IMSE Seminararbeit

Evolution der Datenmodelle

Gruppe 12

Schöndorfer Roman (IMS Era, CODASYL Era),
Nikitina Olena (Semantic Data Model Era, Semistrutkurierte Daten),
Rabizo Daniil (Relationale Era, Entity Relationship),
Fuchs Andreas (Objektorientierte Era, Objektrelationale Era)

7. Juni 2017

Inhaltsverzeichnis

1 Relational Era

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 Organization) 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.

2 The Entity-Raltionship Era

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 genaueren 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

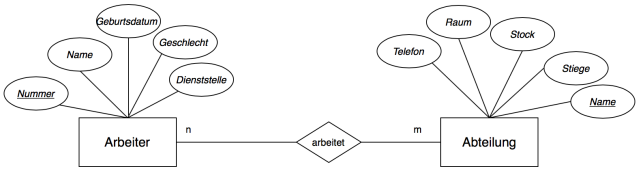
Attributen weggetan, das heißt die ≪Versteckte≫ Entitäten werden festgestellt und abgetrennt.

- 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.

3 Objekt Orientierte Ära

3.1 Motivation:

Damals wie auch heute wurden Daten und Applikation voneinander getrennt. 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* [8]. 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 Form zu repräsentieren.

Im speziellen können im *Object-Relational Impedance Mismatch* folgende Schwierigkeiten ausgemacht werden [9]

Struktur: Eine Klasse besitzt sowohl eine beliebige Struktur mit möglicherweise Pointern/Referenzen oder benutzerdefinierten Datentypen als auch beliebige Semantik definiert durch ihre Methoden und sie ist möglicherweise 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.

3.2 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.

3.2.1 Frühe Ansätze

Bereits in den späten 70ern wurde mit ersten Prototypen wie Pascal-R oder Rigel experimentiert. Es dauerte jedoch bis Mitte der 80er 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 lag 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.

3.2.2 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.

3.3 Objektrelationale Abbildung

Ein anderer Ansatz Impedance Missmatch zu lösen ist, die Abbildung von dem Objetkorientierten auf das Relationale Modell zu automatisieren und für den Entwickler transparent zu machen.

Versuche einer Objektrelationealen Abbildung (ORA) gehen zurück bis auf 1984 [9], 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.

3.3.1 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 [10].

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 [11].

3.4 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 1

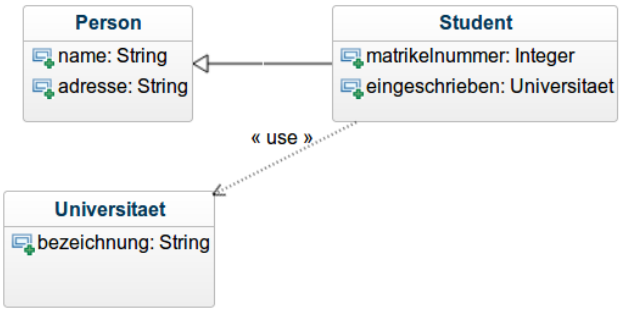


Abbildung 1: Beispiel Objekt Orientierung

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 jedoch 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 und 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 **??**, **??**.

```
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 der 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 [12]. Algorithm 2 zeigt eine einfache OQL abfrage.

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

Algorithm 2: Object Query Language

3.5 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.

4 Objekt Relationale Ära

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 [13].

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 selbst an ihre Bedürfnisse anpassen können.

4.1 Objektrelationale Datenbanken

Dieses Ausgangslage ist der der OODBS nicht unähnlich da hier wie dort sich um einen Impedance Mismatch handelt. Objekt-Relationale ist jedoch im Gegensatz zum "revolutionären" OODBS Ansatz ein "evolutionärer" Ansatz der versucht das Relationale Modell um die folgenden Eigenschaften zu erweitern [12]:

- **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.

zu erweitern.

4.1.1 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 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 [14]. Später kamen noch Vererbung, Referenzen und Mengen hinzu [15].

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.

4.2 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 Standard einen Standard verabschiedet hätte.

Eine weitere Ursache die den Durchbruch verzögerte war dass 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. [16] 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 [17].

4.3 Beispiel SQL:1999

Wir Verwenden das selbe Beispiel wie in Abschnitt 3.4 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 [18]. 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

4.4 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 [12]. Zumindest bieten aber alle großen Anbieter Relationaler Datenbanken manche Features des Objektrelationalen Designs, wie zum Beispiel Stored Procedures, in ihren Produkten an.

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

[9] 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.

[10] 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.

[11] 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.

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

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

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

[15] 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.

[16] 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.

[17] 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.

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