i>canned queries</i>)
Anfragesprache	SQL, XQuery, Volltextsuche
Austauschformat	Binärdaten, XML, HTML, tabellarisch
Kommunikationsprotokoll	HTTP, JDBC, SOAP

Voraussetzung zur
Integration ist
Erreichbarkeit

Prinzipiell gehen wir davon aus, dass jede Datenquelle auf Netzwerkebene erreichbar ist, das Integrationssystem also einen »Kanal« öffnen kann. Hindernisse, die schon dieser Verbindung im Wege stehen, betrachten wir nicht weiter. Auf der konzeptionell nächsthöheren Schicht liegt das *Kommunikationsprotokoll*, das die Reihenfolge der Nachrichten und den Befehlssatz der Kommunikation festlegt. Über dieses Protokoll werden nun *Anforderungen* in Form von Anfragen oder Funktionsaufrufen verschickt und mit den Ergebnissen in einem *bestimmten Format* beantwortet.

Für die Informationsintegration von überragender Bedeutung ist die Möglichkeit, *Anfragen* an eine Datenquelle zu stellen. Ist die Datenquelle eine Datenbank, erwartet man eine deklarative Anfragesprache wie SQL oder XQuery. Anfragen sind in ihrer

Flexibilität unübertreffbar, da das Integrationssystem genau beschreiben kann, welche Daten es in welcher Form haben möchte. Oftmals sind die Zugriffsmöglichkeiten aber weitaus beschränkter und umfassen oft nur einen festen Satz von (parametrisierten) Funktionsaufrufen, beispielsweise in Form von Web-Services. Auch Middleware basiert auf der Festlegung von Schnittstellen, die aus Funktionen bestehen. Der Vorteil dieser *stärkeren Kapselung* liegt darin, dass ein Client das Schema der Quelle nicht kennen muss. Des Weiteren ist es für die Datenquelle sicherer, da sie viel besser vermeiden kann, dass ein Client zum Beispiel mit böser Absicht Anfragen formuliert, die extrem viele Ressourcen beanspruchen.

Zur Verdeutlichung betrachten wir drei fiktive Datenquellen:

- Datenquelle *XFilm* verwaltet Filmdaten in einer XML-Datenbank. Der Zugriff erfolgt über eine HTTP-Schnittstelle (Protokoll: HTTP). Eingebettet in einen HTTP-Request kann ein Client eine XQuery-Anfrage (Anfragesprache: XQuery) senden und erhält als Ergebnis eine XML-Datei (Austauschformat: XML) zurück. Die Anfragen werden typischerweise über ein HTML-Formular abgesetzt und die Ergebnisse über XSLT in einem Browser dargestellt, der Aufruf kann aber auch direkt aus einem Programm erfolgen.
- Datenquelle *FilmService* kann nur über eine Web-Service-Schnittstelle angesprochen werden (Protokoll: SOAP). Diese bietet eine Reihe von Funktionen wie »Suche Film nach Titel« oder »Suche Schauspieler nach Filmen«. Die Funktionen haben jeweils eine Reihe von Parametern (Wörter im Filmtitel, Name von Schauspielern etc.), die die Suche einschränken (Anfragemöglichkeit: parametrisierte Funktionen bzw. Web-Services). Intern werden die Aufrufe in SQL-Befehle umgewandelt; dies ist aber nach außen nicht sichtbar. Die Daten werden dem Client im SOAP-Format übermittelt (Austauschformat: XML).
- Datenquelle *FilmDB* speichert Daten in einer relationalen Datenbank. Für Clients ist ein Zugriff über JDBC möglich (Protokoll: JDBC). Externe Clients haben nur Leserechte und diese nur für manche Daten. Anfragen werden über SQL übermittelt (Anfragesprache: SQL) und die Ergebnisse über den JDBC-Cursor-Mechanismus zurückgeliefert (Austauschformat: binär bzw. tabellarisch).

Anfragesprachen:
Flexibilität und
Genauigkeit

Funktionen: Erhöhte
Sicherheit, stärkere
Kapselung

Drei Beispielquellen

Arten technischer Heterogenität

Entsprechend der Einteilung der verschiedenen Arten von Autonomie kann man technische Heterogenität weiter unterteilen. **Zugriffsheterogenität** bezeichnet Unterschiede in der Authentifizierung und Autorisierung zwischen Informationssystemen. **Schnittstellenheterogenität** umfasst Unterschiede in der technischen Realisierung des Zugriffs. Eine wichtige Unterart von Schnittstellenheterogenität ist *Heterogenität in den Anfragemechanismen*, insbesondere in der Art der verwendeten Anfragesprache. Der Begriff »Anfragemechanismus« ist bewusst sehr weit gefasst und beinhaltet auch HTML-Formulare oder Suchmaschinen im Web.

Überwindung technischer Heterogenität

Verfahren zur Überwindung technischer Heterogenität stellen wir an verschiedenen Stellen dieses Buches dar. Den Umgang mit Datenquellen, die nur eingeschränkte Anfragen erlauben, diskutieren wir in Abschnitt 6.6. Viele Probleme der Netzwerk-, Protokoll- und Formatebene werden in modernen Middleware-Architekturen behandelt, die wir in Abschnitt 10.2 kurz darstellen. Auch die seit einiger Zeit sehr populären Web-Services, die wir in Abschnitt 10.4 vorstellen, sind in erster Linie eine Methode zur Überbrückung technischer Heterogenität. Einen Überblick über Systeme, die technische Heterogenität zwischen verschiedenen relationalen Datenbanken überbrücken (und teilweise auch nicht relationale Quellen einbinden können), geben wir in Abschnitt 10.1.

Gleiche Information, unterschiedliche Darstellung

Mit *syntaktischer Heterogenität* bezeichnen wir Unterschiede in der Darstellung gleicher Sachverhalte. Typische Beispiele sind unterschiedliche binäre Zahlenformate (*little endian* versus *big endian*), unterschiedliche Zeichenkodierung (Unicode versus ASCII) oder unterschiedliche Trennzeichen in Textformaten (*tab-delimited* versus *comma separated values, CSV*).

Wir reduzieren damit die Klasse syntaktischer Heterogenität auf technische Unterschiede in der Darstellung von Informationen. Nicht als syntaktische Heterogenität betrachten wir beispielsweise das Synonymproblem, also die Repräsentation gleicher Konzepte durch unterschiedliche Namen – solche Konflikte fallen in den Bereich semantischer Probleme (Abschnitt 3.3.6) – oder die strukturell unterschiedliche Repräsentation gleicher Konzepte – diese betrachten wir als strukturelle Heterogenität (Abschnitt 3.3.4).

Syntaktische Heterogenität in unserem Sinne kann bei der Informationsintegration meistens leicht überwunden werden. Zahlenformate lassen sich umrechnen, Zeichendarstellungen ineinander transformieren und Trennzeichen durch einfache Parser übersetzen. Aus diesem Grund behandeln wir dieses Thema im Folgenden nicht weiter.

Syntaktische Heterogenität ist nicht problematisch

3.3.3 Heterogenität auf Datenmodellebene

Wie in Kapitel 2 erläutert, beschreiben strukturierte Informationssysteme die von ihnen verwalteten Daten durch Schemata in einem bestimmten Datenmodell. Oftmals ist das zur *Modellierung* verwendete Modell ein anderes als dasjenige zur *Datenverwaltung*, das wieder ein anderes ist als das zum *Datenaustausch* verwendete Modell. Beispielsweise wird bei der Entwicklung von Informationssystemen auf relationalen Datenbanken die Modellierung des Anwendungsbereichs oftmals zunächst im ER-Modell oder in UML vorgenommen. Diese Modelle werden später in einem Übersetzungsschritt in relationale Schemata überführt, auf deren Basis die Implementierung erfolgt. Zum Austausch von Informationen wird heute häufig XML verwendet.

Verwendung unterschiedlicher Datenmodelle

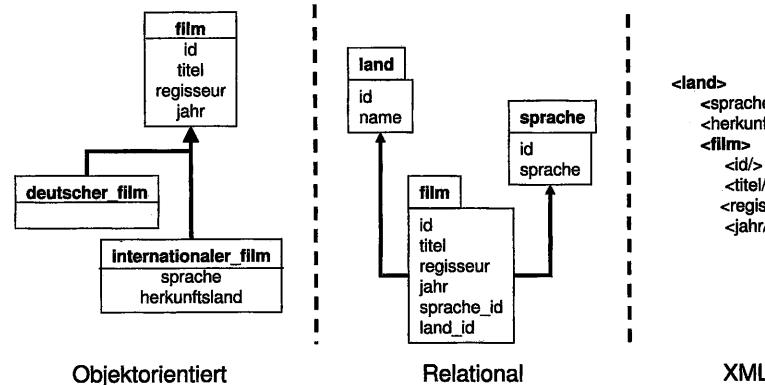
Heterogenität im Datenmodell liegt vor, wenn das Integrationsystem und eine Datenquelle Daten in unterschiedlichen Datenmodellen verwalten. Die Problematik ist unabhängig davon, ob sich die Daten in den verschiedenen Quellen semantisch überlappen, wird aber am deutlichsten, wenn verschiedene Datenquellen gleiche Daten in unterschiedlichen Modellen verwalten.

Als Beispiel sind in Abbildung 3.2 drei Schemata zu sehen, die in unterschiedlichen Datenmodellen vorliegen, aber ungefähr dieselben Informationen darstellen können. Das objektorientierte Modell verwendet eine Spezialisierungsbeziehung, um gemeinsame Attribute deutscher und internationaler Filme in einer Klasse zu vereinen. Das relationale Modell speichert alle Filme in einer Tabelle *film*, verwendet zur Speicherung des Herkunftslands und der Sprache eines Films aber eigene Tabellen, die über Fremdschlüsselbeziehungen mit der Tabelle *film* verbunden sind. Das XML-Modell schließlich verwaltet die Filme geschachtelt unter ihrem Herkunftsland und der Sprache des Films.

Semantik von Modellelementen

Heterogenität im Datenmodell bewirkt nahezu immer auch semantische Heterogenität, da Schemaelemente in unterschiedlichen Datenmodellen immer auch die spezielle, *durch das Modell festgelegte Semantik* besitzen. Beispielsweise unterscheidet sich eine Relation *film* von einer Klasse *film* schon alleine dadurch,

Abbildung 3.2
Drei nahezu
äquivalente Schemata
in unterschiedlichen
Datenmodellen



Metamodelle –
Modelle von
Datenmodellen

dass Klassen über Spezialisierungen bzw. Generalisierungen zu einander in Beziehung stehen – eine Möglichkeit, die im relationalen Modell nicht gegeben ist. Näher gehen wir auf solche Probleme in Abschnitt 3.3.6 ein.

Zur Überbrückung von Datenmodellheterogenität wurde eine Reihe von Techniken entwickelt, die auf *Metamodelle* basieren. Einen entsprechenden Ansatz stellen wir in Abschnitt 5.5 vor. Prinzipiell ist es wesentlich leichter, Schemata semantisch ärmerer Modelle in semantisch reicheren Datenmodelle auszudrücken. Ein relationales Schema kann beispielsweise einfach in ein objektorientiertes Schema übersetzt werden, indem Relationen zu Klassen überführt, Fremdschlüsselbeziehungen in Assoziationen umgewandelt werden und auf Spezialisierung verzichtet wird. Der umgekehrte Weg ist schwieriger, da die Abbildung von Spezialisierungen in ein relationales Schema nicht eindeutig ist (siehe auch Abbildung 3.3). Auch die Darstellung relationaler Daten in XML-Form ist einfach erreichbar, indem Relationen zu Elementen und deren Attribute zu geschachtelten Elementen übersetzt werden. Wiederum erfordert der umgekehrte Weg die nicht eindeutige Repräsentation hierarchischer Beziehungen durch geeignete Hilfsrelationen und Fremdschlüsselbeziehungen.

3.3.4 Strukturelle Heterogenität

Unterschiedliche
Darstellungsweisen

Auch nach Festlegung eines bestimmten Datenmodells gibt es viele Möglichkeiten, ein bestimmtes Anwendungsgebiet durch ein Schema zu beschreiben. Die Wahl eines konkreten Schemas hängt

von einer Vielzahl von Faktoren ab, wie der exakten Abgrenzung des Ausschnitts aus der realen Welt, dessen Modellierung für eine Anwendung notwendig ist, den Vorlieben der Entwickler und den technischen Möglichkeiten des gewählten DBMS.

Wir differenzieren Unterschiede in den Schemata von Datenquellen nach strukturellen und nach semantischen Aspekten:

Strukturelle Heterogenität liegt vor, wenn zwei Schemata unterschiedlich sind, obwohl sie den gleichen Ausschnitt aus der realen Welt erfassen.

Semantische Heterogenität liegt vor, wenn die Elemente verschiedener Schemata sich intensional überlappen (siehe nächsten Abschnitt).

Zum Vorhandensein struktureller Heterogenität gehört also die Tatsache, dass verschiedene Schemata potenziell gleiche Objekte beschreiben, sich also die Bedeutung oder *Intension der Schemaelemente* überlappt. Ob in einer konkreten Instanz tatsächlich gleiche Objekte vorliegen, ist für die Anfragebearbeitung unerheblich, denn das Integrationssystem kann dies vor Ausführen der Anfrage ja nicht wissen. Eine genauere Charakterisierung des Begriffs »Intension eines Schemaelements« geben wir im nächsten Abschnitt und setzen im Folgenden nur ein intuitives Verständnis voraus.

Ursachen struktureller Heterogenität

Strukturelle Heterogenität kann viele Ursachen haben, wie unterschiedliche Vorlieben des Entwicklers, unterschiedliche Anforderungen, Verwendung unterschiedlicher Datenmodelle, technische Systembeschränkungen etc. Sie ergibt sich aus der Designautonomie von Datenquellen. Wir greifen im Folgenden einige besonders häufig anzutreffende Ursachen exemplarisch heraus.

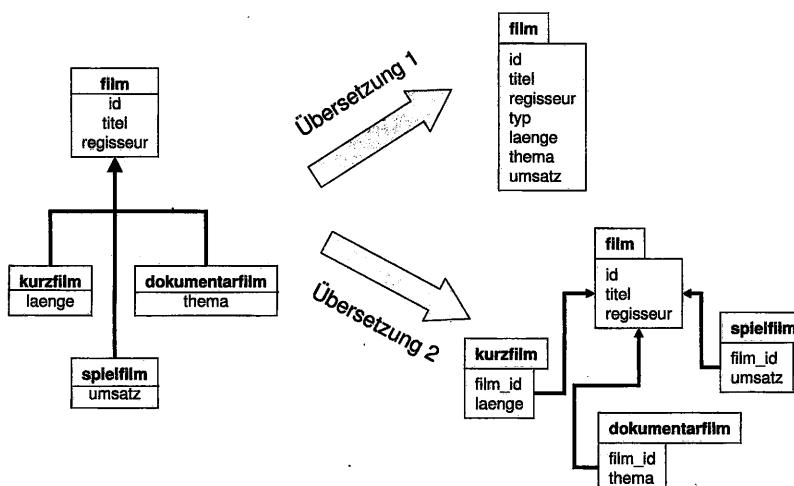
Ein häufiger Grund für das Entstehen struktureller Heterogenität liegt in den Freiheitsgraden in der *Übersetzung von konzeptionellen in logische Modelle*. Beispielsweise kann eine 1:1-Beziehung zwischen zwei Entitätstypen in einem ER-Modell entweder in eine oder in zwei Relationen übersetzt werden. Ebenfalls gibt es eine Reihe unterschiedlicher Möglichkeiten, die Spezialisierungssemantik von objektorientierten Modellen in relationalen Schemata zu repräsentieren. In Abbildung 3.3 sind zwei Möglichkeiten einer solchen objektrelationalen Abbildung (Mapping) dargestellt:

Strukturelle versus
semantische
Heterogenität

Designautonomie

Konzeptionelles →
logisches Modell

Abbildung 3.3
Abbildungen von Spezialisierungsbeziehungen in relationalen Schemata



Objektrelationales Mapping

- Das Modell nach Übersetzung 1 speichert alle Typen von Filmen in einer Relation **film**. Der Typ wird durch ein spezielles Attribut **typ** kodiert. Um semantisch äquivalent zu dem objektorientierten Schema zu sein, müssen die Attribute **laenge**, **umsatz** und **thema** mit *Integritätsbedingungen* an das Attribut **typ** gebunden werden, d.h., die entsprechenden Attribute dürfen in einem konkreten Tupel nur dann mit Werten belegt werden, wenn das Attribut **typ** den entsprechenden Wert hat.
- Das relationale Modell nach Übersetzung 2 speichert jede Klasse des objektorientierten Modells in einer eigenen Relation. Die Spezialisierung wird durch Fremdschlüsselbeziehungen der Kindrelationen zur Vaterrelation **film** dargestellt. Ist die Spezialisierung *exklusiv*, d.h., ist es verboten, dass ein Film gleichzeitig zwei Kindklassen angehört (also zum Beispiel sowohl Kurzfilm als auch Dokumentarfilm ist), so müssen die verschiedenen Fremdschlüssele der Kindrelationen durch zusätzliche *Integritätsbedingungen* dahingehend gesichert werden, dass die Mengen der jeweiligen **film_id** verschiedener Kindrelationen disjunkt sind.

Strukturen werden auf Anwendungen optimiert

Eine weitere, häufig anzutreffende Ursache für strukturelle Heterogenität liegt darin, dass Schemata oftmals für bestimmte Anfragen optimiert werden müssen. So erfordern Anfragen an Schemata, die in der dritten Normalform vorliegen, in der Regel eine

Vielzahl von Joins, da zur Vermeidung von Redundanzen und Anomalien die Daten auf viele Relationen verteilt werden. Da Joins kostenträchtige Operationen sind, werden oftmals durch eine gezielte Denormalisierung bestimmte Anfragen beschleunigt – unter Inkaufnahme der damit verbundenen Konsistenzprobleme.

Eine dritte Ursache liegt in der Freiheit des Modellierers, Attribute des Anwendungsbereichs zu untergliedern oder nicht. Beispielsweise ist es je nach Anwendung mehr oder weniger sinnvoll, das Attribut **adresse** als ein einzelnes Attribut zu speichern oder es in Felder wie **strasse**, **hausnummer**, **postleitzahl**, **ort** etc. zu untergliedern – im Grunde eine Normalisierung nach erster Normalform. An der Semantik der gespeicherten Daten ändert sich dadurch nichts.

Was sind »atomare« Informations-einheiten?

Strukturelle und semantische Heterogenität

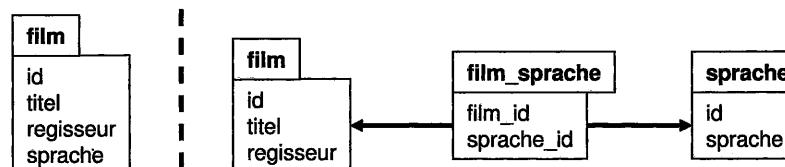
Es ist oftmals schwierig, strukturelle von semantischen Unterschieden zu trennen. Im Falle der in Abbildung 3.3 gezeigten Situation müssen zur Erkennung der semantischen Gleichwertigkeit der beiden relationalen Schemata nicht nur die Schemata selbst, sondern auch deren Integritätsbedingungen herangezogen werden. Fehlen zum Beispiel im oberen relationalen Schema die Integritätsbedingungen für die Attribute der Kindklassen, so ist die Speicherung eines Tupels wie **film(1233, 'The Birds', 'Hitchcock', 'Spielfilm', 109, 'Mad birds attack people', '15.000.000)** möglich – ein Sachverhalt, der im unteren Schema nicht ausdrückbar ist. Damit liegen zusätzlich zu der strukturellen Heterogenität auch Unterschiede in der Semantik der einzelnen Konzepte vor.

Noch deutlicher wird das enge Zusammenspiel von strukturellen und semantischen Problemen in Abbildung 3.4. Im linken Schema liegt eine *1:1*-Beziehung zwischen den Entitätstypen **film** und **sprache** vor, während im rechten Schema die Brückenrelation **film_sprache** eine *m:n*-Beziehung ausdrückt. Damit liegt neben dem schematischen Konflikt auch ein semantischer Konflikt vor, da das Verständnis des Konzepts »Film« in beiden Schemata unterschiedlich ist.

Die Behandlung struktureller und semantischer Konflikte ist deshalb nur schwer zu trennen. Trotzdem hat sich eine Trennung in der Literatur eingebürgert, der wir auch folgen. In den Kapiteln 5 und 6 behandeln wir vornehmlich Algorithmen zur Überbrückung struktureller Konflikte, während sich Kapitel 7 ganz auf semantische Probleme konzentriert.

Kardinalitäten

Abbildung 3.4
Unterschiedliche Kardinalitäten und semantische Heterogenität



Verwendung unterschiedlicher Schemaelemente

3.3.5 Schematische Heterogenität

Elemente des relationalen Datenmodells

Schematische Heterogenität ist ein Spezialfall struktureller Heterogenität, den wir wegen der damit verbundenen spezifischen Schwierigkeiten gesondert behandeln. Schematische Heterogenität zwischen zwei Schemata liegt vor, wenn unterschiedliche Elemente des Datenmodells verwendet werden, um denselben Sachverhalt zu modellieren. So können im relationalen Modell Informationen als Relation, als Attribut oder als Wert modelliert werden. Die Wahl hat weitreichende Konsequenzen, da relationale Anfragesprachen die unterschiedlichen Datenmodellelemente ganz unterschiedlich behandeln.

Betrachten wir die relationalen Schemata in Abbildung 3.5. Im obersten Schema wird die Information, welchen Typ ein Film hat, als *Name der Relation* modelliert. Im zweiten Schema wird der Filmtyp durch die zwei Attribute modelliert, die den Typ boolean haben. Im untersten Schema wird der Filmtyp als Datenwert ('spielfilm' bzw. 'doku') des Attributs typ gespeichert.

Abbildung 3.5
Unterschiedliche Modellierungsvarianten für Filmtypen

Modellierung als Relationen
spielfilm (id, titel, laenge) dokumentarfilm(id, titel, laenge)
Modellierung als Attribute
film(id, titel, laenge, spielfilm, doku)
Modellierung als Attributwerte
film(id, titel, laenge, typ)

Der Unterschied wird deutlich, wenn man überlegt, was bei einer Erweiterung des Anwendungsbereichs um den Filmtyp »Trickfilm« passieren muss:

Relation, Attribut, Wert

- Bei der Modellierung als Relation muss eine neue Relation erzeugt werden.

- Bei der Modellierung als Attribut muss die Relation film um ein Attribut erweitert werden sowie eventuell Integritätsbedingungen angepasst werden.
- Bei der Modellierung durch Werte muss am Schema nichts verändert, sondern lediglich der Wertebereich des Attributs film.typ angepasst werden.

Schematische Heterogenität und Anfragen

Schematische Konflikte sind besonders schwierig, weil sie sich in der Regel nicht durch Mittel einer Anfragesprache überbrücken lassen. In relationalen Sprachen müssen Attribute und Relationen in einer Anfrage explizit benannt werden. Ändert sich etwas in diesen Elementen, so muss die Anfrage geändert werden.

Nehmen wir als Beispiel die Varianten aus Abbildung 3.5 an, jeweils ergänzt um ein Attribut laenge, das die Länge eines Films für alle Filmtypen speichert. In der dritten Variante berechnet dann die folgende Anfrage die durchschnittliche Länge aller Filme gruppiert nach ihrem Typ:

```
SELECT typ, AVG(laenge)
FROM film
GROUP BY typ;
```

Diese Anfrage funktioniert unabhängig davon, welche Filmtypen es gibt. Die gleiche Anfrage an ein Schema, in dem pro Filmtyp eine einzelne Relation existiert (das erste Schema in Abbildung 3.5), muss dagegen mit jedem neuen Filmtyp um eine weitere UNION-Operation erweitert werden:

```
SELECT 'spielfilm', AVG(laenge)
FROM spielfilm
UNION
SELECT 'doku', AVG(laenge)
FROM dokumentarfilm
UNION
...
```

Dies trifft auch auf Anfragen gegen das zweite Schema zu:

```
SELECT 'spielfilm', AVG(laenge)
FROM film
WHERE spielfilm = TRUE
UNION
```

Schematische Konflikte sind meist durch Anfragen nicht überbrückbar

```
SELECT 'doku', AVG(laenge)
FROM film
WHERE doku = TRUE
UNION ...
```

Überbrückung durch Sichten

Für ein Integrationssystem ergeben sich schwierige Probleme: Nehmen wir an, dass Filme aus zwei relationalen Datenquellen q_1 und q_2 integriert werden sollen, wobei q_1 das obere relationale und q_2 das untere relationale Schema aus Abbildung 3.3 besitzt. Um beide Datenquellen gleichzeitig nach Regisseuren durchsuchen zu können, wäre die Definition der folgenden Sichten hilfreich:

```
CREATE VIEW q1_q2
SELECT id, titel, regisseur
FROM q1.film
UNION
SELECT id, titel, regisseur
FROM q2.film;
```

Soll auch der Filmtyp in die Sicht aufgenommen werden, wird die Definition wesentlich komplizierter:

```
CREATE VIEW q1_q2
SELECT id, titel, regisseur, typ
FROM q1.film
UNION
SELECT id, titel, regisseur, 'spielfilm' AS typ
FROM q2.spielfilm
UNION
SELECT id, titel, regisseur, 'kurzfilm' AS typ
FROM q2.kurzfilm
UNION
SELECT id, titel, regisseur, 'doku' AS typ
FROM q2.dokumentarfilm;
```

Insbesondere muss diese Sicht bei jedem neuen Filmtyp in q_2 angepasst werden. Eine Formulierung, die unabhängig von den existierenden Filmtypen ist, ist in SQL schlicht nicht möglich.

Nehmen wir nun an, dass eine Quelle q_3 das zweite Schema in Abbildung 3.5 besitzt. Dann ist es nicht möglich, eine Sicht zu definieren, die nur Filme aus der Schnittmenge von q_1 und q_3 selektiert, also solche Filme, die mit gleichem Titel, Typ und Regisseur in q_1 und q_3 vorhanden sind, ohne in der Sichtdefinition al-

le im Augenblick definierten Filmtypen aufzuzählen. Abhilfe können hier *Multidatenbanksprachen* schaffen, die Erweiterungen für besseren Umgang mit schematischen Konflikten beinhalten (siehe Abschnitt 5.4).

Spracherweiterungen

Schematische Heterogenität in nicht relationalen Datenmodellen

Auch andere Datenmodelle beinhalten verschiedene Elemente, die zu schematischer Heterogenität führen können:

- In objektorientierten Datenmodellen können Informationen als Klassen, Attribute oder Werte modelliert werden. Außerdem kodiert die Spezialisierungsbeziehung semantische Informationen, die auch mit anderen Modellelementen dargestellt werden können.
- In XML-Modellen können Informationen als Element, als Attribut oder als Wert modelliert werden. Darüber hinaus müssen die mit einer Schachtelung von Elementen verbundenen Beziehungen beachtet werden.

Elemente anderer Datenmodelle

3.3.6 Semantische Heterogenität

Werte in einem Informationssystem haben keine inhärente »Bedeutung«. Taucht isoliert die Zahl »1979« auf, so kann dies das Jahr der Produktion eines Films sein, der Preis eines Films in Cent, die Anzahl von Schauspielern, die am Dreh beteiligt waren etc. Daten werden für einen menschlichen Benutzer erst durch *Interpretation* zu Information, und diese Interpretation erfordert sowohl Wissen über die konkrete Anwendung als auch *Weltwissen*. Zur Interpretation von Daten in einem Informationssystem werden deshalb weitere Informationen herangezogen:

- Der Name des Schemaelements, das das Datum beinhaltet.
- Die Position des Schemaelements im gesamten Schema.
- Wissen über den Anwendungsbereich, für den dieses Schema entwickelt wurde.
- Andere Datenwerte, die in diesem Schemaelement gespeichert sind.

Semantik = Interpretation von Daten

Kontext ist notwendig
zur Interpretation

Information im Kontext

Wir fassen diese Informationen unter dem Begriff *Kontext* zusammen. Daten erhalten ihre Bedeutung erst durch Berücksichtigung des Kontextes. Dabei ist zu beachten, dass manche Teile des Kontextes unmittelbar in computerlesbarer Form vorliegen, wie das Schema, während andere Teile nicht explizit modelliert wurden und für Programme nicht erreichbar sind, wie das externe Wissen über die Anwendung. Beispielsweise bezeichnet der Name »Film« in einer Anwendung über Spielfilme sicherlich auf Zelluloid aufgenommene Szenen, während derselbe Name in einem Informationssystem über chemischen Anlagenbau eher Oberflächen von Werkstoffen bezeichnen könnte (oder den Herstellungsprozess von zum Filmen geeignetem Material). Ob ein gegebenes Schema aber zur Verwaltung von Filmen oder von Chemikalien gedacht ist, kann man nur implizit schließen – in keinem Datenmodell ist die Angabe des Kontextes eines Schemas vorgesehen.

Kontext bestimmt die
Semantik

Auch vollkommen identische Schemata können aufgrund eines unterschiedlichen Kontextes unterschiedliche Semantik haben. Stellen wir uns eine französische Videothek vor, die nur französische Filme führt, und eine deutsche Videothek, die nur deutsche Filme verleiht. Das System zur Verwaltung der jeweiligen Filme wurde von einem englischen Softwareanbieter gekauft. Damit verfügen beide Systeme über identische Schemata, aber die Mengen der jeweils zu verwaltenden Filme sind trotzdem disjunkt. Ein System, das beide Filmdatenbanken integrieren will, muss in der Regel diesen nur *implizit vorhandenen Kontext* explizieren, etwa durch ein neu zu schaffendes Attribut sprache.

Semantische Konflikte

Symbole, Konzepte
und Namen

Semantische Konflikte betreffen die Interpretation von *Namen* bzw. *Symbolen*. Wir erläutern die Problematik anhand von Schemaelementen; selbstverständlich muss man aber bei der Integration von Werten genauso mit semantischer Heterogenität kämpfen. Ein Beispiel dafür geben wir in Abschnitt 7.1.

Ein Schemaelement ist zunächst nichts weiter als eine Zeichenkette oder ein Name. Dieser Name bezeichnet zum einen ein Konzept unserer Vorstellungswelt und zum anderen eine Menge von realweltlichen Objekten, die durch dieses Konzept beschrieben werden (siehe Abbildung 3.6).

Extension und
Intension von
Konzepten

Das Konzept bezeichnen wir als die *Intension* des Namens und die Menge der realweltlichen Objekte als seine *Extension*. In Informationssystemen unterscheidet man zusätzlich noch die nur

virtuell vorhandene Extension des Konzepts von der konkreten Extension des Schemaelements innerhalb des Systems. Beispielsweise ist die Intension des Relationennamens »Film« in unseren Beispielen das Konzept »auf Zelluloid aufgezeichnete Szenen«. Die Extension des Konzepts ist die Menge aller jemals gedrehten Filme. Die Extension der Relation film in einer konkreten Filmdatenbank sind dagegen alle zu einem bestimmten Zeitpunkt in dieser Relation enthaltenen Tupel. In der weiteren Betrachtung beziehen wir uns immer nur auf die Extension von Konzepten.

Unterschied zur
konkreten Extension
einer Tabelle

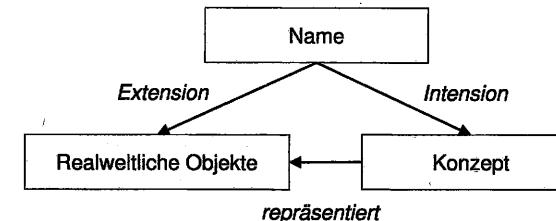


Abbildung 3.6
Extension und
Intension von Namen

Semantische Konflikte treten auf, wenn das Zusammenspiel von Namen und Konzepten in unterschiedlichen Systemen verschieden ist. In dem oben genannten Beispiel mit deutschen und französischen Filmen besitzen zwei Tabellen denselben Namen, haben aber eine geringfügig andere Intension und damit – in diesem Fall – sogar eine disjunkte Extension. Die häufigsten semantischen Konflikte sind Synonyme und Homonyme:

- Zwei Namen sind *Synonyme*, wenn sie dieselbe Intension haben, aber unterschiedlich sind (Schauspieler und Darsteller; »movie« und Film).
- Zwei Namen sind *Homonyme*, wenn sie eine unterschiedliche Intension haben, aber identisch sind. Ein prominentes Homonym in der deutschen Sprache ist das Wort »Bank«, mit den Intensionen »Finanzinstitut« und »Sitzgelegenheit«.

Synonyme und
Homonyme

Schwieriger zu behandeln (und leider auch häufiger) als Synonyme und Homonyme sind Namen, deren Intensionen weder identisch oder vollständig verschieden sind, sondern sich beinhalten oder überlappen. Ein Name ist ein *Hyperonym* eines anderen, wenn der erste Begriff ein Oberbegriff des zweiten ist, die Intension und Extension des einen also die des anderen implizieren bzw. enthalten. In den meisten Fällen sind Intensionen und Extensionen aber weder ineinander enthalten noch disjunkt. So bezeichnet das Konzept »Schauspieler« eine Menge von Realweltobjekten, die

Weitere Konflikte

mit der Extension des Konzepts »Regisseur« zwar überlappt (Regisseure, die auch als Schauspieler auftreten), die Mengen sind aber nicht ineinander enthalten.

Auffinden semantischer Konflikte

Ob zwei Namen Synonyme bzw. Homonyme sind, hängt vom Kontext und vom Betrachter ab, da die durch Namen bezeichneten Konzepte nur individuell verstanden werden können. So werden die Namen »England« und »Großbritannien« in vielen Kontexten synonym verwendet, obwohl sie es nicht sind. Für die meisten Menschen sind sicher die Namen »Prince« und »The artist formerly known as prince« Synonyme — beide bezeichnen einen bestimmten Künstler bzw. die Vorstellung, die wir von diesem Künstler haben. Für manche Menschen symbolisiert die Existenz der zwei Namen aber einen Unterschied — die bezeichneten Realweltobjekte sind zwar identisch, existieren in der Vorstellung aber in Form von zwei Konzepten, die durch ihre temporale Gültigkeit unterschieden sind. In diesem Sinne sind die beiden Namen keine Synonyme mehr.

Konzepte können nur implizit erschlossen werden

Prinzipiell ist es sehr schwierig, semantische Konflikte zu finden und eindeutig zu lösen. Dazu muss man sich klar machen, dass man für die Analyse von Schemata typischerweise nur die Schemata selber und vielleicht einige Beispieldaten zur Verfügung hat. Damit kennt man also nur Namen und einen Ausschnitt ihrer Extensionen und muss damit versuchen, *indirekt* auf die Intensionen der Namen (und damit auf ihre Semantik) zu schließen. Hilfreich dabei ist natürlich auch die *Dokumentation* der Datenquellen, Kenntnisse der Entwickler bzw. Benutzer und der Code von Anwendungen, aber diese liegen, gerade bei der Integration autonomer Quellen, oft nicht vor. Besonders schwierige Fälle ergeben sich überall dort, wo Konzepte nicht genau definiert sind oder bei unterschiedlichen Anwendern eine unterschiedliche Bedeutung haben.

(Unvermeidbare)
Restunschärfe

Andererseits sind eine restlose Aufklärung und Behebung von semantischen Konflikten oftmals nicht möglich und auch nicht notwendig. So kann kein System sicherstellen, dass die in einer Tabelle gespeicherten Daten tatsächlich in der Extension des Tabellenamens liegen. Häufig finden sich Ausnahmen, die zwar semantisch falsch eingeordnet sind, für die sich aber die Erstellung eigener Strukturen und die damit notwendigen Änderungen in Anwendungen nicht lohnen. Allen Informationssystemen wohnt eine gewisse »Restunschärfe« inne, deren Behebung einen unver-

hältnismäßig hohen Aufwand bedeuten würde und für die konkrete Anwendung auch nicht notwendig ist.

Techniken zum Umgang mit semantischen Konflikten basieren oftmals darauf, Namen nicht isoliert zu betrachten, sondern in einen *Kontext einzubetten*. Statt die Identität von Konzepten (bzw. Namen) dann durch den bloßen Vergleich von Zeichenketten zu überprüfen, werden auch die Kontexte berücksichtigt. Wir behandeln entsprechende Techniken in Kapitel 7.

Lösung: Kontext explizieren

Quasi-Synonyme

Beispiele semantischer Heterogenität

Zum besseren Verständnis zählen wir eine Reihe von typischen und leicht erkennbaren semantischen Konflikten auf. Voraussetzung dieser ist immer, dass zwei Schemaelemente aus unterschiedlichen Quellen als sehr ähnlich erkannt wurden. Wir bezeichnen diese Fälle im Folgenden als *Quasi-Synonyme*: Die Konzepte sind zum einen so ähnlich, dass dies zum Erreichen einer homogenen Darstellung des integrierten Ergebnisses berücksichtigt werden muss. Zum anderen ist aber nicht klar, ob die Namen tatsächlich Synonyme sind, was die Integration natürlich stark vereinfachen würde. Das Erkennen von Quasi-Synonymen bei Schemaelementen ist schwierig; wir werden verschiedene Techniken dazu in Abschnitt 5.1 kennen lernen und benennen im Folgenden nur kurz einige Möglichkeiten.

Ein wichtiger Hinweis darauf, dass Tabellen eine ähnliche Intension haben, ist das Vorhandensein gleicher Objekte in ihren konkreten Extensionen. Haben Objekte quellübergreifende oder sogar weltweite Identifikatoren (wie die ISBN von Büchern oder URLs von Webseiten), lässt sich dies relativ leicht feststellen. Sonst muss man Techniken der Duplikaterkennung anwenden. Ebenso ist es nützlich, die Attribute der Tabellen zu betrachten.

Sind zwei Tabellennamen als semantisch sehr ähnlich identifiziert, so deuten die folgenden Situationen auf semantische Konflikte hin – die Namen sind eben nur Quasi-Synonyme:

- Unterschiede in den Integritätsbedingungen
- Fehlende bzw. zusätzliche Attribute
- Verknüpfungen zu unterschiedlichen Tabellen
- Unterschiedliche Kardinalitäten von Beziehungen

Tabellenkonflikte

Um Attribute mit sehr ähnlicher Intension zu finden, ist ebenfalls eine Analyse von konkreten Beispieldaten nützlich. Allerdings liefert dies nur schwache Hinweise, da die Identität der Wertemenge

Attribut- und Wertkonflikte

oder des Wertebereichs weder hinreichend für die gleiche Intension des Attributs ist (zwei Attribute mit Personennamen können auf Kunden, Angestellte, Schauspieler etc. hindeuten) noch disjunkte Mengen eine hinreichende Bedingung für ungleiche Intension darstellen (da beispielsweise gleiche Begriffe ungleich kodiert sein können, siehe unten). Bei quasi-synonymen Attributnamen sollte man auf die folgenden Probleme achten:

- Unterschiede in den Einheiten der Werte (preis in Euro oder DM; laenge in Zentimetern, Metern, Inch etc.), wenn diese nicht modelliert und daher nur implizit bekannt sind.
- Unterschiede in der Bedeutung von Nullwerten und damit auch NOT NULL-Bedingungen.
- Unterschiede in der Bedeutung von Werten, insbesondere die Benutzung inkompatibler Begriffssysteme (siehe auch Abschnitt 7.1.2). Beispielsweise kann eine Quelle das Geschlecht von Personen mit »M« und »W« kodieren, eine andere mit Codes »1« und »2«, eine dritte die Bezeichnungen ausschreiben.
- Unterschiede in der Bedeutung von Skalen. So umfasst die typische Notenskala in Schulen bis zur Oberstufe die Werte 1 bis 6, ab der Oberstufe aber die Werte 0 bis 15, und dies mit umgedrehter Ordnung.

3.4 Transparenz

Vollständige Transparenz, also das Erscheinen eines integrierten Informationssystems als ein lokales, homogenes und konsistentes Informationssystem, ist häufig das Hauptziel der Informationsintegration. Wir werden aber sehen, dass vollständige Transparenz nicht immer erstrebenswert ist, da diese auch einen Informationsverlust für den Benutzer bedeutet.

Man unterscheidet verschiedene Arten der Transparenz. Diese hängen voneinander ab, d.h., manche Arten von Transparenz bedingen andere (siehe Abbildung 3.7). Wir stellen sie im Folgenden in der Reihenfolge dieser Abhängigkeiten vor.

Ortstransparenz

Ortstransparenz verbirgt vor dem Nutzer oder der Anwendung den physischen Ort, an dem die angefragten Daten gespeichert

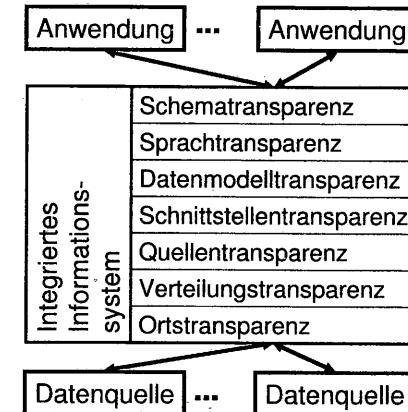


Abbildung 3.7
Arten von
Transparenz

sind. Sie ist damit gegeben, wenn es so erscheint, als ob alle Daten lokal vorhanden sind. Ein Benutzer muss weder Quellenname noch IP-Adresse, Port etc. kennen.

Verteilungstransparenz oder Quellentransparenz

In diesem Fall wird die Tatsache verborgen, dass die zugreifbaren Daten in verschiedenen Quellen gespeichert sind. Der Nutzer hat damit keine Kenntnis über die Schemata von Datenquellen oder die Herkunft von Ergebnissen. Falls Daten redundant gehalten werden, bleibt es für den Nutzer transparent, welche Kopie verwendet wird. Das integrierte System stellt sich wie eine homogene, zentrale Datenbank dar.

Verteilungstransparenz ist nicht immer erwünscht. Es kann für Benutzer sehr wichtig sein festzulegen, dass bestimmte Informationen nur aus bestimmten Quellen entnommen werden. Ein Grund kann sein, dass diese für den Benutzer vertrauenswürdiger sind oder er sie für aktueller hält. Ebenso ist es oftmals wichtig, bei Ergebnissen zu erfahren, aus welcher Quelle sie stammen.

Schnittstellentransparenz

Schnittstellentransparenz verbirgt vor dem Nutzer die Tatsache, dass Datenquellen mit unterschiedlichen Methoden angesprochen werden müssen. Anfragen an das integrierte System, die in der globalen Anfragesprache gestellt werden, müssen dazu mit den jeweiligen Zugriffsmöglichkeiten der Quellen beantwortet werden.

4 Architekturen

Integrierte Systeme sind immer komplexe Anwendungen, insbesondere dann, wenn sie viele und sehr heterogene Datenquellen integrieren. Um dieser Komplexität Herr zu werden, ist ein sorgfältiger Entwurf der Architektur eines Integrationssystems nötig.

Zum einfacheren Entwurf werden Softwaresysteme oftmals in verschiedenen *Schichten* modelliert. Jede Schicht abstrahiert Konzepte der darunter liegenden Schicht. Dieses Schichtenkonzept wird auch beim Entwurf von Datenbankanwendungen eingesetzt und kann auf Integrationssysteme übertragen werden. Der Sprung von monolithischen Datenbanken zu verteilten und dann zu integrierten Informationssystemen macht dabei diverse Erweiterungen der klassischen Datenbankarchitekturen nötig, die im Laufe der letzten ca. zwanzig Jahre vorgeschlagen und realisiert wurden. In diesem Kapitel besprechen wir in ungefährer historischer Folge:

Schichten

Architekturen in ihrer historischen Folge

- Klassische **monolithische Datenbanken**, die auf einem einzelnen Rechner laufen (als Grundlage und Startpunkt aller weiteren Architekturen).
- **Verteilte Datenbanken**, die auf mehreren Rechnern laufen, aber kaum Heterogenität aufweisen.
- **Multidatenbanken**, die mehrere heterogene Datenquellen auf der Anfrageebene integrieren.
- **Föderierte Datenbanken**, die mehrere heterogene Datenquellen auf Schemaebene integrieren.
- **Mediatorbasierte Systeme**, die eine Verallgemeinerung der anderen Ansätze darstellen.
- **Peer-Daten-Management-Systeme**, die die Unterscheidung zwischen Datenquellen und integriertem System auflösen.

Quellenkatalog

Der einfachste Ansatz, verschiedene Datenbestände zu »integrieren«, ist der Aufbau eines *Quellenkatalogs*, d.h. einer Liste aller Quellen, geordnet nach bestimmten Kriterien, wie den enthaltenen Daten, den Zugriffsrechten, der verwendeten Technik etc. (siehe auch Abschnitt 7.1.2). Bei dieser Architektur, die manchmal auch als *metadatenbasierte Integration* bezeichnet wird [39], kann man aber kaum von »Informationsintegration« im Sinne dieses Buches sprechen, da dem Benutzer nur bei der Auswahl von Quellen geholfen wird, aber nicht beim Zugriff auf deren Daten oder bei der Verknüpfung von Daten aus mehreren Quellen. Quellenkataloge bieten damit keine der in Abschnitt 3.4 erläuterten Arten von Transparenz. Trotzdem wird dieses Verfahren häufig angewendet, gerade bei beschränkten Ressourcen und einer Vielzahl von Quellen. Auch das Sammeln von Links auf thematisch verwandte Webseiten in einer Bookmark-Liste bildet bereits einen Quellenkatalog.

Data Warehouses

Eine weitere und sehr weit verbreitete Architektur zur Informationsintegration ist die *Data-Warehouse-Architektur*, die mehrere heterogene Datenquellen durch Transformation und Materialisierung in einer zentralen Datenbank integriert. Da Data Warehouses viele eigene Techniken verwenden, die sich aus ihrem Verwendungszweck, nämlich der Unterstützung der strategischen Unternehmensplanung, ergeben, widmen wir ihnen das Kapitel 9.

Architektur monolithischer Datenbanksysteme**Drei-Schichten-Architektur**

Die Basisarchitektur praktisch aller zentralen Datenbanken ist die *Drei-Schichten-Architektur* [282], die in Abbildung 4.1 dargestellt wird. Jede Schicht bietet eine andere *Sicht* auf die Daten bzw. ein anderes *Schema*. Deshalb werden die Begriffe »Sicht« und »Schicht« in der Literatur oft synonym verwendet. Im Folgenden erläutern wir jede Schicht einzeln.

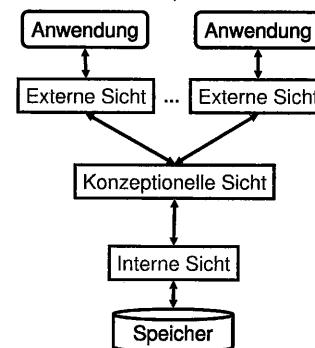
Interne Sicht

Die *interne Sicht* (auch »physische Sicht«) beschäftigt sich mit der Speicherung der Daten auf dem jeweiligen System. Im internen Schema wird festgelegt, welche Daten auf welchem Medium (Festplatte, Magnetband) und an welchem Speicherort (Zylinder, Block) gespeichert werden und welche Zugriffsstrukturen (Indizes) angelegt werden.

Konzeptionelle Sicht

Die *konzeptionelle Sicht* (auch »logische Sicht«) befasst sich mit der konzeptionellen Modellierung der zu speichernden Daten. Im *konzeptionellen Schema* wird festgelegt, welches Datenmodell ver-

Abbildung 4.1
Drei-Schichten-Architektur monolithischer Datenbanken



wendet wird, welche Daten im DBMS gespeichert werden und wie sie in Beziehung stehen. In relationalen Datenbanken geschieht dies durch das Schema mit seinen Relationen, Attributen, Typen und Integritätsbedingungen. Der Entwurf des konzeptionellen Schemas steht am Anfang jeder Datenbankentwicklung.

Die Trennung zwischen interner und konzeptioneller Sicht gewährleistet *physische Datenunabhängigkeit*. Technische Entscheidungen, wie das Anlegen von Indizes oder der Austausch einer Festplatte, sind unabhängig von der Struktur der Daten in den Schemata und können vom Administrator jederzeit geändert werden, ohne dass Anwendungen geändert werden müssten. Umgekehrt kann ein konzeptionelles Schema zunächst anwendungsgerecht entworfen werden, ohne dass bereits Details der Speicherung festgelegt werden müssen.

Die *externe Sicht* (auch »Exportsicht«) beschäftigt sich ebenso wie die konzeptionelle Sicht mit der Modellierung von Daten. Es wird jedoch nicht die gesamte Anwendungsdomäne modelliert, sondern in jeder externen Sicht wird spezifiziert, welcher Teil des konzeptionellen Schemas der jeweiligen Anwendung zur Verfügung gestellt wird. Ein *Exportschema* ist zunächst eine Teilmenge des konzeptionellen Schemas, kann aber auch Transformationen und Aggregationen vornehmen. Zudem werden in der externen Sicht Zugriffsbeschränkungen festgelegt, die es z.B. einer Anwendung verbieten, Stammdaten zu löschen.

Datenbankentwurf

Physische Datenunabhängigkeit

Externe Sicht

Die Trennung zwischen konzeptioneller und externer Sicht gewährleistet *logische Datenunabhängigkeit*. Änderungen im konzeptionellen Schema können prinzipiell unabhängig von den Exportschemata vorgenommen werden, und umgekehrt können die Exportschemata neuen Anforderungen der Anwendungen angepasst werden, ohne das darunter liegende konzeptionelle Sche-

Logische Datenunabhängigkeit

ma zu beeinflussen. Natürlich gibt es auch Änderungen, etwa das Entfernen eines Attributs im konzeptionellen Schema, die nicht ohne Auswirkungen auf die Exportschemata bleiben. Umgekehrt kann nicht jede neue Anforderung an ein Exportschema durch das bestehende konzeptionelle Schema und dessen Daten erfüllt werden. Der Schlüssel zu diesen Problemen liegt in *Abbildungen zwischen Schemata*. Diese Abbildungen sind eines der Kernthemen der Informationsintegration. Dabei geht es allerdings meist nicht um Abbildungen zwischen konzeptionellen und Exportschemata, sondern vielmehr um Abbildungen zwischen den Schemata autonomer Quellen einerseits und dem Schema des integrierten Informationssystems andererseits.

Zunächst wenden wir uns jedoch einzelnen Architekturen zur Informationsintegration zu. Es gibt eine Fülle von Literatur und Architekturvarianten. In den folgenden Abschnitten besprechen wir die wichtigsten und verweisen in Abschnitt 4.8 auf weiterführende Literatur. Einleitend diskutieren wir eine der wichtigsten Entwurfsentscheidungen: Sollen die Daten *virtuell oder materialisiert* integriert werden.

4.1 Materialisierte und virtuelle Integration

Unterschied:
Speicherort

Man kann generell zwei Arten der Informationsintegration unterscheiden: die materialisierte Integration und die virtuelle Integration. Das Unterscheidungsmerkmal zwischen diesen beiden Varianten ist der Ort, an dem die zu integrierenden Daten gespeichert sind. Bei der *materialisierten Integration* werden die zu integrierenden Daten im integrierten System selbst, also an zentraler Stelle, materialisiert. Die Daten in den Datenquellen bleiben zwar erhalten, werden aber nicht zur Anfragebearbeitung herangezogen. Bei der *virtuellen Integration* werden die Daten nur in kleinen Ausschnitten während der Anfragebearbeitung von den Datenquellen zum Integrationssystem transportiert und nach der Berechnung der Antwort wieder verworfen. Der integrierte Datenbestand existiert damit nur virtuell: Aus Sicht des Nutzers gibt es zwar einen homogenen Bestand, aber tatsächlich findet Integration *bei jeder neuen Anfrage* statt.

Die in den folgenden Abschnitten dargestellten Architekturen verwirklichen größtenteils virtuelle Integration (aber siehe dazu auch den Absatz am Ende dieses Abschnitts über hybride und gemischte Architekturen). Diese bildet auch den Schwerpunkt dieses

Schwerpunkt des Buches: Virtuelle Integration

Buches. Eine prominente Architektur, die auf Materialisierung beruht, sind *Data Warehouses*, denen wir Kapitel 9 widmen.

Zunächst untersuchen wir die Unterschiede in Bezug auf den *Datenfluss* und die *Anfragebearbeitung* im System (siehe Abbildung 4.2). Danach diskutieren wir die jeweiligen Vor- und Nachteile.

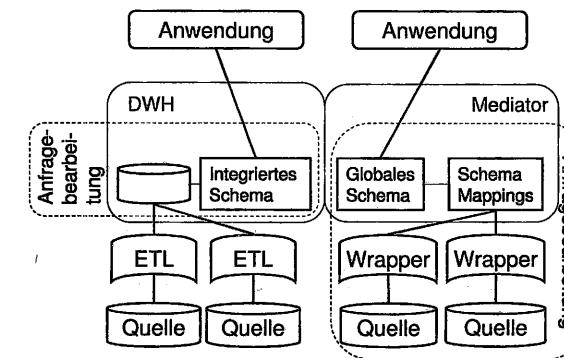


Abbildung 4.2
Materialisierte Integration in einem Data Warehouse im Vergleich zur virtuellen Integration in einem mediatorbasierten Informationssystem

Fluss in materialisierten Systemen

Asynchrone Bereitstellung, Push-Modus

Fluss in virtuellen Systemen

Datenfluss

Zur Einrichtung eines materialisiert integrierten Systems werden die gesamten Daten der Datenquellen in die globale Komponente übertragen und dort persistent, in der Regel mittels eines DBMS, gespeichert. Aktualisierungen und Ergänzungen in den Quelldaten werden im globalen System durch *periodische Update-Operationen*, etwa täglich zu einer Zeit schwacher Systemlast, repliziert. In Bezug auf Anfragen an das System ist die Datenbereitstellung also *asynchron*. Sie erfolgt meistens im *Push-Modus*: Die Daten werden, initiiert durch die Datenquellen, ins Integrationssystem »geschoben«. Nach der Übertragung findet in der Regel ein Prozess zur *Datenreinigung* statt, der die Daten normalisiert, Fehler beseitigt und Duplikate eliminiert, bevor die zentrale Datenbank aktualisiert wird.

In virtuell integrierten Systemen hingegen werden keine Daten global gespeichert, sondern sie verbleiben bei den Datenquellen. Durch Caching können Daten zwar zeitweise auch in der Integrationsschicht eines integrierten Systems vorhanden sein. Caching dient aber vornehmlich der Beschleunigung von Anfragen und gehorcht daher anderen Gesetzmäßigkeiten, die wir hier nicht berücksichtigen. Erst aufgrund einer Anfrage werden Daten von den Datenquellen übertragen – die Bereitstellung der Daten ist daher *synchron*. Dies entspricht dem *Pull-Prinzip* eines *Anfrageplans*: Die Daten werden auf Initiative des Integrationssystems aus den Datenquellen »gezogen«. In der Regel bleibt hierbei keine Zeit für komplexe Datenbereinigungsoperationen.

Synchrone Bereitstellung, Pull-Modus

Bei der virtuellen Integration können Daten kaum gereinigt werden

- dern, dass Änderungen im integrierten System bei Updates aus den Quellen einfach überschrieben werden.
- **Speicherbedarf:** Aufgrund der zentralen Speicherung sämtlicher Datenbestände ist der Speicherbedarf materialisiert integrierter Systeme groß. Bei der virtuellen Integration wird nur sehr wenig Speicher für die Metadaten und gegebenenfalls temporäre Anfrageergebnisse benötigt.
- **Belastung der Quellen:** Bei der materialisierten Integration entsteht durch die Bereitstellung von Updates regelmäßig eine *sehr hohe Last* auf den Quellen, die jedoch planbar ist. Bei der virtuellen Integration entsteht dagegen unregelmäßig eine eher geringere Last, die sich allerdings in Zeiten intensiver Nutzung – meistens praktisch unvorhersehbar – verstärken kann.
- **Datenreinigung:** Daten aus autonomen Datenquellen sind oft nicht von der benötigten Qualität; sie enthalten Datenfehler und Duplikate (siehe Abschnitte 8.1 und 8.2). Die Verfahren zur Datenreinigung sind in der Regel so aufwändig, dass sie nicht im Zuge der Anfragebearbeitung, also innerhalb von Sekunden, durchgeführt werden können. Zudem ist in einigen Fällen der Eingriff durch Experten nötig. Somit kommt Datenreinigung nur für materialisiert integrierte Systeme in Frage.

	Materialisiert	Virtuell
Aktualität	niedrig: Updatefrequenz	hoch: immer aktuell
Antwortzeit	niedrig: lokale Bearbeitung	hoch: Netzwerkverkehr
Komplexität	niedrig: wie DBMS	hoch: Anfrageplanung
Autonomie	Beantwortung von Anfragen	Bereitstellung von Load-Dateien
Anfragemöglichkeiten	unbeschränkt: volles SQL	beschränkt (bei beschränkten Quellen)
Read/Write	beides möglich	nur lesend
Speicherbedarf	hoch: gesamter Datenbestand	niedrig: nur Metadaten
Belastung der Quellen	hoch, aber planbar	ehler niedrig, schlecht planbar
Datenreinigung	möglich	nicht möglich

Tabelle 4.1
Vor- und Nachteile materialisierter und virtueller Integration

Zwischenstufen

Systeme müssen in der Praxis natürlich nicht rein virtuell oder rein materialisiert realisiert werden. *Hybride Architekturen* basieren darauf, dass Teile der Daten materialisiert werden und andere Teile nur virtuell vorliegen. Eine solche Kombination ist sinnvoll, wenn einige Datenquellen sehr oft benötigt werden und/oder komplett als Load-Datei verfügbar sind, während andere Quellen häufigen Datenänderungen unterliegen und/oder nur einen beschränkten Zugriff gestatten. Bei Datenquellen der ersten Art ist eine Materialisierung möglich und nützlich, für Datenquellen der zweiten Art ist die virtuelle Datenhaltung empfehlenswert bzw. eine Materialisierung unmöglich.

Es ist zu beachten, dass virtuelle Integration immer dann notwendig wird, wenn die Daten *logisch verteilt* bleiben. Die physische Verteilung ist davon unabhängig. So kann man auch Systeme bauen, die Datenquellen zentral replizieren, aber ihre Integration erst zum Anfragezeitpunkt durchführen. Diese Herangehensweise erleichtert das Update von Quellen sehr (da nur repliziert wird) und erlaubt trotzdem schnelle Antworten, da zur Anfragezeit kein Netzwerkverkehr notwendig ist. Auch kann man flexibel auf beschränkte Quellen reagieren. Ein entsprechendes System werden wir in Abschnitt 11.5 vorstellen.

4.2 Verteilte Datenbanksysteme

Der Schritt von monolithischen zu verteilten und föderierten Datenbanken gibt eine wichtige Grundannahme auf: Die Daten müssen nicht mehr in einem einzigen System vorliegen, sondern können *physisch und logisch auf mehrere Systeme verteilt* sein.

Unter dem Begriff »verteilte Datenbanken« fasst man Ansätze zusammen, bei denen die Verteilung einer strikten und zentralen Kontrolle unterliegt [220]. Die *Verteilung ist gewollt* und wird geplant. Die am Verbund beteiligten Datenbanken geben ihre *Autonomie vollständig auf*. Dies unterscheidet verteilte Datenbanken von allen im Folgenden vorgestellten Architekturen. Einige Gründe, aus denen man Daten verteilen möchte, haben wir in Abschnitt 3.1 genannt.

Mit der Verteilung soll nicht die Fähigkeit verloren gehen, alle Daten gemeinsam anzufragen. Im Idealfall erzeugt eine verteilte Datenbank für Anwendungen und Nutzer den Eindruck, mit einer einzigen Datenbank zu arbeiten, also *Verteilungstransparenz* zu bieten. Echte Verteilungstransparenz wird in kommerziellen Sys-

Hybride Architekturen

Virtuelle Integration mit materialisierten Daten

Verteilte Datenbestände

Verteilung »by Design«

Ziel: Verteilungstransparenz

Vier-Schichten-Architektur

temen aber oftmals nicht erreicht, immer aber Ortstransparenz (siehe Abschnitt 10.1).

Jeder Rechner im Verbund einer verteilten Datenbank besitzt ein lokales internes und ein lokales konzeptionelles Schema (siehe Abbildung 4.3). Das lokale konzeptionelle Schema spiegelt nur die Daten wider, die auf diesem Rechner verwaltet werden. Je nach *Verteilungsstrategie* kann dies ein Ausschnitt oder das gesamte globale konzeptionelle Schema sein. Zwei typische Strategien sind die vertikale und horizontale Partitionierung:

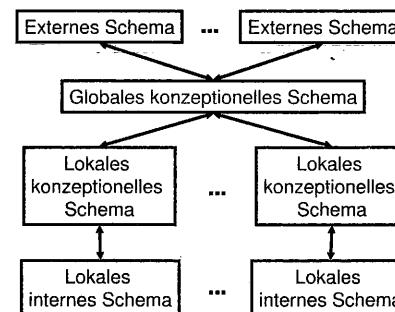
- Bei einer *horizontalen Partitionierung* werden die Daten großer Tabellen tupelweise auf verschiedenen Rechnern verteilt, beispielsweise nach Jahren oder Kunden getrennt. Eine Vereinigung fügt diese Teile wieder zusammen.
- Bei der *vertikalen Partitionierung* werden Daten großer Tabellen attributweise auf unterschiedlichen Rechnern gehalten. Ein gemeinsames Schlüsselattribut auf jeder Partition erlaubt das Zusammenfügen der Teile mittels eines Join.

Darüber hinaus können zur Beschleunigung von Anfragen manche Daten redundant auf verschiedenen Systemen im Verbund gehalten werden. In diesem Fall ist eine strenge Konsistenzkontrolle notwendig.

Oberhalb der lokalen konzeptionellen Schemata gibt es bei verteilten Datenbanken ein *globales konzeptionelles Schema*. Dieses modelliert die gesamte Anwendungsdomäne und ist zentraler Bezugspunkt für die Definition externer Schemata, die die gleiche Rolle wie in der Drei-Schichten-Architektur spielen. Der Entwurf verteilter Datenbanken erfolgt in der Regel in zwei Schritten: Einem Entwurf des globalen Schemas folgt der Verteilungsentwurf, der festlegt, welche Daten wo gespeichert werden sollen.

Globales Schema

Abbildung 4.3
Vier-Schichten-Architektur für
verteilte Datenbanken



Enge Kopplung, keine Heterogenität

Verteilte Datenbanken realisieren eine *enge Kopplung* ihrer Systeme. Aufgrund der strengen Kontrolle bei Entwurf und Betrieb treten die wesentlichen Probleme, die in diesem Buch besprochen werden – etwa strukturelle und semantische Heterogenität – nicht auf. Gleichzeitig ist die Technik, auf deren Grundlage verteilte Datenbanken realisiert werden, unabhängig vom Entwurfsprozess einer Anwendung. Daher sind verteilte Datenbanksysteme sehr geeignete Plattformen für die Informationsintegration. Diesen Aspekt werden wir in Abschnitt 10.1 näher beleuchten.

4.3 Multidatenbanksysteme

Die Multidatenbankarchitektur wurde von Litwin et al. vorgestellt [186]. Eine weitere, ähnliche Variante ist die Import/Export-Architektur von Heimbigner und McLeod [118], die hier nicht näher betrachtet wird.

Multidatenbanksysteme (MDBMS) sind Sammlungen autonomer Datenbanken, die *lose miteinander gekoppelt* sind. Die einzelnen Datenbanken bieten externen Anwendungen die Möglichkeit an, auf die Daten zuzugreifen. Dieser Zugriff erfolgt über eine *Multidatenbanksprache*, mit der mehrere Datenbanken in einer Anfrage angesprochen werden können. Die beteiligten Datenbanken behalten ihre volle *Designautonomie*.

Das wesentliche Merkmal von MDBMS ist das *Fehlen eines globalen konzeptionellen Schemas*¹ (siehe Abbildung 4.4). Stattdessen unterhält jede lokale Datenbank ein Exportschema, das wie in der Drei-Schichten-Architektur festlegt, welcher Teil des lokalen konzeptionellen Schemas externen Anwendungen zur Verfügung gestellt wird.

Es wird bei MDBMS davon ausgegangen, dass alle Datenquellen das gleiche Datenmodell verwenden, also keine Datenmodellheterogenität vorliegt. Ist dies nicht der Fall, muss entweder die lokale Datenquelle oder die Multidatenbanksprache eine Übersetzung in das globale Datenmodell anbieten.

Jede Anwendung kann nun ein eigenes externes Schema erzeugen, das die Exportschemata einer oder mehrerer Quellen integriert. Diese schwierige Integrationsaufgabe bleibt also jeder

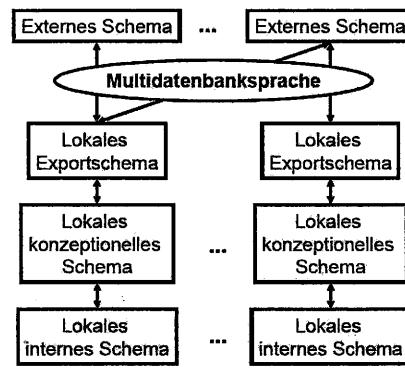
Lose Kopplung

Fehlendes globales Schema

Integration durch Multidatenbank-sprache

¹Diese Definition ist konform mit den meisten anderen Autoren [262, 220, 57], kann aber an anderen Stellen abweichen. Es werden beispielsweise Architekturen vorgeschlagen, die mehrere globale Schemata für einzelne Teilmengen der Datenquellen bilden.

Abbildung 4.4
Die Multidatenbank-architektur



Anwendung selber überlassen². Das MDBMS stellt lediglich eine Multidatenbanksprache zur Verfügung. Solche Sprachen sind in der Regel an SQL angelehnt und erlauben es, (1) innerhalb einer Anfrage auf mehrere Datenbanken zuzugreifen und (2) strukturelle und semantische Unterschiede in den Schemata zu überbrücken. Wir stellen eine dieser Anfragesprachen, SchemaSQL, in Abschnitt 5.4 vor.

4.4 Föderierte Datenbanksysteme

Fünf-Schichten-Architektur

Ungewollte Verteilung und damit Heterogenität

Kanonisches Datenmodell

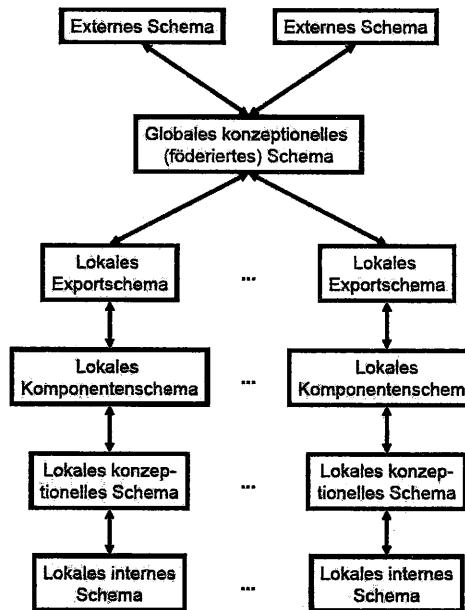
Im Gegensatz zu MDBMS besitzen föderierte Datenbanksysteme (FDBMS) ein globales konzeptionelles (»föderiertes«) Schema [262]. Dieses Schema ist zentraler und stabiler Bezugspunkt für alle externen Schemata und deren Anwendungen. Im Unterschied zu verteilten Datenbanken entsteht das globale Schema aber nach den lokalen Schemata mit dem Zweck, eine *integrierte Sicht auf existierende und heterogene Datenbestände* zu bieten. In föderierten Datenbanken ist Verteilung der Datenquellen also keinesfalls gewollt, sondern ein unausweichliches Übel. Datenquellen in FDBMS bewahren einen hohen Grad an Autonomie.

Man nennt das im globalen Schema verwendete Datenmodell das *kanonische Datenmodell*. Sämtliche lokalen Exportschemata müssen auf dieses globale Schema abgebildet werden (siehe Abbildung 4.5). Das globale Schema kann auf zwei verschiedene Weisen entstehen: Entweder es wird aus den einzelnen Exportschemata zusammengeführt, oder es wird unabhängig von den lokalen Schemata entworfen. Die erste Variante, die so genannte Schemainte-

²Natürlich können entsprechende Anfragen in Sichten zusammengefasst und mehreren Anwendungen zur Verfügung gestellt werden.

gration, wird in Abschnitt 5.1 erläutert. Die zweite Variante, das Schema Mapping, ist Thema von Abschnitt 5.2 und von Kapitel 6.

Abbildung 4.5
Fünf-Schichten-Architektur föderierter Datenbanken (Quelle: [262])



Da die Fünf-Schichten-Architektur die *Referenzarchitektur* für die meisten in diesem Buch besprochenen Techniken und Systeme ist, erläutern wir hier die Rollen der einzelnen Schichten und Schemata ausführlicher als bisher:

- **Lokales konzeptionelles Schema:** Dieses Schema modelliert die Daten der jeweiligen Datenquelle und spielt die gleiche Rolle wie das konzeptionelle Schema der Drei-Schichten-Architektur. Aufgrund der Autonomie der Quellen kann in Bezug auf das globale Schema sowohl strukturelle Heterogenität als auch Datenmodellheterogenität herrschen.
- **Lokales Komponentenschema:** Diese Schemata modellieren, sofern möglich, die Inhalte des lokalen konzeptionellen Schemas in dem kanonischen Datenmodell, also dem Datenmodell des globalen Schemas. Komponentenschemata überwinden also Datenmodellheterogenität.
- **Lokales Exportschema:** Diese Schemata haben die gleiche Rolle wie in der Drei-Schichten-Architektur: Sie stellen die Teilmenge des Komponentenschemata dar, die nach außen sichtbar ist. Die zugreifende Anwendung ist dabei in erster Linie das FDBMS, obwohl natürlich Anwendungen auch direkt auf die Exportschemata der einzelnen Datenquellen zugreifen können.

Referenzarchitektur

Schichten in FDBMS

Varianten

- **Globales konzeptionelles Schema:** Dieses Schema hat in der Literatur viele Namen, u.a. globales Schema, föderiertes Schema, Importschema, *enterprise schema* oder integriertes Schema. Es bildet die integrierte Sicht auf alle Datenquellen und ist im kanonischen Datenmodell modelliert. Vorgehensweisen zur Erstellung des globalen Schemas werden in Abschnitt 4.7.1 besprochen.
- **Externes Schema:** Ebenso wie in der Drei-Schichten-Architektur haben die externen Schemata die Aufgabe, diejenigen Teilmengen des globalen Schemas auszuwählen, die für die jeweiligen Anwendungen sichtbar sein sollen.

Im weiteren Verlauf sprechen wir meist nur von lokalen und globalen Schemata. Wir gehen ohne Beschränkung der Allgemeinheit davon aus, dass (1) alle Schemata im gleichen Datenmodell vorliegen und (2) dass das vollständige lokale Schema für das integrierte System sichtbar und zugreifbar ist.

In [262] erwähnen die Autoren diverse weitere Architekturvarianten, die im Wesentlichen Kombinationen der hier vorgestellten Varianten sind. Eine wichtige Überlegung in diesem Zusammenhang ist, welches System für welches der Schemata verantwortlich ist und insofern wie autonom die Datenquellen sind. In einer föderierten Datenbank geht man in der Regel davon aus, dass den Datenquellen bekannt ist, dass sie Teil einer Föderation sind und dass sie einen Teil ihrer Autonomie aufgeben. Beispielsweise obliegt es den einzelnen Quellen, ein Komponentenschema zu erstellen, das einzig zum Zweck der Föderation benötigt wird. Die Aufgabe der Erstellung der Exportschemata kann nicht eindeutig zugewiesen werden. Je nach Anwendungsszenario wird die Antwort unterschiedlich ausfallen. In einem rein lesenden Szenario muss die Datenquelle in vielen Fällen gar nicht wissen, dass sie Teil einer Föderation ist. In diesem Fall obliegt es dem integrierten System, das Exportschema zu erstellen. Das automatische Auslesen von Daten aus HTML-Seiten, wie es etwa Metasuchmaschinen betreiben, stellt einen solchen Fall dar. Werden innerhalb einer Organisation verschiedene Datenquellen föderiert, ist es aber eher üblich, dass die Verantwortlichen der Datenquellen selbst das Exportschema definieren. Auf diese Weise kann kontrolliert werden, welcher Teil der Daten global sichtbar wird.

4.5 Mediatorbasierte Informationssysteme

Die Mediator-Wrapper-Architektur wurde 1992 von Wiederhold vorgestellt [298] (siehe Abbildung 4.6). Sie stellt eine Verallgemeinerung der vorherigen Architekturen dar, indem sie nur zwei Rollen identifiziert: *Wrapper* sind zuständig für den Zugriff auf einzelne Datenquellen. Dabei überwinden sie Schnittstellen-, technische, Datenmodell- und schematische Heterogenität. *Mediatoren* greifen auf einen oder mehrere Wrapper zu und leisten einen bestimmten Mehrwert, in der Regel die strukturelle und semantische Integration von Daten.

Abbildung 4.6 zeigt ein einfaches Mediator-Wrapper-System. Sämtliche abgebildeten Schemata sind konzeptionell, wir verzichten also auf den Zusatz. Datenquellen ist es in dieser Architektur in der Regel nicht bewusst, Teil eines integrierten Systems zu sein. Ihre Autonomie bleibt erhalten; sie müssen lediglich ein Exportschema anbieten. Wrapper stellen Mediatoren ein Wrapperschema zur Verfügung, das zum einen bereits mittels des kanonischen Datenmodells modelliert ist und zum anderen oftmals erste Datentransformationen vorsieht, um die Integration im Mediator zu erleichtern. Der Mediator verwaltet das globale Mediatorschema und stellt Teile daraus Anwendungen zur Verfügung.

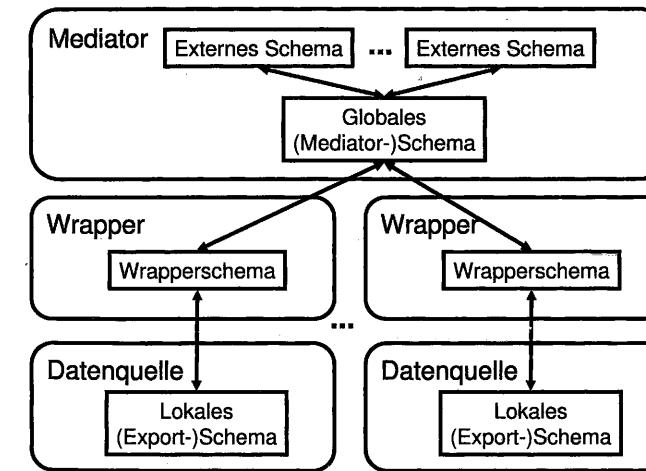


Abbildung 4.6
Mediator-Wrapper-Architektur

Die starre Architektur aus Abbildung 4.6 kann auf verschiedene Weisen erweitert werden. Abbildung 4.7 zeigt eine allgemeinere Variante. Man kann verschiedene Erweiterungen erkennen:

- **Schachtelung von Mediatoren:** Da Mediatoren ein Schema exportieren, können sie selbst wiederum anderen Media-

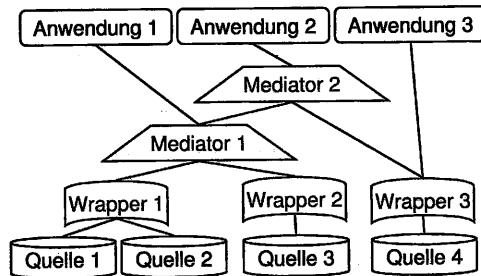
Erweiterung

Wrapper von wrapping – einwickeln
Mediator von mediate – vermitteln

toren als Datenquelle dienen. So können Mediatoren in vielen Ebenen geschachtelt werden, wobei jeder Mediator die Daten weiter anreichert bzw. integriert. Wenn das Gesamtsystem von einer Seite her geplant wird, kann auf Wrapper zwischen Mediatoren verzichtet werden.

- **Wrapper für mehrere Quellen:** Falls mehrere Datenquellen vom gleichen Typ sind, genügt es oft, für diese einen einzigen Wrapper zu entwickeln. Sollen beispielsweise mehrere Oracle-Datenbanken als Datenquellen dienen, kann es genügen, einen einzigen Oracle-Wrapper zu implementieren, der gleichzeitig mehrere Datenquellen anbindet.
- **Zugriff von Anwendungen auf Wrapper:** Da Wrapper eine Schnittstelle zu Datenquellen bieten, die bereits im kanonischen Datenmodell sind, können Anwendungen auch direkt auf Wrapper zugreifen. Sie verzichten dabei jedoch auf den Mehrwert, den ein Mediator bietet, insbesondere auf die Benutzung eines integrierten Schemas.

Abbildung 4.7
Erweiterte Mediator-Wrapper-Architektur



Mediatoren

Wiederhold definiert einen Mediator wie folgt ([298], eigene Übersetzung):

Ein Mediator ist eine Softwarekomponente, die Wissen über bestimmte Daten benutzt, um Informationen für höherwertige Anwendungen zu erzeugen.

Die Aufgabe von Mediatoren ist es also, Daten einen Mehrwert zu geben. Dies kann entweder mittels Daten anderer Quellen (also durch Informationsintegration) geschehen oder auch andere Formen annehmen, wie der folgende Auszug einer Liste in [299] zeigt:

- Suche und Auswahl relevanter Datenquellen
- Datentransformationen zur Konsistenzerhaltung
- Bereitstellung von Metadaten zur Weiterverarbeitung
- Integration von Daten mittels gemeinsamer Schlüssel
- Abstraktion bzw. Aggregation zum besseren Verständnis
- Ordnen von Ergebnissen in einem Ranking
- Filter zur Zugangskontrolle
- Semantisch sinnvolle Ausweitungen von Anfragen

Aufgaben von
Mediatoren

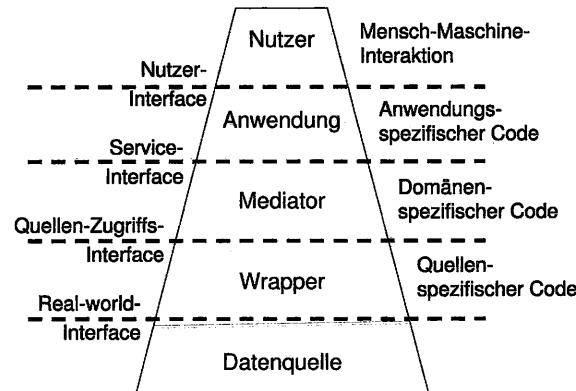
Der Mehrwert wird in der Regel erreicht, indem *Domänenexperten* ihr Wissen in die Implementierung des Mediators einbringen. Bei einem integrierenden Mediator stellen das integrierte Schema und die Abbildung der Quellschemata auf das integrierte Schema dieses Wissen dar; bei der Erweiterung von Anfragen ist das Expertenwissen in dem Thesaurus oder der Ontologie abgelegt, die zur Bestimmung sinnvoller Erweiterungen verwendet wird.

Abbildung 4.8 zeigt die logische Trennung zwischen der Implementierung der Mediatoren und der Datenquellen. In Mediatoren modellieren Experten die Besonderheiten einer Anwendungsdomäne. Wrapper hingegen sind speziell auf einzelne Quellen zugeschnitten.

Mediatoren benötigen
Expertenwissen

Schnittstellen

Abbildung 4.8
Schnittstellen der
Mediator-Wrapper-
Architektur
(nach [299])



Es ist von der Architektur nicht vorgegeben, welchen Funktionsumfang ein Mediator haben muss, damit er diesen Namen auch verdient. Nach [298] sollten Mediatoren so klein und einfach sein, dass sie durch einen einzigen oder höchstens eine kleine Gruppe von Experten entwickelt und gewartet werden können. Das heißt, Mediatoren sollten ein einfaches foderiertes Schema besitzen, eine begrenzte Domäne umfassen und über einfache Schnittstellen verfügen. Wiederhold hat diese Ziele treffend mit »dicken«, »dünnen«,

Optimaler
Funktionsumfang

»schmalen« und »breiten« Mediatoren umschrieben. Abbildung 4.9 verdeutlicht diese Metapher.

Abbildung 4.9
Dicke und dünne
Mediatoren (aus
[299])



Wrapper

Wrapper sind Softwarekomponenten, die die Kommunikation zwischen Mediatoren und Datenquellen realisieren. Wrapper sind jeweils spezialisiert auf eine Ausprägung autonomer, heterogener Quellen. Ihre Aufgaben umfassen:

Aufgaben von Wrappern

- Überwindung von Schnittstellenheterogenität, also z.B. von SQL zu HTML-Formularen.
- Überwindung von Sprachheterogenität, also z.B. den Umgang mit beschränkten Quellen oder verschiedenen Anfragesprachen.
- Herstellung von Datenmodelltransparenz durch Überführung der Daten in das kanonische Datenmodell.
- Überwindung der schematischen Heterogenität durch eine geeignete Abbildung zwischen Quellschema und globalem Schema.
- Unterstützung der globalen Anfrageoptimierung durch die Bereitstellung von Informationen über die Anfragefähigkeiten der Datenquelle (z.B. die Möglichkeit, Prädikate an die Quelle weiterzuleiten) und zu erwartende Kosten.

Ebenso wie Mediatoren sollten Wrapper einfach sein:

Schnelle Implementierung: Wrapper sollten schnell implementiert werden können. Roth und Schwarz nennen als Ziel die Implementierung eines einfachen Wrappers innerhalb weniger Stunden [246].

Erweiterbarkeit: Wrapper sollten aus zwei Gründen leicht erweiterbar sein [246]. Zum einen soll ein erster Wrapper schnell entwickelt werden, um die Machbarkeit zu zeigen. Weitere Fähigkeiten der Datenquelle müssen später hinzugefügt werden. Zum anderen können sich Datenquellen in ihren Schemata und in ihren Fähigkeiten mit der Zeit ändern. Der Wrapper muss mit diesen Änderungen leicht Schritt halten können.

Wiederverwendbarkeit: Wrapper sollten leicht wiederverwendbar sein, so dass im Laufe der Zeit Wrapperbibliotheken gebildet werden können. Ein Wrapper für den JDBC-Zugriff auf eine Oracle-Datenbank sollte sich nicht sehr von einem für den JDBC-Zugriff auf eine SQL-Server-Datenbank unterscheiden.

Wartbarkeit: Wrapper sollten lokal wartbar sein, so dass beispielsweise in föderierten Systemen die Datenquellen ihre eigenen Wrapper entwickeln und warten können.

4.6 Peer-Daten-Management-Systeme

Wünschenswerte
Eigenschaften

Peers sind
gleichberechtigt

Ein zentrales Paradigma der bisherigen Architekturen war die *Trennung zwischen Datenquellen und Integrationssystem*, wobei bei den mediatorbasierten Systemen bereits eine Aufweichung zu beobachten war. Diese Trennung lassen wir nun ganz fallen: Anfragen dürfen an jedes und von jedem an der Integration beteiligten System gestellt werden. Dieses wird dann versuchen, mittels der eigenen und anderer Daten Antworten zu berechnen. In Anlehnung an die Ideen von Peer-to-Peer-(P2P-)Systemen nennt man die resultierende Architektur *Peer-Daten-Management-Systeme* (PDMS, siehe Abbildung 4.10).

Die Rollen der Peers

Ein Peer in einem PDMS erfüllt zugleich die Rolle einer Datenquelle und die Rolle eines Mediators. Peers stellen ein Schema zur Verfügung, speichern Daten gemäß diesem Schema, nehmen Anfragen entgegen und reichen Anfrageergebnisse zurück. In ihrer

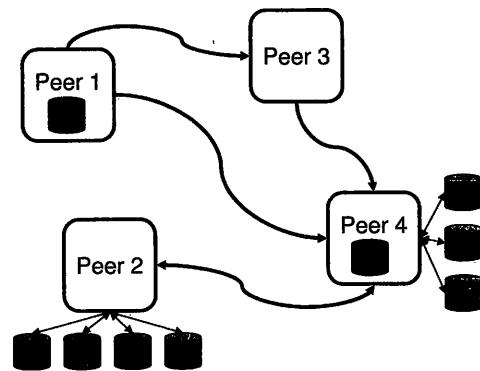
Datenquelle und
Mediator zugleich

**Verbindungen durch
Mappings**

Rolle als Datenquelle kommen die Antworten aus eigenen, lokalen Datenbeständen (Peer 1 in Abbildung 4.10). In ihrer *Rolle als Mediator* benutzen sie andere Peers, um Antworten zu finden. Nach außen, also für anderen Peers des PDMS, ist es kein Unterschied, wo die Daten nun tatsächlich herkommen.

Als Mediator speichern Peers Korrespondenzen oder Schema Mappings zwischen dem eigenen Schema und den Schemata anderer Peers. Mappings beschreiben äquivalente Elemente zwischen zwei Schemata und spezifizieren, wie Daten transformiert werden müssen, wenn sie von einem Peer zum anderen übertragen werden. Sie übernehmen damit im Grunde die Rolle von Wrappern in mediatorbasierten Systemen. Betrachtet man Peers als Knoten und Mappings als Kanten, bilden die Peers eines PDMS ein *Netzwerk von Datenquellen* (siehe Abbildung 4.10). Anfragen an einen Peer werden sowohl mit eigenen Daten beantwortet (sofern vorhanden) als auch über die durch Korrespondenzen verbundenen Peers. Wir werden die entsprechenden Techniken in den Abschnitten 5.2 und 6.4.6 näher erläutern.

Abbildung 4.10
Ein Peer-Daten-
Management-System

**Vor- und Nachteile der PDMS-Architektur**

PDMS befinden sich zwischen zwei Extremen: *Föderierte DBMS* mit einer festen Menge an Datenquellen und festen Zuordnungen zwischen einem globalen und mehreren lokalen Schemata auf der einen Seite und den hochdynamischen *P2P-Systemen* mit ständig wechselnden Beteiligten auf der anderen Seite. In den folgenden Abschnitten vergleichen wir den PDMS-Ansatz mit beiden Extremen.

Vergleich zu föderierten DBMS

Die Vorteile von FDBMS, wie Ortstransparenz und Schematransparenz, gelten auch für PDMS. Der wesentliche Vorteil von PDMS gegenüber FDBMS ist deren Flexibilität. Um eine neue Datenquelle hinzuzufügen, reicht es in der Regel, ein einziges Schema Mapping zu definieren, um in das Netzwerk des PDMS aufgenommen zu werden. Da sämtliche Quellen über Mappings miteinander verbunden sind, kann das Mapping von der neuen Datenquelle zu der ähnlichsten bereits vorhandenen Datenquelle abbilden. So mit fällt die Definition vergleichsweise leicht. Zudem beeinträchtigt das Hinzufügen und Entfernen einzelner Datenquellen das Gesamtsystem nur wenig: Es gibt kein globales Schema, das zu ändern wäre. Lediglich Mappings müssen erzeugt bzw. entfernt werden.

Ein wesentlicher Nachteil von PDMS ist die *komplexe Anfragebearbeitung über viele Peers* hinweg. Anfragen werden vielfach umformuliert, während sie von einem Peer zum nächsten gereicht werden. Dabei müssen Zyklen vermieden werden. Umgekehrt werden Daten auf dem Rückweg durch die Peers vielfach transformiert, was unter anderem einen schlechenden Verlust an Semantik und Qualität bedeuten kann.

Während einem FDBMS alle Datenquellen bekannt sind, die Antworten zu einer Anfrage beisteuern, ist dies in einem PDMS nicht der Fall. Der Peer, an dem die Anfrage gestellt wird, kennt zwar seine direkten Nachbarn und leitet bei Eignung die Anfrage an diese weiter. Diese können jedoch wiederum weitere Peers im Netzwerk befragen, um schließlich eine gebündelte Antwort an den ursprünglichen Peer zurückzusenden. Ohne aufwändigen Austausch von Metadaten weiß dieser nicht, welche Peers am Ergebnis beteiligt waren. Selbst wenn diese oft komplexen Informationen zusammen mit den Antworten geliefert werden, ist es dem anfragenden Peer im Gegensatz zum FDBMS nicht schon zum Anfragezeitpunkt möglich festzulegen, welche Peers Antworten liefern dürfen bzw. sollen.

Vergleich zu P2P-Systemen

PDMS sind als Erweiterung von FDBMS um P2P-Techniken entstanden. Es bestehen jedoch wesentliche Unterschiede zu einfachen P2P-Dateiaustauschsystemen, sowohl auf der logischen Ebene (Schemata, Anfragen, Anfragebearbeitung) als auch in ihrer physischen Ausprägung (Dynamik und Größe). Tabelle 4.2 fasst diese zusammen.

Vorteil: Höhere
Flexibilität

Nachteil: Sehr
komplexe
Anfragebearbeitung

**Unbekannte
Datenquellen**

	P2P	PDMS
Granularität	Vollständige Dateien	Objekte, Tupel, Attribute
Anfragen	Einfach: Suche nach Dateinamen oder Metadaten	Komplex: SQL, XQuery etc.
Wissen um andere Peers	Kein globales Wissen	Kenntnis von Nachbarn (Mappings)
Anfragebearbeitung	Fluten des Netzes, verteilte Hashtabellen etc.	Gezieltes Verfolgen geeigneter Mappings
Schema	Kein Schema (bzw. nur eine Relation)	Komplexes Schema
Anzahl Peers	Hunderttausende	Unter hundert
Dynamik	Hochdynamisch: Kurze Aufenthalte, spontanes Verlassen	Kontrolliert dynamisch: Lange Aufenthalte, An- und Abmeldung

Tabelle 4.2
Vergleich zwischen
P2P-Dateiaustausch und
PDMS

4.7 Einordnung und Klassifikation

Nach der Vorstellung diverser Architekturen betrachten wir nun Kriterien, die bei der Wahl einer Architektur zur Erstellung eines integrierten Informationssystems eine Rolle spielen. Dabei lässt sich keine der Entwurfsentscheidungen eindeutig der einen oder anderen Architektur zuordnen. Das Datenmodell kann beispielsweise unabhängig von der Architektur gewählt werden. Dennoch gibt es oftmals klare Tendenzen, die wir dann jeweils erwähnen.

Anschließend betrachten wir noch einmal Abbildung 3.1 von Seite 50 und klassifizieren die vorgestellten Architekturen innerhalb des aufgespannten Raums.

4.7.1 Eigenschaften integrierter Informationssysteme

Integrierte Informationssysteme lassen sich – neben ihrer Architektur – nach vielerlei Eigenschaften unterscheiden [41], von denen wir im Folgenden eine Reihe kurz vorstellen. Diese Eigenschaften sind nicht unabhängig voneinander. Beispielsweise ist zum Erzielen von Schematransparenz ein integriertes Schema nötig, eine lose Kopplung wird stets im Top-down-Entwurf erreicht, usw.

Strukturierungsgrad

In Kapitel 2 haben wir ausführlich die drei Strukturierungsgrade von Daten besprochen: Daten können strukturiert, semistruk-

turiert oder unstrukturiert sein. Integrationssysteme unterscheiden sich darin, welche Art von Daten sie integrieren können und wie das System selber die integrierten Daten präsentiert. Typische Konstellationen sind Systeme mit einem globalen Schema, die nur strukturierte Quellen aufnehmen, oder Systeme, die auf globaler Ebene nur Suchanfragen gestatten und alle Quellen als unstrukturiert auffassen.

Datenmodell

Wie schon in Abschnitt 2.1 beschrieben, liegen Daten in verschiedenen Datenmodellen vor. Es ist die Aufgabe des integrierten Informationssystems, ein kanonisches Datenmodell vorzugeben und die Daten der Datenquellen entsprechend zu transformieren bzw. transformieren zu lassen. In Abschnitt 2.1.4 haben wir bereits auf die Schwierigkeiten solcher Transformationen hingewiesen. Problematisch ist vor allem die Integration semi- oder unstrukturierter Daten in ein strukturiertes globales Schema.

Transformation von
Datenmodellen

Enge Kopplung:
globales Schema

Enge versus lose Kopplung

Eng gekoppelte integrierte Informationssysteme bieten Nutzern und Anwendungen ein integriertes Schema zur Anfrage an. Es ist dann die Aufgabe des integrierten Systems, vorhandene Heterogenität zu den Datenquellen zu überbrücken. Lose gekoppelte integrierte Informationssysteme hingegen stellen kein integriertes Schema zur Verfügung, sondern lediglich eine Multidatenbanksprache wie etwa SchemaSQL. Strukturelle Heterogenität muss vom Benutzer selbst überbrückt werden, entweder mittels Klau- seln in der Anfrage selbst oder durch eine nachträgliche Bearbeitung der Daten.

Bottom-up- oder Top-down-Entwurf

Man unterscheidet zwei grundlegende Strategien zum Entwurf integrierter Informationssysteme. Im Bottom-up-Entwurf steht die Notwendigkeit der vollständigen Integration einer *festen Menge bestehender und bekannter Datenquellen* im Vordergrund. Beim Top-down-Entwurf hingegen beginnt die Entwicklung mit einem spezifischen Informationsbedarf; relevante *Quellen werden dann nach Bedarf ausgewählt und dem System hinzugefügt*.

Bottom-up-Entwurf. Ein typisches Szenario für den Bottom-up-Entwurf ist der Wunsch eines Unternehmens, integrierten Zugriff

Integration einer festen Menge von Quellen

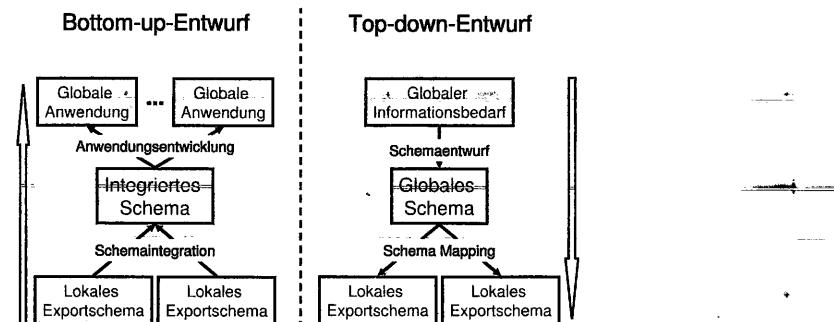
auf alle Unternehmensdaten zu erhalten. Dabei steht fest, welche Datenbanken zu integrieren sind (alle vorhandenen) und dass keine Teile dieser Datenbanken weggelassen werden sollen (vollständige Integration). Eine Änderung in Struktur und Menge der Datenquellen ist eher selten.

Der Bottom-up-Entwurf geht in der Regel einher mit einem strukturierten Schemaintegrationsprozess. Die Schemata der zu integrierenden Quellen werden sorgfältig analysiert und zu einem einheitlichen, integrierten Schema zusammengefasst, über das später auf alle Daten in den Quellen zugegriffen werden kann (siehe Abbildung 4.11).

Integration einer oftmals wechselnden Menge von Quellen

Top-down-Entwurf. Ein typische Szenario für den Top-down-Entwurf ist die Erstellung von Systemen, die eine wechselnde Menge von Webdatenquellen integrieren sollen. Es ist nicht im Vorhinein bekannt, welche Quellen wann zur Verfügung stehen und welches ihre Schemata sind; außerdem ändert sich die Menge geeigneter Quellen ständig. Die Integration vorhandener Schemata ist also unmöglich. Stattdessen wird zunächst ein globales Schema entworfen, das den Informationsbedarf potenzieller Nutzer und Anwendungen deckt. In einem zweiten Schritt werden relevante Quellen in das System eingebunden (siehe Abbildung 4.11). Die Dynamik eines solchen Systems mit oftmalshinzukommenden und ausfallenden Datenquellen erfordert spezielle Mechanismen, die wir in Abschnitt 6.4.1 erläutern werden.

Abbildung 4.11
Bottom-up- vs.
Top-down-Entwurf
integrierter
Informationssysteme



Virtuell oder materialisiert

Den Unterschied zwischen virtueller Integration, die die Daten bei den Datenquellen belässt und nur bei Bedarf integriert, und materialisierter Integration, die sämtliche Daten repliziert und Anfra-

gen dann auf einem zentralen Bestand bearbeitet, wurde bereits ausführlich in Abschnitt 4.1 diskutiert.

Transparenz

Vollständige Transparenz, also das Erscheinen eines integrierten Informationssystems als ein einzelnes, lokal vorhandenes, homogenes und konsistentes Informationssystem, ist das Hauptziel der Informationsintegration (siehe Abschnitt 3.4). Die verschiedenen Arten von Transparenz werden von Architekturen unterschiedlich gut erreicht:

Ortstransparenz: Alle besprochenen Architekturen bieten Ortstransparenz.

Verteilungstransparenz: Verteilungstransparenz bieten nur alle Systeme, die ein globales Schema zur Verfügung stellen. Auch PDMS bieten Verteilungstransparenz; nicht aber MDBMS.

Schnittstellentransparenz: Alle vorgestellten Architekturen bieten auch Schnittstellentransparenz, allerdings in unterschiedlichem Maße. Bei einer materialisierten Integration sind zur Anfragezeit Quellen überhaupt nicht mehr involviert, was eine vollständige Schnittstellentransparenz ermöglicht. Auch verteilte Datenbanken bieten diese trivialerweise, da in ihnen typischerweise keine Schnittstellenheterogenität geduldet wird. Integrierten MDBMS nur Datenbanken, wird ebenfalls Schnittstellentransparenz erreicht. Bei FDBMS, mediatorbasierten Systemen und PDBMS hängt es davon ab, welche Art Datenquellen integriert werden, da nicht alle Beschränkungen im Integrationssystem behoben werden können.

Schematransparenz: Ein Multidatenbanksystem zeichnet sich dadurch aus, dass kein integriertes Schema angeboten wird und somit keine Schematransparenz herrscht. Alle anderen Architekturen bieten Schematransparenz.

Anfrageparadigma

Integrierte Systeme unterscheiden sich in der Art, in der auf sie zugegriffen werden kann. Anfragemöglichkeiten reichen von SQL-Zugriff bis hin zu einfachen Suchanfragen mit Stichworten.

*Hohe Anforderungen
an System und
Quellen*

Canned queries

Navigierender Zugriff

Strukturierte Anfragen: Zum Stellen strukturierter Anfragen muss der Nutzer das globale Schema kennen. Dann kann die Struktur in einer Anfrage verwendet werden, wie etwa in SQL oder XQuery. Strukturierte Anfragen sind sehr mächtig und erlauben sehr komplexe und aufwändige Berechnungen. Deshalb wird oft der für den Nutzer zugängliche Funktionsumfang der Anfragesprache beschränkt, indem etwa keine Sortierung erlaubt wird oder Joins auf höchstens drei Relationen beschränkt werden.

Vorgegebene Anfragen: Um den Funktionsumfang der Anfragen weiter zu beschränken und somit den Anfragebearbeitungsaufwand besser planen zu können, kann ein integriertes Informationssystem eine Menge von Anfragen vorgeben (engl. *canned queries*), die eventuell durch einzelne Parameter verändert werden können. Ein Beispiel ist ein System zur Abfrage von Personendaten, das Nutzern lediglich erlaubt, den Namen einer gesuchten Person anzugeben, nicht aber zu einer gegebenen Telefonnummer den Inhaber des Anschlusses anzufragen.

Suchanfragen: Wenn die Struktur eines Informationssystems nicht bekannt ist, sind Suchanfragen oft die einzige Möglichkeit. Nutzer geben einen Suchbegriff ein und erhalten als Ergebnis eine (geordnete) Liste von Dokumenten oder Datenbankeinträgen, die das Suchwort enthalten. Beispiele solcher Systeme sind WWW-Suchmaschinen.

Browsing: Zugriff auf gespeicherte Informationen kann auch durch Browse ermöglicht werden, also durch *Navigation* in einem Informationssystem. Ein typisches Beispiel sind Produktkataloge, die im WWW zugreifbar sind.

Die Unterscheidung nach dem Anfrageparadigma kann man auch für Datenquellen treffen. Deren Möglichkeiten gelten natürlich indirekt auch für das integrierte System. Strukturierte Anfragen können beispielsweise nicht ohne weiteres an Datenquellen weitergereicht werden, die lediglich Suchanfragen erlauben.

Art der semantischen Integration

Semantische Integration vereint die Daten verschiedener Systeme zu einer integrierten Antwort. Der wichtigste Aspekt semantischer Integration ist die Erkennung verschiedener Repräsentationen gleicher Realweltobjekte (auch: Duplikate) und deren Integration zu einer einheitlichen, konsistenten Repräsentation. Man

kann bei der Art, wie dies vorgenommen wird, verschiedene Abstufungen unterscheiden:

Vereinigung: Die einfachste Methode ist die Vereinigung, bei der Daten unterschiedlicher Quellen lediglich aneinandergereiht werden. Eine semantische Integration der Daten im eigentlichen Sinne findet nicht statt.

Anreicherung: Auf der nächsten Stufe werden Daten nicht vollständig integriert, aber mit *Metadaten angereichert*. Damit kann man beispielsweise auf erkannte Widersprüche hinweisen und potenzielle Duplikate kennzeichnen. Die tatsächliche Integration bleibt aber dem Benutzer überlassen.

Fusion: Auf der höchsten Stufe erkennt und bereinigt die Datenumfusion Repräsentationen semantisch gleicher Dinge automatisch. In Abschnitt 8.3 werden wir verschiedene Varianten genauer besprechen.

*Arten semantischer
Integration*

Read-only oder read-&-write

Man kann integrierte Informationssysteme darin unterscheiden, ob sie schreibenden Zugriff (Einfügen, Ändern und Löschen) auf die Datenquellen zulassen oder nicht. Die große Mehrheit integrierter Systeme erlaubt keinen schreibenden Zugriff:

- Viele Schnittstellen, etwa HTML-Formulare, erlauben keinen schreibenden Zugriff.
- Das *Schreiben auf eine integrierte Sicht* birgt viele Probleme, insbesondere wenn die Sichten Joins oder Aggregationen enthalten [70].
- Das Schreiben auf ein integriertes Schema wirft die Frage auf, welche Datenquelle aktualisiert werden soll. Konzeptuell entspricht dies dem Schreiben auf eine Sicht, die eine Vereinigung (UNION) enthält.
- Die Unterstützung *systemübergreifender Transaktionen*, die für den schreibenden Zugriff notwendig sind, erfordert komplexe Protokolle und schränkt die Autonomie der Quellen ein [294].

*Schreibzugriff
erfordert spezielle
Techniken*

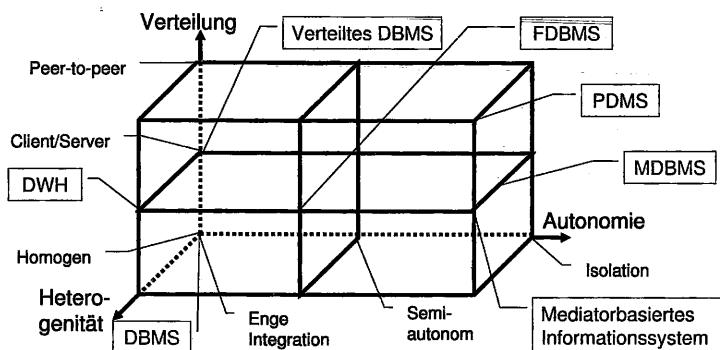
Im weiteren Verlauf des Buches betrachten wir lediglich lesenden Zugriff auf Datenquellen.

Einordnung aller Architekturen**4.7.2 Klassifikation integrierter Informationssysteme**

In Abbildung 4.12 greifen wir die in Kapitel 3 genannten drei Dimensionen integrierter Informationssysteme auf. Diese sind *Verteilung*, *Autonomie* und *Heterogenität*. Außerdem sind in der Abbildung folgende bereits vorgestellten Architekturen eingeordnet:

- Im Ursprungspunkt befinden sich klassische, **monolithische DBMS**. Alle weiteren Architekturen gehen mindestens von verteilten Datenquellen aus.
- Quellen in **verteilten Datenbanksystemen** sind natürlich verteilt, aber nicht autonom, und Heterogenität wird unterdrückt.
- Die Datenquellen von **Multidatenbanken** sind nicht nur verteilt, sondern auch hochgradig autonom. Gleichzeitig ist aber nur eine beschränkte Heterogenität möglich, da sie strukturierte Anfragen ausführen können müssen.
- **Föderierte Datenbanksysteme** gestatten hochgradige Heterogenität, geben dafür aber oft einen Teil ihrer Autonomie auf, um sich an der Föderation zu beteiligen.
- **Mediatorbasierte Informationssysteme** stellen den allgemeinsten Fall von Systemen dar, die noch zwischen Quellen und Integrationssystem unterscheiden.
- **Peer-Daten-Management-Systeme** schließlich machen ähnlich geringe Annahmen an ihre Komponenten wie FDBMS (wenngleich typische Instanzen von PDMS ein gemeinsames Datenmodell verwenden) und basieren darüber hinaus noch auf der größtmöglichen Verteilung der Daten auf Quellen.

Abbildung 4.12
Klassifikation der Architekturen integrierter Informationssysteme nach [220]



Es liegt die Frage nahe, ob nicht für jeden Datenpunkt in dem dreidimensionalen Raum eine Architektur denkbar wäre. Dies wurde von Özsu und Valduriez untersucht [220]. Sie folgern aber, dass nicht jede Variante einer sinnvollen Realisierung entspricht.

4.8 Weiterführende Literatur

Literatur für verteilte Datenbanken geben wir in den Abschnitten 6.7 (bzgl. der Anfrageoptimierung) und 10.5 (bzgl. kommerzieller Systeme). Referenzen für Multidatenbanksprachen finden Sie in Abschnitt 5.6.

Föderierte Datenbanken

Für das tiefere Studium föderierter Datenbanken empfehlen wir drei Quellen: Sheth und Larson geben in einem vielzitierten Überblicksartikel eine Übersicht über das Gebiet [262]. Özsu und Valduriez behandeln auch einige neuere Themen, wie Peer-Daten-Management [220]. Auf Deutsch ist das Lehrbuch von Conrad verfügbar [57], das Architekturen vorstellt, den Entwurfsprozess integrierter Datenbanken erläutert und Transaktionen in integrierten Systemen untersucht.

Mediatorbasierte Systeme

Neben den grundlegenden Artikeln von Wiederhold [298] bzw. Wiederhold und Genesereth [299] sind die Systeme TSIMMIS und Garlic die prominentesten Vertreter mediatorbasierter Systeme.

TSIMMIS ist ein mediatorbasiertes System für strukturierte und unstrukturierte Datenquellen [95]. Mit TSIMMIS wurden das XML-ähnliche Datenmodell Object Exchange Model (OEM) und die zugehörige Anfragesprache OEM-QL eingeführt. Weitere Eigenschaften von TSIMMIS sind die automatische Wrappergenerierung [222], Datenfusion [221] und die Einbindung beschränkter Quellen [181].

Der Kernaspekt von Garlic ist die Anfrageoptimierung mit heterogenen Datenquellen [45]. Die Fähigkeiten der einzelnen Datenquellen werden bei der Generierung von Anfrageplänen mit einbezogen, so dass ganze Teilpläne an die Datenquellen zur Ausführung geleitet werden können [246, 110]. Garlic ist der Vorläufer zu dem IBM-Produkt DiscoveryLink [111] und dem DB2 Information Integrator [135].

TSIMMIS

Garlic