IMSE Seminararbeit

Evolution der Datenmodelle

Gruppe 12

Schöndorfer Roman (IMS Era, CODASYL Era),
Nikitina Olena (Das Semantische Datenmodell, Semistrutkurierte Daten),
Rabizo Daniil (Relationale Ära, Die Entity Relationship Ära),
Fuchs Andreas (Einleitung, Die Objekt Orientierte Ära, Objektrelationale DBS)

7. Juni 2017

Inhaltsverzeichnis

1 Einleitung

Datenbankmodelle existieren schon seit den 1960iger Jahren als das Hierarchische IMS Modell präsentiert wurde. Seitdem haben sich die Datenbankmodell sehr weiter entwickelt. In dieser Arbeit werfen wir einen Blick auf die wichtigsten Evolutionsstufen von Datenbankmodellen, was sie zu der Entwicklung beigetragen haben warum einige davon gescheitert sind.

2 Relationale Ära

Heutzutage sind relationale Datenbanken eine der wichtigsten Teilen aller modernen Informationssystemen. In dem Gebiet der praktischen Programmierung haben sich neue Technologien, Plattformen der Realisation und Umgebungen rasant entwickelt, nichtsdestotrotz bleibt der klassische anerkannte Ansatz zur Modellierung und Projektierung der relationalen Datenbanken aktuell. Relationen Modell und ihre Grundlagen sind weit verbreitet bei der Erstellung von relationalen Datenbanken. Die Darstellung von Daten in Form von einer Gesamtheit von Tabellen hat es gestattet, dass mehrere Nachteile der früheren Datenbanksystemen vermieden werden, und neue Systeme mit der vereinfachten Interface-basierten Verwaltung erzeugt werden. In der heutigen Zeit ist eine große Anzahl von den Datenbanken-Server auf dem Prinzip von relationalen Datenmodell basiert, deswegen möchten wir das Konzept und Geschichte des relationalen Modells näher betrachten.

Das Relationalle Modell erschien infolge der Arbeit von dem britischen Wissenschaftler Edgar Codd bei IBM, seitdem er im Jahr 1970 einen Artikel ≪Relational Model of Data for Large Shared Data Banks≫ veröffentlicht hat. Er ist zum Begründer der modernen Datenbanktechnologie geworden, sodass er dafür im Jahr 1981 den Turing Award ausgezeichnet wurde. Im einleitenden Teil seines Artikels, kritisiert E. Codd hierarchisches Datenbanksystem (IMS) und Netzwerk-Datenbanksystem CODASYL, die nicht eine eindeutige Darstellung von Beziehungen auf der physischen Ebene gewährleisten können [1].

Auf einem einfachen Beispiel zeigt er, dass das gleiche Datenschema in fünf verschiedenen Arten der physischen Organisation von Einträgen dargestellt werden kann. Es wird dadurch komplizierter, dass es sowohl eine Fülle an Zeiger existieren soll, als auch die Einträge (Segmente) organisiert werden sollen, was wiederum die Abhängigkeit zwischen den Daten verstärkt. Edgar Codd versucht diese Abhängigkeit loszuwerden [2] indem er bietet, alle Daten einschließlich die Abhängigkeiten wischen den Objekten in Form von Beziehungen (zweidimensionalen Tabellen) zu speichern. Der Vorteil des relationalen Datenmodells ist seine Deutlichkeit, Klarheit und Einfachheit der physikalischen Implementierung auf einem Computer. Diese Einfachheit und Klarheit waren als Hauptgrund für den Anwender für die weitverbreitete Anwendung. Die Probleme mit der Effizienz von Datenverarbeitung war technisch ziemlich auflösbar. Die Hauptnachteile des relationalen Datenmodells sind unter anderem das Fehlen von Standardmitteln zur Identifizierung einzelnen Datensätzen, Schwierigkeit der Beschreibung der hierarchischen und Netzwerkabhängigkeiten. Das Datenschema besteht aus den einzelnen Tabellen. Jede Spalte dieser Tabellen entspricht einem Attribut (Element in den Netzwerkdatenbanken) und jede Zeile (Tupel) einem Datensatz. Die Reihenfolge der Zeilen kann beliebig sein. Tabellen in dem relationalen

Datenmodell haben eine Menge von Eigenschaften. Die wichtigsten sind [3]:

- Die Tabelle kann keine gleiche Datensätze beinhalten. Alle Tabellen, die diese Bedingung erfüllen heißen Relationen in der Mathematik.
- Die Spalten der Tabelle haben eine bestimmte Reihenfolge, die bei der Erzeugung von der Tabelle festgelegt wird. Die Tabelle kann auch keinen einzigen Dateneintrag beinhalten, aber es muss zumindest eine Spalte vorhanden sein.
- Jede Spalte hat einen eindeutigen Namen(innerhalb der Tabelle), und alle Einträge müssen einen einzigen Datentyp haben.
- Jede Spalte hat einen eindeutigen Namen(innerhalb der Tabelle), und alle Einträge müssen einen einzigen Datentyp haben.
- In der Spalten-Zeilen Kreuzung muss ein atomares Eintrag sein, die nicht aus einer Gruppe von Einträge besteht. Tabelle, die diese Bedingung erfüllen, heißen Normalisiert.
- Relation ist eine bestimmte Abhängigkeit zwischen den übereinstimmenden Werten in den Tabellen (Schlüssel aus einer Tabelle und Fremdschlüssel aus der anderen). Es gibt mehrere arten von Abhängigkeiten, und zwar 1:1, 1:n und n:m, die wir weiter behandeln.

Bei der Verwendung von 1:1 Abhängigkeit kann ein Datensatz in der Tabelle nicht mehr als nur einen verbundenen Datensatz in der Anderen Tabelle haben.Bei der 1:n Verbindung, kann einem Datensatz aus einer Tabelle mehrere Tupel aus der anderen Tabelle entsprechen, wobei jedem Datensatz aus der zweiten Tabelle kann nur ein Datensatz in der ersten entsprechen. Bei der n:m Abhängigkeit können mehrere Datensätze aus der ersten Tabelle den Datensätzen aus den zweiten entsprechen, und umgekehrt. Die n:m Beziehung wird praktisch nicht besonders oft in der relationalen Umgebung realisiert, aber sie lässt sich mit der Einführung von einer zusätzlichen Relation leicht lösen. Eine Menge von Datentypen, die von den relationalen Datenbanksystemen unterstützt werden, umfasst einfache (atomare) Typen, die wir aus den üblichen Programmiersprachen wissen, und zwar: logischer Typ (boolean), numerischer Typ (integer, float), Datum/Zeit sowie ein String Typ. Um das Konzept von Relationen Datenbanken besser zu zu verdeutlichen, haben wir ein einfaches Beispiel mit der Relation ≪Arbeiter≫, die eine eindeutig identifizierende Nummer hat, Vor- und Nachnamen, Geburtsdatum, Geschlecht und Dienststelle. Es gibt auch Relation ≪Abteilung≫, die einen eindeutigen Namen hat, Stiege, Stock, Raum und Telefonnummer. Mit der Relation ≪Arbeiter der Abteilung≫ lässt sich eine n:m Relation darstellen, in der einem Namen der Abteilung wird eine Nummer des Arbeiters zugewiesen.

Relation Arbeiter

Nummer	Vor- Nachname	Geburtsdatum	Geschlecht	Dienststelle
12	Mark Zuckerberg	14.05.1984	M	Leiter
427	James Lebron	30.12.1984	M	Analytiker
732	Bill Gates	28.10.1955	M	Analytiker
1376	Hillary Clinton	26.10.1947	W	Modellierung

Relation Abteilung

Name	Stiege	Stock	Raum	Telefon
Analyse	2	3	3-326	388-33-12
Modellierung	2	4	4-110	388-34-13

Relation Arbeiter in der Abteilung

Name	Nummer
Analyse	732
Analyse	427
Analyse	12
Modellierung	1376

Bestandteile der Tabellen:

Schlüsselattribut - eine Spalte in der Tabelle, deren Einträge alle unterschiedlich sind, weil ein Schlüsselattribut eindeutig identifizierend sein soll. In der Tabelle ≪Arbeiter≫, zum Beispiel, ist Nummer der Schlüsselattribut.

Fremdschlüssel - ein Attribut in der Relation, der auf eine Andere Relation verweist, um eine Beziehung zwischen den Tabellen darzustellen.

Zum Beispiel in der Tabelle ≪Arbeiter der Abteilung≫ sind Name und Nummer Fremdschlüssel, die entsprechend auf Arbeiter und Abteilungen verweisen. Relation-Datenbanksysteme haben zuerst keine große Verbreitung bekommen, da es sehr lange Zeit eine Meinung herrschte, dass es unmöglich wäre, eine effiziente Implementierung solcher Datenbanksystemen zu erreichen, aber die oben beschriebene Vorteile (Einfachheit, Klarheit) und allmähliche Akkumulation der Methoden und Algorithmen zur Organisation und Manipulation von Realtion-Datenbanken haben dazu geführt, dass relationale Datenbanksysteme von dem Weltmarkt alle früher benutze Datenbanken verdrängt haben. Damals im Jahr 1970 hatten die Entwickler erst die Theoretische Aspekte des relationalen Datenbanksystems entworfen, und kurz danach erschienen die erste Prototypen. Das erste Relationale Datenbanksystem war ≪System R≫, die in der Mitte von 1970 Jahren, im Rahmen eines Entwicklungsprojektes von IBM entwickelt wurde. Obwohl die Ideen von Edgar Codd für die Entwicklung von System R zugrunde gelegt wurden, hat er persönlich im Projekt nicht teilgenommen. Für die Arbeit mit Daten wurde eine spezifische Sprache erfunden, die SEQUEL hieß (Structed English QUEry Language) und dann im Laufe der Zeit in SQL umbenennt (Structed Query Language) wurde. Bei dem Schaffen von SQL haben die Entwickler von System R haben die Theorie eines relationalen Modelles ziemlich frei formuliert. Codd hat Ihm zusammen mit dem Programmierer Christopher J. Date verlassen, und ein unabhängiges Beratungsunternehmen gegründet, in dem sie ihre Arbeit mit dem relationalen Modell fortgesetzt haben. Wenn man auch heute die Werke von Christopher J. Date liest, merkt man seine kritische Stellung zu den mehreren Regeln von SQL, zum Beispiel zur Unzulässigkeit von NULL Werten [4].

Allerdings war SQL die erste Sprache, die einen relationalen Ansatz zur Datenstrukturierung realisiert hat. Die erste offizielle Standardsprache SQL - SQL-86 wurde im Jahr 1986 von dem American National Standards Institute (ANSI, American National Standards Institute) und unterstützt von der Internationalen Organisation für Normung (ISO, International Standardization Organizati-

on) angenommen. Dann gab es Modifikationen SQL-89, SQL-92 (oder SQL2), SQL-1999 (SQL3), SQL-2003, SQL 2006, SQL-2008. ≪System R≫ war vor allem ein Forschungsprojekt, und die Vermarktung der relationalen Technologie wurde durch die beiden anderen Entwicklungen - ≪Ingres≫ und ≪Oracle≫ durchgeführt. Ingres DBMS wurde im Universität von Kalifornienparallel zur System R entwickelt und durch das US-Militär unterstützt. Ingres ist zur Basis aller Technologien geworden, die im späteren Zeit in Sybase, Informix und MS SQL Server realisiert wurden. Nachdem Ingres eine weite Verbreitung auf dem Markt bekommen hat, hat eine Gruppe von Forschern aus der Kalifornischen Universität mit dem Entwurf von komplett neuen Postgres (Post Ingres) angefangen, das in 1995 herausgekommen ist. Den größten Einfluss auf kommerzielle Verbreitung hat DBMS Oracle ausgeübt, das von Relation Software, Inc. (jetzt Oracle Corporation) im Jahr 1979 hergestellt wurde. IBM hat auch in diesem Konkurrenzkampf teilgenommen, mit seinem DB2 [2].

Der nächste historische Schritt im Entwicklungsweg von relationalen Datenbanksystemen hat Firma Sybase getan, indem es im Jahr 1987 Sybase SQL Server hergestellt wurde, der dafür geeignet war, um eine hocheffiziente Client-Server Architektur auszubauen. Seit 1988 bis 1996 hat Sybase mit Microsoft zusammengearbeitet, wobei fast dasselbe Produkt realisiert wurde, unter zwei verschiedenen Namen ≪Sybase SQL Server≫ und ≪Microsoft SQL Server≫. In der Mitte von 1990 haben sich die drei größten DMBS Hersteller geformt: Sybase-Oracle-Informix. Die Liste von allen modernen relationalen Datenbanken hast mehr als 100 Bezeichnungen, die im Laufe der letzen 30 Jahre erfunden wurden, deswegen haben wir nur die wichtigsten erläutert, die den grössten Einfluss auf die Geschichte gemacht haben. In der heutigen Zeit eine der wichtigsten Aspekten der Kritik von relationalen Datenbanksystemen ist nicht die mangelhafte Effizienz, sondern eine charakteristische Beschränktheit (direkte Voraussetzung für Einfachheit) solcher Systeme bei der Verwendung in manchen spezifischen Gebieten, in denen äußerst komplexe Datenstrukturen benötigt werden.

3 Die Entity-Relationship Ära

Weiterhin werden wir einige Eigenschaften und Historische Entwicklung einer der populärsten semantischen Modellen - Entity-Relationship Modells betrachten. Das Chen Modell (Entity-Relationship) ist eine semantisches relationales Datenmodell, das alle Objekte der Realwelt auf einzelne unterschiedliche Entitäten teilt, die eine gewisse Abhängigkeit haben [5]. Andererseits hat dieses Modell mit den hierarchischen und Netzwerkdatenmodellen vieles gemeinsam, und wegen seiner Orientation kann dieses Modell als Verallgemeinerung den Netzwerk- und hierarchischen Datenmodellen dienen. Die Haupteinheiten in dem Entity-Relationship Modell sind sogenannte Entities (Entitäten) und Relationships (Beziehungen). Es ist eine Strukturdiagramm in Form eines Graphen, in dem Entitäten als Rechtecke und Beziehungen als Raute dargestellt sind, wobei alle Elemente mit den Linien verbunden sind. Es ist wichtig sich die Definition ≪Entität≫ klarzumachen, darüber hinaus Entität ist ein Objekt, das sich eindeutig identifizieren lässt und sich von den anderen Objekten unterscheidet. Eine konkrete Person, Firma, usw. kann zum Beispiel eine Entität sein. Eine Beziehung (Relationship) - ist eine Assoziation, die zwischen den Entitäten festgestellt wird. Entitäten haben bestimmte Attributen (Eigenschaften), die zur ge-

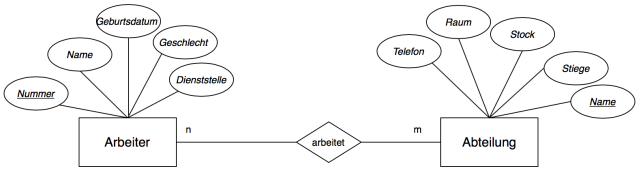
naueren Bestimmung, Identifizierung oder Charakterisierung dient. Es existiert auch eine Definition von Normalformen, genau so wie bei den relationalen Datenbanken, wobei der sinn dieser Definition ist sehr ähnlich. Die Normalformen von ER-Modellen erklären auch den Sinn der Normalisierung von relationalen Datenmodellen. Wir werden hier die ersten drei Normalformen erläutern:

- In der ersten NF werden sich wiederholende Attribute oder Gruppen von Attributen weggetan, das heißt die ≪Versteckte≫ Entitäten werden festgestellt und abgetrennt.
- In der zweiten NF werden alle Attribute beseitigt, die nur von einem Teil des Schlüssels abhängen. Dieser Teil des Schlüssels bestimmt eine eigene Entität.
- In der dritten NF werden alle Attribute beseitigt, die von den anderen Attributen abhängen, die nicht teil des Schlüssels sind. Diese Attribute gehören zu einer anderen Entität.

Zum Beispiel, kann eine Datenbank von einer Firma Informationen beinhalten, die für diese Firma von großem Interesse sind, zum Beispiel Arbeiter und Abteilungen, und eine Beziehung dazwischen. Eine Datenbank kann nicht alle Informationen inkl. komplette Beschreibung einer Entität oder Beziehung beinhalten. Bei der Entwicklung Einers ER-Modells muss man über folgende Informationen verfügen:

- Liste der Entitäten, die abgebildet werden müssen.
- Liste der Attributen dieser Entitäten.
- Beschreibung der Beziehungen zwischen den Entitäten.

Um das Konzept des ER Modells zu verdeutlichen, haben wir ein Beispiel aus der Beschreibung des relationalen Datenmodells genommen und für ER angepasst:



Wie man auf der Diagramm sieht, Nummer ist der Schlüsselattribut beim Arbeiter, und Name ist Schlüsselattribut bei der Abteilung. Es liegt eine n:m Beziehung vor, weil ein Arbeiter kann in mehreren Abteilungen arbeiten, und eine Abteilung kann mehrere Arbeiter beinhalten. In einem ER-Modell wird es zwischen 1:1, 1:n und n:m Beziehungen unterschieden, wie bei dem relationalen Datenmodell. In unserem Fall, ist es eine n:m Beziehung, weil mehrere Arbeiter können in mehreren Abteilungen arbeiten, und eine Abteilung kann mehrere Arbeiter beinhalten. Es gibt auch reflexive Beziehungen. Eine Erweiterung des Chen ER-Modells kann ein Konzept der Spezialisirung oder Generalisierung vorstellen, wobei es eine Abhängigkeit ≪Unterklasse-Oberkalsse≫ zwischen den Entitäten

hergestellt wird. Oberklassen und Unterklassen werden dafür verwendet, dass eine Wiederholung von gemeinsamen Attributen vermieden wird. Die Unterklasse erbt alle Attribute und Abhängigkeiten der Oberklasse. Man unterscheidet zwischen konzeptuellen und physischen ER-Modellen. Die konzeptuellen können die konkrete Eigenschaften einer Datenbank nicht völlig berücksichtigen, aber die physische werden nach der Regeln von Datenbank gebildet stellen ein Prototyp der Datenbank dar. Entitäten aus der ER-Diagramm werden zu den Tabellen, Attribute werden zur Spalten, wobei man die zulässige Datentypen und Namen von Spalten berücksichtigt. Die Beziehungen werden durch Migration der Schlüsselattributen von Eltern-Tabellen und Erzeugung von Fremdschlüsseln gebildet. Um die gegenseitige Abhängigkeit zwischen ER-Modell und relationalen Datenbankmodell zu zeigen, werden wir die wichtigsten Schritte einer Verwandlung von ER zum relatioanalen Modell angeben.

1. Jede einfache Entität wird in eine Tabelle umgewandelt. Einfache Entität ist eine Entität, die keine Unter- bzw. Oberklassen hat.

2. Jedes Attribut wird zu der möglichen Spalte mit dem entsprechenden Namen, und ein passendes Datentyp wird gewählt. Mehrwertige Attribute müssen zerlegt werden und Zusammengesetzte Attribute müssen als Neue Entitäten behandelt werden.

3. Die Schlüsselattribute werden zu dem Primary Keys der Tabelle. Wenn es mehrere Schlüsselattribute gibt, dann muss nur ein passendes ausgewählt werden.

4. Die n:m Beziehungen muss man als eigene Relationen dargestellen, mit den entsprechenden Primary und Foreign Schlüsseln , obwohl die 1:1 und 1:n muss man nicht als eine separate Tabelle machen, sondern nur in einer Relation, die in der Beziehung teilnimmt, entsprechendes Foreign Key angeben.

Es ist eine große Anzahl von grafischen Notationen für die Darstellung von ER-Modellen bekannt. Zum Vorbild ist aber eine Notation geworden, die von Charles Bachmann im Jahr 1969 bereitgestellt wurde [6], und für die Darstellung von Netzwerkmodell der Daten verwendet wurde. Der nächste Schritt wurde im Jahr 1975 von Peter Chen, der eine neue Notation vorgeschlagen hat [7], für die Darstellung von Elementen nicht nur der konzeptuellen Ebene, sondern auch des logischen. In 1986 hat Richard Barker die Ideen von Chen und Bachmann verwendet und ein erfand seine eigene Notation, die dann als Grundlage für die weitere Entwicklungen diente. Für die Entwicklung von ER-Modellen wird spezielles Software benutzt, die sogenannte CASE-mittel, die in der Mitte von 1980 Jahren erschienen. Die Abkürzung CASE hat zwei Auslegungen: Computer-Aided Software Engineering (Entwicklung der Programme mit Hilfe von Computer) und Computer-Aided System Engineering. Heutzutage versteht man unter CASE-programme eine Menge von Instrumenten, die für die Entwicklung der strukturellen Teilen von Programmen, Systemen und Prozessen geeignet ist. Unter dieser Menge gibt es auch Mittel für die Entwicklung von ER-Modellen unter Verwendung von den populärsten Notationen: Chen-Notation, Barker-Notation, Bachmann Notation, usw. Die meistbenutzte Werkzeuge sind CA Erwin Data Modeler (Hersteller - Computer Associates), Oracle Designer

(Oracle Corporation), Power Designer (Sybase), MS Visio, usw. Die meisten Produkte unterstützen oft Barker- und Bachmann Notationen, aber selten Chen Notation. Alle Werkzeuge außer Visio (Microsoft) unterstützen eine Umwandlung des konzeptuelles Schemas in eine physische Beschreibungssprache SQL eines beliebigen Datenbanksystem.

Der Fortschritt von konzeptuellen Darstellung der Daten geht immer weiter. Als Vorschlag für die Richtung der Weiterentwicklung des ER-Modells kann sein, dass es dem Benutzer ermöglicht wird, das konzeptuelles ER Modell zu korrigieren und ändern und die Datenbank wird alle Änderungen und Korrekturen automatisch berücksichtigen, und seine Struktur entsprechend ändern.

4 Das semantische Datenmodell

Die Notwendigkeit für semantische Datenmodelle wurde erstmals Mitte der 70er Jahre von der US-amerikanischen Luftwaffe als Ergebnis des Programms Ïntegrated Computer-Aided Manufacturing"(ICAM) erkannt. Ziel dieses Programms war es, die Fertigungsproduktivität durch die systematische Anwendung der Computertechnik zu erhöhen. Das ICAM-Programm ermittelte einen Bedarf an besseren Analysen- und Kommunikationstechniken für Personen, die an der Verbesserung der Fertigungsproduktivität beteiligt waren. Infolgedessen entwickelte das ICAM-Programm eine Reihe von Techniken, die als IDEF (ICAM Definition) Methoden bekannt sind. [8] Es gab die folgenden Methoden: IDEF0 wurde verwendet, um ein "Funktionsmodellßu erstellen, das eine strukturierte Darstellung der Aktivitäten oder Prozesse innerhalb der Umgebung oder des Systems ist. IDEF1 wurde verwendet, um ein Ïnformationsmodellßu produzieren, das die Struktur und die Semantik von Informationen innerhalb der Umgebung oder des Systems darstellt. IDEF1X war eine semantische Datenmodellierungstechnik. Es wurde verwendet, um ein grafisches Informationsmodell zu erzeugen, das die Struktur und die Semantik von Informationen innerhalb einer Umgebung oder eines Systems darstellt. Die Verwendung dieses Standards erlaubt den Aufbau von semantischen Datenmodellen, die dazu dienen können, das Management von Daten als Ressource, die Integration von Informationssystemen und den Aufbau von Computerdatenbanken zu unterstützen. IDEF2 wurde verwendet, um ein "Dynamikmodellßu produzieren, das die zeitvariablen Verhaltensmerkmale der Umgebung oder des Systems darstellt [9]. In den 1990er Jahren führte die Anwendung semantischer Modellierungstechniken zu den semantischen Datenmodellen der zweiten Art. Ein Beispiel dafür ist das semantische Datenmodell, das als ISO 15926-2 (2002) standardisiert ist und in die semantische Modellierungssprache Gellish (2005) weiterentwickelt wird. Die Definition der Gellish-Sprache ist in Form eines semantischen Datenmodells dokumentiert. Gellish selbst ist eine semantische Modellierungssprache, die verwendet werden kann, um andere semantische Modelle zu erstellen. Diese semantischen Modelle können in Gellish Datenbanken gespeichert werden, wobei es sich um semantische Datenbanken handelt. [10] Das semantische Datenmodell ist eine Methode zur Strukturierung von Daten, um es in einer bestimmten logischen Weise darzustellen. Es handelt sich um ein konzeptionelles Datenmodell, das semantische Informationen enthält, die den Daten und den zwischen ihnen liegenden Beziehungen eine grundlegende Bedeutung verleihen. Dieser Ansatz zur Datenmodellierung und Datenorganisation ermöglicht die einfache Entwick-

lung von Anwendungsprogrammen und auch für die einfache Pflege der Datenkonsistenz bei der Aktualisierung der Daten. Das semantische Datenmodell ist ein relativ neuer Ansatz, der auf semantischen Prinzipien basiert. In der Regel, Singular-Daten oder ein Wort vermittelt keine Bedeutung für den Menschen, aber verbundet mit einem Kontext erbt dieses Wort mehr Bedeutung. In einer Datenbankumgebung wird der Kontext von Daten oft hauptsächlich durch seine Struktur, wie seine Eigenschaften und Beziehungen mit anderen Objekten definiert. So wird in einem relationalen Ansatz die vertikale Struktur der Daten durch explizite referentielle Einschränkungen definiert, aber bei der semantischen Modellierung wird diese Struktur in einer inhärenten Weise definiert. [11] Typischerweise enthalten die Instanzdaten von semantischen Datenmodellen explizit die Beziehungen zwischen den verschiedenen Datenelementen, wie z. B. ¡befindet sich in¿. Um die Bedeutung der Tatsachen aus den Instanzen zu interpretieren, ist es erforderlich, die Bedeutung der Arten von Beziehungen (Relationstypen) zu kennen. Daher standardisieren semantische Datenmodelle typischerweise solche Relationstypen. Die zweite Art von semantischen Datenmodellen erstellen in der Regel semantische Datenbanken. Die Möglichkeit, die Bedeutung in semantischen Datenbanken einzubeziehen, erleichtert den Aufbau verteilter Datenbanken, die ermöglichen die Bedeutung aus dem Inhalt zu interpretieren. Dies bedeutet, dass semantische Datenbanken integriert werden können, wenn sie dieselben (Standard-) Relationstypen verwenden. Dies bedeutet auch, dass sie im Allgemeinen eine breitere Anwendbarkeit als relationale oder objektorientierte Datenbanken haben. [8] Ein semantisches Datenmodell kann grafisch durch ein Abstraktionshierarchiediagramm dargestellt werden, das Datentypen als Felder und deren Beziehungen als Linien anzeigt. Dies geschieht hierarchisch, so dass Typen, die auf andere Typen verweisen, immer über den Typen aufgeführt sind, auf die sie verweisen. Das macht es leichter zu lesen und zu verstehen. [11] In einem semantischen Datenmodell verwendet man die Abstraktionen: Klassifizierung - "instance_of" Beziehungen Aggregation - "has_a" Beziehungen Verallgemeinerung - "is_a" Beziehungen Die logische Datenstruktur eines Datenbankmanagementsystems (DBMS), ob hierarchisch, netz oder relational, kann die Voraussetzungen für eine konzeptionelle Definition von Daten nicht vollständig erfüllen, da sie im Umfang begrenzt und auf die Implementierungsstrategie des DBMS angewiesen ist. Daher die Notwendigkeit Daten aus einer konzeptionellen Sicht zu definieren, hat zur Entwicklung von semantischen Datenmodellierungstechniken geführt. Das heißt, Techniken, um die Bedeutung von Daten im Kontext ihrer Zusammenhänge mit anderen Daten zu definieren. Wie in der Abbildung dargestellt, die reale Welt, in Bezug auf Ressourcen, Ideen, Ereignisse, etc., ist symbolisch in physischen Datenspeichern definiert. Ein semantisches Datenmodell ist eine Abstraktion, die definiert, wie sich die gespeicherten Symbole auf die reale Welt beziehen. So muss das Modell eine wahre Darstellung der realen Welt sein. [8]

Nach Klas und Schrefl (1995) ist das "Ziel der semantischen Datenmodelle" die Erfassung von mehr Bedeutung von Daten durch die Integration von relationalen Konzepten mit leistungsstärkeren Abstraktionskonzepten, die aus dem Künstlichen Intelligenzfeld bekannt sind, zu erfassen. Die Idee besteht darin, hochrangige Modellierungsteile als integraler Bestandteil eines Datenmodells zu erstellen, um die Darstellung von realen Weltsituationen zu erleichtern. Semantisches Datenmodell unterscheidet sich von anderen Datenmodellen jedoch dadurch, dass es auf die Bereitstellung von der Bedeutung der Daten

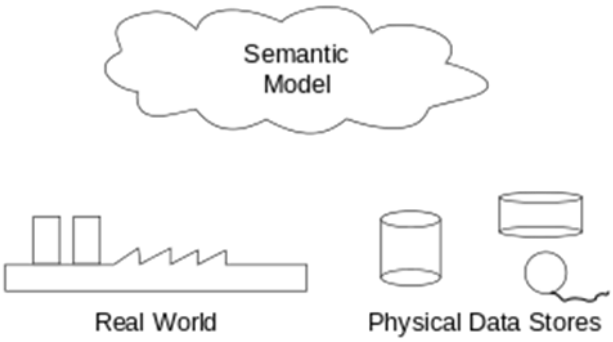


Abbildung 1: Sematisches Modell nach [9]

selbst konzentriert, anstatt nur oder primär auf die Beziehungen und Attribute der Daten. Semantisches Datenmodell bietet ein hochrangiges Verständnis der Daten, indem es weiter weg von den physikalischen Aspekten der Datenspeicherung abstrakt. [8] Ein semantisches Datenmodell kann für viele Zwecke verwendet werden. Einige wichtige Ziele sind: Planung von Datenressourcen: Ein vorläufiges Datenmodell kann verwendet werden, um eine Gesamtansicht der Daten zu liefern, die für die Ausführung eines Unternehmens erforderlich sind. Das Modell kann dann analysiert werden, um Projekte zu identifizieren und zu skalieren, um gemeinsam genutzte Datenressourcen zu erstellen. Aufbau von Shared-Datenbanken: Ein voll entwickeltes Modell kann verwendet werden, um eine anwendungsunabhängige Ansicht von Daten zu definieren, die von Benutzern validiert und dann in eine physikalische Datenbankgestaltung für eine der verschiedenen DBMS-Technologien umgewandelt werden kann. Neben der Generierung von Datenbanken, die konsistent und gemeinsam genutzt werden, können die Entwicklungskosten durch Datenmodellierung drastisch reduziert werden [9]. Auswertung der Vendor-Software: Da ein Datenmodell tatsächlich die Infrastruktur einer Organisation repräsentiert, kann die Vendor-Software gegen das Datenmodell eines Unternehmens ausgewertet werden, um mögliche Inkonsistenzen zwischen der implizierten Infrastruktur und der Art und Weise, wie das Unternehmen tatsächlich tätig ist, zu identifizieren. Integration bestehender Datenbanken: Durch die Definition der Inhalte existierender Datenbanken mit semantischen Datenmodellen kann eine integrierte Datendefinition abgeleitet werden. Mit der richtigen Technologie kann das resultierende Konzeptschema verwendet werden, um die Transaktionsverarbeitung in einer verteilten Datenbankumgebung zu steuern. Das U.S. Air Force Integrated Information Support System (I2S2) ist eine experimentelle Entwicklung und Demonstration von Technologie dieser Art, die auf eine heterogene DBMS-Umgebung angewendet wird. [10] Semantische Datenelemente sind ähnlich wie die Entitäten und Attribute, die wir in einem logischen oder physikalischen Datenmodell finden, wie zum Beispiel "Kunde", "Produkt", "Kreditlimit", "Netzwerk Verkäufe" und so weiter. Was der semantische Modellierer jedoch ansprechen muss, ist der Kontext des Begriffs, des Datenelements und wie er sich auf die in den Rechensystemdaten vorhandenen Datenelemente bezieht. Zum Beispiel ist ein Kunde ein Einzelner - der Einkaufsagent - oder ein Unternehmen? Was in einigen Kontexten vielleicht als "Perspektive" bezeichnet werden könnte, könnte in anderen als "Kunde" be-

zeichnet werden. Ist ein Kunde ein Großhändler oder ist er der Endverbraucher? Ist der Großhändler Kunde auch Kunde genannt? Die Antwort auf diese Fragen ist wahrscheinlich "es hängt davon ab." Und das ist die richtige Antwort, denn es hängt davon ab. Es hängt davon ab, wer fragt und warum. Die Verkaufsabteilung von einer Gesellschaft kann eine klare Linie zwischen Kunden (Käufer) und Interessenten machen. Der semantische Modellierer muss bohren und die Nuance jeder Perspektive erfassen und muss kämpfen, um mit den Geschäftsbenutzern zu arbeiten, um eine Namenskonvention oder Syntax zu entwickeln, die Klarheit bietet. Alle Perspektiven sind im semantischen Modell dargestellt. [11]

5 Die Objekt Orientierte Ära

5.1 Motivation:

Seit dem Aufkommen von Datenbanksystemen ist es üblich die (persistenten) Daten von der Applikation zu trennen. Dadurch werden persistente Daten, üblicherweise durch ein DBMS verwaltet, anders behandelt als programmeigene Daten die nur während der Laufzeit existieren. Um also auf persistenten Daten zu arbeiten und diese zu manipulieren musste also eine Sprache in die andere eingebunden werden - üblicherweise durch Aufrufe aus der Applikation an das DBMS.

Dies hatte zum einen den Nachteil dass Applikationsentwickler sowohl in der Programmiersprache des Programms als auch in der Abfragesprache des DBMS vertraut sein mussten.

Zusätzlich führte die fortschreitender Entwicklung von Programmierparadigmen und dem Durchbruch Objektorientierter Programmierung (OOP) zu einem sogenannten *impedence mismatch* [12]. Dies ist wenn sich die Programmiersprache der Applikation und des DBMS auf konzeptioneller und struktureller weiße Unterscheiden und die damit verbunden Schwierigkeit Daten in einer Form in der anderen zu repräsentieren.

Im speziellen können im *Object-Relational Impedance Mismatch* folgende Schwierigkeiten ausgemacht werden [13]

Struktur: Eine Klasse besitzt sowohl eine beliebige Struktur die möglicherweise Pointern/Referenzen oder benutzerdefinierten Datentypen enthält als auch beliebige Semantik, definiert durch ihre Methoden. Sie ist möglicherweise auch Teil einer Klassenhierarchie. Ein Tupel besteht nur aus reinen Werten (einfachen Datentypen) und ist kein Teil einer Hierarchie.

Instanz: Ein Objekt ist eine Instanz einer Klasse. Ein Tupel ist eine Wahrheitsaussage über ein Universum.

Kapselung: Der zugriff auf die Daten eines Objekts wird über Methoden geregelt. Keine solche Regelung existiert im Relationalen Modell und Daten können beliebig verändert werden.

Identität: Jedes Objekt besitzt eine Identität unabhängig ihres Zustands. Ein Tupel ist über ihren Zustand definiert.

Arbeitsweise: Ein Objektmodell ist ein Netzwerk interagierender Objekte die über Methoden kommunizieren während das das Relationale Modell als Menge betrachtet werden kann und prozedural abgearbeitet werden.

Organisatorisch: Die Besitzer der Applikation und der Datenbank können verschieden sein. Das erschwert es bei Änderungen das Klassenmodell und das Datenbankschema synchron zu halten.

5.2 Objektorientierte Datenbanksysteme (OODBS)

Um diese Probleme zu beheben war es naheliegend den Umweg von der Applikation zu den persistenten Daten über ein (externes) DBMS zu ersparen und einen weg zu Entwickeln der es erlaubt die Objekt in der Programmiersprache selbst persistent zu machen. Deswegen werden die OODBS auch oft *Persistente Programmiersprachen* genannt.

5.2.1 Frühe Ansätze

Bereits in den späten 1970ern wurde mit ersten Prototypen wie Pascal-R oder Rigel experimentiert. Es dauerte jedoch bis Mitte der 1980er dass verschiedene kommerzielle Anbieter persistente Programmiersprachen am Markt anboten. Diese richteten ihre Aufmerksamkeit meist auf den Technischen und Planerischen Bereich. Aufgrund dieser markttechnischen Ausrichtung lag der Fokus dieser frühen OODBS wenig auf Abfragesprachen und Transaktionsmanagement und mehr auf der Performance. Obwohl einige Umsetzungen Innovative Architekturen boten um dieses Ziel umzusetzen fanden persistente Programmiersprachen nur geringe Verbreitung vornehmlich bei CAD-Anwendungen.

Grund für das Scheitern waren das fehlen von Standards und die Abhängigkeit von der eingesetzten Programmiersprache. Applikationen die nicht in der selben Programmiersprache geschrieben sind haben auch keinen Zugriff auf die persistenten Daten. Für die Kunden schien der Vorteil keine Schnittstelle Entwickeln zu müssen als zu geringer Gewinn [**?**] um die Nachteile aufzuwiegen.

Um einige der Nachteile zu beseitigen schickte sich die *Object Database Management Group* (ODMG), ein Zusammenschluss verschiedener Objektorienter Datenbanksystemanbieter, in den 90ern dazu an einen Standard für Objetorientierte Datenbanksysteme inklusive einer an SQL angelehnten Abfragesprache *Object Query Language* zu etablieren. Im Jahr 2000 erschien der Standard 3.0 woraufhin die ODMG als ihr Ziel als erreicht ansah und sich 2001 auflöste [**?**].

Dennoch haben bis heute OODBS nur eine geringe Verbreitung.

5.2.2 Schwierigkeiten

Einer der größten Hindernisse eine Programmiersprache mit DBMS Funktionalität zu erweitern ist, dass die Compiler mit dieser Funktionalität erweitert werden müssen. Dies betrifft nicht nur Compiler verschiedener Sprachen, aber auch Compiler derselben Sprache. Zum Beispiel sollte C++ mit DBMS Funktionalität erweitert werden müssen Compiler wie GCC oder Intel C++ Compiler jeweils dies Unterstützen. Eine solche Unterstützung hat sich bis heute nicht durchgesetzt und die Persistenz muss mit anderen Mitteln - zum Beispiel durch Typbibliotheken wie es die ODMG vorsieht - bereitgestellt werden.

Ein weitere nicht zu unterschätzende Schwierigkeit stellt die Komplexität von Programmiersprachen dar. Aufgrund dieser Komplexität ist es einfach das Programmfehler entstehen die die Datenbestände ungewollt manipulieren. In

einer Welt in der Daten oft einen hohen unternehmerischen Wert darstellen ist dies ein deutliches Hindernis.

5.3 Objektrelationale Abbildung

Ein Ansatz den Impedance Missmatch zu umgehen ist, die Abbildung von dem Objetkorientierten auf das Relationale Modell dem Entwickler aus der Hand zu nehmen und durch ein Framework zu automatisieren. Dies löst zwar nicht direkt den impedance mismatch, diese ist jedoch für den Softwareentwickler im idealen Fall nicht mehr sichtbar.

Versuche einer Objektrelationealen Abbildung (ORA) gehen zurück bis auf 1984 [13], es dauerte jedoch bis in die 90er - zum teil vorangetrieben durch den geringen Fortschritt in OODBS - bis erste Produkte wie *TopLink* auf dem Markt erschienen. Seitem haben sich eine Vielzahl an Produkten für verschiedene Programmiersprachen wie zum Beispiel Hibernate für Java oder das Entity Framework für .NET etabliert.

5.3.1 Schwierigkeiten

Entgegen der Häufigen Annahme kann Objektrelationale Abbildung die Abbildung von der objektorientierten auf das relationale Modell nicht vollständig automatisieren.

Auch wenn Objektrelationale Abbildungen ein hilfreiches Werkzeug für den Entwickler sein können, ist die Verwendung solcher weder trivial noch vollständig automatisiert. Die Verwendung von ORA ohne die Auswirkungen auf die Datenbankperformance zu berücksichtigen kann zu Performance Antipattern führen die langsame Reaktionszeiten oder gar den Stillstand eines System verursachen können [14].

Erschwerend kommt hinzu dass für die verschiedenen ORAs keine einheitlich Standards festgelegt sind. So können gleiche oder ähnliche Konzepte in den verschiedenen ORAs unterschiedlich benannt sein und sie können sich sehr in ihrer Funktionalität unterscheiden [15].

5.4 Beispiel ODMG C++

Angenommen wir haben eine Klasse Person mit den Attributen Name und Adresse sowie eine von der Person abgeleitet Klasse Student mit dem zusätzlichen Attribut matrikelnummer. Es existiert noch eine weitere Klasse Universität und jeder Student ist auf einer Universität eingeschreiben welches durch eine Refernz auf Universität abgebildet wird, siehe Abbildung 2

Für die Klasse die Persistent werden soll muss die ODMG-Klasse d_Object als Superklasse eingebunden werden. Wie Üblich wird diese Eigenschaft an die abgeleiteten Klassen vererbt, es ist also nicht nötig die Klasse d_Object noch einmal einzubinden. Die Persistenz wird dabei während der Erzeugung deklariert. Objekte können andere Objekte über einen smart pointer *d_Ref* referenzieren. Alle primitiven C++ Datentypen bis auf union, bit fields und references werden explizit unterstützt und einige strukturierte Literale wie zum Beispiel String sowie Collection Typen wie Arrays werden ebenso unterstützt. Algorithm 1 zeigt unser Beispiel und wie dieses in C++ mit der Object Definition Language (ODL) persistent gemacht werden könnte **??**, **??**.

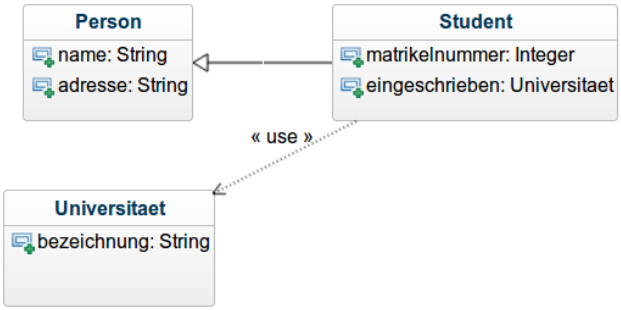


Abbildung 2: Beispiel Objekt Orientierung

```
class Person : public d_Object {
    d_String name;
    d_String adresse;
}
class Student : Person {
    d_Long matrikelnummer;    // d_Long ist äquivalent zu unsigned
     integer
    d_Ref<Universitaet> eingeschrieben;
}
    class Universitaet : public d_Object { d_String bezeichnung;
}
```

Algorithm 1: C++ Object Definition Language

Die Object Query Language (OQL) erlaubt den Zugriff auf die Daten in einer OODBS und ist syntaktisch an SQL angelehnt, bietet jedoch im Vergleich zu SQL:1992 den Vorteil dass man flexibler auf belieb strukturierten Objekten arbeiten kann [16]. Algorithm 2 zeigt eine einfache OQL abfrage.

```
SELECT s.name
FROM s IN studenten
WHERE universitaet = "Uni Wien";
```

Algorithm 2: Object Query Language

5.5 Nachwirkungen

Mit der divergierenden Entwicklung von Programmierparadigmen und Datenbanksystemen, im besonderen die Objektorientierte Programmierung und das Relationale Schema, kam es zu einem deutlichen *Impedance Missmatch*. OODBS machten sich daran dieses Problem zu beheben. Trotz der Anstrengungen konnten sich diese jedoch nicht am Markt durchsetzen der nach wie vor vom Relationalen Datenbanken dominiert wird. Zum einen scheinen die Vorteile als zu gering, zum anderen liegt ihr Fokus weniger auf Geschäftsdaten die jedoch eine mächtige Rolle am Markt spielen. Da die Entwicklung von OODBS in den letzten Jahren stagnierte machte wurde sich zunehmend an die Objektrelationale Abbildung gewandt um das Impedance Mismatch zu beheben. Zwar sind auch diese frei von Problemen und Kritik, dennoch freuen sie sich einer derzeit hohen

Verbreitung.

6 Objektrelationale Datenbanken

Frühe Relationale Systeme wie INGRES waren hauptsächliche an Geschäftsdaten interessiert und boten nur sehr einfache aber dem Markt entsprechende Datentypen wie Ganzzahlen, Gleitkommazahlen oder Strings sowie nur einfache Operationen. Weiters unterstützten sie nur B-Bäume um in einer Relation zu suchen.

Dies ist jedoch nicht für jeden Markt ausreichend. Geographic Information Systems (GIS) ist ein solcher Markt. In GIS sind Koordinaten als Längen- und Breitengrad angegeben. Eine Koordinate zu finden erfordert somit eine zweidimensionale suche. Ein B-Baum ist jedoch nur in der Lage über eine Dimension zu suchen und ist wenig effizient im Vergleich zu auf mehrdimensionale Suche ausgelegte Bäume wie kd-Trees. Schwieriger noch gestaltet sich eine Suche auf eine Fläche für die es in INGRES keine adäquate Operation gibt da diese von eindimensionalen Daten ausgeht [17].

Da jeder Markt eigene Anforderungen stellt ist es wenig zielführend die angeforderten Datentypen und Operationen fest zu codieren - ausgenommen der Hersteller beabsichtigt eine Spezialisierung auf diesen Markt. Anstelle sollte es möglich sein dass Kunden die Datenbank möglichst flexibel selbst an ihre Bedürfnisse anpassen können.

6.1 Objektrelationale Datenbanksysteme

Dieses Ausgangslage ist der der OODBS nicht unähnlich da hier wie dort sich um einen Impedance Mismatch handelt. Objektrelationale Datenbanksysteme (ORDBS) sind jedoch im Gegensatz zum "revolutionären" OODBS Ansatz ein "evolutionärer" Ansatz der versucht das Relationale Modell um die folgenden Eigenschaften zu erweitern [16]:

- **Große Objekte (Large Objects):** Datentypen die es erlauben auch sehr große Attributwerte zu speichern. Eigentlich handelt es sich hier nur um "reine" Werte aber dennoch werden sie vielfach den Objektrelationalen Datenbanken hinzugerechnet.
- **Mengenwertige Attribute:** Erlaubt es einem Attribut einen Menge von Werten zuzuordnen.
- **Geschachtelte Relationen:** Erlaubt Attribute die selbst wieder Relationen sind.
- **Typdeklaration:** Erlaubt es dem Benutzer Anwendungsspezifische Typen anzulegen die dann als Attributtypen verwendet werden können. Dies erlaubt komplexe Objektstrukturen.
- **Referenzen:** Attribute können direkte Referenzen auf Tupel beziehungsweise Objekte derselben oder einer fremden Relation sein. Somit ist man nicht mehr nur auf die Verwendung eines Fremdschlüssels beschränkt.
- **Objektidentität** Tupel können eindeutig identifiziert werden und besitzen eine Identität unabhängig ihres Zustands.

- **Pfadausdrücke:** Referenzattribute machen es erforderlich Pfadausdrücke in der Anfragesprache zu unterstützen
- **Vererbung:** Erlaubt es Generalisierungen beziehungsweise Spezialisierungen umzusetzen.
- **Benutzerdefinierte Operationen:** Erlaubt es den Daten auch eigene Operationen zuzuordnen.

6.1.1 Frühe Ansätze

Postgres war eine der ersten größeren Objektrelationalen Datenbankdesigns. Es gab auch schon zuvor Versuche die rein Relationale Datenbank INGRES zu erweitern welche jedoch eher in einer Zerstückelung des ursprünglichen INGRES-Designs resultierte. Postgres bot in ihrem ursprünglichen Design im Vergleich eine unter anderem verbesserte Unterstützung komplexer Datentypen sowie die Möglichkeit Benutzerdefinierter Datentypen, Operationen und Methoden ohne jedoch vom Ursprünglichen Relationalen Design abzuweichen um die Kompatibilität zu wahren [18]. Später kamen noch Vererbung, Referenzen und Mengen hinzu [19].

Manch andere Datenbanksysteme wie Sybase hatten benutzerdefinierte Funktionen in der Form von *Stored Procedures* ermöglicht welche den Vorteil boten die Anzahl an Transaktionen zu minimieren.

6.2 Schwierigkeiten

So wie auch OODBS hatte auch der Objektrelationale Ansatz das Problem dass sich zunächst keine Standards durchgesetzt haben. Dies ist auch heutzutage noch wahr obwohl der SQL:1999 einen Standard verabschiedet hätte, den aber viele Datenbankanbieter noch nicht realisiert haben oder nicht einhalten und stattdessen eigen Methoden bieten.

Eine weitere Ursache die den Durchbruch verzögerte war dass, ebenso wie bei OODBS, es mit zusätzlichen Features auch zu zusätzlicher Komplexität, sowohl für den Hersteller als auch für den Benutzer, kommt. Der Hersteller muss nicht nur die Funktionalität bereitstellen sondern sie soll auch möglichst Performant sein und Abfragen optimieren können. [20] fand dass manche Abfragen, insbesondere jene die Mengen involvieren, auf Objektrelationalen Datenbank weniger Performant sind als jene die dieses Verhalten auf Relationalen Datenbanken simulieren. Aber auch für den Applikationsentwickler ist die Zunahme an Optionen weniger überschaubar. Insbesondere werden viele der Features kaum verwendet und ein großer Teil der Anwendungsentwickler verwendet die üblichen Features der Relationalen Datenbank und bildet diese auf Applikationsebene auf das objektorientierte Struktur ab. Dies gilt sogar für Objektrelationale Abbildung [21].

6.3 Beispiel SQL:1999

Wir Verwenden das selbe Beispiel wie in Abschnitt 5.4 und wollen diese nun auf einen Objektrelationale Datenbank nach dem SQL:1999 Standard abbilden. Mittels CREATE TYPE kann ein benutzerdefinierter Typ (User Defined Type UDT) angelegt werden. Diese kann auch von einer Oberklasse erben indem

mit dem Schlüsselwort UNDER die Oberklasse angegeben wird [22]. Die UDT kann auch Methoden enthalten und es können andere UDTs referenziert werden. Algorithm 3 zeigt

```
CREATE TYPE person AS
    (name varchar(30), adresse varchar(50));

CREATE TYPE universitaet AS
    (bezeichnung varchar(20))
    METHOD anzahl_studierende( ) RETURNS int;
        --Methode die später definiert wird

CREATE TYPE student UNDER person AS
    (matrikelnummer int,
    eingeschrieben REF(universitaet))    -- Referenz auf universitaet
```

Algorithm 3: Abbildung Klassen mit SQL:1999

6.4 Nachwirkungen

Aufgrund des evolutionären Ansatzes des Objektrelationalen Designs fand es einen höheren Anklang als der revolutionäre Ansatz der Obejktorientierten Datenbanksysteme. Das Postgres Design wurde später Implementiert und ständig erweitert und ist heutzutage unter dem Namen PostgreSQL weit verbreitet.

Der SQL:1999 Standard hatte das Ziel einen standardisiertes Objekt-Relationales Datenmodell einschließlich Anfragesprache zu definieren. Viele dieser Standards sind jedoch bisher von den Datenbanksystemanbietern nicht oder nicht dem Standard entsprechend umgesetzt [16]. Zumindest bieten aber alle großen Anbieter Relationaler Datenbanken manche Features des Objektrelationalen Designs, wie zum Beispiel stored procedures, in ihren Produkten an.

7 Semistrukturierte Daten

Semi-strukturierte Daten ist eine Form von strukturierten Daten, die nicht mit der formalen Struktur von Datenmodellen übereinstimmen, die mit relationalen Datenbanken oder anderen Formen von Datentabellen verknüpft sind, aber dennoch Tags oder andere Marker enthält, um semantische Elemente zu trennen und Hierarchien innerhalb der Daten zu erzwingen. Daher ist es auch als selbstbeschreibende Struktur bekannt. Semi-strukturierte Daten sind Daten, die keine Rohdaten oder typisierte Daten in einem herkömmlichen Datenbanksystem sind. Es ist strukturierte Daten, aber es ist nicht in einem rationalen Modell, wie eine Tabelle oder ein Objekt-basierte Grafik organisiert. Viele im Web gefundene Daten können als semi-strukturiert beschrieben werden. Die Datenintegration nutzt vor allem halbstrukturierte Daten. [23] In halbstrukturierten Daten können die Entitäten, die zu derselben Klasse gehören, unterschiedliche Attribute haben, obwohl sie zusammen gruppiert sind. Die Reihenfolge der Attribute ist nicht wichtig. Seit dem Aufkommen des Internets gibt es immer mehr strukturierte Daten, bei denen Volltextdokumente und Datenbanken nicht mehr die einzigen Datenformen sind und unterschiedliche Anwendungen ein Medium für den Informationsaustausch benötigen. In objektorientierten Datenbanken

findet man oft halbstrukturierte Daten. [24] Arten von Semi-strukturierten Daten

7.1 XML

XML, andere Markup-Sprachen, E-Mail und EDI sind alle Formen von halbstrukturierten Daten. OEM (Object Exchange Model) wurde vor XML als Mittel zur Selbstbeschreibung einer Datenstruktur erstellt. XML wurde von Webdiensten, die mit SOAP-Prinzipien entwickelt wurden, popularisiert. Einige Arten von Daten, die hier als ßemi-strukturiert", vor allem XML, beschrieben werden, leiden unter dem Eindruck, dass sie nicht in der Lage sind, strukturelle Strenge auf der gleichen funktionalen Ebene wie Relational Tables und Rows zu erhalten. In der Tat hat die Sicht von XML als inhärent halb-strukturiert (bisher wurde es als ünstrukturiert"bezeichnet) seine Verwendung für eine erweiterte Palette von datenzentrischen Anwendungen behindert. Sogar Dokumente, die normalerweise als der Inbegriff der Semi-Struktur gedacht sind, können mit praktisch der gleichen Strenge wie Datenbank-Schema entworfen werden, durch das XML-Schema erzwungen und von kommerziellen und kundenspezifischen Softwareprogrammen verarbeitet werden, ohne ihre Benutzerfreundlichkeit zu reduzieren. [25] Angesichts dieser Tatsache könnte XML als "flexible Struktur"bezeichnet werden, die in der Lage ist, menschlich-zentrische Strömung und Hierarchie sowie eine sehr rigorose Elementstruktur und Datentypisierung zu ermöglichen. Das Konzept von XML als "menschlich lesbar"kann jedoch nur so weit genommen werden. Einige Implementierungen / Dialekte von XML, wie die XML-Darstellung des Inhalts eines Microsoft Word-Dokuments, wie sie in Office 2007 und späteren Versionen implementiert sind, verwenden Dutzende oder sogar Hunderte von verschiedenen Arten von Tags, die eine bestimmte Problemdomäne widerspiegeln - im Fall von Word, Formatierung auf der Charakter- und Absatz- und Dokumentenebene, Definitionen von Stilen, Einbeziehung von Zitaten usw. - die in komplexer Weise ineinander verschachtelt sind. Das Verständnis eines solchen XML-Dokuments, ist unmöglich, ohne ein sehr tiefes vorheriges Verständnis der spezifischen XML-Implementierung, zusammen mit Hilfe von Software, die das verwendete XML-Schema zu verstehen. Dieser Text ist nicht mehr "menschlich verständlichäls ein Buch. Die Tags sind Symbole, die bedeutungslos für eine Person sind, die mit der Domäne nicht arbeitet. [26]

7.2 JSON

JSON oder JavaScript Object Notation ist ein offenes Standardformat, das menschlich lesbaren Text verwendet, um Datenobjekte aus Attributwertpaaren zu übertragen. Es wird hauptsächlich verwendet, um Daten zwischen einem Server und einer Webanwendung als Alternative zu XML zu übertragen. JSON wurde von Web-Services entwickelt, die unter Verwendung von REST-Prinzipien entwickelt wurden. Es gibt eine neue Zucht von Datenbanken wie MongoDB und Couchbase, die Daten nativ im JSON-Format speichern und die Profis der halbstrukturierten Datenarchitektur nutzen. [23]

7.3 Vor- und Nachteile der Verwendung eines halbstrukturierten Datenformats

7.3.1 Vorteile

Programmierer, die Objekte von ihrer Anwendung auf eine Datenbank aufhalten, müssen sich nicht um objekt-relationale Impedanzfehlanpassung kümmern, sondern können oft Objekte über eine leichte Bibliothek serialisiert werden. Die Unterstützung für verschachtelte oder hierarchische Daten vereinfacht oft Datenmodelle, die komplexe Beziehungen zwischen Entitäten darstellen. Die Unterstützung für Listen von Objekten vereinfacht Datenmodelle durch Vermeidung von unordentlichen Übersetzungen von Listen in ein relationales Datenmodell. Es kann die Information einiger Datenquellen darstellen, die nicht durch Schema eingeschränkt werden können. Es bietet ein flexibles Format für den Datenaustausch zwischen verschiedenen Arten von Datenbanken. Es kann hilfreich sein, strukturierte Daten als semi-strukturierte (für Browsing-Zwecke) zu sehen. Das Schema kann leicht geändert werden. Das Datenübertragungsformat kann tragbar sein. [25]

7.4 Nachteile

Das traditionelle relationale Datenmodell hat eine populäre und fertige Abfragesprache, SQL. Anfällig für "Müll in, Müll raus"; Durch das Entfernen von Beschränkungen aus dem Datenmodell gibt es weniger Vorbedenken, die notwendig sind, um eine Datenanwendung zu betreiben. Das halbstrukturierte Modell ist ein Datenbankmodell, bei dem es keine Trennung zwischen den Daten und dem Schema gibt und die Menge der verwendeten Strukturen vom Zweck abhängt. Der primäre Kompromiss, der bei der Verwendung eines semi-strukturierten Datenbankmodells gemacht wird, ist, dass Abfragen nicht so effizient gemacht werden können wie in einer beschränkten Struktur, wie zum Beispiel im relationalen Modell. Typischerweise werden die Datensätze in einer halbstrukturierten Datenbank mit eindeutigen IDs gespeichert, auf die mit Zeigern auf ihren Speicherort verwiesen wird. Dies macht Navigations- oder Pfad-basierte Abfragen recht effizient, aber für die Suche über viele Datensätze (wie es in SQL typisch ist), ist es nicht so effizient, weil es um die Festplatte nach Zeigern suchen muss. [27] Das Object Exchange Model (OEM) ist ein Standard, um semi-strukturierte Daten zu erstellen, ein anderer Format ist XML.

7.5 Wie kann man halbstrukturierte Daten verwalten?

In unserer sich schnell verändernden IT-Welt wird es immer wichtiger, sich über verschiedene Datenformen zu informieren und wie (oder wenn) Sie sie verwalten müssen. Strukturierte Daten sind Daten, die in kleine, diskrete Einheiten aufgeteilt wurden. Jede Datenmenge betrifft eine Sache (um ein gutes angelsächsisches Fangwort zu benutzen), zum Beispiel den Nachnamen eines Kunden. Strukturierte Daten werden typischerweise in Tabellen gespeichert. Wenn wir mit unserem Beispiel fortfahren, würde eine Spalte von Daten die Nachnamen aller Kunden auflisten, und jede Zeile würde sich auf einen Kunden beziehen. Diese Tabellen werden in der Regel in einer relationalen Datenbank gespeichert. [26] In sehr vielen Fällen finden wir, dass Daten in der realen

Welt nicht ganz so gut strukturiert sind. Aber wir veranlassen die Datenbank-Struktur auf sie aus dem einfachen Grund, dass dies die Daten leicht abzurufen und abzufragen macht. In der Praxis funktioniert das gut für die Verwaltung der meisten Geschäftsdaten; Aktienkontrolle, Finanzen, Human Resources und andere Unternehmenssysteme reichen sich ganz leicht einer auferlegten Datenstruktur zu. Das Problem ist, dass einige Daten nicht zu einer rigorosen Strukturierung zugänglich sind - und diese Daten werden immer häufiger. Eine Vielzahl von Daten, die für das Unternehmen relevant sind, wird in Dokumenten, Bildern und E-Mails sowie Tweets und anderen Social Media Daten auftauchen. Alle diese Daten können als halbstrukturierte Daten beschrieben werden. [24]

Unsere Möglichkeiten zur Verwaltung von halbstrukturierten Daten sind

1. Ignorieren
2. Erzwinge es in strukturierte relationale Form
3. einen anderen Speichermechanismus annehmen

Ignorieren So viele Daten werden erstellt und gesammelt in halbstrukturierten Formen, die die meisten Unternehmen nicht leisten können, die Ausgießung davon zu ignorieren. Dies ist nur dann möglich, wenn es keinen zwingenden Geschäftsvorteil gibt, solche Daten zu verfolgen und zu analysieren.

Bleiben Sie relational Relationale Datenbanken wurden signifikant verändert, um das zu behandeln, was sich von den Datenbankherstellern als "komplexe Datentypenäuszeichnet. XML ist ein Beispiel: Es wird von vielen als eine hervorragende Möglichkeit angesehen, klassische, halbstrukturierte Daten zu halten. Die meisten gängigen Dokumentenformate sind oder können in XML gerendert werden, und fast alle relationalen Motoren haben nun einen XML-Datentyp, was bedeutet, dass Dokumente oft in einer relationalen Datenbank gespeichert werden können. Aber die zusätzliche Komplexität der Handhabung von semi-strukturierten Daten bedeutet, dass es unvermeidlich ein Kompromiss sein wird, und im Allgemeinen wird das mit langsamen Abrufzeiten gleichsetzen.

Annahme eines anderen Ansatzes Es gibt zunehmendes Interesse an der Annahme alternativer Datenmanagement- und Speichermechanismen. Stellen Sie sich vor, Sie speichern Patienten Röntgenbilder als Bilder. Wir speichern Daten, damit wir sie später abholen können und auch so können wir sie abfragen, aber eine Abfrage gegen ein Röntgenbild ist ein etwas bizarres Konzept, weil das Röntgenbild einfach eine Sammlung von Pixeln ist. Was in der Praxis oft passiert, ist, dass diese und andere semi-strukturierte Daten mit einigen angehängten Metadaten kommen und auch irgendeine Form der Analyse durchlaufen können, um weitere Metadaten zu erzeugen. (Kurz gesagt, Metadaten sind Daten über Daten). Im Falle einer E-Mail können die angehängten Metadaten Länge, Absender, Empfänger, Uhrzeit / Datum und so weiter enthalten. [27] Und nun über Röntgenstrahlen nachdenken, die klassische halbstrukturierte Daten sind. Während man sich nicht ein rohes Röntgenbild abfragen würde, kann man seine Metadaten abfragen. Die angehängten Metadaten könnten Patienten-ID, Arzt-ID, umfangreiche Informationen darüber, wie und wann das Röntgenbild genommen wurde und so weiter. Automatische Analyse des Bildes könnte Metadaten wie Diagnose, Prognose sein. In dieser Lösung

können die halbstrukturierten Daten einfach als Bilddateien im Dateisystem gespeichert werden und die strukturierten Metadaten werden in einer relationalen Datenbank gespeichert und mit dem Bild verknüpft. Eine Abfrage könnte dann alle Röntgenstrahlen für den Arzt ID herausziehen, die gebrochene Gliedmaßen beinhalten und die Bilder anzeigen. [24] Semi-strukturierte Daten bieten das Potenzial für Geschäftsvorteil für jedes Unternehmen, das sie gut behandelt und analysiert.

Literatur

[1] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970.

[2] M. Stonebraker and J. Hellerstein, "What goes around comes around," *Readings in Database Systems*, vol. 4, 2005.

[3] H. Sauer, *Relationale Datenbanken-Theorie und Praxis.* Pearson Deutschland GmbH, 2002.

[4] C. J. Date, *An introduction to database systems.* Pearson Education India, 2006.

[5] P. P.-S. Chen, "The entity-relationship model—toward a unified view of data," *ACM Transactions on Database Systems (TODS)*, vol. 1, no. 1, pp. 9–36, 1976.

[6] C. W. Bachman, "Data structure diagrams," *ACM Sigmis Database*, vol. 1, no. 2, pp. 4–10, 1969.

[7] R. Barker and C. Longman, *Case* Method.* Addison Wesley, 1990.

[8] J. ter Bekke, *Semantic Data Modeling.* Prentice Hall, 2005.

[9] N. Rishe, *Database Design - The Semantic Modelling Approach.* 1992.

[10] K. G. K. Peter Gray and N. W. Paton, "Object-oriented databases: A semantic data model approach," *Prentice-Hall International Series in Computer Science*, 2004.

[11] M. Hammer and D. McLeod, "The semantic data model: a modeling mechanism for data base applications," ACM, 2008.

[12] G. Copeland and D. Maier, "Making smalltalk a database system," *SIGMOD Rec.*, vol. 14, pp. 316–325, June 1984.

[13] C. Ireland, D. Bowers, M. Newton, and K. Waugh, "A classification of object-relational impedance mismatch," in *2009 First International Confernce on Advances in Databases, Knowledge, and Data Applications*, pp. 36–43, March 2009.

[14] T.-H. Chen, W. Shang, Z. M. Jiang, A. E. Hassan, M. Nasser, and P. Flora, "Detecting performance anti-patterns for applications developed using object-relational mapping," in *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, (New York, NY, USA), pp. 1001–1012, ACM, 2014.

[15] A. Torres, R. Galante, M. S. Pimenta, and A. J. B. Martins, "Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design," *Information and Software Technology*, vol. 82, pp. 1 – 18, 2017.

[16] A. Kemper and A. Eickler, *Datenbanksysteme Eine Einführung*, vol. 9. Oldenbourg Verlag, 2013.

[17] M. Stonebraker and J. M. Hellerstein, "What goes around comes around," 2005.

[18] M. Stonebraker and L. A. Rowe, "The design of postgres," *SIGMOD Rec.*, vol. 15, pp. 340–355, June 1986.

[19] M. Stonebraker, L. A. Rowe, and M. Hirohama, "The implementation of postgres," *IEEE Transactions on Knowledge and Data Engineering*, vol. 2, pp. 125–142, Mar 1990.

[20] M. J. Carey, D. J. DeWitt, J. F. Naughton, M. Asgarian, P. Brown, J. E. Gehrke, and D. N. Shah, "The bucky object-relational benchmark," *SIGMOD Rec.*, vol. 26, pp. 135–146, June 1997.

[21] M. Fotache and C. Strîmbei, "Object-relational databases: An area with some theoretical promises and few practical achievements," *Communications of the IBIMA*, vol. 9, 2009.

[22] P. Gulutzan and T. Pelzer, *SQL-99 Complete, Really.* Lawrence, KS, 1999.

[23] P. Buneman, "Tutorial on semi-structured data," 2010.

[24] P. B. Serge Abiteboul and D. Suciu, *Data on the Web. From Relations to Semistructured Data and XML.* Morgan Kaufmann Publishers, 2000.

[25] P. Buneman, "Semistructured data," in *roceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pp. 117–121, ACM, 2007.

[26] "Lore." `http://infolab.stanford.edu/lore/home/index.html`.

[27] D. O. 4. Francois Bry, Michael Kraus and S. Schaffert, "Aktuelles schlagwort ßemi-strukturierte daten"," 2001.