Diplomarbeit
zur Erlangung
des akademischen Grades Diplomingenieur
an der Fakultät für Wirtschaftswissenschaften und Informatik
der Universität Klagenfurt

Objekt-Relationale Datenbanken: Design for Performance

eingereicht von

Simone Kanzian

Betreuer:
O. Univ. Prof. DI. Dr. Johann Eder
Institut für Informatik-Systeme
Forschungsgruppe Betriebliche Informations- und Kommunikationssysteme

Klagenfurt, im Oktober 2002

Ich erkläre hiermit ehrenwörtlich, daß ich die vorliegende Arbeit selbständig verfaßt und außer der im Literaturverzeichnis angeführten Werke keine Hilfen in Anspruch genommen habe.

Diese Arbeit wurde noch keiner Prüfungskommission vorgelegt.

Simone Kanzian

Mein Dank gebührt dem Unternehmen UNiQUARE Financial Solutions GmbH, das mich während der Dauer der Arbeit finanziell unterstützt hat. Weiterer Dank gilt insbesondere Herrn DI Dieter R. Wulz, der mir während meiner Zeit im Unternehmen als Betreuer beratend zur Seite stand. Außerdem möchte ich mich bei meinen Eltern bedanken, die mir das Studium überhaupt ermöglicht haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Wenn heute eine Anwendung eine Datenbank verwendet, handelt es sich dabei meistens um ein relationales System. Relationale Datenbank-Management-Systeme (DBMS) wurden während der letzten Jahre kontinuierlich weiterentwickelt und verbessert. Während dieser Zeit haben sich aber auch die Anforderungen, die an ein DBMS gestellt werden, verändert. Die darzustellenden Realwelt-Ausschnitte sind komplexer geworden, multimediale Daten wie Bilder, Audiodateien und Videodateien gehören mittlerweile zum Alltag. Die zu entwickelnden Anwendungen wurden ebenfalls komplexer, man denke zum Beispiel an ein geographisches Informationssystem oder an ein CAD-CAM-System. Mit den Mitteln, die relationale DBMS anbieten, stößt man bei der Realisierung komplexer Anwendungen oft an Grenzen, die nicht mehr mit Hilfe trivialer Umgehungslösungen innerhalb der Datenbank überwunden werden können ([EN00]).

Objekt-orientierte Datenbanken schienen vielversprechende Ansätze für die Unterstützung komplexer Applikationen zu bieten, haben sich aber letztendlich aus verschiedenen Gründen, die in [CD96] angeführt sind, nicht durchgesetzt. Um den komplexen Anforderungen trotzdem gerecht zu werden, wurden relationale DBMS um objekt-orientierte Funktionalität erweitert. Das daraus entstehende DBMS bezeichnet man als objekt-relationales DBMS und diese Arbeit soll sich näher mit den speziellen Problemen beschäftigen, die auftreten, wenn Datenmodelle für objekt-relationale Datenbanken entwickelt werden.

Zum Thema “Design für objekt-relationale Datenbanken“ findet man in der Literatur bis jetzt nur sehr wenig Beiträge. Für relationale Systeme gibt es genügend Bücher und Papiere, die dem Designer bei der Erstellung konzeptueller und logischer Datenmodelle helfen, für objekt-relationale Systeme ist hier kaum Hilfestellung vorhanden. Man findet zwar Empfehlungen in den Handbüchern der Hersteller, diese beschränken sich aber auf die speziellen Erweiterungen des jeweiligen DBMS und vernachlässigen meistens den Performanz-Aspekt. In dieser Arbeit

möchte ich näher auf diese Problematik eingehen und den konzeptuellen und logischen Design-Prozeß zur Erstellung eines objekt-relationalen logischen Datenmodells genau untersuchen. Nach einer kurzen Einführung in den Bereich der objekt-relationalen Datenbanken in Kapitel 2, gebe ich in Kapitel 3 einen Überblick über den Design-Prozeß im Allgemeinen und über die Sprachen, mit denen ein konzeptuelles Datenmodell erstellt werden kann. Nach der Entwicklung einiger genereller Abbildungsvorschriften, mit deren Hilfe die grundlegenden Konzepte eines konzeptuellen Modells, wie Entitäten und Beziehungen, in ein logisches Modell übergeführt werden können, konzentriere ich mich im Anschluß daran besonders auf die komplexeren Strukturen des konzeptuellen Modells, insbesondere auf Vererbung und Aggregation. Für diese beiden entwickle ich verschiedene Möglichkeiten, wie sie in ein objekt-relationales logisches Modell übergeführt werden können. Diese Möglichkeiten werden in zwei der führenden DBMS praktisch umgesetzt, in Oracle, Version 9.0.2 und in DB2 UDB 7.2, Workgroup Edition.

Da in den Handbüchern der Hersteller nur wenig Auskünfte bezüglich des Performanz-Verhaltens objekt-relationaler Erweiterungen zu finden sind, ist für die beiden DBMS auch noch eine Performanz-Analyse für die Abbildungsmöglichkeiten von Vererbung und Aggregation durchzuführen, um herauszufinden, wo die Schwachstellen der objekt-relationalen Erweiterungen liegen und welche Möglichkeit die größten Vorteile im Hinblick auf Kosten- und Laufzeitverhalten bietet.

Die Arbeit beschäftigt sich nicht mit dem physischen Entwurf objekt-relationaler Datenbankschemata und auch nicht mit den Tuning-Maßnahmen, die eingesetzt werden können, um das Kosten- und Laufzeitverhalten zu verbessern. Nur der logische Design-Schritt, der von einem konzeptuellen Modell in ein logisches Datenbank-Schema ableitet, wird untersucht. Die Kern-Elemente des Design-Prozesses werden zwar behandelt, der Leser dieser Arbeit sollte aber trotzdem zumindest grundlegende Kenntnisse des Design-Prozesses für Datenbankschemata haben und mit SQL vertraut sein.

Beginnen möchte ich nun mit einer kurzen Einführung in die Thematik der objekt-relationalen Datenbanken.

Kapitel 2

Objekt-relationale Datenbanken

2.1 Entwicklung

Durch die Verbreitung objekt-orientierter Programmierung und multimedialer Daten ist die Notwendigkeit von DBMS entstanden, die den "Impedence Mismatch", also die Kluft zwischen dem Typsystem der Anwendung und dem Typsystem der Datenbank, minimieren, wie in [CD96] angeführt wird. Traditionelle relationale Systeme arbeiten mit einer flachen Datenstruktur, mit einfachen, meist alphanumerischen Datentypen. Bilder, geographische Daten und Audio-Daten können in relationalen Systemen gespeichert werden, die Bearbeitung muß aber durch die Anwendung erfolgen. Das entspricht der traditionellen Trennung zwischen Daten und Applikationslogik, die in relationalen Datenbanken vorhanden ist. In einem objekt-orientierten System erfolgt die Trennung zwischen der Schnittstelle eines Objekts und der Implementierung eines Objekts. Zur Implementierung gehören aber sowohl Daten, als auch Applikationslogik. Hier liegt der Unterschied zwischen dem relationalen und dem objekt-orientierten Konzept, der dazu führt, daß eine Abbildung von Objekten auf ein relationales Datenbankschema schwierig zu realisieren ist. Dieser und weitere Unterschiede sind unter anderem in [Sch00] ausführlich erläutert.

Aufgrund der gerade angeführten Problemstellung wurden objekt-orientierte DBMS entwickelt, die eng mit objekt-orientierten Programmiersprachen zusammenarbeiten und dabei datenbankspezifische Funktionalität wie Persistenz, Navigation zwischen den Objekten, sowie Zugriffspfadoptimierung bereitstellen. Daten werden in einer objekt-orientierten Datenbank als Objekte abgespeichert, die durch einen OID eindeutig identifiziert werden. Beziehungen zwischen den Objekten werden durch Referenzen abgebildet, die den OID verwenden. Dabei sind sowohl einzelne Referenzen, als auch Mengen von Referenzen erlaubt, beide Möglichkeiten können entweder unidirektional oder bidirektional sein. In einem relationalen System werden

Beziehungen ausschließlich über Schlüsselwerte realisiert, eine Verbindung von Relationen ist immer bidirektional. Objekt-orientierte DBMS sind laut Aussagen der Hersteller auf Abfragen und Bearbeitung komplexer Daten, zu denen insbesondere auch Audio- und Videodaten zählen, optimiert, sie haben sich trotzdem nicht am Markt durchgesetzt. Verschiedene Gründe dafür sind in [CD96] aufgeführt. Einer davon ist, daß es trotz aller Bemühungen nicht gelungen ist, einen vollständigen Standard für objekt-orientierte Datenbanken zu entwerfen, der von den Herstellern auch implementiert wurde. Ein objekt-orientiertes DBMS ist eng an eine bestimmte objekt-orientierte Sprache gebunden und die Robustheit, Skalierbarkeit und Fehlertoleranz der objekt-orientierten Systeme reicht nicht an die der relationalen Systeme heran. Viele Unternehmen, die sich gerade erst mit der Idee relationaler Systeme angefreundet hatten, waren nicht bereit, auf ein objekt-orientiertes System zu wechseln. In der Forschung wurde erkannt, daß relationale Systeme nicht unbedingt unzureichend für die Anforderungen einer objekt-orientierten Applikation sind. Hugh Darwen erklärt in [DD95], wie man durch korrekte und vollständige Umsetzung des Relationenkalküls, insbesondere durch die Einführung von Domänen, viele Eigenschaften objekt-orientierter Systeme in ein relationalen DBMS übernehmen kann.

Unabhängig von den Entwicklungen in der Forschung haben auch Hersteller von DBMS relationale Systeme um objekt-orientierte Konzepte erweitert, da ihnen dieser Weg im Gegensatz zur vollständigen Neuentwicklung eines objekt-orientierten DBMS einfacher schien. Aus den zunächst in [CD96] angeführten “Extended Relational Database Systems“ wurden so im Lauf der Zeit objekt-relationale DBMS. Ein neuer SQL-Standard, der unter anderem die Anforderungen an ein objekt-relationales System vereinheitlichen soll, ist bereits seit Jahren in Arbeit, wann der objekt-relationale Teil verabschiedet wird, ist derzeit nicht absehbar.

Welche Konzepte ein objekt-relationales DBMS enthalten soll, um als objekt-relational bezeichnet werden zu können, wird in den folgenden Abschnitten besprochen. An dieser Stelle ist zu bemerken, daß es hierfür noch keinen Standard gibt, bestenfalls Empfehlungen und daß auch noch keiner der führenden Hersteller, wie Oracle oder IBM, alle Konzepte unterstützt. Da teilweise der bestehende SQL92-Standard noch nicht vollständig realisiert ist, wird es wohl noch einige Jahre dauern, bis objekt-relationale DBMS alle gewünschten Erweiterungen umgesetzt haben.

2.2 Konzepte

Grundsätzlich soll ein DBMS laut [SBM99] vier Konzepte unterstützen, um als objekt-relational zu gelten und zwar:

- Basisdatentyp-Erweiterung
- Komplexe Objekte
- Regelsystem
- Vererbung

2.2.1 Basisdatentyp-Erweiterung

Relationale DBMS haben vordefinierte Datentypen, mit denen gearbeitet werden kann, die sogenannten Basisdatentypen. Für diese Basisdatentypen ist eine Anzahl von Operationen implementiert, mit denen die Daten bearbeitet, gelesen und geschrieben werden können. Der Benutzer selbst kann den Typ weder verändern noch definieren, er wird vom Hersteller in das DBMS eingebaut und für Zugriffe optimiert. Beispiele für Basisdatentypen sind

- Zeichenketten variabler und fester Länge, besser bekannt als CHAR und VARCHAR,
- natürliche Zahlen, ganze Zahlen und Gleitkommazahlen,
- Datumswerte und Zeitstempel sowie
- Boolean-Werte (die allerdings nicht in allen DBMS vorhanden sind)

Alle in einem Modell der Realwelt benötigten Datentypen müssen auf diese Basisdatentypen abgebildet werden. Diese Abbildung ist häufig sehr aufwendig für den Designer, vor allem wenn das Typsystem einer objekt-orientierten Anwendung auf ein relationales Datenbankschema abgebildet werden muß. Die dabei auftretenden Probleme wurden in Abschnitt 2.1 schon angesprochen. Um diese Probleme zu umgehen, hat man in einem objekt-relationalen DBMS die Möglichkeit geschaffen, eigene Datentypen und die dazugehörigen Operationen zu definieren und in das DBMS einzubinden. Ist ein neuer Typ im System bekanntgemacht, kann er wie ein Basisdatentyp verwendet werden. Das Laufverzeitverhalten der Abfragen sollte dabei durch die Verwendung des benutzerdefinierten Datentyps nicht negativ beinflußt werden. In der Praxis gibt es eine Reihe von Optimierungsmethoden für benutzerdefinierte Datentypen, die ein Administrator einsetzen kann, auf die hier nicht näher eingegangen wird. Es wird also eine Möglichkeit für den Benutzer geschaffen, die von seinem System benötigten Datentypen selbst

in das DBMS einzubauen, ohne darauf warten zu müssen, daß sie vom Hersteller implementiert werden, wie es bisher in relationalen Systemen der Fall war.

2.2.2 Komplexe Datentypen

Ein objekt-relationales DBMS muß auch in der Lage sein, zusammengesetzte Datentypen sowie Mengen und Referenzen dieser Datentypen zu unterstützen. Ein zusammengesetzter Datentyp besteht aus mindestens einem Attribut, das wiederum auf einem Basisdatentyp, einem benutzerdefinierten Datentyp oder einem zusammengesetzten Datentyp aufbauen kann. Der Unterschied zu einem benutzerdefinierten Datentypen liegt darin, daß ein benutzerdefinierter Datentyp maximal ein Attribut haben kann, während ein komplexer Datentyp aus mehreren Attributen bestehen kann. Die Schachtelung zwischen zusammengesetzten Datentypen, das heißt daß ein Attribut eines Datentyps als Typ wieder einen zusammengesetzten Datentyp haben kann, ist beliebig tief. Rekursion ist nicht erlaubt, ein zusammengesetzter Datentyp kann keine Attribute enthalten, die den Datentyp selbst verwenden. Die Idee hinter zusammengesetzten Datentypen ist die, daß Daten, die konzeptuell zu einem gemeinsamen Objekt gehören, auch in der Datenbank logisch gemeinsam gespeichert werden. Bei relationalen Systemen werden die Attribute einer Tabelle auf verschiedene Spalten aufgeteilt, es ist nicht mehr anhand der Tabelle alleine erkennbar, welche Daten zusammengehörig sind, das konzeptuelle Modell muß hier zur Information herangezogen werden. Verwendet man für ein Attribut einen zusammengesetzten Datentyp, bleiben die Daten in einer logischen Einheit gespeichert und es ist auch im logischen Datenmodell sofort erkennbar, welche Daten zusammengehören. Ein zusammengesetzer Datentyp kann als Typ für eine Tabelle verwendet werden oder auch als Datentyp für einzelne Attribute einer Tabelle.

Wird ein zusammengesetzter Datentyp dem System bekannt gemacht, werden automatisch zwei weitere Datentypen angelegt: ein Typ, der die Menge des zusammengesetzten Datentyps darstellt und ein Typ, mit dessen Hilfe auf den zusammengesetzten Datentyp referenziert werden kann. Es gilt also: Wenn ein zusammengesetzter Datentyp ein gültiger Datentyp im System ist, dann sind auch seine Menge und eine Referenz auf ihn gültige Datentypen.

Referenzen sind eine Alternative für das Primär-Fremdschlüssel-Konzept, das in relationalen Datenbanken Beziehungen zwischen Relationen abbildet. Mit Referenzen wird eine Navigationsmöglichkeit zwischen Tabellen angeboten, die unabhängig von den Werten ist, die in der Tabelle gespeichert sind. Eine Referenz ist vergleichbar mit einem Zeiger einer Programmiersprache, der einen Verweis auf ein Objekt in Form eines OID enthält. Ein OID muß systemweit

eindeutig sein, das DBMS kümmert sich selbst um die Einhaltung dieser Bedingung. Während Beziehungen in einer relationalen Datenbank immer bidirektional sind, also in beide Richtungen aufgelöst werden können, sind Beziehungen durch Referenzen unidirektional. Eine Referenz kann immer nur in eine Richtung aufgelöst werden, was entscheidenden Einfluß auf das Design des Datenbankschemas hat. Der Designer muß wissen, in welche Richtung die Beziehung aufgelöst werden soll, eine Entscheidung, die ihm im relationalen Design erspart bleibt. Um mehrwertige Attribute zu unterstützen, muß auch ein Datentyp für eine Menge von Referenzen vorhanden sein.

Für alle zusammengesetzten Datentypen müssen auch Funktionen angelegt werden, die das Ergebnis einer Abfrage, die Werte eines zusammengesetzten Datentypen zurückliefert, in geeigneter Weise darstellen.

2.2.3 Regelsystem

Eine weitere Eigenschaft, die ein objekt-relationales DBMS laut [SBM99] haben sollte, ist die Möglichkeit ein Regelsystem aufzubauen, das komplexe Nebenbedingungen, die nicht mit herkömlichen Constraints realisiert werden können, umsetzen soll. Die Lösung, die dafür zur Verfügung steht, sind Trigger. Vier verschiedene Arten von Triggern sind notwendig um ein Regelsystem aufzubauen:

- Update-Update Trigger
- Query-Update Trigger
- Update-Query Trigger und
- Query-Query Trigger,

wobei auch “Insert“ und “Delete“ hier unter den Begriff “Update“ fallen. Ein Trigger kann vor oder nach einer Operation auf der Tabelle ausgeführt werden (BEFORE- bzw. AFTER-Trigger) und er kann für alle betroffenen Datensätze oder einmal pro SQL-Statement aktiviert werden (FOR-EACH-ROW- bzw. FOR-EACH-STATEMENT-Trigger). Während die erste Variante in fast allen, derzeit am Markt vertretenen DBMS implementiert ist, sind die letzteren eigentlich kaum bekannt.

Bei einem Update-Update-Trigger wird durch eine Update-Operation auf einer Tabelle eine weitere Update-Operation auf derselben oder einer anderen Tabelle ausgelöst. Wenn zum Beispiel Geld von einem Konto abgehoben, also die Soll-Seite erhöht wird, dann muß auch

gleichzeitig die Haben-Seite verringert werden, mit Hilfe eines Update-Update-Triggers. Ein Update-Update Trigger kann sehr vielseitig verwendet werden. Er wird meistens dazu benutzt um die Integrität der Daten zu gewährleisten, aber auch um Datensätze einzufügen, die voneinander abhängen.

Bei einem Query-Update-Trigger löst eine Query auf eine Tabelle ein Update aus. Ein Query-Update Trigger kann für Log-Tabellen verwendet werden, in dem man jeden Zugriff eines Benutzers auf eine Tabelle mitprotokolliert. Bei jeder Query des Benutzers wird die Log-Tabelle upgedatet. Derzeit verwenden fast alle DBMS dafür ein Trace-System, anstelle eines Query-Update Triggers.

Ein Update-Query Trigger, bei dem ein Update eine Query auslöst, könnte ein Benachrichtigungs-System implementieren, bei dem in Folge eines Updates auf einer Tabelle eine Benachrichtigung an den Benutzer geschickt wird. Auch hier verwenden fast alle Hersteller ihre eigenen Systeme.

Ein Query-Query Trigger wird dazu benutzt eine Query durch eine andere zu ersetzen. Dadurch könnte man Datensätze, die für mehrere Instanzen den gleichen Wert haben, nur einmal in der Datenbank eintragen und bei Abfragen eine entsprechende Ersetzung durch eine andere Query vornehmen. Diese Vorgehensweise wirft allerdings eine Reihe von Problemen auf, die berücksichtigt werden müssen. Was geschieht zum Beispiel, wenn der Datensatz aktualisiert wird und dadurch für einige der Instanzen ungültig wird? Was geschieht, wenn der Datensatz aus der Datenbank gelöscht wird? Der hierbei entstehende Overhead steht in keiner Relation zum Aufwand, der berücksichtigt werden muß, wenn die Daten für jede Instanz gespeichert werden. Ein Beispiel aus [SBM99] für einen Query-Query-Trigger wäre die Zuordnung eines Gehaltes zu mehreren Mitarbeitern, die alle gleichviel verdienen. Physisch wird das Gehalt dabei nur für einen der Mitarbeiter gespeichert. Wenn Abfragen auf das Gehalt seiner Kollegen gestellt werden, werden diese durch eine Abfrage auf das Gehalt des Mitarbeiters ersetzt. Probleme treten nun auf, wenn dieser Mitarbeiter eine Gehaltserhöhung bekommt oder das Unternehmen verläßt. Dafür muß man sich eine Strategie überlegen, damit die Integrität der Gehaltsdaten für die anderen Mitarbeiter gewährleistet bleibt. Diese wird jedoch mehr an Aufwand erfordern, als die Variante, bei der das Gehalt für jeden Mitarbeiter gespeichert wird. Verdient hier ein Mitarbeiter plötzlich mehr muß nur sein Datensatz aktualisiert werden, die Daten seiner Kollegen bleiben unverändert. Verläßt bei der Query-Query Trigger Variante der Mitarbeiter das Unternehmen, für den das Gehalt definiert ist, müssen alle Trigger, die darauf aufbauen geändert werden. Der Aufwand von Updates und die Gefährdung der Datenkonsistenz

sind schwerwiegende Gründe, die gegen die Verwendung von Query-Query-Triggern sprechen. Es gibt sicher andere Verwendungsmöglichkeiten, wo ihr Einsatz von Vorteil ist, aber zur Darstellung sogenannter “virtueller Attribute“, wie oben beschrieben, sollten sie nicht verwendet werden.

Der Einsatz von Triggern erfordert außerdem einiges an Vorsicht und Feingefühl. Durch ihre leichtfertige Verwendung kann eine Reihe von Problemen auftreten. So gerät man zum Beispiel durch Trigger, die sich gegenseitig immer wieder aktivieren, sogenannte “zyklische Trigger“, sehr leicht in eine Endlosschleife. Weiters kann es zu unvorhersehbaren bzw. nicht nachvollziehbaren Ergebnissen kommen, wenn zwei oder mehr Trigger auf das gleiche Ereignis reagieren. Es ist von Bedeutung zu welchem Zeitpunkt ein Trigger aktiviert wird, da der Zeitpunkt Einfluß auf den Zustand nach der Ausführung des Triggers haben kann, vor allem, wenn mehrere Trigger gleichzeitig aktiv sind. Außerdem muß berücksichtigt werden, daß bei Abbruch einer Transaktion des Triggers auch die nachfolgenden Transaktionen desselben Triggers nicht mehr ausgeführt werden. Man kann also leicht erkennen, daß Trigger vorsichtig eingesetzt werden sollten, da das Nachvollziehen von Fehlern, die durch Trigger entstehen, besonders schwierig ist, was vor allem auf die undefinierte Reihenfolge in der Ausführung zurückzuführen ist. Zudem sollte darauf geachtet werden, daß keine Applikationslogik durch Trigger in das DBMS eingebaut wird.

2.2.4 Vererbung

Eine weitere Anforderung an ein objekt-relationales DBMS ist Vererbung. Dabei spielt sowohl die Vererbung von Daten als auch die Vererbung von Funktionen eine Rolle. Ein Vorteil von Vererbung ist Codewiederverwendung und die daraus resultierende Arbeitsersparnis.

In einem DBMS ist vor allem die Vererbung von Attributen wichtig, um eine Hierarchie von zusammengesetzten Datentypen zu definieren, man spricht hier von Schema-Vererbung. Nicht in den Bereich der Vererbung fallen benutzerdefinierte Datentypen, obwohl es auf den ersten Blick scheint, das diese Eigenschaften des Basisdatentyps, auf dem sie aufbauen, erben. Bei Vererbung soll das DBMS dem Designer die Arbeit abnehmen, sich selbst darum kümmern zu müssen, welche Attribute innerhalb einer Vererbungshierarchie weitergegeben werden. Im Allgemeinen werden Attribute eines zusammengesetzten Datentyps entlang einer Hierarchie von oben nach unten vererbt, das heißt ein Subtyp erbt alle Attribute seines Supertyps. Der Supertyp enthält dabei generelle Attribute einer Entität, also alle Attribute, die bei den zu der Hierarchie gehörenden Subklassen vertreten sein sollen. Der Subtyp enthält zusätzlich zu den geerbten Attributen noch die für ihn speziellen Attribute. Da auch Mehrfachvererbung unterstützt werden soll, kann ein Subtyp Attribute von mehreren Supertypen erben. Hier

können Probleme bei der Vererbung von Attributen auftreten, die in verschiedenen Superklassen gleich benannt sind, aber eine unterschiedliche Bedeutung haben. Das DBMS kann hier nicht selbst entscheiden, welches der beiden Attributen vererbt werden soll und reagiert mit einem Fehler. Es liegt also beim Designer, solche Fälle aufzulösen, nicht beim DBMS. Wie in [MS00] angeführt, sieht der SQL3-Standard derzeit keine Unterstützung von Mehrfachvererbung in objekt-relationalen DBMS vor. Welche Methoden es gibt, um Mehrfachverbung ohne die explizite Unterstützung des DBMS zu realisieren, ist ebenfalls in [MS00] ausführlich dargelegt und wird von mir in Abschnitt 3.9 noch besprochen. Vererbung hat den Vorteil, daß Konstrukte des konzeptuelles Modells, die Vererbung verwenden, auch in der Datenbank abgebildet werden können. Die Meta-Information über die Struktur der Daten bleibt damit auch im logischen Modell erhalten und der Benutzer muß sich nicht selbst um die korrekte Übernahme aller Attribute einer Vererbungshierarchie kümmern. Prinzipiell sollte auch Instanzenvererbung von objekt-relationalen DBMS unterstützt werden, das heißt innerhalb einer Vererbungshierarchie sollten nicht nur die Attribute und ihre Definitionen vererbt werden, sondern auch die tatsächlichen Werteausprägungen der Attribute, die diese für einen bestimmten Datensatz annehmen.

Ein Beispiel für eine Vererbungshierarchie ist in Abbildung 2.1 veranschaulicht. Hier sind die Beziehungen zwischen Kunden und Angestellten einer Bank dargestellt. Sowohl ein Kunde als auch ein Angesteller ist eine natürliche oder juristische Person, die einen Name, eine Adresse und ein Alter besitzt. Diese zählen also zu den generellen Attributen und sind damit in der Superklasse “Person“ vertreten. “Kunde“ und “Angestellter“ sind Subklassen von “Person“, erben also die Attribute von “Person“ und besitzen zusätzlich noch spezifische Attribute. So erhält ein Angestellter ein Gehalt und ist einer Abteilung zugeordnet, während ein Kunde eine Kontonummer und ein Konto besitzt. Nun kann aber ein Angestellter natürlich auch ein Konto bei der Bank haben und ist damit sowohl ein Subtyp der Klasse “Angestellter“ als auch ein Subtyp der Klasse “Kunde“ und muß dementsprechend Attribute von beiden erben. Die Attribute der Superklasse “Person“ dürfen nur einmal an den Kunden, der auch Angestellter ist, vererbt werden. Bei mehrdeutigen Attribute, die denselben Name haben, aber eine unterschiedliche Bedeutung, muß der Designer den Konflikt auflösen. Die aktuellen objekt-relationalen DBMS der beiden führenden Hersteller Oracle und IBM fordern derzeit, daß ein Subtyp nicht zwei Supertypen, die in der Hierarchie nebeneinander liegen, untergeordnet werden kann. Ansonsten können beliebig viele Superklassen in einer Linie über der Subklasse stehen.

Nicht nur die Attribute eines zusammengesetzen Datentyps sollen vererbt werden, sondern auch die Methoden und Funktionen, die für den Datentyp angelegt wurden. Dabei wird das Prinzip des Polymorphismus eingesetzt. Erbt ein Subtyp eine Methode von seinem Supertyp,

die er selbst schon besitzt, dann wird automatisch die Methode des Subtyps verwendet. Nur wenn das DBMS bei Aufruf einer Methode keine für den Subtyp definierte Methode findet, sucht es entlang der Hierarchie nach einer passenden Methode. Einem Name können dabei auch unterschiedliche Implementierungen der Methoden zugeordnet sein. Das DBMS entscheidet dann zur Laufzeit mit Hilfe der aktuellen Parameter anhand der Signatur der Methoden, also der Anzahl und dem Typ der übergebenen Parameter, welche Methode eingesetzt werden soll. Es liefert einen Fehler, wenn entlang der Hierarchie keine passende Methode gefunden werden kann. Auch hier gilt, daß Mehrfachvererbung, also der Fall, daß mehrere Methoden in Frage kommen und keine eindeutige Auswahl erfolgen kann, vom Designer aufgelöst werden muß, nicht vom DBMS selbst. Dasselbe gilt analog für Funktionen. In weiterer Folge wird ab hier nur noch von Methoden gesprochen, da es auf konzeptueller Ebene zwischen Methoden und Funktionen keinen Unterschied gibt.

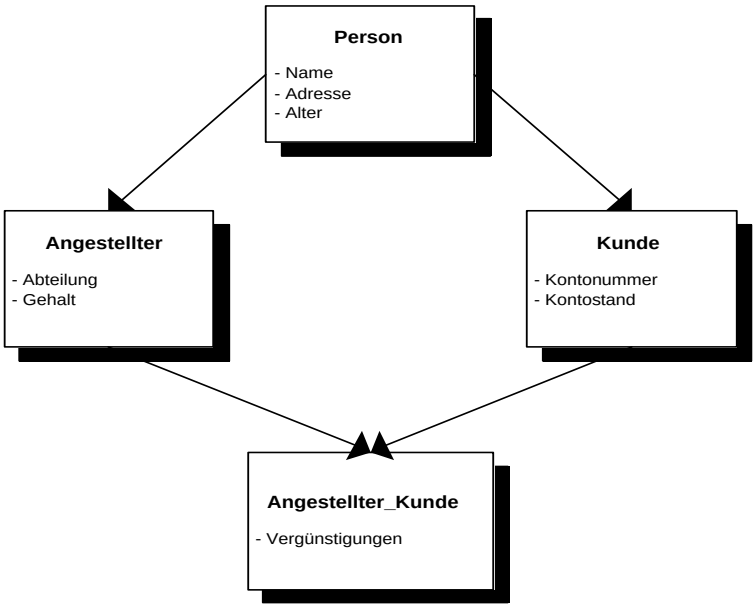


Abbildung 2.1: Vererbungshierarchie mit Mehrfachvererbung

Im nächsten Kapitel möchte ich nun ausführlich auf das Design objekt-relationaler Datenbanken eingehen. Das wesentliche Ziel dabei soll die Entwicklung von Varianten sein, mit denen ein konzeptuelles Modells in ein objekt-relationales logisches Modell übergeführt wird. Für relationale Datenbanken gibt es bereits viele Bücher und Arbeiten, die sich mit diesem Thema beschäftigen. Die dort eingesetzten Lösungen sind hinreichend bekannt, ebenso ihre Auswirkungen auf das Laufzeitverhalten von Abfragen. Durch die Verwendung objekt-relationaler Konzepte hat man eine zusätzliche Anzahl an Varianten, mit der Modellierungsaufgaben gelöst werden können.

Jede Variante hat unterschiedliche Auswirkungen auf die Performanz von Abfragen, weshalb die entwickelten Varianten mit zwei konkreten DBMS, nämlich Oracle und DB2, im Hinblick auf ihr Kosten- und Laufzeitverhalten getestet werden.

Kapitel 3

Design objekt-relationaler Datenbanken

Datenbank-Design ist der Prozeß, der mit dem Erstellen eines konzeptuellen Datenmodells gemäß der im Analyseprozeß identifizierten Anforderungen beginnt. Das konzeptuelle Datenmodell stellt die Strukturen der Daten und die Zusammenhänge zwischen den Daten in einer vom DBMS losgelösten graphischen Notation dar. Weiters werden mit Hilfe von Datenfluß-Diagrammen, Use-Case-Diagrammen und Szenarios identifzierte Datenflüsse aufbereitet. Aus dem konzeptuellen Modell wird ein logisches Modell abgeleitet, das die Daten in eine relationale Struktur bringt, Relationen identifiziert, Primärschlüssel und Fremdschlüssel identifiziert und Beziehungen zwischen den Daten auflöst. Aus dem logischen Modell entsteht in weiterer Folge das physische Datenbankschema, Relationen werden in Tabellen übergeführt, Attributen wird ein Datentyp zugewiesen, Indizes werden angelegt, Cluster gebildet und Tabellen partitioniert. Man führt also die Daten von einer abstrahierten Ebene in ein konkretes DBMS unter Berücksichtigung seiner speziellen Eigenschaften über. Auf konzeptueller Ebene darf es noch keine Rolle spielen, welches DBMS schlußendlich für das physische Schema verwendet wird. Da der Design-Prozeß für den Erfolg einer Datenbank eine wesentliche Rolle spielt, soll er an dieser Stelle noch näher erläutert werden.

3.1 Der Design-Prozeß

Der vollständige Ablauf des Entwurfsprozesses einer Datenbank ist in Abbildung 3.1 zu sehen.

Zunächst werden während der Analysephase gemeinsam mit zukünftigen Benutzern Anforderungen identifiziert, die an die zu entwickelnde Anwendung gestellt werden. Mit Hilfe dieser

Anforderungen wird das konzeptuelle Datenmodell erstellt, das einen Ausschnitt aus der Realwelt darstellt.

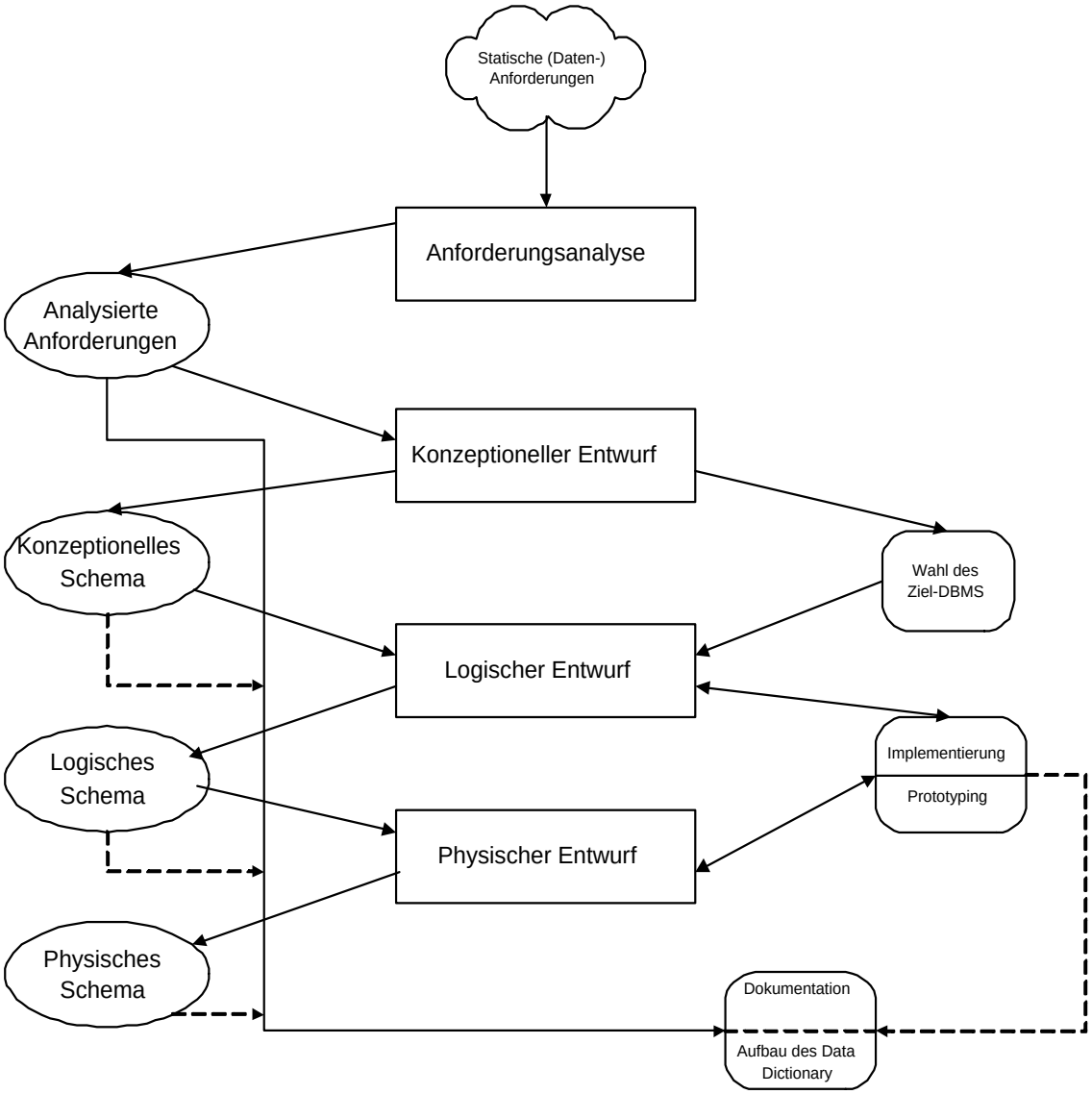


Abbildung 3.1: Entwurfsprozeß einer Datenbank [Vos99]

Abstraktion ist bei der Modellierung von Realweltausschnitten unbedingt erforderlich. Es ist unmöglich die Realität vollständig zu modellieren, man muß versuchen, sie auf die für die Anwendung wesentlichen Punkte zu reduzieren. Dabei geht man meist schrittweise vor, indem man zunächst verwandte Bereiche der Anforderungen klassifiziert und diese formal mit Hilfe einer speziellen Notation beschreibt. Man kann dabei mit allgemeinen Beschreibungen

beginnen und schrittweise verfeinern, was einer Top-Down-Vorgehensweise entspricht. Man kann aber auch mit detailierten Beschreibungen beginnen und diese verallgemeinern, also eine Bottom-Up-Vorgehensweise verwenden. Es liegt im Ermessen des Designers, welche der beiden Varianten er bevorzugt. Wenn man die Anforderungen gruppiert, also zu sogenannten "Sichten" zugeteilt hat, erfolgt eine Integration dieser Sichten, das heißt man verknüpft die Sichten miteinander um so eine Gesamtsicht auf die Daten zu erhalten. Der Input des konzeptuellen Entwurfs ist das Anforderungsdokument, der Output ist ein konzeptuelles, abstrahiertes Datenmodell, das unabhängig vom gewählten DBMS die Daten in eine wohlgeordnete Struktur bringt.

Am Ende des konzeptuellen Entwurfs muß man sich entscheiden, welches DBMS man in weiterer Folge verwenden will. Die Wahl eines DBMS kann von verschiedenen Faktoren beeinflußt werden, zum Beispiel von

- Hersteller-Preferenzen,
- verfügbare Datentypen und verfügbare Konzepte,
- Laufzeitverhalten, gemessen durch Benchmarks,
- bereits vorhandene Hardware, Software und DBMS,
- Migrationskosten (wenn von einer bestehenden Datenbank migriert werden soll),
- Kosten für die Neuanschaffung und Wartung eines DBMS,
- Personalkosten für die Ausbildung oder Neueinstellung von Administratoren, sowie Schulungskosten für Benutzer

Das DBMS und das konzeptuelle Modell sind also zwei Input-Faktoren für den logischen Entwurf, aber es gibt noch zwei weitere Faktoren, die von entscheidender Wichtigkeit sein können, nämlich das Mengengerüst und das Transaktionsprofil. Ein Mengengerüst liefert Informationen über die Anzahl der Datensätze, die in den Sichten des konzeptuellen Entwurfs zu erwarten sind. Das Transaktionsprofil erteilt Auskunft darüber welche Operationen auf dem Datenbankschema erfolgen, ob auf Datensätze überwiegend lesend oder schreibend zugegriffen wird und wie oft innerhalb eines bestimmten Zeitraums eine Operation auftritt. Weiters wird im Transaktionsprofil auch definiert, ob die Operation sofort ausgeführt wird (online) oder ob sie im Batch-Betrieb erfolgt. Die Anzahl der Datensätze und die Zugriffsart auf die Datensätze spielen eine wesentliche Rolle im logischen Design. Eine genaue Definition von Mengen- und Transaktionsprofil ist in Abschnitt 3.4 zu finden. In der Phase des logischen Designs wird das konzeptuelle Modell mit Hilfe von definierten Transformationsregeln unter

Miteinbeziehung von Mengengerüst und Transaktionsprofil in ein logisches Modell abgebildet. Die spezifischen Charakteristika des gewählten DBMS, wie zum Beispiel Datentypen, Referenzen und Vererbung, werden hier bereits berücksichtigt. Für relationale DBMS sind die Transformationsregeln sehr gut bekannt und in der Literatur verankert. Zum logischen Entwurf gehört auch die Normalisierung des Datenmodells, deren Ziel die Vermeidungen von Redundanzen im Datenmodell ist. Einschränkungen, die sich aus der Anforderungsanalyse ergeben, müssen ebenfalls in den logischen Entwurf integriert werden. Der Output des logischen Entwurfs ist ein Datenbankschema des eingesetzten DBMS.

Am Ende des Entwurfsprozesses steht der physische Entwurf, dessen Input das logische Datenbankschema ist. Das Ziel des phyischen Entwurfs ist die Optimierung der Zugriffspfade auf die Datensätze des logischen Modells. Man versucht hier in einem ersten Anlauf Zugriffe auf die Tabellen weitgehend zu optimieren. Das geschieht mit Hilfe von Clustering, Partioning und Indexdefinitionen. Physischer Entwurf erfolgt noch vor der Inbetriebnahme des Datenbankschemas. Beispiele dafür sind

- Blockgröße: Wieviel Speicherblöcke werden dem DBMS im Hauptspeicher zugeteilt und in welcher Größe kann Speicherplatz nachgefordert werden?
- Verwendung von Indizes: Auf welche Attribute bzw. Attributkombinationen wird besonders häufig zugegriffen? Diese sollte man möglichst optimiert speichern.
- Clustering: Inhaltlich zusammengehörige Objekte werden in gemeinsamen Speicherblöcken verwaltet.
- Denormalisierung: Normalformen werden aufgelöst, das heißt kontrollierte Redundanz wird eingeführt um Zugriffszeiten zu optimieren. Hat man zum Beispiel zwei Relationen, die in Anfragen häufig gemeinsam benötigt und durch einen Join verbunden werden, ist es sinnvoll diese beiden Relationen wieder zu einer einzigen zu verbinden.

Der Output des physischen Entwurfs ist ein für die Anwendung optimiertes Datenbankschema, das möglichst geringe Zugriffszeiten erlaubt. Ist das Datenbankschema eine Zeitlang im Einsatz, konnte also das Verhalten der Operationen über einen bestimmten Zeitraum hinweg beobachtet werden, kann noch weiter optimiert werden indem man Schwachstellen, die erst im Betrieb ersichtlich wurden, beseitigt. Diese Optimierungen bezeichnet man als “Datenbank-Tuning“.

Eine wichtige Phase, die den gesamten Entwurfsprozeß begleitet, ist die Dokumentationsphase. Bei Re-Design und Wartung spart eine detaillierte Dokumentation viel Arbeitszeit und damit

auch Geld. Änderungen im Datenmodell können effizienter umgesetzt werden, wenn frühere Entscheidungen anhand der Dokumentation nachvollziehbar sind.

Durch den Einsatz von objekt-relationalen Erweiterungen kommt beim logischen Design noch eine Fülle von Entscheidungen dazu, die bei der Abbildung eines konzeptuellen Modells auf ein logisches Modell berücksichtigt werden müssen, was die Arbeit des Designers erschwert. Mit objekt-relationaler Modellierung kann man logische Datenmodelle erstellen, die sehr viel näher am konzeptuellen Modell orientiert sind und in denen viel Information aus dem konzeptuellen Modell in die Datenbank übernommen wird, was allerdings nicht immer unbedingt von Vorteil sein muß, da Elemente des konzeptuellen Modells oft sehr viel Komplexität beinhalten. Zum Thema Modellierung und Datenbank-Design gibt es, wie bereits erwähnt, für relationale Datenbanken bereits ausreichend Literatur, mit derselben Thematik haben sich in der Welt der objekt-relationalen Datenbanken bis jetzt noch sehr wenig Autoren beschäftigt. Es gibt einige Vorschläge, wie man die Objekte und Beziehungen eines konzeptuellen Datenmodells in eine objekt-relationale Datenbank abbildet (siehe auch [SBM99], [MP01] und [MS00]), wirklich ausführliche Literatur ist allerdings nicht vorhanden. Diese Arbeit soll sich ausführlicher mit den Abbildungsmöglichkeiten eines konzeptuellen Modells auf eine objekt-relationale Datenbank und deren Auswirkungen auf das Laufzeitverhalten beschäftigen und dem interessierten Benutzer einen Eindruck davon vermitteln, worauf es bei der Erstellung objekt-relationaler Datenbankmodelle ankommt. Dazu werden zunächst zwei konzeptuelle Design-Modelle, das Entity-Relationship-Modell und die Unified Modeling Language, vorgestellt.

3.2 Das Entity-Relationship Modell

Ein Design-Modell hat sich im Laufe der Jahre in der Datenbankwelt verbreitet und bewährt: das Entity-Relationship-Modell von Peter Chen ([Che76]). Aus einem ER-Modell können mit einer Reihe von definierten Abbildungsvorschriften sehr einfach relationale Datenbank-Schemata erstellt werden. Das Modell selbst ist einfach zu lesen und leicht zu erlernen, was wohl auch der Grund für seinen großen Erfolg war.

Die grundlegenden Konzepte im ER-Modell sind folgende:

- Entitäten
- Beziehungen zwischen Entitäten
- Attribute

Eine Entität ist ein Objekt der realen Welt, das unabhängig von anderen Objekten existieren kann. Eine Entität hat einen Name und Attribute, die ihr zugeordnet sind. Ein Attribut stellt eine bestimmte Eigenschaft der Entität dar und nimmt einen oder mehrere definierte Werte an. Ein Attribut kann komplex oder atomar sein, ein- oder mehrwertig, sein Wert kann direkt gespeichert oder errechnet werden. Der Wertebereich eines Attributs wird als Domäne bezeichnet. Von einer Entität können beliebig viele Instanzen vorhanden sein, die alle gemeinsame Attribute, aber unterschiedliche Werte haben. Diese Instanzen werden durch den Typ einer Entität definiert. Alle definierten Entitäten, die sich zu einem beliebigen Zeitpunkt in der Datenbank befinden, bezeichnet man als Entitäten-Menge. Jede Entität wird durch ein oder mehrere Attribute eindeutig identifziert, den sogenannten Schlüsselattributen. Zwischen Entitäten sind Beziehungen vorhanden, die Zusammenhänge zwischen den Entitäten darstellen. Beziehungen können selbst wieder einen Name und Attribute besitzen, die Beziehungseigenschaften definieren. Die Anzahl der teilnehmenden Entitäten, die Kardinalität, wird ebenfalls durch eine Beziehung festgelegt, ebenso wie eventuelle Einschränkungen, die aufgrund der Anforderungen identifiziert wurden. Hat eine Entität keine eigenen Schlüsselattribute, die sie identifizieren, spricht man von einer sogenannten “schwachen“ Entität. Diese wird durch eine Beziehung zu einer anderen “starken“ Entität definiert, man spricht von einer identifizierenden Beziehung. Eine “schwache“ Entität kann also nicht unabhängig von anderen Entitäten existieren. Ein Beispiel für eine “schwache“ Entität sind Adressen, die einer Person zugeordnet werden. Eine Adresse kann in einer Datenbank im Normalfall nicht alleine existieren. Es ist nicht sinnvoll Adressen zu speichern, ohne sie Personen zuzuordnen. Die Adresse ist also eine “schwache“ Entität, die über ihre Beziehung zu einer Person, einer “starken“ Entität identifiziert wird.

Die graphische Notation der Konzepte ist denkbar einfach, wie in Abbildung 3.2 zu sehen ist. Das ursprüngliche ER-Modell besteht ausschließlich aus diesen drei Konzepten. Für eine große Zahl von Anwendungen ist diese Darstellung ausreichend. In den späten 70er Jahren hat sich allerdings herausgestellt, daß viele Datenbanken zum Beispiel aus den Bereichen Multimedia und geographische Informationssysteme komplexere Anforderungen an ein Datenmodell stellen. Man hat sich daher entschlossen, das ER-Modell um Vererbung, Spezialisierung und Generalisierung zu erweitern, woraus sich das “Extended-Entity-Relationship“-Modell ergeben hat, für das es allerdings keinen einheitlichen Standard gibt. In dieser Arbeit wird das EER-Modell aus [EN00] verwendet. Das Konzept der Vererbung wurde bereits in Abschnitt 2.2.4 ausreichend erläutert und soll an dieser Stelle nicht mehr wiederholt werden. Die Idee der Spezialisierung und Generalisierung werden im nächsten Abschnitt genau erklärt.

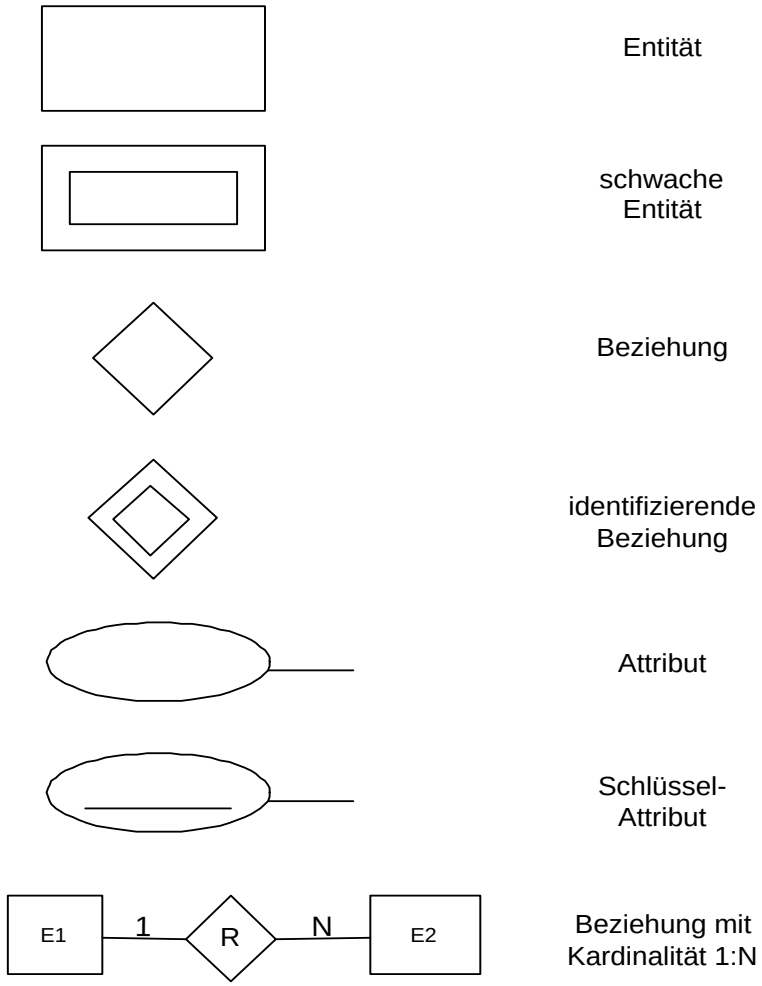


Abbildung 3.2: ER Modell Notation (vereinfacht aus [EN00])

3.2.1 Spezialisierung und Generalisierung

Bei Spezialisierung handelt es sich um eine besondere Form der Vererbung, bei der einem Entitätstyp eine Menge von Subklassen zugeordnet wird. Der Entitätstyp wird damit zu einer Superklasse, die die generellen Attribute der Entität enthält. Eine Subklasse besteht nur aus Attributen, die sie von den anderen Subklassen unterscheidet und die eine Entität eindeutig zu dieser Subklasse zuordnen. Eine Entität kann entweder nur einer einzigen Subklasse zugeordnet sein, man spricht dann von disjunkter Spezialisierung, oder sie kann mehreren Subklassen zugeordnet sein, was als überlappende Spezialisierung bezeichnet wird. Eine Entität kann auch keiner Subklasse zugeordnet sein, hier handelt es sich um eine sogenannte partielle Spezialisierung. Muß eine Entität hingegen einer Subklasse zugeordnet sein, spricht

man von totaler Spezialisierung. Welche der vier Arten zum Einsatz kommt, wird vom Designer anhand der Anforderungen der Analysephase entschieden. Ein graphisches Beispiel einer Spezialisierung anhand des Personenverhältnisses einer Bank ist in Abbildung 3.3 zu sehen.

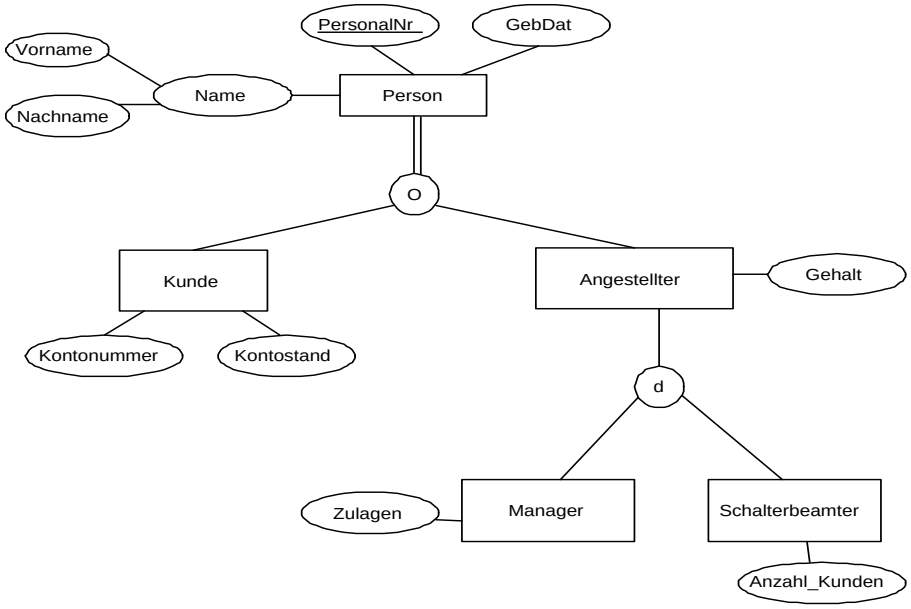


Abbildung 3.3: Spezialisierung in einem EER-Modell

Die Notation ist folgendermaßen zu verstehen: "Person" ist die Superklasse dieses EER-Modells. Eine Person wird durch das Attribut *PersonalNr* eindeutig identifziert und hat außerdem noch einen Name und ein Geburtsdatum. Spezialisierungen von "Person" sind die Subklassen "Kunde" und "Angestellter". Eine Person kann ein Kunde, ein Angestellter oder beides sein, die Spezialisierung ist also überlappend. Im EER-Diagramm wird die Spezialisierung durch einen Kreis angezeigt. Handelt es sich um eine überlappende Spezialisierung, befindet sich im Kreis der Buchstabe "o" für "overlapping". Die doppelte Linie oberhalb des Kreises signalisiert, daß es sich um eine totale Spezialisierung handelt, das heißt jede Person muß zu mindestens einer Subklasse zugeordnet werden. Die Subklassen enthalten sogenannte spezifische Attribute, die sie voneinander unterscheiden. Im Beispiel sind die Attribute *Kontonummer* und *Kontostand* spezifische Attribute der Klasse "Kunde", während *Gehalt* ein spezifisches Attribut der Klasse "Angestellter" ist. Ein Angestellter kann wiederum ein Manager oder ein Schalterbeamter sein, aber nicht beides. Hier ist eine weitere Spezialisierung vorhanden, die nicht total ist, das heißt ein Angesteller muß nicht notwendigerweise ein Manager oder ein Schalterbeamter sein. Es handelt sich also um eine partielle Spezialisierung, was in der Notation durch eine einfache Linie zwischen Superklasse und Kreis angezeigt wird. Allerdings handelt es sich zum eine disjunkte

Spezialisierung, ein Manager kann nicht gleichzeitig ein Schalterbeamter sein und umgekehrt, Im Kreis befindet sich also der Buchstabe “d“ für “disjoint“. *Zulagen* und *Anzahl_Kunden* sind spezifische Attribute der Klassen “Manager“ und “Schalterbeamter“.

Eine Generalisierung ist das Gegenteil einer Spezialisierung. Hier werden die gemeinsamen Attribute verschiedener Entitätstypen gesucht und zu zu einer Superklasse abstrahiert. Man könnte beim Modellieren der Personen einer Bank also auch von den Klassen “Manager“, “Schalterbeamter“ und “Kunde“ ausgehen und daraus im Zuge einer Generalisierung die Superklassen “Angestellter“ und “Person“ abstrahieren. Ansonsten wird für eine Generalisierung die gleiche Notation verwendet, wie für eine Spezialisierung.

3.2.2 Vereinigung

Bei Spezialisierung und Generalisierung existiert für alle Subklassen jeweils eine Superklasse. Es kann aber durchaus notwendig werden, daß mehrere Superklassen für eine Subklasse definiert werden müssen. Die Subklasse repräsentiert eine Vereinigung der Superklassen-Entitäten. Eine Veranschaulichung dieser Idee ist in Abbildung 3.4 zu sehen.

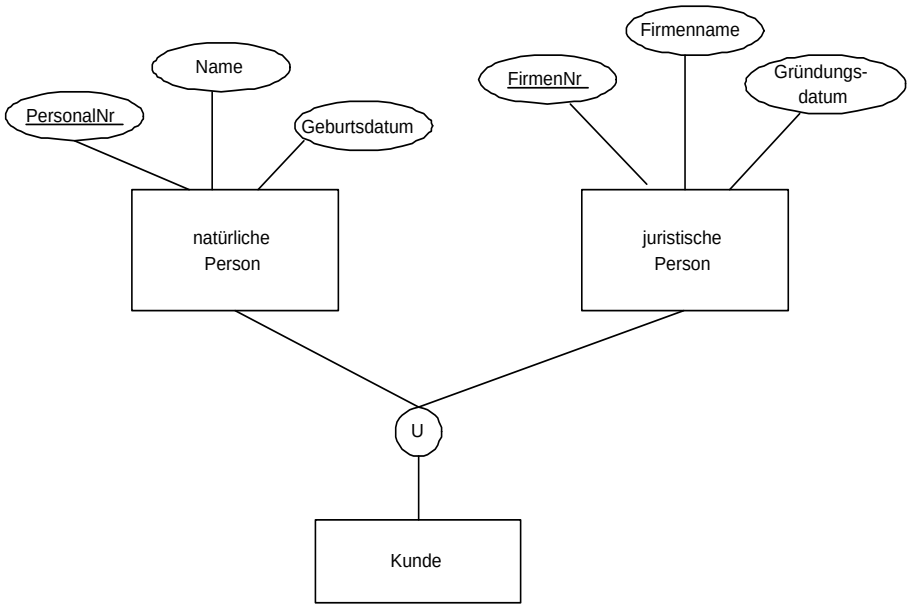


Abbildung 3.4: Vereinigung in einem EER-Modell

Die Subklasse “Kunde“ erbt hier die Attribute von zwei Superklassen, nämlich von “natürliche Person“ und von “juristische Person“. Eine Vereinigung wird in EER durch den Buchstabe “u“ im Kreis dargestellt. Es handelt sich um eine partielle Vereinigung, da eine natürliche oder juristische Person nicht notwendigerweise auch ein Kunde sein muß, was durch die einfache

Linie vom Kreis zu der Subklasse signalisiert wird.

Vereinigung ist nicht zu verwechseln mit sogenannten geteilten Subklassen. Eine geteilte Subklasse erbt Attribute von allen ihren Superklassen, es kann also auch eine Mehrfachvererbung auftreten, während bei einer Vereinigung die Subklasse immer nur die Attribute einer einzigen Superklasse erbt. Eine totale Vereinigung, bei der jede Entität einer Superklasse der Subklasse zugeordnet werden muß, kann auch als Generalisierung dargestellt werden. Die Superklassen der Vereinigung werden dabei zu den Subklassen der Generalisierung, analog gilt dasselbe für die Superklasse.

ER/EER ist ein sehr beliebtes Datenmodell, das auch noch häufig eingesetzt wird. Problematisch wird der Einsatz von ER für Datenbanken mit vielen Entitäten, die zahlreiche Attribute besitzen. Durch die platzraubende Notation wird das Diagramm schnell unübersichtlich und schwer lesbar. Weiters gibt es keine Möglichkeit Methoden, die für einen Entitätstyp definiert sind, darzustellen. Man hat daher in den letzten Jahren vermehrt nach Modellierungsalternativen gesucht, aus denen sich schließlich die Object Modeling Technique (OMT) und ihr Nachfolger die Unified Modeling Language (UML) als geeignet herauskristallisiert haben.

3.3 Die Unified Modeling Language

Die Unified Modeling Language ([Obj02]) wurde von Rumbaugh, Booch und Jacobson aus der Object Modeling Technique entwickelt. Das Ziel dieser Modellierungsmethode ist es, alle Diagramme, die in der Software-Entwicklung verwendet werden, zu vereinigen und eine Standardnotation anzubieten. Dazu zählen zum Beispiel Klassendiagramme, Use-Case-Diagramme, Datenflußdiagramme, Objekt-Diagramme, Statecharts und Sequenzdiagramme. Eingesetzt wird UML hauptsächlich im Bereich des Software-Engineering, vor allem für objekt-orientierte Software. Ein Teil daraus, die statischen Klassendiagramme eignen sich aber auch gut für konzeptuelles Datenbankdesign. Besonders vorteilhaft ist der Einsatz von UML für das Design von objekt-orientierten und objekt-relationalen Datenbanken, da sich ein objekt-orientiertes konzeptuelles Modell intuitiv auf ein objekt-orientiertes bzw. objekt-relationales Datenbankschema abbilden läßt.

Das Paradigma ist dasselbe wie bei ER-Diagrammen, es gibt aber Unterschiede in der Terminologie und der Notation. So entspricht ein Entitätstyp eines ER-Diagramms einer Klasse in UML, eine Entität selbst wird als Objekt bezeichnet. Beziehungen werden zu Assoziationen und in UML können auch die Methoden einer Klasse dargestellt werden. Vererbung,

Generalisierung/Spezialisierung und Vereinigung sind ebenfalls vorhanden, lehnen sich aber in UML stärker an den Bereich des Software-Engineering an, man spricht also hier auch von Instanzenvererbung, nicht nur von Schemavererbung. Neben Attributen werden auch die Methoden und Funktionen einer Superklasse an ihre Subklassen vererbt. Ein Überblick über die UML-Notation ist in Abbildung 3.5 zu sehen.

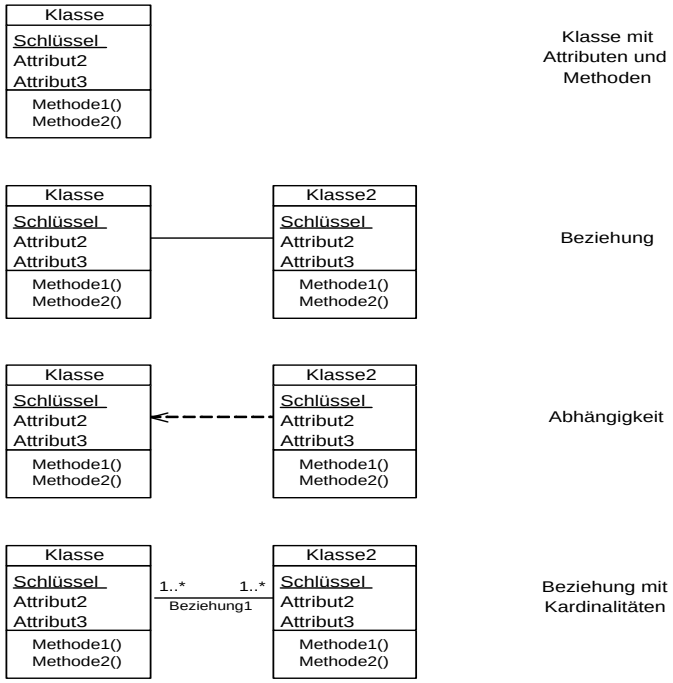


Abbildung 3.5: UML Notation

UML-Notation ist im Vergleich zu ER platzsparender, was bei umfangreichen Datenmodellen zu besserer Übersichtlichkeit und Lesbarkeit führt.

Da UML ein objekt-orientiertes Modell ist, ist Vererbung natürlich in der Notation enthalten. Spezialisierung, Generalisierung und Vereinigungen als Sonderformen der Vererbung sind daher ebenfalls vertreten. Die Notation ist in Abbildung 3.6 dargestellt.

Eine disjunkte oder überlappende Generalisierung, Spezialisierung oder Vereinigung erkennt man an der Farbe der Pfeilspitzen. Sind diese weiß, handelt es sich um überlappende Vererbung, sind sie schwarz, wird disjunkte Vererbung verwendet.

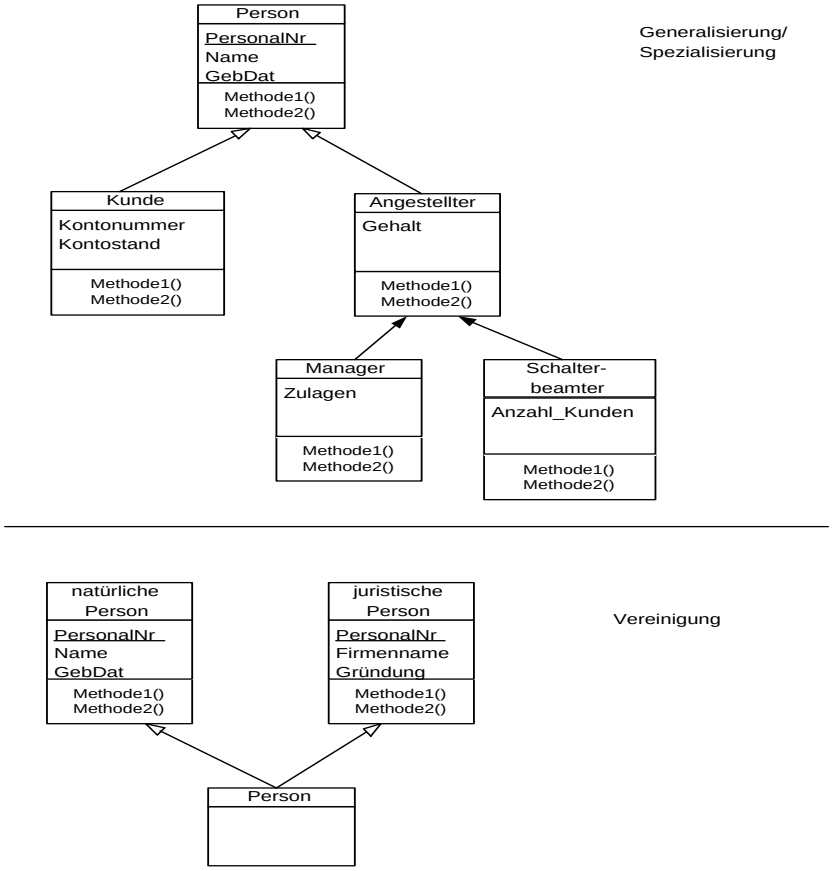


Abbildung 3.6: Generalisierung/Spezialisierung und Vereinigung in UML

3.3.1 Aggregation

UML bietet im Gegensatz zu ER Aggregationsbeziehungen an, also die Komposition zusammengehöriger Teile zu einem Ganzen. Dabei entsteht ein zusammengefaßtes Objekt (ein Aggregat), dessen Verhalten und Behandlung auch immer seine einzelnen Teile betrifft. Ein Beispiel für ein Aggregat ist ein Fahrrad, das aus einem Rahmen, zwei Reifen, verschiedenen Schrauben, einem Sattel und einer Lenkstange besteht. Bewegt sich das Fahrrad, bewegen sich auch seine Teile, das Fahrrad ist also ein Aggregat, das aus einzelnen Komponenten zusammengesetzt ist. Die Notation für eine Aggregationsbeziehung ist in Abbildung 3.7 zu sehen. Eine alternative Bezeichnung für eine Aggregationsbeziehung ist *Is-Part-Of*-Beziehung.

Damit sind die wichtigsten Bestandteile konzeptueller Modellierung sowohl in ER/EER als auch in UML zusammengefaßt. Die Abbildung des konzeptuellen Datenmodells in ein logisches Datenmodell mit Hilfe objekt-relationaler Erweiterungen soll nun im nächsten Kapitel

ausführlich dargelegt werden. Der Schwerpunkt wird dabei vor allem auf die Abbildung von Generalisierung, Spezialisierung, Vereinigung und Aggregation gelegt. Da sich UML für objekt-relationales Datenbank-Design besser eignet, wird in weiterer Folge als zugrundeliegendes konzeptuelles Modell ausschließlich UML verwendet. Es wird außerdem empfohlen, bei der Erstellung objekt-relationaler Datenbanken die Unified Modeling Language dem Entity-Relationship-Modell, aus Gründen der Übersichtlichkeit und Nähe der Paradigmen des konzeptuellen und objekt-relationalen logischen Schemas, vorzuziehen.

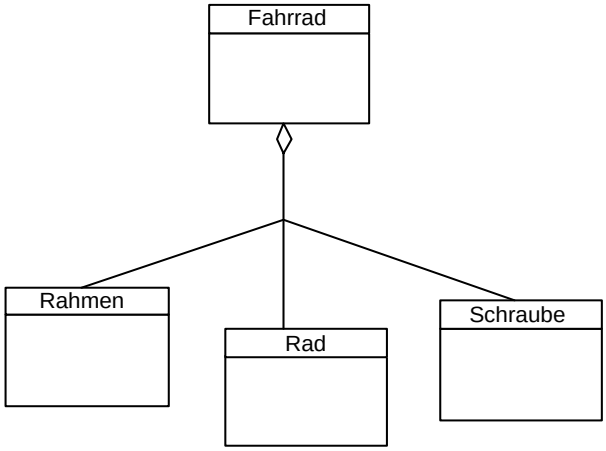


Abbildung 3.7: Aggregationsbeziehung

3.4 Mengen- und Transaktionsprofil

Die folgenden Definitionen sind im Wesentlichen dem Buch “Conceptual Database Design“ von Batini, Ceri und Navathe [BCN92] entnommen. Bei Mengen- und Transaktionsprofilen geht es um die Optimierung des logischen Datenmodells im Hinblick auf die Last, die auf diesem Datenbankschema in Zukunft entstehen wird. Anhand dieser Daten können Entscheidungen getroffen werden, welche Variante verschiedener Abbildungsverfahren des logischen Designs für die zu entwickelnde Datenbank die beste Performance bietet und demnach einzusetzen ist.

3.4.1 Mengenprofil

Beim Erstellen eines Mengenprofils sind folgende Informationen relevant:

- durchschnittliche Anzahl der Instanzen pro Entität

- durchschnittliche Anzahl teilnehmender Instanzen pro Beziehung
- durchschnittliche Kardinalität einer Instanz, die an einer Beziehung teilnimmt

Aus diesen Informationen werden sogenannte *Datenvolumen*-Tabellen erstellt, in denen die Zahlen tabellarisch aufbereitet werden. Ein Beispiel für eine Datenvolumen-Tabelle ist in Abbildung 3.8 zu sehen. Als zugrundeliegendes Schema wurde das Beispiel aus Abbildung 3.11 verwendet, wobei davon ausgegangen wird, daß einer Person mehrere Dienstautos zugeordnet werden können.

Konzept	Typ	Anzahl
Person	E	1000
Dienstauto	E	2000
Person_hat_Dienstauto	R	3000
PersonalNr	A	1000
Vorname	A	1000
Nachname	A	1000
Kennzeichen	A	2000
PS	A	2000
Kilometerstand	A	2000

Abbildung 3.8: Beispiel einer Datenvolumen-Tabelle

Für jede Entität, für jede Beziehung und für jedes Attribut ist die durchschnittliche Anzahl seiner Instanzen festgehalten. Die durchschnittliche Kardinalität einer Instanz für eine Beziehung kann aus dieser Tabelle berechnet werden. So sind einer Person durchschnittlich drei Dienstautos zugeordnet, was durch eine Division der Instanzen der Beziehung *Person_hat_Dienstauto* durch die Instanzen der Entität *Person* errechnet werden kann. Im Beispiel gibt es keine mehrwertigen Attribute, daher ist die durchschnittliche Anzahl der Attribute immer gleich der durchschnittlichen Anzahl der Instanzen.

3.4.2 Transaktionsprofil

Ein Transaktionsprofil beschäftigt sich mit den Operationen, die auf das Datenbankschema zugreifen werden. Da eine Applikation meistens eine Vielzahl von verschiedenen Operationen hat, ist es zuviel Aufwand sich mit den Details jeder einzelnen zu beschäftigen. Daher werden nur die wichtigsten Transaktionen, von denen vermutet wird, daß sie am meisten Einfluß auf

das Laufzeitverhalten haben, genau untersucht. Relevante Informationen über Operationen sind folgende:

- Elemente des konzeptuellen Schemas, auf die in der Operation zugegriffen wird, einschließlich der Zugriffspfade
- Art des Zugriffs (lesend oder schreibend)
- durchschnittliche Anzahl der Instanzen, die von einem Zugriff betroffen sind
- Häufigkeit der Operation im Beobachtungszeitraum
- Typ der Operation (Online oder Batch)

Diese Informationen werden mit Hilfe von zwei Tabellen dargestellt, der Operations-Häufigkeits-Tabelle und der Operations-Zugriffshäufigkeits-Tabelle. Erstere wird pro Schema einmal erstellt und beinhaltet Informationen über alle Operationen, wie zum Beispiel die Häufigkeit und den Typ der Operation. Letztere wird für jede Operation einmal erstellt und beinhaltet Informationen über Zugriffspfade, Anzahl der betroffenen Instanzen und Art des Zugriffs. Aus Gründen der Übersichtlichkeit werden mehrere Operationen in einer Operations-Zugriffshäufigkeits-Tabelle zusammengefaßt.

Name/Kurzbeschreibung	Häufigkeit	Typ (Online/Batch)
O1: Anlegen einer Person	3x/Tag	OL
O2: Anlegen eines Dienstautos	6x/Tag	OL
O3: Zuordnung von Dienstautos zu Personen	12x/Tag	OL
O4: Abfragen von Zuordnungen	150x/Tag	OL
O5: Abfragen von Personen	600x/Tag	OL
O6: Abfragen von Dienstautos	325x/Tag	OL
O7: Löschen einer Person	1x/Woche	B
O8: Löschen eines Dienstautos	1x/Woche	B
O9: Löschen von Zuordnungen	3x/Tag	OL

Abbildung 3.9: Beispiel einer Operations-Häufigkeits-Tabelle

Operationen, die als Batch ausgeführt werden, werden häufig nur in die Operations-Häufigkeits-Tabelle aufgenommen, Operations-Zugriffshäufigkeits-Tabellen werden für diese Operationen aber nicht erstellt. Das Laufzeitverhalten spielt hier meistens keine große Rolle, da Batch-Operationen selten oder zu Zeiten durchgeführt werden, in denen wenig Zugriffe auf die

Datenbank erfolgen, zum Beispiel in der Nacht. Daher sollte man sich bei der Erstellung eines logischen Datenmodells an der Optimierung für Online-Transaktionen orientieren, da hier das Laufzeit- und Antwortzeitverhalten eine wesentliche Rolle spielt.

Ein Beispiel für eine Operations-Häufigkeits-Tabelle ist Abbildung 3.9. In der Tabelle ist zu sehen, daß in diesem System häufig Abfragen auf die Relationen getätigt werden, während Modifikationen der Daten eher selten erfolgen. Für einige der identifizierten Operationen werden nun Operations-Zugriffshäufigkeits-Tabellen erstellt. Ein Beispiel ist in Abbildung 3.10 zu sehen und zwar für die Operationen O1 bis O6. Für jede Operation wird festgehalten, auf welche Elemente des konzeptuellen Modells zugegriffen wird und um welchen Typ von Elementen es sich dabei handelt. Weiters wird für jeden Zugriff notiert, ob es sich um einen lesenden oder schreibenden Zugriff handelt und die durchschnittliche Anzahl der vom Zugriff betroffenen Instanzen.

Name/Kurzbeschreibung	Konzept	Konzept-Typ	Read/Write	Durchschnitt. Anzahl betroffener Instanzen
O1: Anlegen einer Person	Person	Entität	W	3
O2: Anlegen eines Dienstautos	Dienstauto	Entität	W	6
O3: Zuordnung von Dienstautos zu Personen	Dienstauto	Entität	R	12
	Person	Entität	R	12 x 1,5 = 18
	Person_hat_ Dienstauto	Beziehung	W	12 x 1,5 = 18
O4: Abfragen von Zuordnungen	Person	Entität	R	150
	Dienstauto	Entität	R	150 x 2 = 300
	Person_hat_ Dienstauto	Beziehung	R	150 x 2 = 300
O5: Abfragen von Personen	Person	Entität	R	600
O6: Abfragen von Dienstautos	Dienstauto	Entität	R	325

Abbildung 3.10: Beispiel einer Zugriffshäufigkeits-Tabelle

Betrachtet man beispielsweise Operation 3 (O3), die Zuordnung von Dienstautos zu Personen, genauer, findet man folgendes heraus: Im Zuge dieser Operation werden zuerst Dienstautos gelesen, die zugeordnet werden sollen. Danach werden die Personen gelesen, denen Dienstautos zugeordnet werden sollen. Im Schnitt werden pro Dienstauto 1,5 Personen zur Zuordnung benötigt, daher muß für jede Instanz eines Dienstautos die 1,5-fache Anzahl an Personen gelesen werden. Sind sowohl Dienstautos als auch Personen gelesen, muß ein Datensatz für jede Zuordnung in der Beziehungstabelle festgehalten werden. Auch hier handelt es sich um die 1,5-fache Anzahl von Dienstautos, da pro Dienstauto durchschnittlich 1,5 Zuordnungen festgehalten werden müssen.

Das Mengen- und Transaktionsprofils eines konzeptuellen Schemas wird zur Entscheidungshilfe beim Erstellen eines logischen Modells herangezogen, wenn man mehrere Abbildungsvarianten hat, mit denen Konzepte aufgelöst werden können. Anhand der im Mengen- und Transaktionsprofil gesammelten Informationen kann man entscheiden, welche Variante für das betrachtete Datenbankschema die geeignetste ist. Für das Mengen- und Transaktionsprofil spielt es keine Rolle, ob in weiterer Folge auf ein relationales oder ein objekt-relationales Schema abgebildet wird. Das Mengen- und Transaktionsprofil wird für das konzeptuelle Schema erstellt und ist neben den konzeptuellen Schema und dem DBMS ein weiterer Input für den logischen Design-Prozeß.

Welche Abbildungsvorschriften man nun bei der Erstellung objekt-relationaler, logischer Datenbankschemata hat, soll in den nächsten Abschnitten definiert werden.

3.5 Generelle Abbildungsvorschriften

UML berücksichtigt - im Gegensatz zu einem ER-Modell - sowohl die statischen als auch die dynamischen Komponenten des Datenmodells. Diese Arbeit soll sich auf die Abbildung des statischen Teils, also eines Klassenmodells, in ein logisches Datenmodell konzentrieren. Die dynamische Komponente, zu der State-Chart-, Use-Case- und Aktititäten-Diagramme gehören, wird hier nicht behandelt.

Zunächst steht man vor der Aufgabe Klassen und ihre Beziehungen in ein logisches Datenmodell abzubilden. Dabei wird für jede Klasse ein zusammengesetzter Datentypen angelegt, dessen Attribute den Klassenattributen entsprechen. Dabei können für Attribute die Basisdatentypen des DBMS verwendet oder benutzerdefinierte Datentypen angelegt werden. Auf dem zusammengesetzten Datentyp wird eine Tabelle definiert. Für Tabellen, denen ein zusammengesetzter Datentyp zugrunde liegt, können Beziehungen zu anderen Tabellen mit Hilfe von Referenzen abgebildet werden. Ein Beispiel soll diese grundlegende Vorgehensweise veranschaulichen.

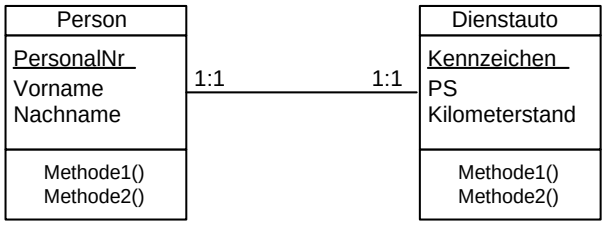


Abbildung 3.11: Beispiel Dienstautoverwaltung

In Abbildung 3.11 ist ein kleines Datenmodell zu sehen, das eine Klasse enthält, in der Personen verwaltet werden und eine weitere Klasse, in der Dienstautos verwaltet werden. Jeder Person soll dabei ein Dienstauto zugeordnet werden. Ein Dienstauto kann nur einer Person zugeordnet werden.

Für das Attribut *Name* der Klasse "Person" wird ein zusammengesetzer Datentyp definiert, dessen Attribute dem Vor- und Nachname entsprechen. Für die Klassen "Person" und "Dienstauto" wird jeweils ein zusammengesetzter Datentyp definiert, der die Attribute der Klasse abbildet.

```
create type name_typ {
          vorname     VARCHAR(20),
          nachname    VARCHAR(20)
}

create type dienstauto_typ {
               kennzeichen         VARCHAR(50),
               PS                  SMALLINT,
               kilometerstand      VARCHAR(10),
}

create type person_typ {
              personalNr             CHAR(5),
              name                   name_typ,
              ref_dienstauto         ref(dienstauto_type)
}
```

Man sieht, daß im zusammengesetzten Datentyp "person_typ" ein Attribut verwendet wird, das wiederum auf einem zusammengesetzten Datentyp aufbaut, nämlich das Attribut *Name*. Diese Schachtelung kann beliebig tief sein. Auf den zusammengesetzten Datentypen "dienstauto_typ" und "person_typ" werden nun Tabellen definiert. Eine Tabelle, die auf einem zusammengesetzten Datentyp beruht, wird auch als *getypte* Tabelle bezeichnet.

```
create table dienstautos of type dienstauto_typ;

create table personen of type person_typ;
```

Die 1:1-Beziehung zwischen diesen beiden Tabellen wird durch die Referenz abgebildet. Die referenzierende Tabelle "Person" enthält das Attribut, in dem der OID eines bestimmten Dienstautos

gespeichert ist. Man könnte in diesem Fall, da es sich um eine 1:1-Beziehung handelt, die Referenz auch umdrehen und in der Tabelle “Dienstautos“ ein Attribut definieren, das eine Referenz auf eine bestimmte Person enthält. Hat man eine 1:N-Beziehung, wird das referenzierende Attribut der Tabelle auf der “N-Seite“ der Beziehung zugeordnet. So könnte ein Dienstauto auch mehreren Personen zugeordnet werden, wenn es nur für Dienstfahren zur Verfügung gestellt wird, nicht rund um die Uhr. In diesem Fall enthält die Tabelle “Personen“ eine Referenz auf die Tabelle “Dienstautos“. Mehrere Instanzen, also Personen, können dann auf dasselbe Dienstauto verweisen. Einem Mitarbeiter können auch beliebig viele Dienstautos zugeteilt werden, je nachdem welche Fahrzeuge gerade verfügbar sind, kann er wählen, mit welchem er fahren möchte. In diesem Fall liegt eine M:N-Beziehung vor. Für M:N-Beziehungen wird eine eigene Tabelle definiert, deren einzige Attribute Referenzen auf die an der Beziehungen teilnehmenden Tabellen sind. Für die Mitarbeiter-Dienstauto-Beziehung wird also eine Tabelle mit zwei Referenz-Attributen definiert, eine Referenz auf den Mitarbeiter und eine auf das Dienstauto. In der Tabelle werden als Instanzen Mitarbeiter-Dienstauto-Paare gespeichert, die einander zugeordnet sind. Diese Abbildungsmöglichkeit von Beziehungen hat große Ähnlichkeit mit der Abbildung in relationalen Datenbanken, mit dem einzigen Unterschied, daß die Primär- und Fremdschlüssel durch Referenzen ersetzt werden. Da auch Mengen von Referenzen in einem objekt-relationalen DBMS unterstützt werden sollen, ist es auch möglich eine M:N-Beziehung so abzubilden, daß in jeder der teilnehmenden Tabellen ein Attribut als Menge von Referenzen definiert wird, das die Referenzen auf die Partner-Tabelle enthält, die teilnehmenden Tabellen referenzieren sich also gegenseitig. Das Prinzip dieser Abbildungsart ist in Abbildung 3.12 zu sehen.

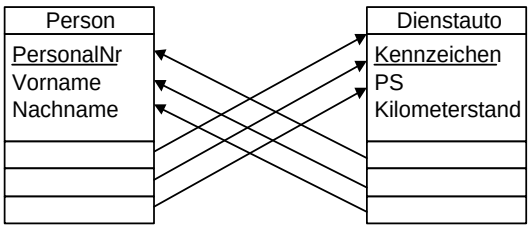


Abbildung 3.12: Auflösung einer M:N-Beziehung durch Mengen von Referenzen

Für mehrwertige Attribute hat man zwei verschiedene Möglichkeiten, das Attribut aus dem konzeptuellen Modell in ein Attribut aus dem logischen Modell überzuführen. Man kann zunächst eine eigene Tabelle für das mehrwertige Attribut anlegen, die ein referenzierendes Attribut auf die Eltern-Tabelle, der das mehrwertige Attribut zugeordnet ist, enthält. Diese Variante entspricht im wesentlichen der relationalen Lösung mit dem Unterschied, daß anstelle eines Fremdschlüssels eine Referenz verwendet wird. Eine weitere Variante ist die Verwendung

von Mengen. Wie bereits im Abschnitt 2.2.2 erwähnt, wird für jeden Datentyp im System, sei es ein Basisdatentyp, ein benutzerdefinierter oder zusammengesetzter Datentyp, auch ein Typ für die Menge dieses Datentyps angelegt. Ein mehrwertiges Attribut kann nun in seiner Elterntabelle auch als Menge eines Datentyps definiert werden. Damit werden die unterschiedlichen Werte dieses Attributs direkt in der Elterntabelle gespeichert und nicht mehr außerhalb, wie es bei der ersten Lösungsvariante der Fall war. Das hat den Vorteil, daß bei einer Abfrage auf das mehrwertige Attribut keine Referenzen aufgelöst werden müssen, sondern direkt auf die Eltern-Tabelle zugegriffen werden kann. Dargestellt sind diese beiden Varianten in Abbildung 3.13. Die Unterteilung des Attributes *Telefonnummern* in mehrere Bereiche im unteren Teil der Abbildung soll veranschaulichen, daß es sich bei den Telefonnummern um eine Menge handelt.

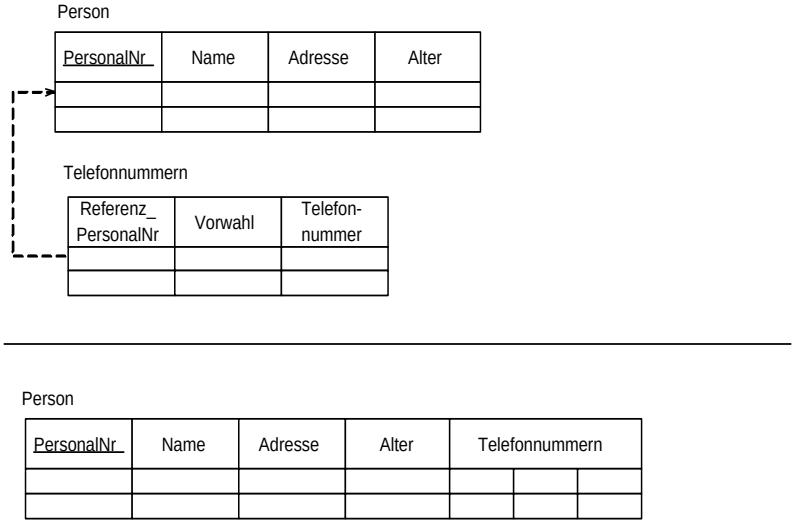


Abbildung 3.13: Abbildung mehrwertiger Attribute

3.6 Vererbung

Vererbung ist, wie bereits in Abschnitt 2.2.4 beschrieben, eines der komplexesten Konzepte des konzeptuellen Modells. Aufgrund dieser Komplexität ist Vererbung im logischen Designprozeß eine der am schwierigsten aufzulösenden Beziehungen. Es gibt in relationaler logischer Modellierung eine Reihe von Möglichkeiten, wie Vererbung auf Tabellen abgebildet werden kann, mit der Verwendung objekt-relationaler Erweiterungen kommen noch zusätzliche Varianten hinzu. Generelle Probleme, die bei Vererbung auftreten können, sind Konflikte zwischen den ererbten Attributen und den Attributen der Subklasse. Tritt so ein Konflikt bei einfacher Vererbung auf sollte man zunächst prüfen, ob es sich nicht um einen Fehler im Design handelt.

Da eine Subklasse nur spezielle Attribute enthält, die sie von der Superklasse unterscheiden, sollte es nicht vorkommen, daß Attribute mit demselben Namen vorhanden sind. Relevant wird diese Problemstellung erst bei Mehrfachvererbung, wo es relativ häufig zu solchen Konflikten kommen kann, vor allem dann, wenn die Superklassen selbst wieder gemeinsame Vorfahren haben. Wie bei Mehrfachvererbung mit Strukturkonflikten umgegangen wird, ist in Abschnitt 3.9 genau definiert. In einer einfachen Vererbungsbeziehung sollten solche Strukturkonflikte allerdings nicht auftreten, da es sich dabei meist um einen Design-Fehler handelt. Die beiden getesteten DBMS unterstützen die Auflösung solcher Strukturkonflikte nicht, es ist bei beiden nicht möglich, daß ein Atttribut des Subtypen denselben Name wie ein Attribut des Supertypen hat.

Das Beispiel, das hier zur Veranschaulichung der Abbildung verwendt wird, ist konfliktfrei. Zunächst wird die Abbildung von Vererbung für ein relationales Datenmodell gezeigt, danach die Abbildung für ein objekt-relationales Datenmodell. Die Überleitungen basieren auf dem Beispielschema von Abbildung 3.14.

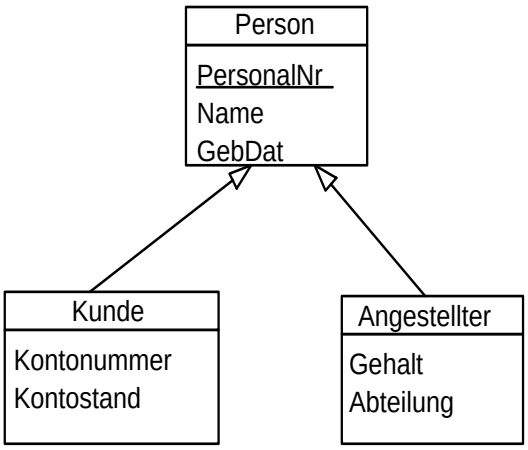


Abbildung 3.14: konzeptuelles Beispielmodell für Vererbung

3.6.1 Abbildung in einem relationalen Datenmodell

In einem relationalen Datenmodell hat man vier Möglichkeiten eine Vererbungsbeziehung in ein logisches Schema überzuführen.

Variante A - Superklasse mit Subklassen: Die Superklasse K wird in eine Tabelle umgewandelt, die Attribute $\{a_1..a_m\}$ der Superklasse werden Attribute der Tabelle, der Schlüssel

k der Superklasse wird zum Primärschlüssel der Tabelle. Für jede Subklasse S_i wird ebenfalls eine Tabelle angelegt. Die Attribut von S_i werden dabei zu Attributen der Tabelle. Zusätzlich wird das Schlüsselattribut k der Superklasse in jede Tabelle einer Subklasse übernommen, wo es als Primärschlüssel fungiert. Variante A, angewandt auf das Beispiel aus Abbildung 3.14 ist in Abbildung 3.15 zu sehen. Bei dem Beispiel handelt es sich um eine Spezialisierungsbeziehung, die wie in Abschnitt 3.2.1 bereits erwähnt, eine Sonderform der Vererbung darstellt.

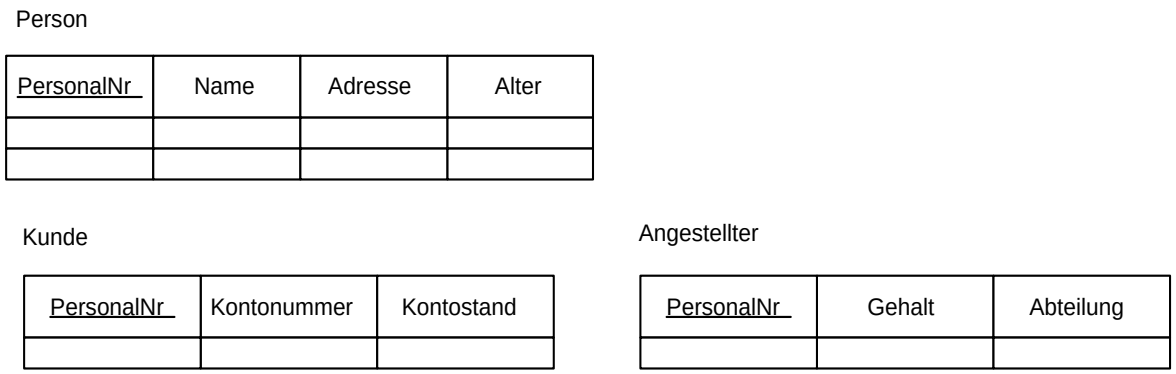


Abbildung 3.15: Relationales Modell - Vererbung aufgelöst nach Variante A

Die Subklassen werden bei Abfragen über das Schlüsselattribut mit der Superklasse verbunden. Variante A kann sowohl für totale oder partielle Vererbung als auch für disjunkte oder überlappende Vererbung verwendet werden. Allerdings sollte man nicht vergessen, daß ein Join eine kostenintensive Operation ist. Hat also die Superklasse nur wenige Attribute, ist Variante B empfehlenswert.

Variante B - Nur Subklassen: Der Join, der in Variante A die Subklassen mit der Superklasse verbindet, wird bei Variante B in die Tabellen "eingebaut". Dazu wird für jede Subklasse Si eine Tabelle angelegt, die neben den Attributen der Subklasse auch die Attribute $\{a_1..a_m\}$ der Superklasse enthält. Das Schlüsselattribut der Superklasse wird zum Primärschlüssel in jeder Tabelle einer Subklasse. Variante B ist in Abbildung 3.16 zu sehen. Variante B ist nicht für alle Arten von Vererbung geeignet. Bei einer partiellen Vererbung gehen Instanzen, die keiner Subklasse angehören, verloren. Bei überlappender Vererbung werden alle Attribute, die von der Superklasse geerbt werden, redundant gespeichert. Variante B sollte daher vor allem bei totaler, disjunkter Vererbung eingesetzt werden, oder bei Superklassen, die nur wenig Attribute beinhalten.

Kunde

Personal_ Nr	Name	Adresse	Alter	Kontonummer	Kontostand

Angestellter

Personal_ Nr	Name	Adresse	Alter	Gehalt	Abteilung

Abbildung 3.16: Relationales Modell - Vererbung aufgelöst nach Variante B

Variante C - Einzelrelation mit Unterscheidungsattribut: Eine Tabelle wird angelegt, die sowohl die Attribute der Superklasse als auch die Attribute aller Subklassen enthält. Das Schlüsselattribut der Superklasse wird zum Primärschlüssel der Tabelle. Zusätzlich wird ein Attribut benötigt, das signalisiert, zu welcher Subklasse die Instanzen zugehörig sind. Variante C ist in Abbildung 3.17 veranschaulicht.

Person

Personal_ Nr	Name	Adresse	Alter	Kontonummer	Kontostand	Gehalt	Abteilung	Subklasse_ Typ

Abbildung 3.17: Relationales Modell - Vererbung aufgelöst nach Variante C

Variante C ist nur auf disjunkte Vererbung anwendbar, da das differenzierende Attribut für jede Instanz maximal einen Wert enthalten kann. Allerdings ist sowohl totale als auch partielle Vererbung möglich. Gehört eine Instanz keiner Subklasse an enthält das differenzierende Attribut keinen Wert.

Variante D - Einzelrelation mit bool'schen Attributen: Variante D ähnelt in ihrem Verfahren sehr Variante C, mit einem Unterschied. Für jede Subklasse wird ein bool'sches Attribut definiert, das anzeigt, ob eine Instanz dieser Subklasse zugeordnet ist oder nicht. Diese Darstellung ist in Abbildung 3.18 zu sehen. Durch die bool'schen Attribute eignet sich Variante D auch für überlappende Vererbung. Die Zugehörigkeit einer Instanz zu den verschiedenen

Subklassen wird für jede Subklasse separat verwaltet. Ist für alle bool'schen Attribute kein Wert definiert, ist die Instanz keiner Subklasse zugeordnet, es handelt sich dann um partielle Vererbung.

Person

Personal_ Nr	Name	Adresse	Alter	Kontonummer	Kontostand	Gehalt	Abteilung	Ist_Kunde	Ist_ Angestellter

Abbildung 3.18: Relationales Modell - Vererbung aufgelöst nach Variante D

Das Resultat von Variante C und D ist eine einzige große Tabelle, die sehr viele Attribute hat. Variante A und B verteilen die Subklassen auf eigene Tabellen. Welche Variante bevorzugt wird hängt vor allem davon ab, welche Operationen im Transaktionsprofil definiert wurden, wie diese auf die Sub- und Superklassen zugreifen und welches Performanz-Verhalten sich daraus ergibt. Subklassen mit vielen Attributen sollten in einer disjunkten Vererbungsbeziehung mit Hilfe von Variante A oder B abgebildet werden, da sonst Tabellen mit einer großen Anzahl von Null-Values entstehen. Wenn allerdings bei Variante A die Subklassen immer in Verbindung mit der Superklasse benötigt werden, muß ein Join durchgeführt werden, der mit hohen Kosten verbunden ist. Ebenso muß bei Variante B, wenn nur die Superklassen-Attribute benötigt werden, eine UNION-Operation durchgeführt werden, die ebenfalls sehr kostenintensiv ist. Hat man nur wenige Subklassen mit einer geringen Anzahl von Attributen, eignen sich Variante C und D besser für die Abbildung.

Das Laufzeitverhalten dieser vier Varianten im Hinblick auf gegebene Mengen- und Transaktionsprofile ist bekannt und in der einschlägigen Literatur zu finden. In einem objekt-relationalen Datenmodell können diese vier Varianten zur Abbildung von Vererbung ebenfalls eingesetzt werden. Zusätzlich gibt es eine Reihe von weiteren Varianten, die den eben vorgestellten zum Teil sehr ähnlich sind, die aber objekt-relationale Erweiterungen bei der Abbildung verwenden. Auf diese Varianten und ihr Laufzeitverhalten soll in den nächsten Abschnitten besonderes Augenmerk gelegt werden.

3.6.2 Abbildung in einem objekt-relationalen Datenmodell

Bildet man Vererbung von einem konzeptuellen Modell in ein objekt-relationales logisches Datenmodell ab, bieten sich fünf Möglichkeiten an, wie eine Vererbungsbeziehung aufgelöst

werden kann. Die meisten sind der relationalen Abbildung sehr ähnlich, basieren aber auf objekt-relationalen Erweiterungen. Zusätzlich wird Vererbung vom DBMS selbst angeboten, das heißt die Vererbungsbeziehung aus dem konzeptuellen Datenmodell kann direkt in der Datenbank abgebildet werden.

Als Beispiel soll die Personen-Verwaltung einer Bank dienen, deren Datenstruktur in Abbildung 3.14 zu sehen ist. Es gibt vier Personen: Eine Person ist Kunde und Angesteller, eine weitere ist nur Kunde. Eine Person ist nur Angestellter und eine ist weder Kunde noch Angestellter (Abbildung 3.19).

Personalnr	Name	Adresse	Alter	Kontonummer	Kontostand	Gehalt	Abteilung
25	Mustermann	Hauptstraße 46, 9020 Klagenfurt	34	18880000005	4.500,--	900,--	EDV
30	Müller	Blumenweg 34, 9500 Villach	25	24540000033	6.300,--		
35	Maier	Lannerweg 9, 9201 Krumpendorf	28			1.300,--	Management
40	Huber	Weiherstraße 56, 9020 Klagenfurt	37				

Abbildung 3.19: Beispieldaten von Personen

Diese vier Personen dienen als Beispiel für disjunkte und überlappende, sowie totale und partielle Vererbung. Eine Person, die Kunde und Angestellter ist, repräsentiert totale und überlappende Vererbung. Eine Person, die nur Kunde oder nur Angestellter ist, repräsentiert totale und disjunkte Vererbung. Eine Person, die weder Kunde noch Angestellter ist, repräsentiert partielle Vererbung. Bei den folgenden Abbildungsvarianten soll anhand dieser vier Personen gezeigt werden, ob sich die Variante für totale, partielle, disjunkte oder überlappende Vererbung eignet.

Variante I - Subklassen mit Referenz auf Superklasse: Für die Superklasse wird zunächst ein zusammengesetzter Datentyp definiert, der die Attribute der Superklasse enthält. Auf diesem Datentyp wird in Folge eine Tabelle definiert. Das Schlüsselattribut wird zum Primärschlüssel. Für jede Subklasse S_i wird ebenfalls ein zusammengesetzter Datentyp mit den Attributen der Subklasse angelegt. Für jeden der zusammengesetzten Datentypen wird eine Tabelle angelegt. In jeder Tabelle einer Subklasse wird weiters ein Attribut definiert, das auf die Tabelle der Superklasse referenziert. Variante I ist in Abbildung 3.20 dargestellt. Variante I hat sehr viel Ähnlichkeit mit Variante A aus Abschnitt 3.6.1. Anstelle von Primär- und Fremdschlüsseln werden jedoch Referenzen verwendet. Das DBMS löst die Referenzen bei Bedarf in einen Join auf, außerdem kümmert es sich um die Erhaltung der referentiellen Integrität der Daten. Ändert

sich der Primärschlüssel einer Instanz, bleibt eine Referenz durch die Verwendung eines OID davon normalerweise unberührt.

Abbildung 3.20: Objekt-relationales Modell - Vererbung aufgelöst nach Variante I

Abfragen auf die Tabellen der Super- und Subklassen können durch die Verwendung von Punkt-Notation elegant und übersichtlich formuliert werden. Die Join-Bedingung, die im relationalen Modell zur Verbindung der Super- und Subklassen angegeben werden muß, entfällt. Allerdings muß der Designer hier schon eine Entscheidung treffen, in welche Richtung referenziert werden soll. In der Abbildung referenzieren die Tabellen "Kunde" und "Angestellter" auf die Tabelle "Person". Man könnte aber den Verlauf der Referenzen auch ohne weiteres umdrehen, sodaß die Tabelle "Person" auf die Tabellen "Kunde" und "Angestellter" referenziert. Der Designer muß hier also entscheiden, in welche Richtung referenziert wird. Soll eine bidirektionale Beziehung abgebildet werden, müssen beide teilnehmenden Tabellen einander gegenseitig referenzieren. Die Abbildung einer Beziehung, wie sie in einem relationalen Modell mit Hilfe eines Fremdschlüssels erfolgt, ist immer bidirektional, die Richtung, in der der Join später erfolgen wird, ist hier in der Design-Phase noch nicht von Bedeutung.

Eine Referenz kann weiters immer nur auf eine getypte Tabelle zeigen, nie auf eine relationale. Um eine getypte Tabelle zu erweitern, muß ihr zugrundeliegender Datentyp geändert werden. Dazu müssen sämtliche Daten aus der Tabelle exportiert, die Tabelle gelöscht und neu angelegt werden, da es nicht möglich ist, den Datentyp zu modifizieren, wenn Tabellen auf ihm definiert sind. Eine Erweiterung der getypten Tabelle ist damit im Gegensatz zur Erweiterung einer relationalen Tabelle aufwendig zu realisieren. Da aber bei der Verwendung von Referenzen ausschließlich getypte Tabellen verwendet werden können, kann die Erweiterung oder die

Änderung des Datenmodells sehr problematisch werden.

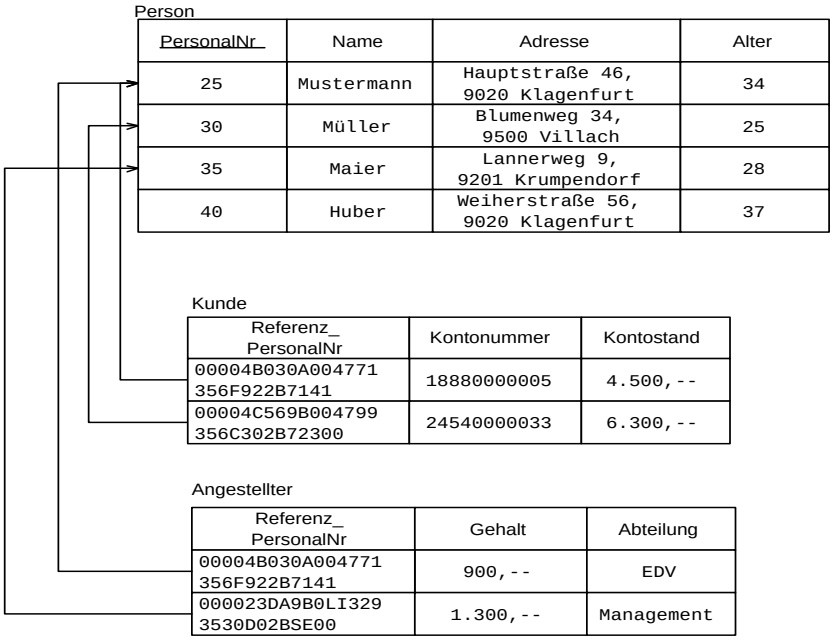


Abbildung 3.21: Variante I - Zuordnung der Beispiel-Instanzen

Variante I eignet sich für partielle, totale, disjunkte und überlappende Vererbung, wie in Abbildung 3.21 zu sehen ist. Alle vier Personen können ohne Probleme in die Tabellen eingetragen werden. Personen, die weder Kunden noch Angestellte sind, haben keinen Eintrag in den Subklassen-Tabellen. Personen, die sowohl Kunden als auch Angestellte sind, haben in beiden Subklassen-Tabellen einen Eintrag, wobei darauf zu achten ist, daß die Referenz auf denselben Datensatz in der Superklassen-Tabelle zeigt.

Variante II - Subklassen mit zusammengesetztem Datentyp der Superklasse: Für die Attribute der Superklasse wird ein zusammengesetzter Datentyp definiert. Für jede Subklasse S_i wird eine Tabelle angelegt, die als Attribute den zusammengesetzten Datentyp der Superklasse und die Attribute der Subklasse enthält. Das Schlüsselattribut der Superklasse wird der Primärschlüssel in der Tabelle der Subklasse. Dieses Prinzip ist in Abbildung 3.22 dargestellt.

Variante II kann nicht für partielle Vererbung verwendet werden. Superklassen, die keiner Subklasse angehören, gehen verloren, da sie keiner passenden Tabelle zugeordnet werden können. Gibt es in diesem Fall Personen, die weder Kunden noch Angestellte sind, können diese keiner der beiden Tabellen aus Abbildung 3.22 zugeordnet werden, der Datensatz geht verloren. Es

muß also totale Vererbung, die disjunkt oder überlappend ist, vorhanden sein. Bei überlappender Vererbung werden alle Attribute der Superklasse redundant gespeichert. Ist ein Angestellter zum Beispiel auch Kunde, müssen seine Personendaten sowohl in der Angestellten- als auch in der Kunden-Tabelle gespeichert werden, wie in Abbildung 3.23 zu sehen ist. Die Person "Huber" geht verloren, da sie weder Kunde noch Angestellter ist, während die Personendaten der Person "Mustermann" sowohl in der Kunden- als auch in der Angestellten-Tabelle gespeichert werden müssen. Hat man eine Superklasse mit vielen Attributen, sollte besser auf Variante I zurückgegriffen werden, da redundante Speicherung auch zu beträchtlichem Mehraufwand bei Bewahrung der Datenkonsistenz führt.

Kunde

Person				Kontonummer	Kontostand
Personal Nr	Name	Adresse	Alter		

Angestellter

Person				Gehalt	Abteilung
Personal Nr	Name	Adresse	Alter		

Abbildung 3.22: Objekt-relationales Modell - Vererbung aufgelöst nach Variante II

Kunde

Person				Kontonummer	Kontostand
Personal Nr	Name	Adresse	Alter		
25	Mustermann	Hauptstraße 46, 9020 Klagenfurt	34	18880000005	4.500,--
30	Müller	Blumenweg 34, 9500 Villach	25	24540000033	6.300,--

Angestellter

Person				Gehalt	Abteilung
Personal Nr	Name	Adresse	Alter		
25	Mustermann	Hauptstraße 46, 9020 Klagenfurt	34	900,--	EDV
35	Maier	Lannerweg 9, 9201 Krumpendorf	28	1.300,--	Management

Abbildung 3.23: Variante II - Zuordnung der Beispiel-Instanzen

Variante III - Einzelrelation mit zusammengesetztem Datentyp der Super- und Subklassen und Unterscheidungsattribut: Für die Attribute der Superklasse wird ein zusammengesetzter Datentyp definiert. Für jede Subklasse S_i wird ein zusammengesetzter Datentyp für die Attribute von S_i angelegt. Eine Tabelle wird erstellt, deren Attribute auf den zusammengesetzten Datentypen der Superklasse und der Subklassen aufbauen. Zusätzlich wird ein Attribut benötigt, das signalisiert, welcher Subklasse eine Instanz zugeordnet ist. Die resultierende Tabelle hat also i + 2 Attribute, ein Attribut pro Datentyp einer Subklasse, eines für die Superklasse und eines für die Zuordnung zu den Subklassen. Der Aufbau der Tabelle ist in Abbildung 3.24 veranschaulicht.

Person				Kunde		Angestellter		Subklasse_Typ
Personal Nr	Name	Adresse	Alter	Konto-nummer	Konto-stand	Gehalt	Abteilung	

Abbildung 3.24: Objekt-relationales Modell - Vererbung aufgelöst nach Variante III

Variante III kann nur für disjunkte Vererbung eingesetzt werden, da das unterscheidende Attribut - wie bei Variante C der relationalen Abbildung - nur jeweils einen Wert annehmen kann. Hat man eine große Anzahl von Subklassen oder Subklassen mit vielen Attributen, ist das Ergebnis von Variante III eine Tabelle, die viele Null-Values enthält. Möchte man diese vermeiden, muß eine andere Variante gewählt werden. Sowohl totale als auch partielle Vererbung kann mit Variante III abgebildet werden. Wie in Abbildung 3.25 zu sehen ist, fehlt der Datensatz für die Person "Mustermann". Diese Person ist sowohl Kunde als auch Angestellter, was in der Tabelle von Variante III nicht abgebildet werden kann. Dazu kommt noch, daß verhältnismäßig viele Felder der Tabelle leer sind.

Person				Kunde		Angestellter		Subklasse_Typ
Personal Nr	Name	Adresse	Alter	Konto-nummer	Konto-stand	Gehalt	Abteilung	
30	Müller	Blumenweg 34, 9500 Villach	25	24540000033	6.300,--			K
35	Maier	Lannerweg 9, 9201 Krumpendorf	28			1.300,--	Management	A
40	Huber	Weiherstraße 56, 9020 Klagenfurt	37					NULL

Abbildung 3.25: Variante III - Zuordnung der Beispiel-Instanzen

Variante IV - Einzelrelation mit zusammengesetztem Datentyp der Super- und Subklassen mit bool'schen Attributen: Wenn man Variante III auch für überlappende

Vererbung einsetzen möchte, ist eine Änderung notwendig. Um einer Instanz die Zugehörigkeit zu mehreren Subklassen zu erlauben, muß anstelle eines Attributs zur Unterscheidung der Subklasse ein zusammengesetzter Datentyp definiert werden, der ein bool'sches Attribut für jede Subklasse enthält. Die bool'schen Attribute der Subklassen, denen eine Instanz angehört, werden auf den Wert *1* gesetzt, die anderen erhalten den Wert *0* oder bleiben undefiniert. Bei Abfragen wird für jede Instanz geprüft, welchen Wert die bool'schen Attribute des zusammengesetzten Datentypen haben. Die resultierende Tabelle ist in Abbildung 3.26 dargestellt.

Person				Kunde		Angestellter		Subklasse_Typ	
Personal_Nr	Name	Adresse	Alter	Konto-nummer	Konto-stand	Gehalt	Abteilung	Kunde	Angestellter

Abbildung 3.26: Objekt-relationales Modell - Vererbung aufgelöst nach Variante IV

Durch Anwendung von Variante IV kann sowohl partielle und totale als auch disjunkte und überlappende Vererbung aufgelöst werden. Wegen der Tabellenstruktur eignet sich die Variante allerdings am besten für totale, überlappende Vererbung, bei anderen Vererbungsvarianten entstehen viele Felder mit Null-Values. Die Zuordnung der Instanzen ist in Abbildung 3.27 zu sehen. Alle vier Personen können in die Tabelle eingetragen werden. Die bool'schen Attribute des Typen "Subklasse_Typ" zeigen an, zu welchen Subklassen die Instanz gehört.

Person				Kunde		Angestellter		Subklasse_Typ	
Personal_Nr	Name	Adresse	Alter	Konto-nummer	Konto-stand	Gehalt	Abteilung	K	A
25	Mustermann	Hauptstraße 46, 9020 Klagenfurt	34	18880000005	4.500,--	900,--	EDV	1	1
30	Müller	Blumenweg 34, 9500 Villach	25	24540000033	6.300,--			1	0
35	Maier	Lannerweg 9, 9201 Krumpendorf	28			1.300,--	Management	0	1
40	Huber	Weiherstraße 56, 9020 Klagenfurt	37					0	0

Abbildung 3.27: Variante IV - Zuordnung der Beispiel-Instanzen

Variante V - Einzelrelation mit zusammengesetztem Datentyp der Super- und Subklassen mit mehrwertigem Attribut: Variante IV kann durch den Einsatz eines mehrwertigen Attributes variiert werden. Anstelle des zusammengesetzten Datentyps, der ein bool'sches Attribut für jede Subklasse enthält, kann ein mehrwertiges Attribut definiert werden, in dem die Bezeichnungen der Subklassen, denen die Instanz zugehörig ist, eingetragen werden. Bei Abfragen wird geprüft, welche Subklassen im mehrwertigen Attribut vorhanden, beziehungsweise welche nicht vorhanden sind. Die Variante ist in Abbildung 3.28 veranschaulicht.

Person				Kunde		Angestellter		Subklasse_ Menge
Personal Nr	Name	Adresse	Alter	Konto-nummer	Konto-stand	Gehalt	Abteilung	

Abbildung 3.28: Objekt-relationales Modell - Vererbung aufgelöst nach Variante V

Diese Variante kann für partielle, totale, disjunkte und überlappende Vererbung eingesetzt werden, wie in Abbildung 3.29 zu sehen ist. Das Attribut zur Subklassen-Zuteilung kann hier mehrere Werte enthalten. So gehört die Person "Mustermann" den Subklassen Kunde und Angestellter an, das Zuteilungsattribut enthält beide Werte. Wie schon bei Variante IV ist allerdings die Verwendung von totaler, überlappender Vererbung zu empfehlen, um eine große Anzahl von Null-Values zu vermeiden.

Person				Kunde		Angestellter		Subklasse_ Menge
Personal Nr	Name	Adresse	Alter	Konto-nummer	Konto-stand	Gehalt	Abteilung	
25	Mustermann	Hauptstraße 46, 9020 Klagenfurt	34	18880000005	4.500,--	900,--	EDV	K, A
30	Müller	Blumenweg 34, 9500 Villach	25	24540000033	6.300,--			K
35	Maier	Lannerweg 9, 9201 Krumpendorf	28			1.300,--	Management	A
40	Huber	Weiherstraße 56, 9020 Klagenfurt	37					NULL

Abbildung 3.29: Variante V - Zuordnung der Beispiel-Instanzen

Variante VI - Vererbung durch das DBMS: Die Vererbung, die vom DBMS angeboten wird, wird ausgenutzt. Mit Hilfe von komplexen Datentypen und getypten Tabellen wird die Vererbungshierarchie des konzeptuellen Modells in die Datenbank übernommen. Die Vererbungshierarchie wird dabei zunächst anhand zusammengesetzter Datentypen abgebildet. Dabei wird ein zusammengesetzter Datentyp für die Superklasse definiert. Danach wird für jede Subklasse ein zusammengesetzter Datentyp angelegt, der dem Datentyp der Superklasse untergeordnet ist. Vom Datentyp der Superklasse erbt der Datentyp der Subklasse alle Attribute. Ist eine mehrstufige Hierarchie vorhanden, werden Attribute entlang der Hierarchie weitervererbt. Dasselbe gilt für Methoden. Für jeden der zusammengesetzten Datentypen wird eine Tabelle angelegt. Die mit Hilfe von Variante VI aufgebaute Typhierarchie ist in Abbildung 3.30 graphisch veranschaulicht.

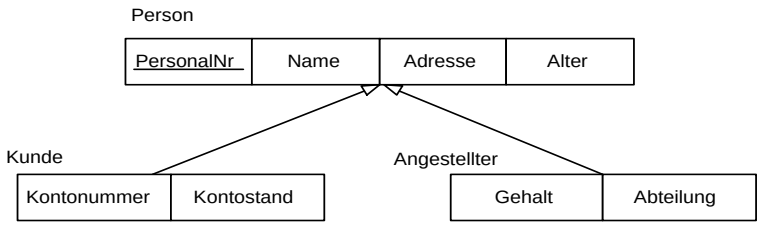


Abbildung 3.30: Vererbung mit Hilfe zusammengesetzter Datentypen des DBMS

Die resultierenden Tabellen entsprechen in etwa den Tabellen aus Abbildung 3.22, nur das noch eine Tabelle für den zusammengesetzten Datentyp der Superklasse hinzukommt. Das Resultat von Variante VI sind also drei Tabellen. Der Unterschied zwischen Variante II und Variante VI liegt im Zeitpunkt, zu dem die Vererbung erfolgt. Bei Variante II werden die Attribute der Superklasse erst bei der Definition der Tabelle durch ein Attribut, das auf dem zusammengesetzten Datentyp der Superklasse aufbaut, übernommen. Die Abbildung der Vererbung erfolgt also manuell durch den Benutzer. Bei Variante VI erfolgt die Vererbung schon bei der Typdefinition und wird vom DBMS übernommen.

Die Syntax zur Erstellung einer Vererbungshierarchie wird in den Abschnitten 3.6.3 und 3.6.5 für zwei ausgewählte DBMS genau definiert.

Wie in der Abbildung zu sehen ist, müssen in den Datentypen keine Fremdschlüssel oder Referenzen abgebildet werden. Das DBMS übernimmt die Abbildung der Vererbungsbeziehung zwischen den Datentypen selbst und kümmert sich um die Vererbung von Attributen und Methoden. Das DBMS unterstützt dabei keine Mehrfachvererbung. Mehrfachvererbung ist ein sehr komplexes Thema und ihre Einbettung in ein DBMS ist alles andere als trivial. Daher hat man sich entschlossen die Auflösung von Mehrfachvererbung dem Datenbank-Designer zu überlassen. Methoden dafür werden in Abschnitt 3.9 vorgestellt. Variante VI unterstützt sowohl partielle und totale als auch disjunkte und überlappende Vererbung. Bei partieller Vererbung hat eine Instanz der Superklasse keine Instanzen in den Tabellen der Subklassen. Bei totaler Vererbung sind nur Einträge in den Tabellen der Subklassen vorhanden. Wenn auch noch überlappende Vererbung eingesetzt wird, müssen die Attribute der Superklasse redundant in allen zugeordneten Subklassen gespeichert werden.

Variante VII - Vererbung durch das DBMS mit nicht-instanzierbarem Root-Datentyp: Die letzte Variante, wie Vererbung in einem objekt-relationalen Datenmodell aufgelöst werden kann, ist die Verwendung eines nicht-instanzierbaren Root-Datentypen. Dabei

wird für die Attribute der Superklasse ein zusammengesetzter Datentyp definiert, der nicht instanzierbar ist. Das heißt, daß für diesen Datentyp keine Tabelle angelegt werden kann. Unter diesem nicht instanzierbaren Datentyp werden zusammengesetzte Datentypen für die Subklassen definiert, die die Attribute des nicht instanzierbaren Datentyps erben. Für jeden zusammengesetzten Datentyp einer Subklasse wird eine Tabelle angelegt. Diese Tabelle enthält aufgrund der Vererbungsbeziehung sowohl die Attribute der Subklasse, als auch die Attribute der Superklasse. Das Resultat entspricht Variante VI, mit dem Unterschied, daß nur totale Vererbung verwendet werden kann, da keine Instanzen angelegt werden können, die keiner Subklasse zugeordnet sind.

Ein Überblick über die gerade vorgestellten Varianten ist in Abbildung 3.31 zu sehen.

Vererbungsart / Variante	total	partiell	disjunkt	überlappend	Anzahl der Tabellen	Besonderheiten
Variante I : Vererbung mit getypten Tabellen und Referenzen	X	X	X	X	Superklasse + Anzahl der Subklassen	Zugriff mittels Punkt-Notation, DBMS kümmert sich um referentielle Integrität
Variante II : Eine Tabelle pro Subklasse, Attribute der Superklasse als zusammengesetzter Datentyp	X		X	X	Anzahl der Subklassen	Redundanz bei überlappender Vererbung
Variante III : Eine Tabelle, die die Attribute der Klassen als zusammengesetzten Datentyp enthält, ein Attribut für Zuordnung zu einer Subklasse	X	X	X		1	Überprüfung des Zuordnungsattribut auf erlaubte Werte
Variante IV: Wie Variante III, Attribut für die Zuordnung als zusammenge-setzter Datentyp, der ein Attribut pro Subklasse hat	X	X	X	X	1	Zuordnungsdatentyp besteht aus bool'schen Attributen
Variante V: Wie Methode III, Attribut für die Zuordnung als mehrwertiges Attribut	X	X	X	X	1	
Variante VI: Vererbung mit des DBMS - zusammengesetzte Datentypen und getypte Tabellen	X	X	X	X	Superklasse + Anzahl der Subklassen	DBMS kümmert sich um die Vererbung
Variante VII: Wie Variante VI, aber Root-Typ ist nicht instantierbar	X		X	X	Anzahl der Subklassen	Für die Superklasse kann keine Tabelle angelegt werden

Abbildung 3.31: Überblick Vererbungsvarianten

Um zu entscheiden, welche der acht Varianten man zur Abbildung auswählt, muß ein Mengen- und Transaktionsprofil erstellt werden. Vor allem das Transaktionsprofil wird zur Auswahl ei-

ner Variante herangezogen. Als erster Schritt muß festgestellt werden, ob es sich um disjunkte oder überlappende, totale oder partielle Vererbung handelt. Dadurch wird die Auswahl bereits eingeschränkt:

- Wenn totale, partielle, disjunkte und überlappende Vererbung eingesetzt wird, bleiben Variante I, IV, V und VI über, zwischen denen gewählt werden kann.
- Wenn partielle Vererbung nicht erlaubt ist, kann zwischen Variante II und VII gewählt werden.
- Wenn überlappende Vererbung nicht erlaubt ist, bleibt Variante III über.

Für die verbleibenden Möglichkeiten werden die Operations-Häufigkeits-Tabelle und die Zugriffshäufigkeitstabelle herangezogen. Dabei werden zunächst aus der Operations-Häufigkeitstabelle die Operationen ausgewählt, die am häufigsten durchgeführt werden und Online-Transaktionen sind. Diese Operationen sind sozusagen die kritischen Operationen, für die die Datenbank optimiert werden sollte. Mit Hilfe der Zugriffshäufigkeitstabelle werden diese Operationen weiter analysiert, um herauszufinden, welche Entitäten und Attribute von der Operation betroffen sind. In Abbildung 3.32 ist die Operationshäufigkeitstabelle für das Vererbungsbeispiel zu sehen.

Name/Kurzbeschreibung	Häufigkeit	Typ (Online/Batch)
O1: Abfragen von Kundendaten	300x/Tag	OL
O2: Abfragen von Kunden- und Angestelltendaten	350x/Tag	OL
O3: Personen, die älter als 30 sind	60x/Tag	OL
O4: Durchschnittsvermögen aller Kunden	20x/Tag	OL
O5: Durchschnittsgehalt aller Angestellten	25x/Tag	OL
O6: Anlegen eines Kunden	30x/Tag	OL
O7: Anlegen eines Angestellten	1x/Tag	OL
O8: Anlegen einer Person, die weder Kunde noch Angestellter ist	10x/Tag	OL
O9: Löschen von Kunden	1x/Woche	B
O10: Löschen von Angestellten	1x/Woche	B
O11: Löschen von Personen	1x/Woche	B

Abbildung 3.32: Operationshäufigkeitstabelle für das Vererbungsbeispiel

Die drei Operationen O9, O10 und O11 werden aufgrund der Tatsache, daß es sich um Batch-Operationen handelt, nicht näher untersucht. Von den verbleibenden Operationen sind O1, O2 und O3 diejenigen, die innerhalb eines Tages am häufigstens ausgeführt werden und die daher einen Großteil der gesamten Transaktionen, die auf dem Datenbankschema ausgeführt werden, ausmachen. Auf diese drei Operationen hin sollte das Datenbankschema optimiert werden, sie werden also näher untersucht. Entscheidend ist hierbei, um welche Kategorie von Operationen es sich handelt. Zunächst ist festzustellen, daß alle drei Operationen nur Abfragen sind, also nur lesend auf die Tabellen zugreifen. Bei O1 handelt es sich weiters um eine Operation, die Daten der Subklasse in Verbindung mit Daten der Superklasse benötigt. O2 benötigt Daten mehrerer Subklassen, kann also offensichtlich nur bei überlappender Vererbung eingesetzt werden. O3 benötigt ausschließlich Daten der Superklasse, unabhängig von den Subklassen. Um zu entscheiden, welche Variante geeignet ist, werden zunächst die logischen Zugriffszahlen für die verschiedenen Varianten berechnet.

Zunächst wird O1 betrachtet, eine Operation, die auf eine Subklasse in Verbindung mit der Superklasse zugreift, unter der Voraussetzung, es handelt sich um disjunkte, überlappende, totale und partielle Vererbung. Die Anzahl der logischen Zugriffe für die verschiedenen möglichen Varianten ist in Abbildung 3.33 zu sehen.

Name/Kurzbeschreibung	Konzept	Konzept-Typ	Read/Write	Anzahl logischer Zugriffe
Variante I:	Person	Entität	R	300
	Kunde	Entität	R	300
				600
Variante IV	Person	Entität	R	300
Variante V	Person	Entität	R	300
Variante VI	Person	Entität	R	300
	Kunde	Entität	R	300
				600

Abbildung 3.33: Anzahl der logischen Zugriffe für Operation 1

O1 wird laut der Operationshäufigkeitstabelle 300 mal am Tag ausgeführt. Wird Variante I eingesetzt, muß also 300 mal ein Datensatz der Superklasse “Person“ und 300 mal ein Datensatz der Subklasse “Kunde“ gelesen werden, um die gewünschten Daten zu erhalten. Das ergibt eine Gesamtzugriffszahl von 600 logischen Zugriffen. Dasselbe gilt für Variante VI. Wird hingegen Variante IV oder Variante V eingesetzt, die beide eine Einzelrelation verwenden, erfolgen jeweils nur 300 logische Zugriffe, da die gewünschten Daten in einem Zugriff aus der Einzelrelation

ausgelesen werden können.

Für O2, eine Operation, die mehrere Subklassen in Verbindung mit der Superklasse benötigt, ist die Anzahl der logischen Zugriffe in Abbildung 3.34 zu sehen.

Name/Kurzbeschreibung	Konzept	Konzept-Typ	Read/Write	Anzahl logischer Zugriffe
Variante I:	Person	Entität	R	350
	Kunde	Entität	R	350
	Angestellter	Entität	R	350
				1050
Variante IV	Person	Entität	R	350
Variante V	Person	Entität	R	350
Variante VI	Person	Entität	R	350
	Kunde	Entität	R	350
	Angestellter	Entität	R	350
				1050

Abbildung 3.34: Anzahl der logischen Zugriffe für Operation 2

Unter Einsatz von Variante I muß jede Subklasse und die Superklasse einmal gelesen werden. Bei 350 Ausführungen am Tag ergibt das eine Gesamtanzahl von 1.050 logischen Zugriffen. Dasselbe gilt auch hier für Variante VI. Wird hingegen Variante IV oder V eingesetzt können alle benötigten Attribute in einem Zugriff aus der Einzelrelation ausgelesen werden, was eine Gesamtanzahl von 350 logischen Zugriffen ergibt.

O3 benötigt nur Daten der Superklasse, was zu den logischen Zugriffszahlen in Abbildung 3.35 führt.

Name/Kurzbeschreibung	Konzept	Konzept-Typ	Read/Write	Anzahl logischer Zugriffe
Variante I:	Person	Entität	R	60 * 20000 = 1.200.000
Variante IV	Person	Entität	R	60 * 20000 = 1.200.000
Variante V	Person	Entität	R	60 * 20000 = 1.200.000
Variante VI	Person	Entität	R	60 * 20000 = 1.200.000

Abbildung 3.35: Anzahl der logischen Zugriffe für Operation 3

O3 stellt eine Abfrage, die alle Personen zurückliefert, die älter als 30 Jahre sind. Unter

der Annahme, daß dies ungefähr die Hälfte der Personen betrifft, müssen jeweils 20.000 Datensätze gelesen werden und das 60 mal am Tag. Unter Einsatz von Variante I und VI muß jeweils nur die Tabelle der Superklasse gelesen werden, bei Variante IV und V die Einzelrelation. Das ergibt für alle Varianten eine Gesamtzugriffszahl von 1.200.000 logischen Zugriffen.

O3 benötigt, wie sich nach dieser Betrachtung herausgestellt hat, die größte Anzahl logischer Zugriffe, obwohl sie nur 60 mal am Tag ausgeführt wird. Allerdings spielt es für O3 keine Rolle, welche der vier möglichen Varianten verwendet wird, die Anzahl logischen Zugriffe ist immer gleich. Daher wird zur Wahl O2 herangezogen, die die nächsthöhere Anzahl logischer Zugriffe aufweist. Für O2 ist entweder Variante IV oder V zu wählen, da diese geringere Zugriffszahlen aufweisen, als Variante I oder VI. Diese Entscheidung begünstigt auch gleichzeitig O1, da hier das gleiche Ergebnis vorliegt. In diesem Fall wäre also in weiterer Folge zu untersuchen, ob Variante IV oder V die bessere Laufzeit aufweist, da die logischen Zugriffszahlen gleich sind, und eine dementsprechende Entscheidung zu treffen.

Wenn partielle Vererbung nicht erlaubt ist, also entweder Variante II oder Variante VII zum Einsatz kommt, unterscheiden sich die Ergebnisse der logischen Zugriffszahlen. Zunächst wird wieder O1 untersucht, der Zugriff auf eine Subklasse in Verbindung mit der Superklasse, was zu den logischen Zugriffszahlen in Abbildung 3.36 führt.

Name/Kurzbeschreibung	Konzept	Konzept-Typ	Read/Write	Anzahl logischer Zugriffe
Variante II	Kunde	Entität	R	300
Variante VII	Kunde	Entität	R	300

Abbildung 3.36: Anzahl der logischen Zugriffe für Operation 1 bei totaler Vererbung

Sowohl bei Variante II als auch bei Variante VII ergibt sich die gleiche Anzahl logischer Zugriffe, da in beiden Fällen nur auf die Subklassentabelle zugegriffen wird, die die benötigten Daten enthält. Die Gesamtanzahl der logischen Zugriffe ist 300.

O2 benötigt mehrere Subklassen und damit auch mehr logische Zugriffe, wie in Abbildung 3.37 zu sehen ist. Auch hier ergibt sich für Variante II und VII die gleiche Anzahl logischer Zugriffe, was nicht verwunderlich ist, da sich die beiden Varianten nur durch die Art der Datentyp-Vererbung unterscheiden. Bei Variante II erfolgt diese manuell durch den Designer, bei Variante VII erfolgt sie durch das DBMS mittels nicht instanzierbarem Root-Datentyp. Die resultierende Tabelle ist aber in beiden Fällen dieselbe.

Name/Kurzbeschreibung	Konzept	Konzept-Typ	Read/Write	Anzahl logischer Zugriffe
Variante II	Kunde	Entität	R	350
	Angestellter	Entität	R	350
				700
Variante VII	Kunde	Entität	R	350
	Angestellter	Entität	R	350
				700

Abbildung 3.37: Anzahl der logischen Zugriffe für Operation 2 bei totaler Vererbung

In Abbildung 3.38 sind die Zugriffszahlen für O3 zu sehen, die nur die Attribute der Superklasse benötigt.

Name/Kurzbeschreibung	Konzept	Konzept-Typ	Read/Write	Anzahl logischer Zugriffe
Variante II	Kunde	Entität	R	60 * 10.000 = 600.000
	Angestellter	Entität	R	60 * 10.000 = 600.000
				1.200.000
Variante VII	Kunde	Entität	R	60 * 10.000 = 60.000
	Angestellter	Entität	R	60 * 10.000 = 60.000
				1.200.000

Abbildung 3.38: Anzahl der logischen Zugriffe für Operation 3 bei totaler Vererbung

Wenn man davon ausgeht, daß die Hälfte der Personen, die über 30 Jahre alt sind, Kunden und die andere Hälfte Angestellte sind, ergibt sich eine Gesamtzugriffszahl von 1.200.000 logischen Zugriffen. Auch hier ist die Anzahl der logischen Zugriffe für beide Varianten gleich.

Die Untersuchung ergibt keine eindeutige Entscheidung zugunsten einer Variante bei keiner der untersuchten Operationen. In diesem Fall muß anhand der Performance-Analyse entschieden werden, ob Variante II oder Variante VII eingesetzt werden soll.

Ist überlappende Vererbung nicht erlaubt, kann nur Variante III eingesetzt werden. Eine weitere Analyse ist nicht notwendig. Wie aber in den Abschnitten 3.6.3 und 3.6.5 zu sehen ist, bieten die Hersteller Oracle und DB2 für Vererbung noch jeweils eine weitere Variante (Variante VIII) an, die überlappende Vererbung unterstützt. Ob dann Variante III oder Variante VIII zu wählen ist, ist den Performance-Analysen des jeweiligen DBMS zu entnehmen.

Die Untersuchungen haben ergeben, daß in den meisten Fällen mehr als eine Variante zur Auswahl überbleibt. In so einem Fall kann die Entscheidung nicht mehr mit Hilfe des Mengen- und Transaktionsprofils getroffen werden, es muß auf das Laufzeitverhalten der Varianten in einem bestimmten DBMS zurückgegriffen werden. Eine Analyse des Laufzeitverhaltens der verschiedenen Varianten für Oracle ist in Abschnitt 3.6.4 zu finden, für DB2 in Abschnitt 3.6.6.

Aus diesem Abschnitt geht klar hervor, daß mit der Verwendung objekt-relationaler Datenmodelle eine große Zahl an Abbildungsmöglichkeiten für Vererbung hinzukommt. In den nächsten beiden Abschnitten werden die in diesem Abschnitt abgehandelten sieben Varianten der Vererbungsauflösung in einem objekt-relationalen Datemmodell in zwei konkret ausgewählten DBMS, nämlich Oracle Version 9.2.0 und IBM DB2 UDB Version 7.2, realisiert. Zusätzlich zur Erstellung der Tabellen werden auch die Zugriffsmöglichkeiten auf die enthaltenen Datensätze gezeigt.

3.6.3 Vererbung in Oracle

Oracle bietet objekt-relationale Erweiterungen seit der Version 8i an. In Oracle können alle sieben Varianten der Vererbungsauflösung verwendet werden, vorausgesetzt, es handelt sich um Oracle 9i. Die Version 8i bietet noch keine vom DBMS realisierte Vererbung an, sondern nur zusammengesetzte Datentypen, Referenzen und mehrwertige Attribute. Für Oracle 8i entfallen demnach die Varianten VI und VII der Vererbungsauflösung. Diese Arbeit greift auf ein Oracle DBMS der Version 9.2.0 zurück, es werden also alle sieben Varianten realisiert.

Variante I

Bei Verwendung von Variante I wird zunächst der Datentyp für die Superklasse definiert:

```
CREATE OR REPLACE TYPE person_typ AS OBJECT (
        personalnr      CHAR(5),
        name            VARCHAR(60),
        adresse         VARCHAR(100),
        person_alter    NUMBER(2)
);
/
```

Für die Subklassen werden zwei weitere Datentypen gebraucht, die ein Attribut besitzen, das auf die Superklasse referenziert:

```
CREATE OR REPLACE TYPE varI_kunde_typ AS OBJECT (
        kontonummer     CHAR(11),
        kontostand      NUMBER(10,2),
        ref_person      REF person_typ
);
/
CREATE OR REPLACE TYPE varI_angestellter_typ AS OBJECT (
        gehalt       NUMBER(7,2),
        abteilung    VARCHAR(50),
        ref_person   REF person_typ
);
/
```

Bei diesen beiden Datentypen referenziert das Attribut *ref_person* auf eine Instanz der Superklasse. Nachdem die Datentypen nun bekannt sind, müssen noch Tabellen definiert werden, die auf diesen Datentypen aufbauen.

```
CREATE TABLE varI_person OF person_typ;
ALTER TABLE varI_person ADD PRIMARY KEY (personalnr);

CREATE TABLE varI_kunde OF varI_kunde_typ (
ref_person REFERENCES varI_person
);

CREATE TABLE varI_angestellter OF varI_angestellter_typ (
ref_person REFERENCES varI_person
);
```

Variante I ist damit in der Datenbank abgebildet. Die dabei verwendeten getypten Tabellen werden in Oracle als “Object Tables“ bezeichnet. Möchte man nun zum Beispiel den Datensatz des Kunden mit der Personalnummer “25“ selektieren, verwendet man dafür folgende Abfrage:

```
SELECT k.kontonummer, k.kontostand, k.ref_person.name
FROM varI_kunde k
WHERE k.ref_person.personalnr = '25';
```

Zu beachten ist bei der Abfrage, daß der Tabellen-Alias k angegeben werden muß. Um die Query zu formulieren wird Punkt-Notation verwendet. Die Join-Bedingung muß nicht angegeben

werden, ebensowenig die Tabelle der Superklasse. Das DBMS erkennt die Referenz und löst sie selbständig auf.

Möchte man alle Attribute der Superklasse selektieren, kann man Punkt-Notation für jedes einzelne Attribut verwenden, oder man kann die Referenz mit Hilfe des Operators *deref* auflösen. Mit *deref* erhält man die referenzierte Instanz der Superklasse als Objekt zurück, ohne daß jedes einzelne Attribut im SELECT-Teil der Abfrage angegeben werden muß. Die Abfrage sieht folgendermaßen aus:

```
SELECT DEREF(k.ref_person)
FROM varI_kunde k
WHERE k.ref_person.personalnr = '25';
```

Die Instanz wird als Objekt zurückgegeben:

```
DEREF(K.REF_PERSON)(PERSONALNR, NAME, ADRESSE, ALTER)
--------------------------------------------------------------------------------
PERSON_TYP('25', 'Max Mustermann', 'Hauptstrasse 48, 9020 Klagenfurt', '34')
```

Viele Anwendungen können allerdings mit einem Ergebnis dieser Art nicht umgehen, sondern erwarten, daß die Attribute in der gewohnten relationalen Form zurückgegeben werden. In diesem Fall muß jedes Attribut einzeln mit Punkt-Notation selektiert werden.

Soll ein neuer Datensatz in die Kunden-Tabelle eingefügt werden, muß in das referenzierende Attribut der OID der zu referenzierenden Instanz eingetragen werden. Diesen OID erhält man mit Hilfe des Operators *REF*:

```
INSERT INTO varI_kunde VALUES (
       SELECT '18880000005', '4500', ref(p)
       FROM varI_person p
       WHERE p.personalnr = '25');
```

Um eine Instanz in die Kunden-Tabelle einzufügen muß also zuerst die Instanz der zugeordneten Person selektiert und in eine Referenz umgewandelt werden, was durch den Teil *ref(p)* in der SELECT-Anweisung geschieht. Wird ein OID verwendet, der vom System generiert wird, sollte ein Insert-Statement dieser Art verwendet werden. Reicht es allerdings aus, einen OID zu benutzen, der nur in der Tabelle, in der er verwendet wird, eindeutig sein muß, kann man eine andere Lösung verwenden. Diese liegt darin, in der Superklasse einen Primärschlüssel zu definieren und diesen als OID zu verwenden. Die Tabelle wird mit folgendem Statement angelegt:

```
CREATE TABLE varI_person OF person_typ (personalnr PRIMARY KEY)
                                        OBJECT IDENTIFIER IS PRIMARY KEY;
```

Wenn nun Daten in die Subklasse eingefügt werden sollen, muß der OID für einen bestimmten Datensatz nicht mehr aus der Superklasse selektiert werden, sondern er kann aus dem Primärschlüsselwert des Datensatzes mit Hilfe der Funktion *make_ref* generiert werden:

```
INSERT INTO varI_kunde
VALUES ('18880000005', '4500', make_ref(varI_Person, '25'));
```

Trägt der Benutzer selbst Sorge dafür, daß der Primärschlüssel der Superklasse systemweit eindeutig ist, kann diese Lösung auch dann verwendet werden, wenn ein systemweit eindeutiger OID benötigt wird. Allerdings ist es für einen Benutzer äußert schwierig diese Bedingung selbst sicherzustellen, man sollte in diesem Fall einen systemgenerierten OID verwenden..

Für die Tabelle mit den Angestellten gelten analog dieselben Statements.

Variante II

Bei Abbildung von Variante II kann auf den Datentyp *person_typ* aus Variante I zurückgegriffen werden. Wir erinnern uns, daß bei Variante II für jede Subklasse eine Tabelle angelegt wird, die auch alle Attribute der Superklasse enthält. Die Attribute der Superklasse sind im zusammengesetzten Datentyp *person_typ* definiert. Um sie in die Tabelle der Subklasse zu übernehmen, wird dort ein Attribut definiert, dessen zugrundeliegender Datentyp *person_typ* ist. In Oracle werden für die Definition der Subklassen-Tabellen folgende Statements verwendet:

```
CREATE TABLE varII_kunde (
        person       person_typ,
        kontonummer  CHAR(11),
        kontostand   NUMBER(10,2)
);
ALTER TABLE varII_kunde ADD PRIMARY KEY (person.personalnr);

CREATE TABLE varII_angestellter (
        person      person_typ,
        gehalt      NUMBER(7,2),
        abteilung   VARCHAR(50)
);
ALTER TABLE varII_angestellter ADD PRIMARY KEY (person.personalnr);
```

Als Primärschlüssel der beiden Tabellen wird das Attribut *personalnr* der Superklasse verwendet. Auf die Attribute der Superklasse kann wieder mit Hilfe der Punkt-Notation zugegriffen werden:

```
SELECT k.person.name
FROM varII_kunde k
WHERE k.personalnr = '25';
```

Auch hier muß - wie immer bei Verwendung der Punkt-Notation - der Tabellen-Alias verwendet werden.

Variante III

Bei Variante III wird eine einzige Tabelle angelegt, die die Attribute der Superklasse, die Attribute aller Subklassen und ein Attribut zur Zuordnung der Instanz zu einer Subklasse enthält. Bei einer relationalen Datenbank geht bei dieser Variante Information über die Struktur der Daten verloren. In einer objekt-relationalen Datenbank wird für zusammengehörige Attribute, das heißt für die Attribute der Superklasse und für die Attribute jeder Subklasse, ein zusammengesetzter Datentyp definiert.

Für die Attribute der Superklasse kann der in Variante I definierte Datentyp *person_typ* verwendet werden, für die Attribute der Subklassen müssen folgende Datentypen angelegt werden:

```
CREATE OR REPLACE TYPE kunde_typ AS OBJECT (
        kontonummer     CHAR(11),
        kontostand      NUMBER(10,2)
);
/
CREATE OR REPLACE TYPE angestellter_typ AS OBJECT  (
        gehalt          NUMBER(7,2),
        abteilung       VARCHAR(50)
);
/
```

Diese Datentypen werden bei der Definition der Tabelle verwendet, um Information über die Struktur der Daten zu erhalten:

```
CREATE TABLE varIII_person (
        person          person_typ,
        kunde           kunde_typ,
        angestellter    angestellter_typ,
```

```
        subklasse_typ   VARCHAR(20) CONSTRAINT check_subklasse
                                CHECK (subklasse_typ IN ('Kunde', 'Angestellter'))
);
ALTER TABLE varIII_person ADD PRIMARY KEY (person.personalnr);
```

Das Attribut *personalnr* des Superklassen-Typs "person" wird der Primärschlüssel der Tabelle. Betrachtet man die Tabelle ist auf den ersten Blick klar, welche Attribute Daten der Superklasse beziehungsweise der Subklassen enthalten. Der Zugriff auf die Daten erfolgt wiederum mit Punkt-Notation.

Variante IV

Wird Vererbung mit Variante IV abgebildet, resultiert daraus im Wesentlichen dieselbe Tabelle wie bei Verwendung von Variante III. Der einzige Unterschied liegt in der Definition des Attributes *subklasse_typ*. Bei Variante IV wird für dieses Attribut ein weiterer zusammengesetzter Datentyp definiert, der für jede Subklasse ein bool'sches Attribut enthält, das signalisiert, ob eine Instanz dieser Subklasse zugeordnet ist. Damit kann im Gegensatz zu Variante III auch überlappende Vererbung abgebildet werden. Da die resultierende Tabelle ansonsten der Tabelle aus Variante III sehr ähnlich ist, wird das Statement zum Anlegen der Tabelle hier nicht erneut wiedergegeben.

Variante V

Bei Verwendung von Variante V wird das Attribut, das die Zugehörigkeit einer Instanz zu einer Subklasse definiert, als mehrwertiges Attribut angelegt. Mehrwertige Attribute können in Oracle entweder als VARRAY oder als Nested Table abgebildet werden. Ein VARRAY ist eine Art Vektor, der eine definierte Anzahl an Elementen, die auf Basisdatentypen aufbauen, enthält. Auf die einzelnen Werte eines VARRAYs kann in SQL nicht zugegriffen werden, das heißt, es kann immer nur der gesamte Inhalt des VARRAYs auf einmal manipuliert werden. Bei Anwendung von Variante V wird immer der gesamte Inhalt benötigt, da alle Subklassen, denen eine Instanz zugeordnet ist, gezeigt werden sollen. Daher wird Vererbung in diesem Fall als VARRAY abgebildet, nicht als Nested Table. Es wäre wenig sinnvoll hier einen Nested Table zu erstellen, der nur ein einziges Attribut enthält. Ist es allerdings aus irgendeinem Grund nötig auf einzelne Werte des mehrwertigen Datentyps zuzugreifen, muß eine Nested Table benutzt werden. Die Verwendung einer Nested Table wird in Abschnitt 3.8 genau erklärt.

Um ein VARRAY zu verwenden, muß zunächst ein benutzerdefinierter Datentyp für das VARRAY angelegt werden:

```
CREATE OR REPLACE TYPE subklasse_typ_varray as VARRAY(2) OF VARCHAR(20);
/
```

Da im Beispiel nur zwei Subklassen vorhanden sind, reicht es, ein VARRAY anzulegen, das maximal zwei Elemente enthalten kann. Natürlich ist es auch möglich, daß dieses VARRAY nur ein oder kein Element enthält.

Als nächstes wird die Tabelle definiert:

```
CREATE TABLE varV_person_varray (
        person              person_typ,
        kunde               kunde_typ,
        angestellter        angestellter_typ
        subklasse_varray    subklasse_typ_varray
);
```

Diese Tabelle hat große Ähnlichkeit mit jener von Variante III, der wesentliche Unterschied liegt hier jedoch in der Definition des Attributes *subklasse_varray*. Um nun herauszufinden, welchen Subklassen eine Instanz dieser Tabelle zugeordnet ist, wird folgende Abfrage verwendet:

```
SELECT V.*
FROM varV_person P, TABLE(P.subklasse_varray) V
WHERE P.person.personalnr = '25';
```

Dieses Statement liefert als Ergebnis alle Elemente des VARRAYs in einer flachen, relationalen Form, die von den meisten Anwendungen als Ergebnis einer Abfrage erwartet wird:

```
COLUMN_VALUE
--------------
Kunde
Angestellter
```

Die Umwandlung der Daten in diese Form wird als "Unnesting" bezeichnet. Alternativ kann man auch folgende Query verwenden:

```
SELECT P.subklasse_varray
FROM varV_person_varray P;
```

Diese Query liefert nicht die einzelnen Werte des VARRAYs in aufbereiteter Form, sondern das VARRAY als Gesamtes:

```
SUBKLASSE_VARRAY
-----------------------------------------
SUBKLASSE_TYP_VARRAY('Kunde', 'Angestellter')
```

Dieses Resultat hat keine ansprechende Form und ist für Anwendungen schwierig zu interpretieren. Allerdings ist die Abfrage etwas performanter, da im Gegensatz zum "Unnesting" kein impliziter Join der Tabelle mit sich selbst erfolgen muß. Hier liegt die Entscheidung beim Benutzer, ob er aus Gründen der Lesbarkeit und Einfachheit der Interpretation die minimale Verzögerung der Antwort in Kauf nimmt.

Will man die Werte des VARRAYs mit einem bestimmten Wert vergleichen, können zwei Fälle auftreten:

- Man möchte prüfen, ob ein einzelner Wert im VARRAY vorhanden ist. Ein Beispiel wäre die Selektion aller Kunden. Ob der Kunde dann auch Angestellter ist oder nicht, ist nicht relevant. In diesem Fall verwendet man folgendes Statement:

  ```
  SELECT P.person.name, P.person.person_alter, P.kunde.kontonummer,
         P.kunde.kontostand
  FROM varV_person_varray P, TABLE(P.subklasse_varray) V
  WHERE P.person.personalnr = '25' AND
        V.COLUMN_VALUE = 'Kunde';
  ```

  Dieses Statement überprüft, ob die Person mit der Personalnummer "25" ein Kunde ist und liefert Daten über den Kunde zurück, unabhängig davon, ob der Kunde auch Angestellter ist oder nicht.

- Im zweiten Fall möchte man Daten über eine Person abfragen, die sowohl Kunde als auch Angestellter ist. In diesem Fall ist die Prüfung des VARRAYs etwas komplizierter:

  ```
  SELECT P.person.name, P.person.person_alter, P.kunde.kontonummer,
         P.kunde.kontostand
  FROM varV_person_varray P, TABLE(P.subklasse_varray) V,
       TABLE(P.subklasse_varray) V2
  WHERE P.person.personalnr = '12' AND
        V.COLUMN_VALUE = 'Kunde' AND
        V2.COLUMN_VALUE = 'Angestellter;
  ```

Die Anzahl der VARRAY-Elemente, die überprüft werden sollen, ist hier ausschlaggebend dafür, wie oft das VARRAY als Tabelle selektiert werden muß. Im Beispiel möchte man prüfen, ob die Werte "Kunde" und "Angestellter" im VARRAY enthalten sind, das VARRAY muß also zunächst zweimal als Tabelle mit einem Alias selektiert werden. Jeder Alias wird zur Prüfung eines Wertes herangezogen, die Bedingungen werden mit *AND* verknüpft. Nur so kann das gesamte Array auf bestimmte Werte überprüft werden. Folgendes Statement, das zunächst naheliegend erscheinen würde, funktioniert nicht:

```
SELECT P.person.name, P.person.person_alter, P.kunde.kontonummer,
       P.kunde.kontostand
FROM varV_person_varray P, TABLE(P.subklasse_varray) V
WHERE P.person.personalnr = '12' AND
      V.COLUMN_VALUE = 'Kunde' AND
      V.COLUMN_VALUE = 'Angestellter';
```

Dieses Statement liefert nie einen Datensatz zurück, auch wenn beide Werte im VARRAY vorhanden sind. Es muß also wenn auf mehrere Werte des VARRAYs geprüft werden soll, in jedem Fall das erste Statement verwendet werden.

Variante VI

Bei Variante VI wird die Vererbung eingesetzt, die das DBMS selbst anbietet. In Oracle kann Vererbung nur auf Datentypen angewandt werden, vererbt werden Attribute und Methoden. Vererbung in Tabellen, bei der auch die Daten vererbt werden, ist nicht möglich.

Um eine Vererbungshierarchie gemäß Variante VI abzubilden, werden zunächst die passenden Datentypen benötigt:

```
CREATE OR REPLACE TYPE inherit_person_typ AS OBJECT (
        personalnr      CHAR(5),
        name            VARCHAR(60),
        adresse         VARCHAR(100),
        person_alter    NUMBER(2)
        ) NOT FINAL;
/
```

```
CREATE OR REPLACE TYPE inherit_kunde_typ under inherit_person_typ (
        kontonummer     CHAR(11),
        kontostand      NUMBER(10,2)
        );
/
CREATE OR REPLACE TYPE inherit_angestellter_typ under inherit_person_typ (
        gehalt          NUMBER(7,2),
        abteilung       VARCHAR(50)
        );
/
```

Beim Datentyp der Superklasse - *inherit_person_typ* - muß das Schlüsselwort "NOT FINAL" angegeben werden, um dem DBMS mitzuteilen, daß dieser Datentyp als Supertyp für andere Datentypen verwendet wird. Wird nichts angegeben, wird automatisch "FINAL" verwendet. *inherit_kunde_typ* und *inherit_angestellter_typ* werden als Subtypen von *inherit_person_typ* angelegt und erben damit die Attribute einer Person.

Für jeden der Datentypen wird nun eine Tabelle definiert, die auf dem Datentypen aufbaut. Da ein Subtyp die Attribute des Supertypen erbt, sind diese auch in der Tabelle enthalten, die diesen Datentypen verwendet. Die Tabellen werden mit folgenden Statements angelegt:

```
CREATE TABLE varVI_person OF inherit_person_typ;
ALTER TABLE varVI_person ADD PRIMARY KEY (personalnr);

CREATE TABLE varVI_kunde OF inherit_kunde_typ;
ALTER TABLE varVI_kunde ADD PRIMARY KEY (personalnr);

CREATE TABLE varVI_angestellter OF inherit_angestellter_typ;
ALTER TABLE varVI_angestellter ADD PRIMARY KEY (personalnr);
```

Bei den Tabellen handelt es sich wieder um Object Tables. Da Variante VI für partielle Vererbung eingesetzt werden kann, muß auch für die Superklasse eine Tabelle angelegt werden, die diejenigen Instanzen enthält, die keiner Subklasse zugeordnet werden.

Variante VII

Variante VII entspricht Variante VI, mit dem einzigen Unterschied, daß der zusammengesetzte Datentyp der Superklasse nicht instanzierbar ist, wie im folgenden Statement zu erkennen ist:

```
CREATE OR REPLACE TYPE varVII_person_typ AS OBJECT (
        personalnr      CHAR(5),
        name            VARCHAR(60),
        adresse         VARCHAR(100),
        person_alter    NUMBER(2)
        ) NOT INSTANTIABLE NOT FINAL;
/
```

Für diesen Datentyp kann zwar eine Tabelle angelegt werden, in diese Tabelle können aber keine Datensätze eingefügt werden. Der Datentyp wird in erster Linie zur Vererbung von Attributen und Methoden an seine Subtypen benutzt, er ist nicht dafür gedacht in Tabellen verwendet zu werden. Variante VII kann damit nur für totale Vererbung eingesetzt werden, hier jedoch für disjunkte und überlappende.

Variante VIII

Oracle 9i bietet noch eine weitere Variante der Vererbung an. Wenn eine Vererbungshierarchie mit zusammengesetzten Datentypen abgebildet wird, wie zum Beispiel in Variante VI oder Variante VII, reicht es, eine Tabelle anzulegen, die auf dem Supertyp aufbaut. In diese Tabelle können dann auch Datensätze eingefügt werden, die der Struktur eines Subtypen entsprechen. Wenn also gemäß dem Beispiel eine Tabelle angelegt wird, die auf dem Datentyp *inherit_person_typ* basiert, dann können in diese Tabelle auch Instanzen der Datentypen *inherit_kunde_typ* und *inherit_angestellter_typ* eingefügt werden.

Das DBMS geht grundsätzlich davon aus, daß sämtliche Objekte, die in dieser Tabelle gespeichert werden, Instanzen des Supertypen sind. Eine Abfrage auf Instanzen eines Subtypen muß mit Hilfe der Funktionen *VALUE* und *TREAT*, die vom DBMS angeboten werden, erfolgen. *VALUE* nimmt ein Table-Alias einer getypten Tabelle als Parameter und gibt Instanzen, die in dieser Tabelle gespeichert sind, als Objekt zurück. Die Struktur eines solchen Ergebnisses wurde in Variante I bereits gezeigt:

```
SELECT VALUE(p) FROM varVIII_person p WHERE VALUE(p) IS OF (inherit_kunde_typ);
```

Diese Abfrage liefert alle Instanzen zurück, die vom Typ *inherit_kunde_typ* sind. Will man auf einzelne Attribute eines Subtypen zugreifen, muß *VALUE* in Verbindung mit der Funktion *TREAT* verwendet werden. Mit Hilfe dieser Funktion kann man dem DBMS mitteilen, welchen Datentyp es einer Instanz zuweisen soll.

```
SELECT name, TREAT(VALUE(p) AS inherit_kunde_typ).kontonummer
FROM varVIII_person p
WHERE personalnr = '25';
```

In dieser Abfrage wird dem DBMS mitgeteilt, daß es der zurückgelieferten Instanz den Datentyp *inherit_kunde_typ* zuweisen soll. Damit kann in weiterer Folge unter Verwendung der Punkt-Notation auf die speziellen Attribute der Subklasse zugegriffen werden.

Diese Notation funktioniert nicht, wenn ein bestehender Datensatz geändert werden soll. Ein Update-Statement muß folgendermaßen aussehen:

```
UPDATE varVIII_person p
SET p =
 inherit_kunde_typ(p.personalnr, p.name, p.person_alter, p.adresse, '10819555300',
                   '4800')
WHERE p.personalnr = '25';
```

Wie aus diesem Statement ersichtlich ist, muß immer die ganze Instanz neu geschrieben werden, auch wenn sich nur ein Wert ändert. Unter dem Tabellen-Alias ansprechen und übernehmen kann man dabei nur Werte für die Attribute des Supertypen, Werte für die Attribute des Subtypen müssen immer neu angegeben werden, auch wenn sie sich nicht geändert haben. Ein Update auf eine Tabelle, die mit Variante VIII angelegt wurde, ist also sehr umständlich, vor allem, wenn eine Instanz eines Subtypen geändert werden soll, der viele Attribute hat.

Bei Anwendung dieser Variante bleiben sozusagen die Attribute der Subklassen vor dem Benutzer verborgen, es sei denn, er möchte sie explizit auswählen. Geeignet ist diese Variante also vor allem dann, wenn hauptsächlich partielle Vererbung verwendet wird, bei der die speziellen Attribute der Subklassen selten benötigt werden. In jedem anderen Fall empfehle ich die Verwendung von Variante VI oder VII, da der Zugriff und vor allem die Modifikation der Daten bei Variante VIII sehr kompliziert ist.

Damit sind alle Möglichkeiten Vererbung in Oracle aufzulösen, besprochen. Ein Überblick über die einzelnen Varianten, Vererbungsarten, die sie unterstützen und Anzahl der entstehenden Tabellen ist in Abbildung 3.39 zu sehen. Anhand einer Performance-Analyse soll untersucht werden, wie sich die einzelnen Varianten zur Laufzeit verhalten und welche Variante abhängig vom Mengen- und Transaktionsprofil verwendet werden soll.

Vererbungsart / Variante	total	partiell	disjunkt	überlappend	Anzahl der Tabellen	Besonderheiten
Variante I : Vererbung mit Object Tables und Referenzen	X	X	X	X	Superklasse + Anzahl der Subklassen	Zugriff mittels Punkt-Notation, Operatoren DEREF und REF
Variante II : Eine Tabelle pro Subklasse, Attribute der Superklasse als Object Type	X		X	X	Anzahl der Subklassen	Redundanz bei überlappender Vererbung
Variante III : Eine einzige Tabelle, Attribute als Object Types, ein Attribut für Zuordnung zu Subklasse	X	X	X		1	CHECK-Constraint für das Attribut der Zuordnung
Variante IV: Wie Variante III, Attribut für die Zuordnung als Object Type	X	X	X	X	1	Object Type für die Zuordnung hat ein bool'sches Attribut für jede Subklasse
Variante V: Wie Methode III, Attribut für die Zuordnung als mehrwertiges Attribut	X	X	X	X	1	Mehrwertiges Attribut als VARRAY
Variante VI: Vererbung mit Object Type Hierachie	X	X	X	X	Superklasse + Anzahl der Subklassen	Redundanz bei überlappender Vererbung
Variante VII: Wie Variante VI, aber Root-Typ ist nicht instantierbar	X		X	X	Anzahl der Subklassen	Redundanz bei überlappender Vererbung
Variante VIII: Wie Variante VI, Tabelle wird nur für den Root-Typ angelegt	X	X	X		1	Totale Vererbung prinzipiell möglich, aber nicht zu empfehlen; aufwendige Update-Statements; Redundanz bei überlappender Vererbung

Abbildung 3.39: Überblick - Vererbungsvarianten in Oracle

3.6.4 Performance-Studie in Oracle

Nach der Anwendung der allgemeinen Abbildungsvorschriften aus Abschnitt 3.6.2 bleiben eine oder mehrere Varianten zur Abbildung von Vererbung über. Hier muß anhand des Laufzeitverhaltens und der Kostenanalyse eine Wahl getroffen werden. Grundsätzlich sollte die Variante, die die schnellsten Antwortzeiten und die geringsten Kosten aufweist, gewählt werden. Meistens ist die Entscheidung aber komplexer, da Kosten und Laufzeit sehr oft nicht denselben Verlauf haben. Worauf bei der Entscheidung für eine bestimmte Variante zu achten ist, wird in diesem Abschnitt genau besprochen.

Um die Laufzeitanalyse durchzuführen und die verbleibenden Varianten für das Vererbungsschema zu identifizieren, werden zunächst ein Mengen- und ein Transaktionsprofil benötigt. In Abbildung 3.40 ist ein fiktives Mengenprofil für das Vererbungsschema zu sehen.

Konzept	Typ	Anzahl
Person	E	40.000
Kunde	E	20.000
Angestellter	E	20.000
PersonalNr	A	40.000
name	A	40.000
adresse	A	30.000
person_alter	A	40.000
kontostand	A	20.000
kontonummer	A	20.000
gehalt	A	20.000
abteilung	A	20.000

Abbildung 3.40: Mengenprofil für das Vererbungsschema

Es gibt 40.000 Personen, davon sind 10.000 weder Kunde noch Angestellter. 10.000 Personen sind nur Kunden, 10.000 Personen sind nur Angestellte. Weitere 10.000 sind sowohl Kunden als auch Angestellte. Von allen Personen ist die Adresse bekannt. In Abbildung 3.41 ist die Operations-Häufigkeits-Tabelle für das Beispiel zu sehen.

Elf Operationen sind definiert, drei davon werden als Batch durchgeführt und sind damit nicht relevant für weitere Analysen. Die restlichen Operationen sind ganz unterschiedlicher Art: Es gibt welche, die nur auf Subklassen zugreifen, andere greifen nur auf die Superklasse zu, weiter benötigen sowohl die Super- als auch die Subklasse. Anhand der Operations-Häufigkeits-Tabelle wird die Kategorie von Operationen bestimmt, die am häufigsten durchgeführt wird. Die Wahl einer Abbildungsvorschrift wird dann so getroffen, daß die dominierende Kategorie von Abfragen möglichst optimal unterstützt wird, wie in Abschnitt 3.6.2 gezeigt wurde. In dieser Arbeit erfolgt der Vergleich jedoch für alle identifizierten Operationskategorien, nicht nur für die dominierende.

Zuerst muß die Einteilung in disjunkte, überlappende, totale und partielle Vererbung berücksichtigt werden, da nicht jede Variante alle Arten von Vererbung realisieren kann. Einige Operationen können damit mit einigen Varianten nicht durchgeführt werden. Durch die Berücksichtigung von totaler, partieller, disjunkter und überlappender Vererbung können folgende Varianten miteinander verglichen werden:

Name/Kurzbeschreibung	Häufigkeit	Typ (Online/Batch)
O1: Abfragen von Kundendaten	300x/Tag	OL
O2: Abfragen von Kunden- und Angestelltendaten	350x/Tag	OL
O3: Personen, die älter als 30 sind	60x/Tag	OL
O4: Durchschnittsvermögen aller Kunden	20x/Tag	OL
O5: Durchschnittsgehalt aller Angestellten	25x/Tag	OL
O6: Anlegen eines Kunden	30x/Tag	OL
O7: Anlegen eines Angestellten	1x/Tag	OL
O8: Anlegen einer Person, die weder Kunde noch Angestellter ist	10x/Tag	OL
O9: Löschen von Kunden	1x/Woche	B
O10: Löschen von Angestellten	1x/Woche	B
O11: Löschen von Personen	1x/Woche	B

Abbildung 3.41: Operations-Häufigkeits-Tabelle für das Vererbungsschema

- Variante I, Variante IV, Variante V und Variante VI - Diese vier erlauben totale, partielle, disjunkte und überlappende Vererbung.
- Variante II und Variante VII - Diese beiden erlauben totale, disjunkte und überlappende Vererbung, aber keine partielle Vererbung.
- Variante III und Variante VIII - Diese beiden erlauben totale, partielle und disjunkte, aber keine überlappende Vererbung.

Das Laufzeitverhalten wurde durch Abfragen und Modifikationen analysiert, die die definierten Operationen realisieren. Die Abfragen wurden auf den Tabellen jeder Variante ausgeführt und zwar auf unterschiedlichen Größenordnungen des identifizierten Mengenprofils, wobei die vierfache Datenmenge die Obergrenze bildete. Aufgezeichnet wurden für jede Variante die Kosten und die Antwortzeit für die gestellte Abfrage, wobei sowohl für die Kosten als auch für die Laufzeit festgehalten werden muß, daß es sich um die Standard-Installation der Oracle-Instanz handelt, an der keine Tuning-Maßnahmen vorgenommen wurden. Indizes wurden nur vom DBMS selbst für das Primärschlüssel-Attribut “Personalnr“ angelegt. Die Testdaten der Instanzen wurden automatisch durch Skripts generiert. Die Analyse selbst erfolgte auf einem Athlon XP 1800+ mit 256 MB RAM.

Getestet wurden die Abfragen aus Abbildung 3.41, also O1, O2, O3, O4 und O5. O1 greift nur auf eine einzige Subklasse zu, O2 greift auf mehrere Subklassen zu. O3 verwendet nur Daten der Superklasse, O4 und O5 beinhalten eine Aggregationsfunktion. Die Abfragen auf Kunden-

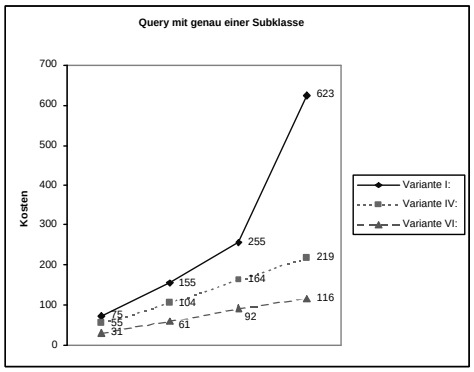
und Angestelltendaten wurden so gestellt, daß einmal viele und einmal wenig Datensätze als Ergebnis zurückgeliefert wurden. Weiters wurde die Anlage einer Person, die nur Kunde ist, sowie die Modifikation einer Person, die nur Kunde ist, analysiert. Ausgewertet und graphisch dargestellt werden die Ergebnisse für die Abfragen O1, O2, O3 und O4, die viele Datensätze als Ergebnis liefern und zwar aus folgendem Grund: Einschränkungen auf eine Tabelle wurden immer über die Personalnummer definiert, die in jeder Tabelle der Primärschlüssel ist. Dadurch wurde bei der Abfrage auf wenig Datensätze immer auf den Index zugegriffen, was zur Folge hat, daß das Kostenverhalten der Abfrage unabhängig von der Anzahl der zu durchsuchenden Datensätze ist. Deshalb werden in weiterer Folge hier nur die Abfragen betrachtet, bei denen sich das Kostenverhalten durch die Menge der Datensätze ändert.

Variante I, IV, V und VI

Wenn sowohl totale und partielle als auch disjunkte und überlappende Vererbung eingesetzt wird, kommt eine von vier Varianten zum Einsatz:

- Variante I, die Vererbung mit Object Tables und Referenzen unterstützt,
- Variante IV, bei der eine Einzelrelation gebildet und die Zuordnung zu einer Subklasse anhand eines zusammengesetzten Datentypen unterschieden wird,
- Variante V, bei der ebenfalls eine Einzelrelation gebildet wird, die Zuordnung zu einer Subklasse aber anhand eines VARRAYs erfolgt oder
- Variante VI, bei der eine Vererbungshierarchie von zusammengesetzten Datentypen, die das DBMS anbietet, eingesetzt wird.

Der Kostenverlauf und das Laufzeitverhalten für eine Abfrage, die auf genau eine Subklasse in Verbindung mit der Superklasse zugreift, ist in Abbildung 3.42 zu sehen, wobei hier nur die Varianten I, IV und VI in die Grafik aufgenommen wurden. Variante V benötigt derart hohe Kosten, daß die Grafik nicht mehr aussagekräftig ist, wenn Variante V darin enthalten ist. Es handelt sich dabei um eine Größenordnung, die ungefähr das dreihundertfache der Kosten der anderen Varianten beträgt. Die hohen Kosten von Variante V ergeben sich durch den Zugriff auf das VARRAY der Tabelle. Auch die Laufzeiten sind, wie in der Abbildung zu erkennen ist, schlechter als die der anderen drei Varianten. Daher ist für Abfragen, die auf genau eine Subklasse zugreifen, Variante V generell nicht zu empfehlen.

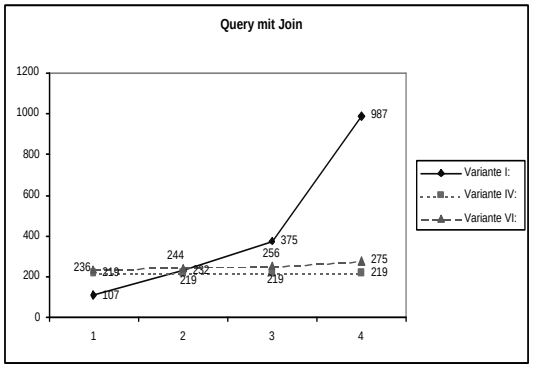


Laufzeit

Anzahl Datensätze	Variante I	Variante IV	Variante V	Variante VI
40.000	1,06 sec	3,01 sec	3,09 sec	2,09 sec
80.000	2,08 sec	3,05 sec	9,05 sec	3,04 sec
120.000	5,06 sec	7,0 sec	16,03 sec	5,05 sec
160.000	14,03 sec	8,08 sec	19,06 sec	8,07 sec

Abbildung 3.42: Kostenverlauf und Laufzeit der Varianten I, IV, V und VI - Zugriff auf genau eine Subklasse

Wie in der Grafik zu sehen ist, hat Variante I die höchsten Kosten im Vergleich mit Variante IV und VI, während die Laufzeiten vor allem bei geringeren Anzahlen von Datensätzen signifikant besser sind. Variante VI hat bei einer geringeren Anzahl von Datensätzen schlechtere Laufzeiten, benötigt aber die niedrigsten Kosten. Variante IV liegt sowohl bei den Kosten als auch bei der Laufzeit im Mittelfeld. Für Abfragen, die auf genau eine Subklasse zugreifen, ist also Variante VI, die Vererbung des DBMS zu empfehlen, da hier niedrige Kosten mit geringen Laufzeiten verbunden werden können, wenn die Relationen viele Datensätze beinhalten. Bewegt sich die Anzahl der Datensätze in einem kleineren Bereich (weniger als hundertausend) und ist eine schnelle Antwortzeit von Bedeutung, sollte Variante I - der Einsatz von Referenzen - bevorzugt werden. Bei Kostenminimierung ist auf jeden Fall Variante VI einzusetzen.



Laufzeit

Anzahl Datensätze	Variante I	Variante IV	Variante V	Variante VI
40.000	3,06 sec	9,03 sec	6,01 sec	10,0 sec
80.000	7,04 sec	13,05 sec	9,09 sec	13,0 sec
120.000	11,09 sec	20,06 sec	18,08 sec	17,02 sec
160.000	19,08 sec	23,02 sec	23,08 sec	22,09 sec

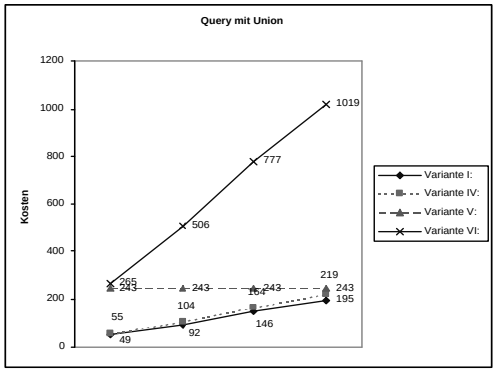
Abbildung 3.43: Kostenverlauf und Laufzeit der Varianten I, IV, V und VI - Zugriff auf mehrere Subklassen

Auch für Abfragen, die auf mehr als eine Subklasse zugreifen, sind die Kosten für Variante V durch den Zugriff auf das VARRAY dermaßen hoch (dreihundertfache Größenordnung), daß

Variante V nicht in die Grafik aufgenommen wurde. Die Laufzeit von Variante V liegt im gleichen Bereich mit den Varianten IV und VI, wie in Abbildung 3.43 zu erkennen ist, trotzdem sollte Variante V aufgrund der extrem hohen Kosten nicht eingesetzt werden.

Während die Kosten für Variante IV und VI nahezu identisch sind, beginnt Variante I zunächst mit sehr niedrigen Kosten, die aber mit der Anzahl der Datensätze sprunghaft ansteigen. Allerdings weist Variante I die besten Laufzeiten auf, weshalb sie auch zu empfehlen ist, wenn kurze Antwortzeiten das ausschlaggebende Kriterium sind. Sollen die Kosten möglichst gering gehalten werden, ist für Abfragen, die auf mehrere Subklassen zugreifen, Variante IV oder VI zu empfehlen, wobei durch den sehr geringen Unterschied, sowohl bei den Kosten als auch bei der Laufzeit, dem Designer die freie Wahl bleibt.

Das Kostenverhalten und die Laufzeiten für Abfragen, die nur Daten der Superklasse benötigen, ist in Abbildung 3.44 zu sehen.



Laufzeit

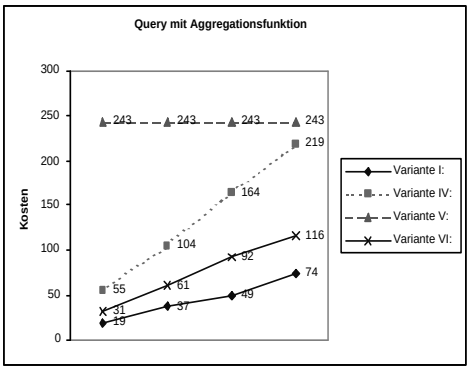
Anzahl Datensätze	Variante I	Variante IV	Variante V	Variante VI
40.000	21,0 sec	43,09 sec	44,0 sec	22,05 sec
80.000	56,06 sec	1:53 min	1:27 min	58,02 sec
120.000	1:05 min	2:09 min	2:10 min	1:13 min
160.000	1:28 min	3:31 min	2:54 min	3:43 min

Abbildung 3.44: Kostenverlauf und Laufzeit der Varianten I, IV, V und VI - Zugriff nur auf die Superklasse

Hier ist auch Variante V in der Grafik enthalten, da bei dieser Abfrage nicht auf das VARRAY zugegriffen werden muß und die Kosten somit in einem annehmbaren Bereich bleiben und im Vergleich mit den anderen Varianten sogar im Mittelfeld liegen. Interessant ist die Beobachtung, daß die benötigten Kosten von Variante V unabhängig von der Anzahl der Datensätze nahezu konstant verlaufen, während die Kosten der anderen drei Varianten mit der Anzahl der Datensätze ansteigen. Nicht zu empfehlen ist hier Variante VI, die sehr hohe Kosten aufweist. Diese erklären sich dadurch, daß bei Einsatz von Variante VI eine Abfrage, die nur auf die Superklasse zugreift, durch die redundante Speicherung der Superklassen-Attribute die Tabellen aller Subklassen und die Tabelle der Superklasse durch ein UNION miteinander verbinden muß. Ein UNION ist eine äußerst kostenaufwendige Operation, wie in der Grafik zu erkennen ist.

Die Laufzeiten der Varianten I und VI sind bei einer geringen Anzahl von Datensätzen nahezu identisch und deutlich kürzer als die Laufzeiten von Variante IV und V. Erst bei größeren Datenmengen hebt sich Variante I mit einer signifikant kürzeren Laufzeit von den anderen drei Varianten ab. Da Variante VI aber wie bereits erwähnt sehr hohe Kosten benötigt, ist in jedem Fall unabhängig von der Anzahl der Datensätze Variante I zu empfehlen, wenn Abfragen dominieren, die nur auf Daten der Superklasse zugreifen.

Abfragen, die eine Aggregationsfunktion beinhalten, favorisieren eindeutig Variante I, wie in Abbildung 3.45 zu erkennen ist. Kosten und Laufzeit sind für Variante I minimal, vor allem in Vergleich zu Variante V. Variante V weist zwar konstante Kosten, unabhängig von der Datenmenge auf, diese sind allerdings sehr viel höher, als bei den anderen Varianten, auch wenn die Tabellen nur wenig Datensätze enthalten. Die Laufzeiten werden bei Variante IV und VI sehr stark von der Anzahl der Datensätze beeinflußt, während sie bei Variante I und V immer im Bereich von null Sekunden liegen. Dominieren also Abfragen mit Aggregationsfunktionen in der Operationshäufigkeitstabelle ist Variante I sehr zu empfehlen, da sie Kosten und Laufzeit minimiert.



Laufzeit

Anzahl Datensätze	Variante I	Variante IV	Variante V	Variante VI
40.000	0 sec	0,02 sec	0,08 sec	0,02 sec
80.000	0 sec	0,04 sec	0,05 sec	0,04 sec
120.000	0,03 sec	6,03 sec	0,04 sec	3,01 sec
160.000	0,05 sec	5,05 sec	0,05 sec	3,08 sec

Abbildung 3.45: Kostenverlauf und Laufzeit der Varianten I, IV, V und VI - Abfrage mit Aggregationsfunktion

Wie aus der Analyse ersichtlich ist, sollte im Fall der Laufzeitminimierung fast immer Variante I, also die Vererbungsabbildung mit Referenzen und Object Tables, eingesetzt werden, außer im Falle einer Abfrage, die auf genau eine Subklasse in Verbindung mit der Superklasse zugreift, wenn die Tabellen viele Datensätze enthalten. In diesem Fall ist Variante VI zu empfehlen. Spielt die Kostenminimierung eine große Rolle, ist grundsätzlich Variante IV, die Einzelrelation mit Zuordnung zu Subklassen mit Hilfe eines zusammengesetzten Datentypen, zu favorisieren. Es treten aber auch Fälle auf, in denen Variante I oder Variante VI die niedrigeren oder zumindest gleich hohe Kosten bieten. Ein Überblick darüber, welche Variante wann gewählt werden soll,

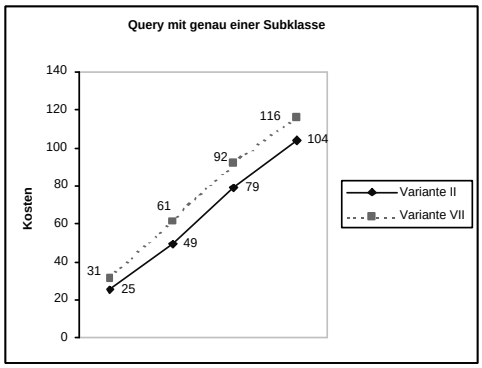
ist in Abbildung 3.46 zu sehen.

Variante I, IV, V und VI	Genau eine Subklasse in Verbindung mit der Superklasse		Mehrere Subklassen in Verbindung mit der Superklasse		Nur Superklasse		Abfrage mit Aggregationsfunktion	
	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze
Minimale Kosten	VI	VI	IV, VI	IV, VI	I, IV	I, IV	I	I
Minimale Laufzeit	I	VI	I	I	I	I	I	I

Abbildung 3.46: Entscheidungshilfe zur Auswahl geeigneter Varianten im Hinblick auf Kosten- und Laufzeitminimierung

Variante II und Variante VII

In diesem Abschnitt wird das Kosten- und Laufzeitverhalten von Variante II (eine Tabelle pro Subklasse, Attribute der Superklasse als Object Type) und Variante VII (Vererbung mit Object Type Hierarchie, Root-Typ ist nicht instanzierbar) verglichen. Die Gemeinsamkeit zwischen den beiden Varianten liegt darin, daß sie keine partielle Vererbung unterstützen. In Abbildung 3.47 ist das Kostenverhalten und die Laufzeiten einer Query zu sehen, die auf genau eine Subklasse zugreift.



Laufzeit

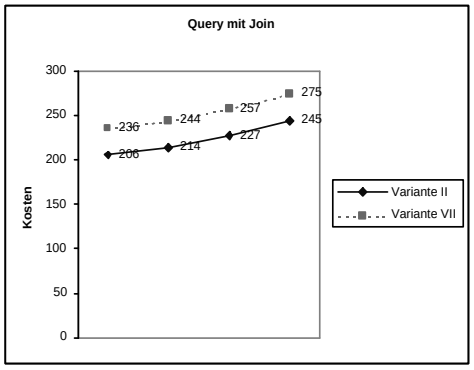
Anzahl Datensätze	Variante II	Variante VII
40.000	3,05 sec	2,08 sec
80.000	3,04 sec	3,04 sec
120.000	5,05 sec	5,05 sec
160.000	8,06 sec	8,07 sec

Abbildung 3.47: Kostenverlauf und Laufzeit von Variante II und Variante VII - Zugriff auf genau eine Subklasse

In der Abbildung ist zu erkennen, daß beide Varianten einen linearen Kostenverlauf im Hinblick auf die Größenordnung der Daten haben, wobei Variante VII etwas höhere Kosten aufweist. Die Laufzeiten unterscheiden sich nur minimal. Wenn nur Queries eingesetzt werden, die auf genau eine Subklasse zugreifen, liegt die Entscheidung welche Variante verwendet werden

soll, wenn die Laufzeit minimiert werden soll, also primär im Ermessen des Designers. Bei Kostenminimierung ist Variante II zu bevorzugen.

In Abbildung 3.48 ist der Kostenverlauf und die Laufzeit beider Varianten für eine Query zu sehen, die auf mehr als eine Subklasse zugreift. Hier wird ein Join über mehrere Tabellen benötigt, was sich natürlich auf den Kostenverlauf auswirkt. Durch den Join steigen die Kosten für die Abfrage, der Unterschied zwischen den beiden Varianten ist aber auch hier relativ gering. Variante VII benötigt etwas mehr Kosten als Variante II. Die Laufzeiten sind anfangs noch gleich, bei einer größeren Anzahl von Datensätzen, weist Variante VII kürzere Laufzeiten auf. Je nachdem ob Kosten oder Laufzeit das Entscheidungskriterium sind, wird die Entscheidung zugunsten von Variante VII (weniger Kosten) oder zugunsten von Variante VII (bessere Laufzeit) fallen, wenn die Relationen eine große Anzahl von Datensätzen beinhalten. Bei Datensatzmengen unter hunderttausend ist das Laufzeitverhalten nahezu identisch, hier sollte aufgrund der geringeren Kosten Variante II gewählt werden.

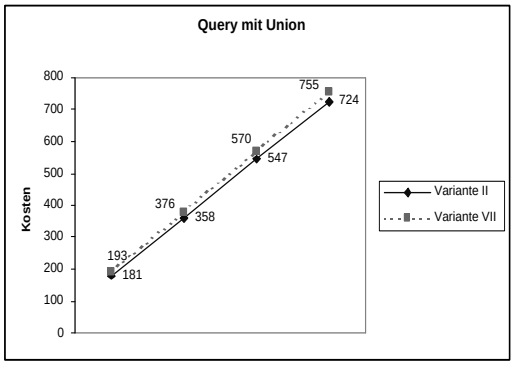


Laufzeit

Anzahl Datensätze	Variante II	Variante VII
40.000	10,0 sec	10,07 sec
80.000	13,04 sec	13,07 sec
120.000	20,07 sec	17,0 sec
160.000	26,0 sec	22,06 sec

Abbildung 3.48: Kostenverlauf und Laufzeit von Variante II und Variante VII - Zugriff auf mehrere Subklassen

Queries, die nur auf die Daten der Superklasse zugreifen, benötigen ein UNION-Statement, um die Daten zu vereinigen. Der Kostenverlauf und die Laufzeiten beider Varianten für eine UNION-Abfrage ist in Abbildung 3.49 zu sehen. Es ist ein linearer Kostenverlauf gegeben, bei dem Variante VII etwas höhere Kosten aufweist, als Variante II. Allerdings können unter Verwendung von Variante VII signifikant kürzere Laufzeiten beobachtet werden, wenn es sich um Datenmengen im Bereich von hundertzwanzigtausend Datensätzen handelt. Bei größeren Datenmengen gleichen sich die Laufzeiten an. Dominieren also Abfragen, die nur Daten der Superklasse benötigen und die Laufzeit soll möglichst minimal gehalten werden, ist Variante VII zu wählen. Sollen die Kosten niedrig gehalten werden, ist Variante II besser geeignet.

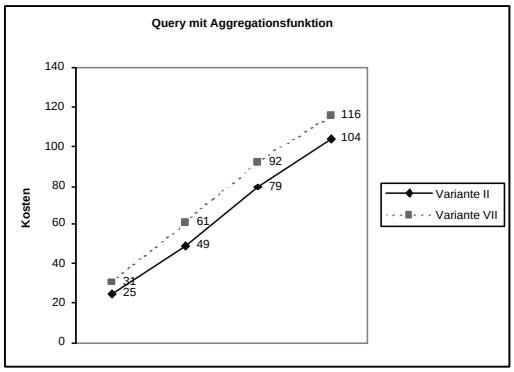


Laufzeit

Anzahl Datensätze	Variante II	Variante VII
40.000	21,05 sec	11,07 sec
80.000	56,06 sec	28,07 sec
120.000	1:09 min	38,03 sec
160.000	1:51 min	1:52 min

Abbildung 3.49: Kostenverlauf und Laufzeit von Variante II und Variante VII - Zugriff erfolgt nur auf die Superklasse

Abfragen, die eine Aggregationsfunktion beinhalten, ergeben im wesentlichen dasselbe Bild, wie in Abbildung 3.50 zu sehen ist. Im Beispiel wurde die Funktion *AVG* als Aggregationsfunktion verwendet.



Laufzeit

Anzahl Datensätze	Variante II	Variante VII
40.000	0 sec	0,02 sec
80.000	0,02 sec	0,03 sec
120.000	2 sec	2,08 sec
160.000	2,06 sec	2,05 sec

Abbildung 3.50: Kostenverlauf und Laufzeit von Variante II und Variante VII - Verwendung einer Aggregationsfunktion

Auch hier verlaufen die Kosten linear, mit etwas höheren Kosten bei Variante VII. Die Laufzeiten unterscheiden sich kaum. Für die Entscheidung sind also die Kosten ausschlaggebend, wobei Variante II zu bevorzugen ist.

Die Analyse ergibt für Variante II und Variante VII ein sehr ähnliches Kosten und Laufzeitverhalten. Dieses Resultat ist nicht sehr überraschend, da die Tabellen eigentlich identisch sind, mit dem einzigen Unterschied, daß die Tabellen von Variante VII Object Tables sind, deren Typ Attribute von einem Supertyp erbt. Die Tabellen in Variante II sind keine Object Tables, sondern relationale Tabellen, die aus zusammengesetzten Attributen bestehen. Die Kosten

von Variante VII sind etwas höher als die Kosten von Variante II, der Unterschied ist aber sehr gering. Abfragen die Attribute genau einer Subklasse und die dazugehörigen Attribute der Superklasse benötigen, haben nur geringe Kosten. Abfragen, die Daten der Superklasse unabhängig von den Subklassen benötigen, haben hier natürlich sehr hohe Kosten, da ein UNION-Statement angewandt werden muß, um die Superklasseattribute aller Subklassen zu vereinigen. Dasselbe gilt für Abfragen, die Daten mehr als einer Subklasse benötigen. Hier muß anstelle des UNION-Statements ein Join angewandt werden, der ebenfall sehr kostenintensiv ist. Die Laufzeit beider Varianten ist fast identisch, kleine Unterschiede zugunsten von Variante VII treten nur bei Abfragen auf, die ein UNION-Statement beinhalten. Die Kosten von Abfragen mit Aggregationsfunktionen entsprechen denen von Abfragen, die auf genau eine Subklasse zugreifen. Die vergleichsweise kürzere Laufzeit ergibt sich dadurch, daß nur ein Datensatz zurückgeliefert wird, das Ergebnis kann also schneller dargestellt werden. Als Entscheidungshilfe kann Abbildung 3.51 dienen, in der die Ergebnisse zusammengefaßt sind.

Variante II und VII	Genau eine Subklasse in Verbindung mit der Superklasse		Mehrere Subklassen in Verbindung mit der Superklasse		Nur Superklasse		Abfrage mit Aggregationsfunktion	
	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze
Minimale Kosten	II	II	II	II	II	II	II	II
Minimale Laufzeit	II, VII	II, VII	II, VII	VII	VII	VII	II, VII	II, VII

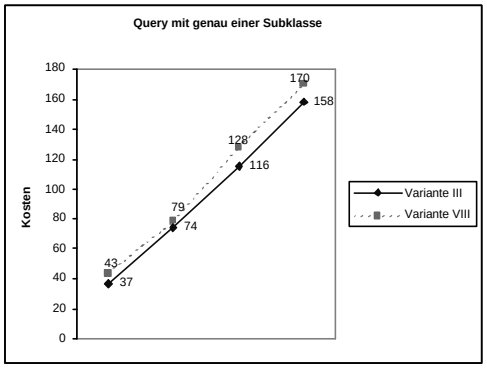
Abbildung 3.51: Entscheidungshilfe zur Wahl zwischen Variante II und VII im Hinblick auf Kosten- und Laufzeitminimierung

Variante III und Variante VIII

In diesem Abschnitt werden Variante III (Einzelrelation mit Subklassen und Superklasse als zusammengesetzte Attribute) und Variante VIII (Vererbungshierarchie mit einer einzelnen Object Table) miteinander verglichen. Da beide keine überlappende Vererbung unterstützen, konnten hier keine Abfragen analysiert werden, die auf mehrere Superklassen zugreifen. Untersucht wurden Abfragen, die auf genau eine Subklasse in Verbindung mit der Superklasse zugreifen, Abfragen, die nur auf Daten der Superklasse zugreifen, sowie Abfragen, die eine Aggregationsfunktion beinhalten.

In Abbildung 3.52 ist der Kostenverlauf und das Laufzeitverhalten einer Abfrage, die die Daten eines Kunden zurückliefert, die also auf genau eine Subklasse in Verbindung mit der Superklasse

zugreift.



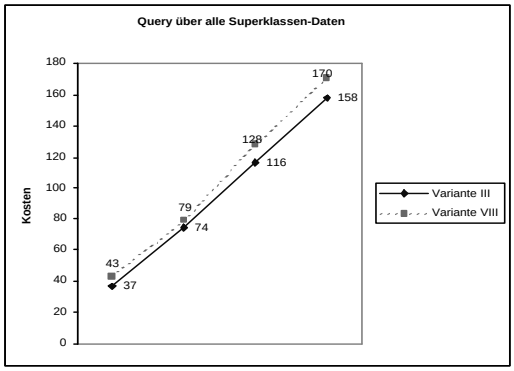
Laufzeit

Anzahl Datensätze	Variante III	Variante VIII
40.000	0,02 sec	1,01 sec
80.000	0,05 sec	1,04 sec
120.000	4,01 sec	4,07 sec
160.000	6,01 sec	8,02 sec

Abbildung 3.52: Kostenverlauf und Laufzeit von Variante III und Variante VIII - Zugriff auf genau eine Subklasse

Die Kosten steigen linear mit der Anzahl der Datensätze, wobei Variante VIII höhere Kosten benötigt als Variante III und auch die schlechteren Laufzeiten aufweist. Außerdem sind Abfragen unter Verwendung von Variante VIII, wie in Abschnitt 3.6.3 beschrieben, aufwendig zu formulieren und schlecht lesbar. Werden also großteils Abfragen gestellt, die auf genau eine der Subklassen zugreifen, ist Variante III zu bevorzugen.

Abfragen, die nur Daten der Superklasse benötigen, unabhängig von der Subklasse, der die Instanz zugeordnet ist, haben ebenfalls einen nahezu linearen Kostenverlauf, wie in Abbildung 3.53 zu sehen ist.



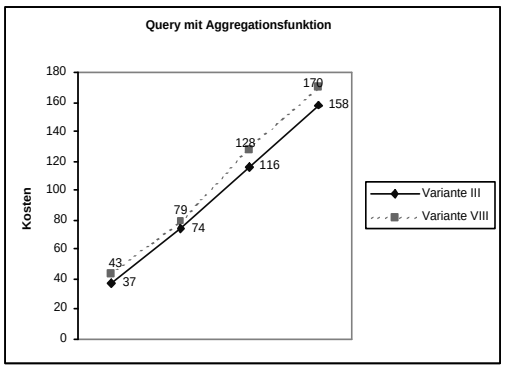
Laufzeit

Anzahl Datensätze	Variante III	Variante VIII
40.000	21,02 sec	10,04 sec
80.000	55,09 sec	27,08 sec
120.000	1:04 min	32,01 sec
160.000	1:44 min	1:44 min

Abbildung 3.53: Kostenverlauf und Laufzeit von Variante III und Variante VIII - Zugriff nur auf Daten der Superklasse

Auch hier sind die benötigten Kosten für Variante VIII geringfügig höher, während die Laufzeit

im Vergleich mit Variante III signifikant kürzer ist, vor allem wenn es sich um eine geringere Anzahl von Datensätzen handelt. Es muß entweder Laufzeit oder Kostenminimalität als Entscheidungskriterium herangezogen werden. Prinzipiell ist für schnellere Laufzeiten Variante VIII zu bevorzugen, wobei die aufwendige Formulierung der Abfragen in die Entscheidung miteinbezogen werden sollte. Für Minimierung der Kosten ist Variante III zu wählen.



Laufzeit

Anzahl Datensätze	Variante III	Variante VIII
40.000	0,01 sec	0,02 sec
80.000	0,03 sec	0,03 sec
120.000	4,03 sec	4,03 sec
160.000	5,05 sec	5,02 sec

Abbildung 3.54: Kostenverlauf und Laufzeit von Variante III und Variante VIII - Abfrage mit Aggregationsfunktion

Der Kostenverlauf und das Laufzeitverhalten von Abfragen, die eine Aggregationsfunktion beinhalten - im Beispiel die Durchschnittsbildung von Kontoständen - ist in Abbildung 3.54 zu sehen. Für derartige Abfragen unterscheiden sich die beiden Varianten kaum. Für Kostenminimierung ist Variante III zu wählen, für die Laufzeit kann keine Entscheidung zugunsten einer der beiden Varianten getroffen werden, da die Laufzeiten nahezu identisch sind. Die Unterschiede im Hinblick auf die Kosten sind minimal, daher sollten zur Entscheidung ob Variante III oder Variante VIII eingesetzt wird, zusätzlich die bereits vorgestellten Kategorien von Abfragen herangezogen werden, die entweder nur auf die Superklasse oder auf Subklassen in Verbindung mit der Superklasse zugreifen.

Variante III und VIII	Genau eine Subklasse in Verbindung mit der Superklasse		Nur Superklasse		Abfrage mit Aggregationsfunktion	
	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze
Minimale Kosten	III	III	III	III	III	III
Minimale Laufzeit	III	III	VIII	VIII	III, VIII	III, VIII

Abbildung 3.55: Entscheidungshilfe zur Wahl zwischen Variante III und VIII im Hinblick auf Kosten- und Laufzeitminimierung

Ein Überblick darüber, wann Variante III und wann Variante VIII eingesetzt werden soll, ist in Abbildung 3.55 zu sehen.

Erzeugen und Modifizieren von Datensätzen

Wenn nicht Abfragen den Großteil der Operationen ausmachen, sondern Modifikationen und Anlage der Datensätze, sollte man die Kosten für INSERT- und UPDATE-Statements als Entscheidungsgrundlage heranziehen. Auch hier wird zuerst durch Berücksichtigung von totaler, partieller, disjunkter und überlappender Vererbung die Auswahl der geeigneten Varianten eingeschränkt.

Variante	Insert	Update
I	608	401
IV	515	2
V	572	2
VI	275	2

Abbildung 3.56: Kosten eines Insert- und Update-Statements - Varianten I, IV, V und VI

Wenn sowohl partielle, totale, disjunkte und überlappende Vererbung möglich sind, also Variante I, IV, V oder VI zum Einsatz kommen, sollte auf keinen Fall Variante I - die Abbildung von Vererbung mit Hilfe von Referenzen - gewählt werden, wenn Abfragen und Modifikationen in der Operationshäufigkeitstabelle dominieren. Wie in Abbildung 3.56 zu sehen ist, weist Variante I signifikant höhere Kosten für Insert- und Update-Statements auf, als die anderen drei Varianten. Nur Variante V hat ähnlich hohe Kosten, durch den Zugriff auf das VARRAY, die Kosten für ein Update sind aber deutlich geringer.

Die hohen Kosten von Variante I sind schnell erklärt. Wird ein neuer Datensatz eingefügt, der einer Subklasse zugeordnet ist, muß zunächst der Eintrag der Superklassenattribute in die Tabelle der Superklasse erfolgen. Danach muß der OID des eben eingefügten Datensatzes aus der Superklassen-Tabelle selektiert und in eine Referenz umgewandelt werden, bevor die Attribute der Subklasse in die entsprechende Tabelle eingefügt werden. Auch wenn auf dem eindeutigen Attribut “Personalnr“ der Superklassen-Tabelle ein Index liegt, kostet die Selektion des OID viel Zeit, die bei anderen Varianten nicht nötig ist. Umgehen kann man diese Selektion nur, indem man wie in Abschnitt 3.6.3 bei Variante I beschrieben, den Primärschlüssel der Superklassen-Tabelle als OID verwendet. Dadurch können die Kosten und die Laufzeit für ein Insert-Statement zwar verringert werden, allerdings wird das Abfrage-Verhalten dadurch

deutlich schlechter. Variante I schneidet bei Abfragen sehr gut ab, ist also primär für Datenbanken zu empfehlen, bei denen nur selten neue Datensätze erzeugt oder bestehende Datensätze modifiziert werden. Anderenfalls ist Variante VI zu empfehlen, da hier beim Einfügen von Datensätzen die niedrigsten Kosten entstehen.

Wenn partielle Vererbung nicht erlaubt ist, kommt entweder Variante II oder Variante VII in Frage. Die Kosten für die jeweiligen Insert- und Update-Statements sind in Abbildung 3.57 zu sehen.

Variante	Insert	Update
II	235	2
VII	275	2

Abbildung 3.57: Kosten eines Insert- und Update-Statements - Variante II und Variante VII

Beide Varianten haben dieselben Kosten bei der Änderung von Datensätzen. Da ein Primärschlüsselattribut existiert, kann der Datensatz, der geändert werden soll, rasch durch Angabe der Personalnummer unter Miteinbeziehung des Index gefunden werden. Die Kosten sind hier also sehr gering. Bei der Neuanlage eines Datensatzes benötigt Variante II geringere Kosten und ist daher zu bevorzugen.

Ist nur disjunkte Vererbung erlaubt, kommt entweder Variante III oder Variante VIII in Frage. Die Kosten für die Insert- und Update-Statements sind in Abbildung 3.58 zu sehen.

Variante	Insert	Update
III	363	2
VIII	394	2

Abbildung 3.58: Kosten eines Insert- und Update-Statements - Variante III und Variante VIII

Die Kosten für die Modifikation eines Datensatzes sind, wie auch schon beim Vergleich von Variante II und VII, sehr gering, was auch hier daran liegt, daß ein Index zur Suche des Datensatzes verwendet wird. Allerdings sollte in die Entscheidung miteinbezogen werden, daß Variante VIII gemäß Abschnitt 3.6.3 ein sehr kompliziertes Statement zur Modifikation eines Datensatzes benötigt, bei dem auch immer Attribute modifiziert werden müssen, die sich nicht geändert haben. Bei der Neuanlage eines Datensatzes benötigt Variante VIII höhere Kosten. Bei disjunkter Vererbung und Insert- und Update-Statements als Auswahlkriterium ist daher

auf jeden Fall Variante III zu bevorzugen.

Hält sich die Anzahl der Abfragen die Waage mit der Anzahl der Modifikationen und Neuanlage von Datensätzen, müssen die Vor- und Nachteile, die durch den Einsatz der verschiedenen Varianten entstehen, sorgfältig gegeneinander abgewogen werden. So wird in diesem Fall Variante I sicher zu vermeiden sein, da das gute Kosten- und Laufzeitverhalten bei Abfragen durch das signifikant schlechte Kosten- und Laufzeitverhalten beim Erzeugen und Modifizieren wieder aufgehoben wird. Hier muß eine Variante gefunden werden, die in beiden Fällen möglichst gute Ergebnisse liefert. Auch hier ist zu entscheiden, ob vermehrt Wert auf kurze Antwortzeiten oder minimale Kosten gelegt wird.

Die Performance-Studie hat gezeigt, daß neben der Struktur der Daten auch das Mengen- und Transaktionsprofil der zu erstellenden Datenbank bekannt sein muß, wobei vor allem das Transaktionsprofil von entscheidender Bedeutung ist. Je nachdem ob es sich bei der Mehrheit der Operationen um Abfragen handelt, die auf genau eine Subklasse in Verbindung mit der Superklasse zugreifen, um Abfragen, die nur auf die Superklasse zugreifen, um Abfragen, die auf mehrere Subklassen zugreifen, um Abfragen, die Aggregationsfunktionen beinhalten oder um Insert- und Update-Statements, ist der Einsatz einer Variante vorteilhaft gegenüber den anderen. Man wird daher aus dem Transaktionsprofil die Operationskategorie wählen, die dominierend gegenüber den anderen ist und erst dann anhand der Kosten- und der Laufzeitanalyse entscheiden, welche Variante am geeignetsten ist. Dazu wiederum ist es notwendig zu wissen, ob das Hauptaugenmerk auf einer schnellen Antwortzeit und damit verbunden einer kurzen Laufzeit oder auf der Kostenminimierung und damit geringen Auslastung der Datenbank liegen soll. Je nachdem welcher der beiden Faktoren die größere Bedeutung besitzt, wird eine andere Variante zur Vererbungsrealisierung gewählt.

Generell ist hinzuzufügen, daß für jede neue Datenbank, wenn genügend Zeit und Budget vorhanden ist, die Kosten- und Laufzeit-Analyse im Hinblick auf die speziellen Operationen der zu erstellenden Datenbank durchgeführt werden sollte, um ein möglichst optimales Ergebnis zu erreichen. Weiters ist zu berücksichtigen, daß Kosten und Laufzeit der verschiedenen Varianten durch physisches Datenbank-Design noch verbessert werden können, worauf in dieser Arbeit keine Rücksicht genommen wurde.

3.6.5 Vererbung in DB2 UDB

DB2 bietet als objekt-relationale Erweiterungen benutzerdefinierte und zusammengesetzte Datentypen an, sowie Referenzen und Vererbung. Mehrwertige Attribute werden nicht un-

terstützt, daher können in DB2 auch nicht alle sieben Varianten der Vererbungsauflösung angewandt werden. Variante V, die den Einsatz eines mehrwertigen Attributes erfordert, kann nicht realisiert werden. Zusätzlich bietet DB2 eine weitere Variante an, bei der nicht nur Datentypen sondern auch Tabellen in eine Vererbungshierarchie miteingebunden werden. Bei Selektion von Datensätzen der Superklassentabelle werden dabei auch immer Datensätze der Subklassentabellen mitselektiert, doch dazu später mehr.

Variante I

Um Variante I in DB2 zu realisieren, müssen zunächst die zusammengesetzten Datentypen definiert werden:

```
CREATE TYPE varI_person_typ as (
        personalnr      CHAR(5),
        name            VARCHAR(60),
        adresse         VARCHAR(100),
        person_alter    DECIMAL(2)
        )
        REF USING INT
        MODE DB2SQL;

CREATE TYPE varI_kunde_typ as (
        kontonummer     CHAR(11),
        kontostand      DECIMAL(10,2),
        ref_person      REF(varI_person_typ)
        )
        REF USING INT
        MODE DB2SQL;

CREATE TYPE varI_angest_typ as (
        gehalt          DECIMAL(7,2),
        abteilung       VARCHAR(50),
        ref_person      REF(varI_person_typ)
        )
        REF USING INT
        MODE DB2SQL;
```

Für jeden Datentyp wird angegeben, welcher Basisdatentyp zur internen Darstellung einer Referenz verwendet werden soll. Im Beispiel ist es der Datentyp Integer, was dem DBMS durch die Klausel "REF USING INT" mitgeteilt wird. Wird kein Wert angegeben, wird zur Darstellung der Referenz der Datentyp VARCHAR(16) FOR BITDATA verwendet. Ein Datentyp muß mit "MODE DB2SQL" abgeschlossen werden.

Für jeden der Datentypen wird nun eine Tabelle definiert:

```
CREATE TABLE varI_person OF varI_person_typ (
            REF IS oid USER GENERATED,
            oid WITH OPTIONS UNIQUE,
            personalnr WITH OPTIONS UNIQUE NOT NULL
);

CREATE TABLE varI_kunde OF varI_kunde_typ (
            REF IS oid USER GENERATED,
            ref_person WITH OPTIONS SCOPE varI_person REFERENCES varI_person(oid)
);

CREATE TABLE varI_angestellter OF varI_angest_typ (
            REF IS oid USER GENERATED,
            ref_person WITH OPTIONS SCOPE varI_person REFERENCES varI_person(oid)
);
```

Die Tabellen werden als getypte Tabellen angelegt (Typed Tables in DB2). Mit der Klausel "REF IS oid USER GENERATED" wird dem DBMS mitgeteilt, in welcher Spalte der Tabelle der OID gespeichert werden soll. Diese Spalte darf nicht zum zusammengesetzten Datentypen gehören, auf dem die Tabelle definiert ist. Die Schlüsselwörter "USER GENERATED" müssen in jedem Fall angegeben werden, auch wenn der OID vom System generiert wird. Beim Einfügen von Datensätzen hat der Benutzer die Möglichkeit, den OID selbst anzugeben oder mittels der Funktion *generate_unique()* vom DBMS generieren zu lassen. Für das DBMS spielt es allerdings keine Rolle, ob der Benutzer einen speziellen Wert als OID hinschreibt, oder ob dieser Wert durch eine Funktion generiert wird. Daher muß die Klausel "USER GENERATED" in jedem Fall angegeben werden. Wir erinnern uns, daß sich in Oracle das DBMS wirklich selbst um die Erstellung und Verwaltung des OID kümmert. Dort muß der OID beim Erstellen eines Datensatzes in keiner Form vom Benutzer angegeben werden. Genau hier liegt der Unterschied zwischen einem "system-generated" und einem "user-generated" OID. Auch wenn in DB2

mittels einer Funktion ein eindeutiger Wert generiert wird, muß diese Funktion beim Erstellen eines Datensatzes trotzdem vom Benutzer aufgerufen werden, daher handelt es sich auch hier um einen vom Benutzer generierten OID.

Für das referenzierende Attribut einer Tabelle kann durch Verwendung des Schlüsselwortes "SCOPE" angegeben werden, auf welche Tabelle referenziert werden soll und in weiterer Folge durch das Schlüsselwort "REFERENCES" in welcher Spalte der OID der zu referenzierenden Tabelle zu finden ist. Durch diese Einschränkungen wird die Einhaltung der referentiellen Integrität vom DBMS überwacht. Es ist ohne weiteres möglich eine Referenz zu definieren, die auf eine nicht vorhandene Tabelle referenziert, man spricht dann von einer *Dangling Reference*. Wird keine Einschränkung beim Erstellen der Tabelle angegeben, muß diese auf jeden Fall bei Verwendung des referenzierenden Attributes in einer Abfrage angegeben werden. Dadurch werden Abfragen wieder komplex und unübersichtlich. Daher ist es auf jeden Fall vorteilhaft, den Scope des referenzierenden Attributes bereits bei der Erstellung der Tabelle zu definieren.

Eine Abfrage auf die Tabellen, die eine Referenz verwendet, muß folgendermaßen aussehen:

```
SELECT ref_person->name, kontostand
FROM varI_kunde
WHERE ref_person->personalnr = '25';
```

Diese Abfrage funktioniert allerdings nur, wenn beim Erstellen der Tabelle ein Scope für das referenzierende Attribut angegeben wurde. Ansonsten weiß das DBMS nicht, auf welche Tabelle referenziert wird, man muß folgende Abfrage verwenden:

```
SELECT CAST(ref_person AS REF(varI_person_typ) SCOPE varI_person)->name, kontostand
FROM varI_kunde
WHERE CAST(ref_person AS REF(varI_person_typ) SCOPE varI_person)->personalnr = '25';
```

Variante II

Bei Variante II wird für jede Subklasse eine Tabelle angelegt, die neben den Attributen der Subklasse auch die Attribute der Superklasse als zusammengesetzten Datentyp enthält. Da es in DB2 nicht möglich ist, ein Attribut eines zusammengesetzten Datentypen als Primary Key zu verwenden oder als "Unique" zu deklarieren, wird die Personalnummer nicht in den zusammengesetzten Datentyp der Superklasse aufgenommen, sondern als normales Attribut in die Tabelle übernommen.

```
CREATE TYPE person_typ AS (
        name            VARCHAR(60),
        adresse         VARCHAR(100),
        person_alter    DECIMAL(2)
        )
        REF USING INT
        MODE DB2SQL;

CREATE TABLE varII_kunde (
        personalnr      CHAR(5) NOT NULL,
        person          person_typ,
        kontonummer     CHAR(11),
        kontostand      DECIMAL(10,2)
        );
ALTER TABLE varII_kunde ADD PRIMARY KEY (personalnr);

CREATE TABLE varII_angestellter (
        personalnr      CHAR(5) NOT NULL,
        person          person_typ,
        gehalt          DECIMAL(7,2),
        abteilung       VARCHAR(50)
        );
ALTER TABLE varII_angestellter ADD PRIMARY KEY (personalnr);
```

Im Gegensatz zu Variante I handelt es sich hier nicht um eine Typed Table. Um auf das Attribut zuzugreifen, das auf dem zusammengesetzten Datentypen definiert ist, muß eine Abfrage folgender Art verwendet werden:

```
SELECT person..name, kontostand
FROM varII_kunde
WHERE personalnr = '25'
```

Das Attribut als Gesamtes kann nur dann selektiert werden, wenn eine sogenannte Transform-Function zur Verfügung steht, die die Instanz als Parameter übernimmt und ihre Attribute in eine lesbare Form bringt. Das DBMS stellt diese Transform-Functions nicht zur Verfügung, es liegt in der Verantwortung des Benutzers diese Funktionen zu schreiben.

Wenn in DB2 ein zusammengesetzter Datentyp angelegt wird, werden gleichzeitig sogenannte *Konstruktor-*, *Mutator-* und *Observer-Methoden* für den Datentyp definiert. Mit einer Mutator-Methode kann der Inhalt einer Instanz verändert werden. Mit einer Observer-Methode wird der Inhalt einer Instanz ausgelesen. Mutator- und Observer-Methoden werden für jedes Attribut des zusammengesetzten Datentypen angelegt und tragen den Name des entsprechenden Attributs. Der Operator ".." gefolgt vom Name der Methode signalisiert dem DBMS, daß ein Methodenaufruf erfolgt, bei dem eine Mutator- oder Observer-Methode aufgerufen wird. Mit der Konstruktor-Methode wird eine leere Instanz des Datentyps erzeugt, die in weiterer Folge befüllt werden kann. Der Aufruf einer Konstruktor-Methode erfolgt mit *datatype_name()* gefolgt von den Mutator-Methoden der einzelnen Attribute.

Beim Einfügen eines Datensatzes in eine Tabelle muß für jedes Attribut, das auf einem zusammengesetzten Datentypen definiert ist, die Konstruktor-Methode des Datentypen aufgerufen werden, gefolgt von den Mutator-Methoden der Attribute:

```
INSERT INTO varII_kunde VALUES
         ('25',
          person_typ()..name('Max Mustermann')
                      ..adresse('Hauptstrasse 48, 9020 Klagenfurt')
                      ..person_alter(34),
          '1888000005',
          8000,00
         );
```

Die Instanz des zusammengesetzten Datentypen wird mit Hilfe der Konstruktor-, Mutator- und Observer-Methoden angelegt, die restlichen Attribute werden in gewohnter Weise wie bei Verwendung einer relationalen oder getypten Tabelle befüllt.

Variante III

Bei Verwendung von Variante III werden zunächst Datentypen für jede Subklasse gebraucht. Für die Superklasse kann der Datentyp, der bei Variante II angelegt wird, wiederverwendet werden.

```
CREATE TYPE kunde_typ AS (
        kontonummer     CHAR(11),
        kontostand      DECIMAL(10,2)
        )
        REF USING INT
        MODE DB2SQL;
```

```
CREATE TYPE angestellter_typ AS (
        gehalt          DECIMAL(7,2),
        abteilung       VARCHAR(50)
        )
        REF USING INT
        MODE DB2SQL;
```

Als nächstes wird eine Tabelle angelegt, die ein Attribut für den Datentyp der Superklasse und jeweils ein Attribut für den Datentyp jeder Subklasse enthält. Zusätzlich muß das Attribut *personalnr* in die Tabelle mit aufgenommen werden, um einen Primärschlüssel anlegen zu können. Wie bereits in Variante II erklärt, ist es in DB2 nicht möglich, ein Attribut eines zusammengesetzten Datentypen als Primärschlüssel zu verwenden, daher wird das Attribut aus dem Datentyp herausgenommen und als relationales Attribut angelegt. Ein weiteres Attribut wird für die Zuordnung einer Instanz zu einer bestimmten Subklasse benötigt.

```
CREATE TABLE varIII_Person (
        personalnr      CHAR(5) NOT NULL,
        person          person_typ,
        kunde           kunde_typ,
        angestellter    angestellter_typ,
        subklasse_typ   VARCHAR(20) CONSTRAINT check_subklasse
                             CHECK (subklasse_typ IN ('Kunde', 'Angestellter')));
ALTER TABLE varIII_person ADD PRIMARY KEY (personalnr);
```

Auf die Attribute der zusammengesetzten Datentypen kann wie schon in Variante II mit Hilfe der Observer-Methoden zugegriffen werden.

Variante IV

Variante IV kann im wesentlichen genau wie Variante III realisiert werden, mit dem einzigen Unterschied, daß das Attribut *subklasse_typ* als Datentyp einen zusammengesetzten Datentypen mit bool'schen Attributen für die verschiedenen Subklassen verwendet:

```
CREATE TYPE subklasse_typ AS (
        kunde           CHAR(1),
        angestellter    CHAR(1)
        )
        REF USING INT
        MODE DB2SQL;
```

Die anzulegende Tabelle entspricht jener von Variante III, allerdings wird das Attribut *subklasse_typ* mit dem zusammengesetzten Datentyp anstelle des Basisdatentyps VARCHAR(20) definiert.

Variante V

Variante V kann in DB2 nicht realisiert werden, da das DBMS keine mehrwertigen Attribute unterstützt.

Variante VI

Bei Verwendung von Variante VI werden die zusammengesetzten Datentypen in einer Vererbungshierarchie angelegt, wie es auch bei Oracle der Fall war.

```
CREATE TYPE inherit_person_typ AS (
        personalnr      CHAR(5),
        name            VARCHAR(60),
        adresse         VARCHAR(100),
        person_alter    DECIMAL(2)
        )
        NOT FINAL
        MODE DB2SQL;

CREATE TYPE inherit_kunde_typ UNDER inherit_person_typ AS (
        kontonummer     CHAR(11),
        kontostand      DECIMAL(10,2)
        )
        MODE DB2SQL;

CREATE TYPE inherit_angest_typ UNDER inherit_person_typ AS (
        gehalt          DECIMAL(7,2),
        abteilung       VARCHAR(50)
        )
        MODE DB2SQL;
```

Für jeden dieser Datentypen wird eine Tabelle angelegt.

```
CREATE TABLE varVI_person OF inherit_person_typ (REF IS oid USER GENERATED);
```

```
CREATE TABLE varVI_kunde OF inherit_kunde_typ (REF IS oid USER GENERATED);

CREATE TABLE varVI_angest OF inherit_angest_typ (REF IS oid USER GENERATED);
```

An die Subtypen vererbt werden hier - wie auch schon in Oracle - nur die Attribute und Methoden des Supertypen, es erfolgt keine Vererbung der Daten, wenn Datensätze in die Tabellen eingefügt werden.

Variante VII

Variante VII verwendet Vererbung von Attributen und Methoden mittels einer Typhierarchie, allerdings mit einem nicht instantierbaren Superklassen-Typ.

```
CREATE TYPE varVII_person_typ AS (
        personalnr      CHAR(5),
        name            VARCHAR(60),
        adresse         VARCHAR(100),
        person_alter    DECIMAL(2)
        )
        NOT FINAL NOT INSTANTIABLE
        MODE DB2SQL;

CREATE TYPE varVII_kunde_typ UNDER varVII_person_typ AS (
        kontonummer     CHAR(11),
        kontostand      DECIMAL(10,2)
        )
        MODE DB2SQL;

CREATE TYPE varVII_angest_typ UNDER varVII_person_typ AS (
        gehalt          DECIMAL(7,2),
        abteilung       VARCHAR(50)
        )
        MODE DB2SQL;
```

Für den Datentyp *varVII_person_typ* kann keine Tabelle angelegt werden, im Gegensatz zu Oracle, wo die Tabelle angelegt werden konnte, es aber nicht möglich war, Datensätze einzufügen. In DB2 ist es erst gar nicht möglich, eine Tabelle für diesen Datentypen zu definieren. Tabellen

können nur für die Datentypen der Subklassen angelegt werden, es kann also nur totale Vererbung eingesetzt werden.

Variante VIII

DB2 bietet auch erste Ansätze zur Datenvererbung. Dazu können die eben angelegten Datentypen verwendet werden, der Unterschied kommt beim Anlegen der Tabellen zu Tage. Die Tabellen werden nämlich selbst in eine Vererbungshierarchie eingebunden, diese wird also nicht nur mit den Datentypen sondern zusätzlich auch mit den Tabellen abgebildet:

```
CREATE TABLE varVIII_person OF inherit_person_typ (REF IS oid USER GENERATED);

CREATE TABLE varVIII_kunde OF inherit_kunde_typ
                        UNDER varVIII_person INHERIT SELECT PRIVILEGES;

CREATE TABLE varVIII_angest OF inherit_angest_typ
                        UNDER varVIII_person INHERIT SELECT PRIVILEGES;
```

Die Tabelle *varVIII_person* übernimmt die Funktion einer Superklasse. Die Tabellen *varVIII_kunde* und *varVIII_angest* werden durch das Schlüsselwort *UNDER* dieser Tabelle untergeordnet und erben die Zugriffsrechte. Wenn ein Datensatz in eine dieser Tabellen eingefügt wird, muß sein OID entlang der gesamten Tabellenhierarchie eindeutig sein. Damit wird Datenvererbung möglich gemacht. Wenn ein Datensatz, der einer Subklasse zugeordnet ist, angelegt wird, muß er in die entsprechende Tabelle, die die Subklasse repräsentiert, eingefügt werden, mit allen Attributen der Superklasse. Ein Datensatz, der keiner Subklasse zugeordnet ist, wird in die sogenannte Root-Tabelle *varVIII_person* eingefügt.

```
INSERT INTO varVIII_person VALUES
            (inherit_person_typ('1'),
             '1',
             'Max Mustermann',
             'Hauptstrasse 48, 9020 Klagenfurt',
             34
            );

INSERT INTO varVIII_kunde VALUES
            (inherit_kunde_typ('2'),
            '2',
            'Peter Mueller',
```

```
'Musterstrasse 25, 9020 Klagenfurt',
25,
'18880000005',
8000
);
```

Wenn nun eine Abfrage auf die Daten der Superklasse erfolgt, liefert diese auch alle Instanzen zurück, die in den Subklassen enthalten sind.

```
SELECT * FROM varVIII_person
```

liefert also nicht nur die Instanz mit der Personalnummer '1', sondern auch die Instanz mit der Personalnummer '2'. Wenn nur die Instanzen der Superklasse zurückgegeben werden sollen, muß die Abfrage

```
SELECT * FROM ONLY(varIII_person)
```

verwendet werden. Der wesentliche Vorteil dabei ist, daß Redundanz vermieden wird. Jede Instanz muß nur einmal angelegt werden, die Attribute der Superklasse werden nicht redundant gespeichert, wie es der Fall ist, wenn Vererbung nur entlang der Typhierarchie erfolgt. Die Realisierung der Datenverbung im DBMS macht aber auch gleichzeitig überlappende Vererbung unmöglich. Da der OID eines Datensatz entlang der Tabellenhierarchie eindeutig sein muß, ist es nicht möglich, dieselbe Person zweimal anzulegen. Umgehen kann man diese Einschränkung, in dem man einen neuen OID bei gleicher Personalnummer vergibt. Allerdings müssen die Attribute der Superklasse damit wieder redundant gespeichert werden und das DBMS erkennt nicht, daß es sich eigentlich um dieselbe Person handelt. Es wird also empfohlen Variante VIII nur für disjunkte Vererbung zu verwenden.

Eine Übersicht über die Anwendung der verschiedenen Vererbungsvarianten in DB2 ist in Abbildung 3.59 zu sehen.

Wie man sieht erfüllen Oracle und DB2 großteils die Anforderungen, die an ein objekt-relationales DBMS gestellt werden, allerdings nicht vollständig. Während Oracle Vererbung gerade erst in Version 9i implementiert hat, bietet DB2 keine Unterstützung mehrwertiger Datentypen an.

Vererbungsart / Variante	total	partiell	disjunkt	überlappend	Anzahl der Tabellen	Besonderheiten
Variante I : Vererbung mit Typed Tables und Referenzen	X	X	X	X	Superklasse + Anzahl der Subklassen	Zugriff mit Operator „->“, SCOPE für Referenz sollte angegeben werden
Variante II : Eine Tabelle pro Subklasse, Attribute der Superklasse als Structured Type	X		X	X	Anzahl der Subklassen	Redundanz bei überlappender Vererbung, Primärschlüssel nicht im Structured Type
Variante III : Eine einzige Tabelle, Attribute als Structured Types, ein Attribut für Zuordnung zu Subklasse	X	X	X		1	CHECK-Constraint für das Attribut der Zuordnung, Primärschlüssel nicht im Structured Type
Variante IV: Wie Variante III, Attribut für die Zuordnung als Structured Type	X	X	X	X	1	Structured Type für die Zuordnung hat ein bool'sches Attribut für jede Subklasse
Variante V: nicht realisierbar	-	-	-	-	-	
Variante VI: Vererbung mit Structured Type Hierachie	X	X	X	X	Superklasse + Anzahl der Subklassen	Redundanz bei überlappender Vererbung
Variante VII: Wie Variante VI, aber Root-Typ ist nicht instantierbar	X		X	X	Anzahl der Subklassen	Redundanz bei überlappender Vererbung
Variante VIII: Vererbungshierarchie wird mit Structured Types und Tabellen abgebildet	X	X	X		Superklasse + Anzahl der Subklassen	Auch Daten werden vererbt, eindeutiger OID entlang der gesamten Tabellenhierarchie

Abbildung 3.59: Überblick - Vererbungsvarianten in DB2

3.6.6 Performance-Studie in DB2

In diesem Abschnitt sollen die gerade vorgestellten Varianten der Vererbungsabbildung in DB2 im Hinblick auf ihr Kosten- und Laufzeitverhalten untersucht werden. Dabei werden wie auch schon bei der Performance-Studie in Oracle diejenigen Varianten miteinander verglichen, die dieselben Eigenschaften bezüglich totaler, partieller, disjunkter und überlappender Vererbung haben. Variante I (Vererbung mit Hilfe von getypten Tabellen und Referenzen) wird mit Variante IV (Vererbung mit einer Einzelrelation und einem Structured Type für die Zuordnung zu einer Subklasse) sowie mit Variante VI (Vererbung durch eine Structured Type Hierarchie) verglichen. Alle drei Varianten unterstützen sowohl totale und partielle als auch disjunkte und überlappende Vererbung. Variante II (eine Tabelle pro Subklasse, die die Attribute der Superklasse als Structured Type enthält) wird mit Variante VII (Vererbung durch Structured Type Hierarchie mit nicht-instanzierbarem Supertyp) verglichen, da beide durch die Struktur ihrer Tabellen partielle Vererbung ausschließen. Schließlich wird Variante III (Einzelrelation mit einem Attribut für die Zuordnung zu einer Subklasse) mit Variante VIII (Vererbung durch

eine Hierarchie von Typed Tables) verglichen. Die letzten beiden Varianten haben gemeinsam, daß keine überlappende Vererbung möglich ist.

Die Kosten- und Laufzeit-Analyse wurde auf einem Athlon XP 1800+ mit 256 MB RAM unter Windows 2000 durchgeführt. Beim DBMS handelte es sich um die Standard-Installation der IBM DB2 UDB 7.2 Workgroup Edition, an der keine physischen Tuning-Maßnahmen durchgeführt wurden.

Name/Kurzbeschreibung	Häufigkeit	Typ (Online/Batch)
O1: Abfragen von Kundendaten	300x/Tag	OL
O2: Abfragen von Kunden- und Angestelltendaten	350x/Tag	OL
O3: Personen, die älter als 30 sind	60x/Tag	OL
O4: Durchschnittsvermögen aller Kunden	20x/Tag	OL
O5: Durchschnittsgehalt aller Angestellten	25x/Tag	OL
O6: Anlegen eines Kunden	30x/Tag	OL
O7: Anlegen eines Angestellten	1x/Tag	OL
O8: Anlegen einer Person, die weder Kunde noch Angestellter ist	10x/Tag	OL
O9: Löschen von Kunden	1x/Woche	B
O10: Löschen von Angestellten	1x/Woche	B
O11: Löschen von Personen	1x/Woche	B

Abbildung 3.60: Operationshäufigkeitstabelle für das Vererbungschema

Die ausgewählten Operationen sind in Abbildung 3.60 zu sehen. Es handelt sich hierbei um dieselbe Operationshäufigkeitstabelle, die auch in Abschnitt 3.6.4 verwendet wurde. Für die Analyse von Abfragen wurden die Operationen O1, O2, O3, O4 und O5 herangezogen. O1 entspricht einer Abfrage, die auf genau eine Subklasse in Verbindung mit der Superklasse zugreift. Diese Abfrage wurde zweimal gestellt, wobei einmal viele Datensätze als Ergebnis zurückgeliefert wurden und einmal nur einige wenige. O2 entspricht einer Abfrage, die auf mehrere Subklassen in Verbindung mit der Superklasse zugreift, was nur bei überlappender Vererbung möglich ist. Auch diese Abfrage wurde zweimal gestellt, mit vielen und wenig Datensätzen als Ergebnis. Für die Varianten III und VIII war O2 nicht relevant, da beide keine überlappende Vererbung unterstützen. Bei O3 handelt es sich um eine Abfrage, die nur auf Daten der Superklasse zugreift, unabhängig von der Zugehörigkeit zu einer Subklasse. O4 und O5 sind Abfragen, die eine Aggregationsfunktion - die Funktion *AVG* - beinhalten. Für jede Variante wurde weiters eine Kostenanalyse für das Anlegen und die Modifikation eines Datensatzes untersucht. Die Laufzeit wurde hier außer acht gelassen, da sie für alle sieben

Möglichkeiten in etwa denselben Wert hat.

Konzept	Typ	Anzahl
Person	E	40.000
Kunde	E	20.000
Angestellter	E	20.000
PersonalNr	A	40.000
name	A	40.000
adresse	A	30.000
person_alter	A	40.000
kontostand	A	20.000
kontonummer	A	20.000
gehalt	A	20.000
abteilung	A	20.000

Abbildung 3.61: Mengengerüst für das Vererbungsschema

Das Mengengerüst, das als Ausgangsbasis für die Abfragen diente, ist in Abbildung 3.61 zu sehen. Wie auch schon in der Oracle-Performance-Studie existieren 40.000 Personen, von denen 10.000 nur Kunden und 10.000 nur Angestellte sind. Weitere 10.000 sind sowohl Kunden als auch Angestellte und die verbleibenden 10.000 sind weder Kunden noch Angestellte. Das Mengengerüst wurde während der Analyse dreimal um 40.000 Datensätze erhöht, sodaß mit 160.000 Datensätzen als Obergrenze getestet wurde. Dabei wurde beobachtet, wie sich die Kosten und die Laufzeit einer Abfrage mit der Anzahl der Datensätze verändern. Begonnen wurde mit der Kosten- und Laufzeitanalyse für Variante I, IV und VI.

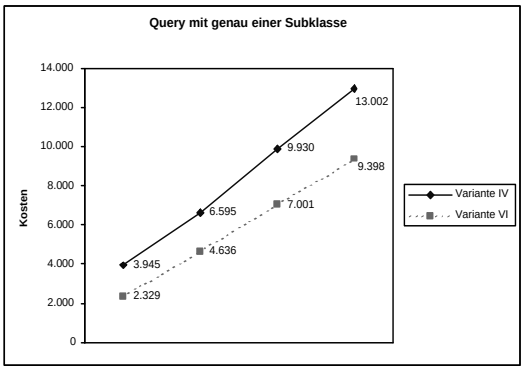
Variante I, IV und VI

Wenn totale, partielle, disjunkte und überlappende Vererbung eingesetzt werden soll, kann zwischen folgenden Varianten gewählt werden:

- Variante I - Vererbung wird mit Hilfe von getypten Tabellen abgebildet. Die Superklasse wird als Typed Table angelegt, die Subklassen referenzieren auf die Superklassen-Tabelle.
- Variante IV - eine Einzelrelation wird angelegt, die die Attribute der Superklasse und der Subklassen als Structured Types enthält. Die Zuordnung einer Instanz zu einer Subklasse erfolgt mit ebenfalls durch ein Structured Type Attribut.

- Variante VI - Die Attribute der Superklasse werden durch die Vererbung, die das DBMS für Structured Types anbietet, an die Subklassen weitergegeben. Es wird also eine Datentyp-Hierarchie aufgebaut, bei der allerdings keine Daten, sondern nur Attribute vererbt werden.

Das Kostenverhalten und die Laufzeiten für eine Abfrage, die auf genau eine Subklasse in Verbindung mit der Superklasse zugreift, ist für diese drei Varianten in Abbildung 3.62 zu sehen. Bei dieser Query werden als Ergebnis viele Datensätze zurückgegeben. Die Abfragen, die nur einige wenige Datensätze zurückliefern, haben für alle drei Varianten das gleiche Kosten- und Laufzeitverhalten, mit minimalen Unterschieden. Aus diesem Grund wird hier verstärkt Wert auf Abfragen gelegt, die viele Datensätze als Ergebnis zurückliefern, da hier zum Teil große Unterschiede sichtbar werden.



Laufzeit

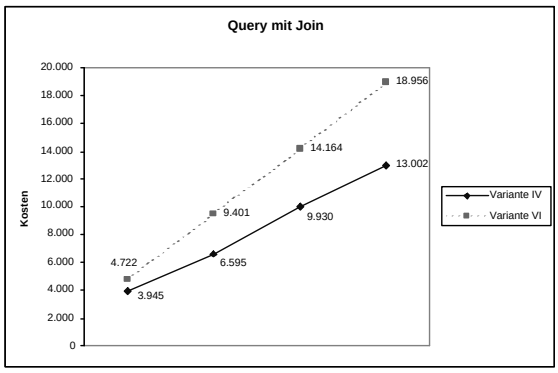
Anzahl Datensätze	Variante I	Variante IV	Variante VI
40.000	0,74 sec	0,36 sec	1,38 sec
80.000	5,46 sec	2,41 sec	3,7 sec
120.000	7,32 sec	4,37 sec	4,88 sec
160.000	9,01 sec	6,53 sec	6,02 sec

Abbildung 3.62: Kosten und Laufzeit der Varianten I, IV und VI - Abfrage, die auf genau eine Subklasse zugreift

Was in der Abbildung auffällt ist, daß Variante I nicht im Kosten-Diagramm abgebildet ist und zwar aus folgendem Grund: Die Kosten für eine Abfrage sind bei Verwendung von Variante I dermaßen hoch, daß das Diagramm durch Aufnahme von Variante I nicht mehr aussagekräftig ist. Die Kosten bewegen sich im zweihundertfachen Bereich von Variante IV, sind also signifikant höher, weshalb im Variante I, wenn Kosten minimiert werden sollen, auf keinen Fall zu empfehlen ist. Auch bei den Laufzeiten schneidet Variante I schlechter ab als die anderen beiden, mit einem Unterschied im Sekundenbereich bei einer Größenordnung von mehr als 80.000 Datensätzen. Daher ist von der Verwendung von Variante I generell abzuraten, der Vorteil der kürzeren und intuitiveren Abfragen durch die Verwendung von Referenzen wird durch das schlechte Kosten- und Laufzeitverhalten zunichte gemacht.

Ein Vergleich zwischen Variante IV und VI zeigt, daß Variante IV höhere Kosten benötigt, aber etwas bessere Laufzeiten aufweist. Bei Variante VI kann die Prüfung auf die Zugehörigkeit zu einer bestimmten Subklasse entfallen, da durch die Aufteilung in separate Tabellen für jede

Subklasse, die Zugehörigkeit ohnehin klar ist. Sollen daher die Kosten minimiert werden, ist Variante VI, die Vererbung mittels einer Typhierarchie, zu bevorzugen. Wenn kurze Laufzeiten das ausschlaggebende Kriterium sind, ist allerdings Variante IV zu bevorzugen, zumindest dann, wenn es sich um eine Abfrage handelt, die auf genau eine Subklasse in Verbindung mit der Superklasse zugreift. Im Beispielschema ist das der Fall, wenn nur die Daten eines Kunden abgefragt werden sollen, ohne Rücksicht darauf, ob der Kunde auch Angestellter ist, oder nicht.



Laufzeit

Anzahl Datensätze	Variante I	Variante IV	Variante VI
40.000	0,71 sec	0,14 sec	0,43 sec
80.000	2,47 sec	2,09 sec	1,60 sec
120.000	4,49 sec	3,43 sec	3,17 sec
160.000	6,35 sec	4,05 sec	4,36 sec

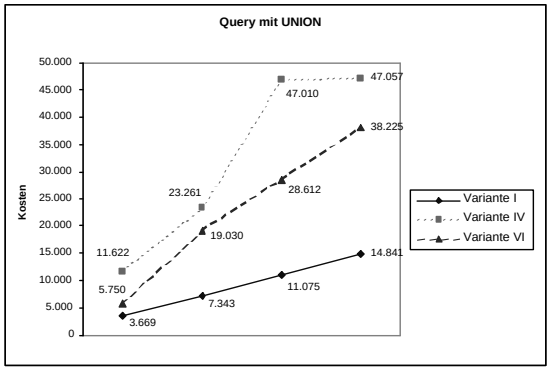
Abbildung 3.63: Kosten und Laufzeit der Varianten I, IV und VI - Abfrage, die auf mehrere Subklassen zugreift

Werden hauptsächlich Abfragen verwendet, die auf mehrere Subklassen in Verbindung mit der Superklasse zugreifen, ergibt sich ein Kosten- und Laufzeitverhalten, wie es in Abbildung 3.63 zu sehen ist. Eine Abfrage dieser Art ist zum Beispiel das Abfragen der Daten von Personen, die sowohl Kunden als auch Angestellte sind.

Im Gegensatz zu einer Abfrage, die nur auf eine Subklasse zugreift, hat Variante IV hier das bessere Kostenverhalten, da es sich um eine Einzelrelation handelt, in der der Join, der bei Variante VI notwendig ist, schon enthalten ist. Es müssen also nur zusätzliche Attribute selektiert werden. Variante IV hat bei wenig Datensätzen (bis 40.000) das bessere Laufzeitverhalten, es wird allerdings mit Zunahme der Datensätze schlechter im Vergleich zu Variante VI. Ab einer Größenordnung von 160.000 Datensätzen verbessert sich die Laufzeit wieder. Für eine Minimierung der Laufzeiten ist daher keine eindeutige Entscheidung zugunsten einer der beiden Varianten zu treffen, unter Berücksichtigung der Kosten ist auf jeden Fall Variante IV, die Einzelrelation, zu wählen. Variante I, die Vererbung durch Einsatz von Referenzen, ist auch bei einer Abfrage, die auf mehrere Subklassen zugreift, auf keinen Fall zu empfehlen. Die Kosten bewegen sich auch hier im zweihundertfachen Bereich im Vergleich zu den anderen Varianten, weshalb Variante I auch hier nicht in das Kosten-Diagramm aufgenommen wurde. Die Laufzeiten sind

um einiges schlechter als die der anderen beiden Varianten, deshalb ist von Variante I abzuraten.

Wenn Abfragen gestellt werden, die nur die Daten der Superklasse benötigen, unabhängig von der Zuordnung zu einer bestimmten Subklasse, ergibt sich das Kostenverhalten und die Laufzeiten von Abbildung 3.64. Da hier bei Variante I bei einem Zugriff keine Referenzen aufgelöst werden müssen, bewegen sich die Kosten in einem annehmbaren Bereich, sodaß Variante I in das Kostendiagramm aufgenommen werden kann.



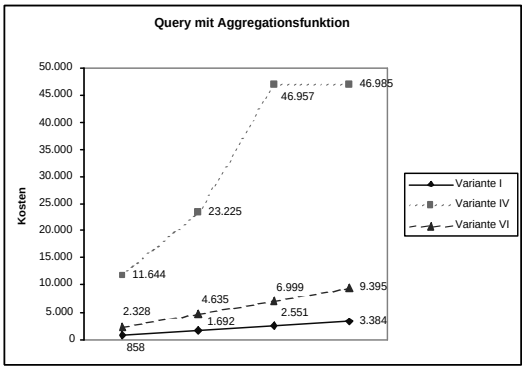
Laufzeit

Anzahl Datensätze	Variante I	Variante IV	Variante VI
40.000	16,86 sec	14,90 sec	17,45 sec
80.000	35,79 sec	30,44 sec	36,58 sec
120.000	1:33,57 min	1:05,66 sec	54,57 sec
160.000	1:10,50 min	59,58 sec	1:12,35 min

Abbildung 3.64: Kosten und Laufzeit der Varianten I, IV und VI - Abfrage, die nur auf Daten der Superklasse zugreift

Wie in der Abbildung ersichtlich ist, weist Variante I hier sogar die geringsten Kosten auf, was nicht weiter überraschend ist, da alle Daten aus der Superklassen-Tabelle selektiert werden können, eine Verbindung zu anderen Tabellen ist nicht nötig. Bei Verwendung von Variante IV erfolgt der Zugriff auf die einzelnen Attribute des Structured Types, der die Superklasse repräsentiert. Es wird also sozusagen eine Mutator-Methode aufgerufen, was dazu führt, daß die Kosten des Zugriffs höher sind, als bei einem Zugriff auf eine getypte Tabelle, wie es bei Variante I der Fall ist. Variante VI hat bei einer Abfrage dieser Art die höchsten Kosten, was sich dadurch erklärt, daß ein UNION zwischen allen Subklassen-Tabellen und der Tabelle der Superklasse nötig ist, um die Daten aller Personen unabhängig von der Zuordnung zu einer Subklasse zu erhalten. Da ein UNION eine sehr kostenintensive Operation ist, sind die Kosten hier auch signifikant höher als bei den anderen Varianten. Sollen die Kosten minimiert werden, ist also Variante I zu empfehlen. Allerdings weist Variante I die schlechtesten Laufzeiten auf. Sollen also die Laufzeiten minimiert werden, ist Variante IV einzusetzen. Variante VI hat großteils die schlechtesten Laufzeiten und auch die höchsten Kosten, ist also in keinem Fall zu empfehlen, wenn hauptsächlich Abfragen gestellt werden, die ausschließlich auf Daten der Superklasse zugreifen.

Wenn die Mehrheit der Abfragen eine Aggregationsfunktion beinhaltet, zum Beispiel die Errechnung des Durchschnittsvermögens aller Kunden, ergibt sich das Kosten- und Laufzeitverhalten aus Abbildung 3.65.



Laufzeit

Anzahl Datensätze	Variante I	Variante IV	Variante VI
40.000	0,05 sec	0,18 sec	0,07 sec
80.000	0,08 sec	0,26 sec	0,11 sec
120.000	0,11 sec	0,38 sec	0,14 sec
160.000	0,15 sec	0,57 sec	0,18 sec

Abbildung 3.65: Kosten und Laufzeit der Varianten I, IV und VI - Abfrage, die eine Aggregationsfunktion beinhaltet

Variante I weist auch hier wieder die geringsten Kosten auf, da nur auf die Subklasse zugegriffen werden muß und keine Referenzen auf die Superklasse aufgelöst werden müssen. Dasselbe gilt für Variante VI, ein Join zu den Daten der Superklasse ist hier nicht notwendig. Bei Variante IV wird zwar auch kein Join durchgeführt, da sich hier aber alle Datensätze, egal ob sie einer Subklasse zugeordnet sind oder nicht, in einer Tabelle befinden, steigen mit der Anzahl der Datensätze die Kosten für die Abfrage. Variante IV weist hier auch deutlich schlechtere Laufzeiten auf, ist daher auf keinen Fall zu empfehlen. Sollen Kosten minimiert werden, ist Variante I einzusetzen, soll die Laufzeit minimiert werden, ist ebenfalls Variante I einzusetzen, da sie neben den niedrigsten Kosten auch die kürzesten Laufzeiten hat.

Ein Überblick, welche Variante im Hinblick auf Kosten- und Laufzeitminimierung zu wählen ist, ist in Abbildung 3.66 zu sehen.

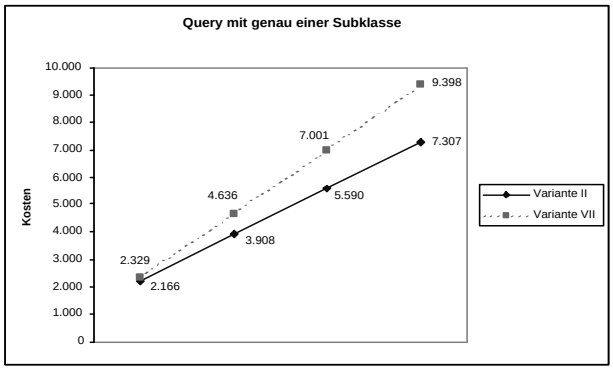
Variante I, IV und VI	Genau eine Subklasse in Verbindung mit der Superklasse		Mehrere Subklassen in Verbindung mit der Superklasse		Nur Superklasse		Abfrage mit Aggregationsfunktion	
	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze
Minimale Kosten	VI	VI	IV	IV	I	I	I	I
Minimale Laufzeit	IV	IV, VI	IV, VI	IV, VI	IV	IV	I	I

Abbildung 3.66: Entscheidungshilfe zur Wahl zwischen Variante I, IV und VI

Variante II und VII

Variante II, Subklassentabellen, die die Superklassen-Attribute als Structured Type enthalten, und Variante VII, eine Vererbungshierarchie von Structured Types mit einem nicht-instanzierbaren Root-Typ, kommen dann zum Einsatz, wenn partielle Vererbung nicht erlaubt ist. Variante II und VII sind einander hinsichtlich der Struktur der Tabellen ähnlich. Trotzdem sind die Ergebnisse der Kosten- und Laufzeit-Analyse sehr unterschiedlich.

Abfragen, die auf genau eine Subklasse in Verbindung mit der Superklasse zugreifen und ein umfangreiches Ergebnis zurückliefern, ergeben ein Kosten- und Laufzeitverhalten, wie es in Abbildung 3.67 zu sehen ist.



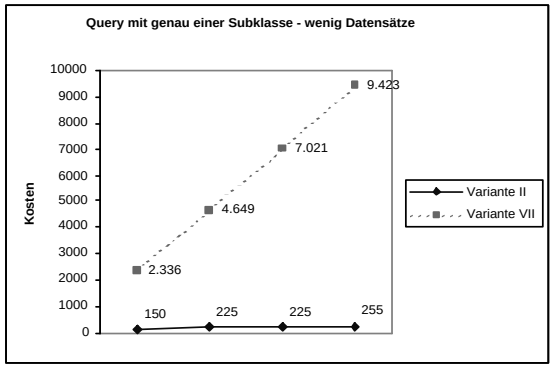
Laufzeit

Anzahl Datensätze	Variante II	Variante VII
40.000	1,78 sec	1,41 sec
80.000	5,24 sec	3,99 sec
120.000	7,88 sec	5,59 sec
160.000	5,69 sec	7,37 sec

Abbildung 3.67: Kosten und Laufzeit der Varianten II und VII - Abfrage, genau eine Subklasse

Die Abbildung zeigt, daß Variante II weniger Kosten verursacht als Variante VII, weshalb bei Kostenminimierung Variante II zu empfehlen ist. Allerdings weist Variante II schlechtere Laufzeiten auf, wenn die Datensätze in einer Größenordnung von unter 120.000 liegen. Ab 160.000 Datensätzen hat Variante II kürzere Laufzeiten. Wenn also kurze Laufzeiten wichtig sind und die Anzahl der Datensätze gering ist, sollte auf Variante VII zurückgegriffen werden, ansonsten auf Variante II.

Interessant ist beim Vergleich von Variante II und VII auch der Vergleich für eine Abfrage, die auf genau eine Subklasse in Verbindung mit der Superklasse zugreift, aber nur einige wenige Datensätze zurückliefert. Grundsätzlich würde man in diesem Fall ein sehr ähnliches Kosten- und Laufzeitverhalten erwarten, tatsächlich bestehen hier aber große Unterschiede, wie in Abbildung 3.68 zu sehen ist.

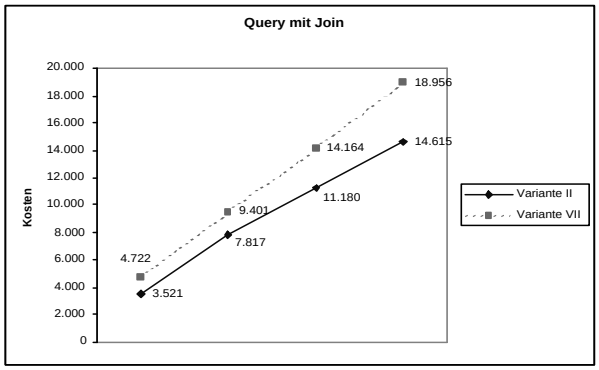


Laufzeit

Anzahl Datensätze	Variante II	Variante VII
40.000	0,05 sec	0,07 sec
80.000	0,04 sec	0,09 sec
120.000	0,04 sec	0,12 sec
160.000	0,05 sec	0,15 sec

Abbildung 3.68: Kosten und Laufzeit der Varianten II und VII - Abfrage, genau eine Subklasse, wenig Datensätze als Ergebnis

Variante VII hat signifikant schlechtere Kosten, als Variante II. Dazu kommt, daß die Kosten von Variante VII mit der Anzahl der Datensätze ansteigen, während die Kosten von Variante II nahezu immer gleich bleiben. Genau dasselbe gilt auch für die Laufzeiten. Die Laufzeit von Variante VII steigt mit der Anzahl der Datensätze an und ist schlechter als die von Variante II, die unabhängig von der Anzahl der Datensätze gleich bleibt. Der große Unterschied erklärt sich dadurch, daß bei Variante II ein Index auf dem Primärschlüsselattribut der relationalen Tabellen liegt. Bei Variante VII handelt es sich um getypte Tabellen, der Index liegt also auf dem OID, nicht auf dem Attribut, das in Variante II den Primärschlüssel darstellt. Ein Index könnte hier in jedem Fall Abhilfe schaffen und die Kosten und Antwortzeiten von Variante VII verbessern. Da IBM aber empfiehlt, den OID zur eindeutigen Identifizierung heranzuziehen, wurde in diesem Performance-Test bei Tabellen, die einen OID haben, kein Primärschlüssel angelegt. Ansonsten wird zur Kosten- und Laufzeitminimierung die Verwendung von Variante II empfohlen oder die Anlage eines Index auf dem Attribut, das in Variante II der Primärschlüssel ist.



Laufzeit

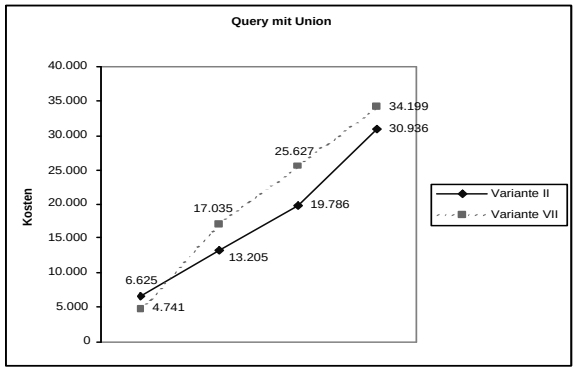
Anzahl Datensätze	Variante II	Variante VII
40.000	1,62 sec	1,22 sec
80.000	4,57 sec	3,49 sec
120.000	7,36 sec	6,04 sec
160.000	7,98 sec	7,75 sec

Abbildung 3.69: Kosten und Laufzeit der Varianten II und VII - Abfrage, mehrere Subklassen in Verbindung mit der Superklasse

Eine Abfrage, die auf mehrere Subklassen in Verbindung mit der Superklasse zugreift, ergibt das Kosten- und Laufzeitverhalten aus Abbildung 3.69. Das Kostendiagramm zeigt einen nahezu linearen Kostenverlauf für beide Varianten, wobei Variante II etwas niedrigere Kosten verursacht, aber eindeutig die höheren Laufzeiten hat, unabhängig von der Anzahl der Datensätze. Sollen also Kosten minimiert werden, empfiehlt sich der Einsatz von Variante II, die Verwendung einer Tabelle pro Subklasse, die die Attribute der Superklasse als Structured Type enthält. Wenn kurze Laufzeiten wichtig sind, sollte Variante VII eingesetzt werden, die Verwendung einer Typhierarchie mit einem nicht-instanzierbaren Root-Typ.

Für Abfragen, die auf mehrere Subklassen in Verbindung mit der Superklasse zugreifen, aber nur wenige Datensätze als Ergebnis zurückliefern, gilt dasselbe wie für Abfragen, die nur auf eine Subklasse zugreifen. Der Kosten- und Laufzeitverlauf von Variante VII steigt mit der Anzahl der Datensätze, während der Kosten- und Laufzeitverlauf von Variante II konstant bleibt. Auch hier kann durch Einsatz eines Index für das Attribut, das in Variante II der Primärschlüssel ist, das Kosten- und Laufzeitverhalten von Variante VII verbessert werden.

Wenn nur Daten der Superklasse, unabhängig von der Zugehörigkeit zu einer Subklasse, benötigt werden, verursacht Variante II zunächst höhere Kosten als Variante VII, wie in Abbildung 3.70 zu sehen ist.



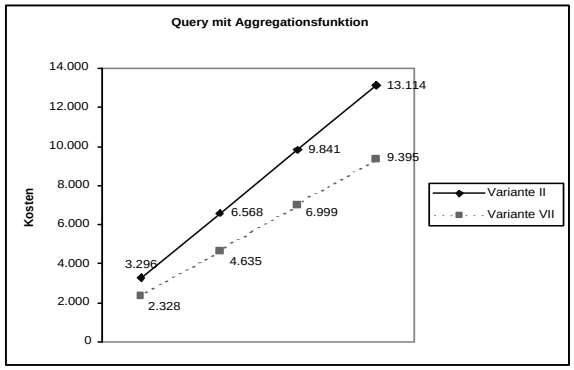
Laufzeit

Anzahl Datensätze	Variante II	Variante VII
40.000	10,35 sec	8,24 sec
80.000	19,61 sec	18,12 sec
120.000	29,38 sec	26,50 sec
160.000	39,37 sec	35,30 sec

Abbildung 3.70: Kosten und Laufzeit der Varianten II und VII - Abfrage auf Attribute der Superklasse

Steigt die Anzahl der Datensätze jedoch, dreht sich dieses Bild um und Variante II verursacht weniger Kosten als Variante VII. Ab einer Größenordnung von ungefähr 50.000 Datensätzen ist daher Variante II bei Kostenminimierung zu bevorzugen. Allerdings weist Variante II durchgehend schlechtere Laufzeiten auf, als Variante VII. Ist Laufzeitminimierung daher das relevante Entscheidungskriterium, ist Variante VII zu empfehlen.

Werden hauptsächlich Abfragen eingesetzt, die eine Aggregationsfunktion beinhalten, ist - wie in Abbildung 3.71 erkenntlich - in jedem Fall Variante VII zu bevorzugen, egal ob die Kosten oder die Laufzeit das ausschlaggebende Kriterium sind.



Laufzeit

Anzahl Datensätze	Variante II	Variante VII
40.000	0,07 sec	0,07 sec
80.000	0,13 sec	0,10 sec
120.000	0,16 sec	0,14 sec
160.000	0,22 sec	0,19 sec

Abbildung 3.71: Kosten und Laufzeit der Varianten II und VII - Abfrage mit Aggregationsfunktion

Variante VII verursacht unabhängig von der Anzahl der Datensätze weniger Kosten als Variante II und hat außerdem kürzere Laufzeiten, daher wird ihr Einsatz bei Abfragen mit Aggregationsfunktionen empfohlen, auch wenn die Unterschiede zwischen den Laufzeiten sehr gering sind.

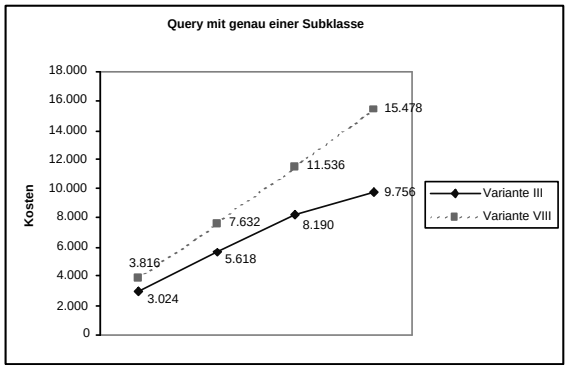
Ein Überblick darüber, welche Variante bei einer bestimmten Kategorie von Abfragen das bessere Kosten- und Laufzeitverhalten aufweist, ist in Abbildung 3.72 zu sehen.

Variante II und VII	Genau eine Subklasse in Verbindung mit der Superklasse		Mehrere Subklassen in Verbindung mit der Superklasse		Nur Superklasse		Abfrage mit Aggregationsfunktion	
	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze
Minimale Kosten	II	II	II	II	VII	II	VII	VII
Minimale Laufzeit	VII	II	VII	VII	VII	VII	VII	VII

Abbildung 3.72: Entscheidungshilfe zur Wahl zwischen Variante II und VII

Variante III und Variante VIII

Wenn überlappende Vererbung nicht erlaubt ist, wird entweder Variante III oder Variante VIII eingesetzt. Variante III ist die Abbildung von Vererbung durch eine Einzelrelation, die sowohl die Attribute der Subklassen als auch die Attribute der Superklasse als Structured Types enthält. Die Zuordnung eines Datensatzes zu einer Subklasse erfolgt mit Hilfe eines relationalen Attributes. Bei Variante VIII erfolgt die Vererbung mit Hilfe einer Typ- und Tabellenhierarchie, wie sie von DB2 angeboten wird. Beim Einsatz von Variante VIII werden nicht nur Attribute von Datentypen vererbt, sondern auch die Daten selbst.



Laufzeit

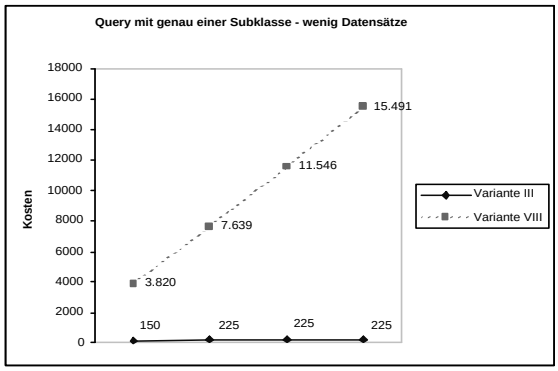
Anzahl Datensätze	Variante III	Variante VIII
40.000	0,08 sec	0,11 sec
80.000	0,28 sec	2,70 sec
120.000	0,56 sec	4,63 sec
160.000	0,92 sec	6,55 sec

Abbildung 3.73: Kosten und Laufzeit der Varianten III und VIII - Abfrage auf genau eine Subklasse in Verbindung mit der Superklasse

Werden primär Abfragen eingesetzt, die auf genau eine Subklasse in Verbindung mit der Superklasse zugreifen, ergibt sich das Kosten- und Laufzeitverhalten aus Abbildung 3.73. Variante III, also die Einzelrelation, verursacht hier geringere Kosten, als der Einsatz der vom DBMS angebotenen Vererbungshierarchie. Auch die Laufzeiten von Variante III sind kürzer. Die Erklärung für dieses Verhalten liegt darin, daß bei Variante III durch die Einzelrelation kein Join zwischen der Subklasse und der Superklasse nötig ist. Variante VIII nützt zwar die Tabellenhierarchie des DBMS, die aber offensichtlich schlechter angesprochen wird, als die relationale Tabelle von Variante III. Dominieren also Abfragen, die auf eine Subklasse in Verbindung mit der Superklasse zugreifen, ist Variante III zu bevorzugen.

Bei Abfragen, die auf eine Subklasse in Verbindung mit der Superklasse zugreifen, aber nur wenig Datensätze als Ergebnis zurückliefern, kommt es zu einem ähnlichen Kosten- und Laufzeitverhalten, wie es beim Vergleich von Variante II und VII zu beobachten war und wie es in Abbildung 3.74 zu sehen ist. Die Kosten und die Laufzeit von Variante III bleiben unabhängig von der Anzahl der Datensätze durch den Index, der auf dem Primärschlüsselattribut definiert

ist, konstant. In der Tabellenhierachie von Variante VIII ist auf diesem Attribut kein Index definiert, da hier der OID anstelle des Primärschlüssel eingesetzt wird. Daher steigen die Kosten und die Laufzeit mit der Anzahl der Datensätze an. Im Gegensatz zu Variante VII, wo durch die Definition eines Index Abhilfe geschafft werden konnte, kann innerhalb einer Tabellenhierarchie auf einer Subtabelle kein Index definiert werden. Als einzige Lösung bleibt die Definition eines Index auf der Superklassen-Tabelle, was sich auf das Verhalten der gesamten Tabellen-Hierachie auswirkt. Ansonsten wird die Verwendung von Variante III empfohlen.



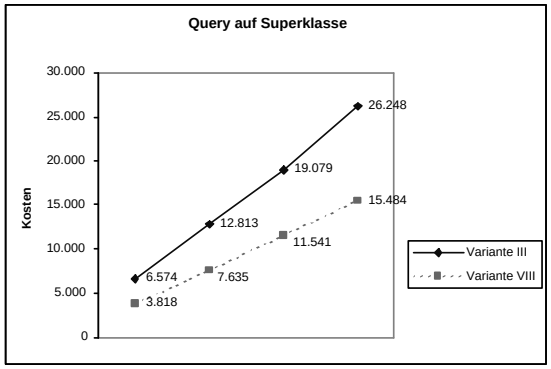
Laufzeit

Anzahl Datensätze	Variante III	Variante VIII
40.000	0,02 sec	0,08 sec
80.000	0,02 sec	0,12 sec
120.000	0,02 sec	0,18 sec
160.000	0,07 sec	0,23 sec

Abbildung 3.74: Kosten und Laufzeit der Varianten III und VIII - Abfrage auf genau eine Subklasse in Verbindung mit der Superklasse, wenig Datensätze als Ergebnis

Da bei den Varianten III und VIII überlappende Vererbung nicht möglich ist, können auch keine Abfragen getätigt werden, die auf mehrere Subklassen in Verbindung mit der Superklasse zugreifen. Die Untersuchung für Abfragen dieser Art entfällt damit.

Abfragen, die nur auf Daten der Superklasse zugreifen, ergeben das in Abbildung 3.75 ersichtliche Kosten- und Laufzeitverhalten.



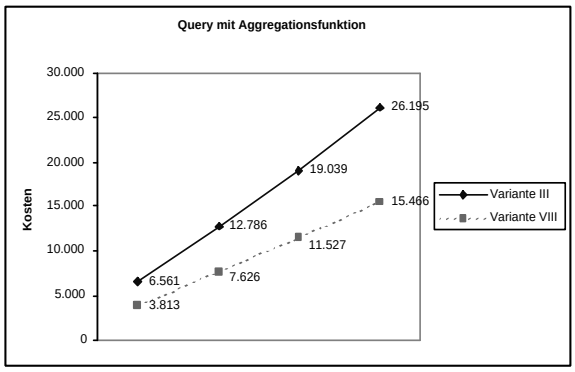
Laufzeit

Anzahl Datensätze	Variante III	Variante VIII
40.000	7,31 sec	8,06 sec
80.000	15,05 sec	17,26 sec
120.000	23,39 sec	26,63 sec
160.000	29,12 sec	34,71 sec

Abbildung 3.75: Kosten und Laufzeit der Varianten III und VIII - Abfrage auf die Superklasse

Hier verursacht Variante VIII eindeutig weniger Kosten, hat aber höhere Laufzeiten. Sollen also die Kosten minimiert werden, ist Variante VIII zu bevorzugen, während bei Laufzeitminimierung Variante III empfehlenswert ist.

Bei Abfragen, die Aggregationsfunktionen beinhalten, ist Variante VIII klar im Vorteil, wie in Abbildung 3.76 zu erkennen ist.



Laufzeit

Anzahl Datensätze	Variante III	Variante VIII
40.000	0,12 sec	0,08 sec
80.000	0,18 sec	0,11 sec
120.000	0,25 sec	0,16 sec
160.000	0,35 sec	0,20 sec

Abbildung 3.76: Kosten und Laufzeit der Varianten III und VIII - Abfrage mit Aggregationsfunktion

Variante VIII - die Tabellenhierarchie des DBMS - weist hier sowohl einen besseren Kostenverlauf, als auch bessere Laufzeiten gegenüber Variante III - Einzelrelation mit Structured Types für Sub- und Superklassen-Attribute - auf. Daher ist sowohl für Kosten- als auch für Laufzeitminimierung bei Abfragen mit Aggregationsfunktionen Variante VIII zu bevorzugen.

Eine Entscheidungshilfe für die Wahl zwischen Variante III und VIII ist in Abbildung 3.77 zu sehen.

Variante III und VIII	Genau eine Subklasse in Verbindung mit der Superklasse		Nur Superklasse		Abfrage mit Aggregationsfunktion	
	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze
Minimale Kosten	III	III	VIII	VIII	VIII	VIII
Minimale Laufzeit	III	III	III	III	VIII	VIII

Abbildung 3.77: Entscheidungshilfe zur Wahl zwischen Variante III und VIII

Erzeugen und Modifizieren von Datensätzen

Wenn nicht Abfragen den Großteil der Operationen auf der Datenbank ausmachen, sondern das Erzeugen und das Ändern von Datensätzen, sollten die Kosten dafür als Entscheidungsgrundlage herangezogen werden. Die Laufzeit ist hier für alle Varianten gleich, kann also vernachlässigt werden. Für die Kosten eines Insert-Statements wurde eine Person, die nur Kunde ist, neu angelegt. Waren dafür mehrere Insert-Statements notwendig, weil mehrere Tabellen betroffen waren, wurden die Kosten addiert. Für eine Update-Operation wurde der Kontostand eines Kunden geändert.

Im Vergleich zwischen Variante I (Vererbung mit getypten Tabellen und Referenzen), Variante IV (Einzelrelation mit Structured Types für die Sub- und Superklassenattribute, sowie für die Zuordnung eines Datensatzes zu einer Subklasse) und Variante VI (Vererbung durch eine Typhierarchie) schneidet Variante I beim Erzeugen von Datensätzen eindeutig am schlechtesten ab, wie in Abbildung 3.78 zu sehen ist.

Variante	Insert	Update
I	50	3.536
IV	25	100
VI	25	9.430

Abbildung 3.78: Kosten eines Insert- und Update-Statements - Variante I, IV und VI

Die hohen Kosten für das Insert-Statement ergeben sich dadurch, daß bei Variante I, im Gegensatz zu den anderen beiden Möglichkeiten, zwei Tabellen vom Erzeugen des Datensatzes betroffen sind. Die Kosten pro Insert-Statement betragen 25, addiert man beide Statements, kommt man auf die doppelten Kosten. Die hohen Update-Kosten für Variante I ergeben sich dadurch, daß zum Suchen des betroffenen Datensatzes eine Referenz aufgelöst werden muß. Variante VI erfordert für ein Update noch höhere Kosten als Variante I, was wieder auf das Fehlen eines Index auf dem Attribut, das in Variante IV den Primärschlüssel stellt, zurückzuführen ist. Um den richtigen Datensatz zu finden, führt das DBMS einen Full-Table-Scan durch. Auch hier kann ein Index Abhilfe schaffen. Dominieren also Insert-Statements ist Variante IV oder VI empfehlenswert. Machen Update-Statements einen Großteil der Operationen aus, sollte entweder Variante IV gewählt werden, oder bei Verwendung von Variante VI ein Index auf dem eindeutigen Attribut definiert werden.

Beim Vergleich von Variante II und Variante VII ergibt sich ein ganz ähnliches Bild, wie beim Vergleich zwischen Variante IV und VI (Abbildung 3.79). Die Kosten für ein Insert-Statement

sind für beide Varianten gleich, die Kosten für ein Update-Statement sind bei Variante VII aus denselben Gründen, die gerade bei Variante VI genannt wurden, signifikant höher.

Variante	Insert	Update
II	25	100
VII	25	9.430

Abbildung 3.79: Kosten eines Insert- und Update-Statements - Variante II und VII

Dominieren also Insert-Statements bei nicht erlaubter partieller Vererbung hat der Designer die freie Wahl zwischen Variante II, eine Tabelle pro Subklasse, die die Superklassenattribute als Structured Type enthält, und Variante VII, der Typhierarchie mit einem nicht instanzierbaren Root-Typ. Für Update-Statements wird der Einsatz von Variante II empfohlen. Wenn Variante VII verwendet werden soll, muß zumindest ein Index auf dem eindeutigen Attribut, das in Variante II der Primärschlüssel ist, definiert werden.

Dasselbe wie für Variante II und VII gilt analog für den Vergleich zwischen Variante III - Einzelrelation mit Structured Types für Sub- und Superklassenattribute, wobei ein einzelnes Attribut für die Zuordnung eines Datensatzes zu einer Subklasse eingesetzt wird - und Variante VIII - Vererbung mittels einer Typ- und Tabellenhierarchie. Die Kosten für Insert- und Update-Statements sind in Abbildung 3.80 zu sehen.

Variante	Insert	Update
III	25	100
VIII	25	15.486

Abbildung 3.80: Kosten eines Insert- und Update-Statements - Variante III und VIII

Beim Erzeugen von Datensätzen sind die beiden Varianten gleichwertig, beim Modifizieren eines Datensatzes ist Variante III zu bevorzugen, oder ein Index auf das eindeutige Attribut in der Root-Tabelle von Variante VIII zu definieren. Auf den Subklassen-Tabellen innerhalb der Tabellenhierarchie von Variante VIII kann kein Index definiert werden. Innerhalb einer Tabellenhierarchie muß ein Index immer auf der Superklassen-Tabelle definiert werden, er wird in weiterer Folge auch für Abfragen auf den Subklassen-Tabellen vom DBMS verwendet.

3.7 Spezialisierung und Generalisierung

Da Spezialisierung und Generalisierung spezielle Formen von Vererbung sind, können die in Abschnitt 3.6 vorgestellten Konzepte ohne Änderungen übernommen werden. Die generelle Klasse entspricht dabei der Superklasse, während die speziellen Klassen als Subklassen zu betrachten sind.

3.8 Aggregation

Aggregation ist neben Vererbung ein weiteres UML-Abstraktionskonzept, das oft verwendet wird um Wissen über die Struktur der Daten implizit auszudrücken. Bei Aggregation handelt es sich um die Ableitung neuer Entitäten, die aus Teilen zusammengesetzt sind. Aggregation wird daher oft als *Is-Part-Of*-Beziehung bezeichnet. Durch das Zusammensetzen dieser Teile entsteht eine neue Entität, ein sogenanntes Aggregat, wie bereits in Abschnitt 3.3.1 beschrieben wurde.

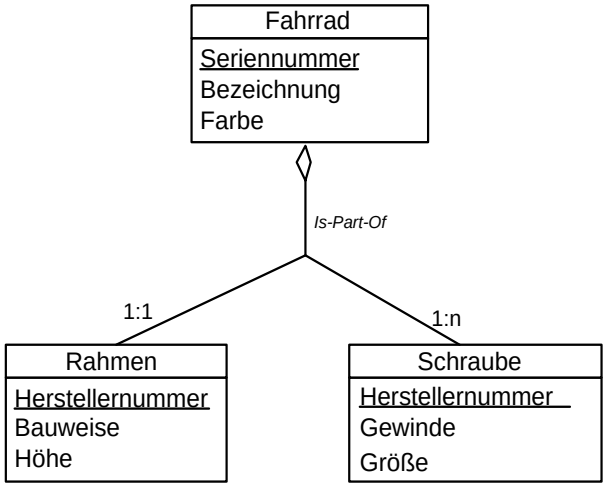


Abbildung 3.81: Aggregationsbeziehung (*Is-Part-Of*-Beziehung)

Das Verhalten und die Behandlung dieses Aggregats wirkt sich dabei immer auch auf seine Teile aus, nicht nur auf das Aggregat selbst. Jeder Teil ist eine eigenständige Entität, meistens allerdings nur eine schwache. Das heißt, wenn das Aggregat gelöscht wird, dann werden in den meisten Fällen auch seine Teile gelöscht. Ein Beispiel für ein solches Aggregat ist ein Fahrrad. Es besteht aus verschiedenen Teilen, die zusammen das Fahrrad ausmachen. Wenn ein Fahrrad fährt, bewegen sich auch seine Teile vorwärts, das heißt sie verhalten sich genauso wie das Aggregat. Die Aggregation “Fahrrad“, die auch schon in Abschnitt 3.3.1 kurz beschrieben

wurde, ist in Abbildung 3.81 noch einmal genau zu sehen. Das Beispiel wird in weiterer Folge zur Erläuterung der Abbildungsmöglichkeiten einer Aggregation in ein objekt-relationales Datenbankschema verwendet.

Ein Fahrrad besteht aus den Teilen “Rahmen“ und “Schraube“. Natürlich besteht ein Fahrrad aus vielen weiteren Teilen, das Beispiel wurde stark vereinfacht. Bei dem Teil “Rahmen“ handelt es sich um eine 1:1-Beziehung, das heißt das Fahrrad hat genau einen Rahmen. Das Fahrrad hat eine beliebige Anzahl an Schrauben, die den Rahmen mit den restlichen Teilen verbinden. Der Rahmen und die Schrauben bilden jeweils eigene Klassen, die durch eine *Is-Part-Of*-Beziehung mit dem Fahrrad verbunden sind. Wird das Fahrrad entfernt, werden damit auch sein Rahmen und seine Schrauben entfernt. Es handelt sich dabei wie bereits erwähnt um schwache Entitäten, die ohne die Entität “Fahrrad“ nicht existieren können. Natürlich gibt es auch Fälle, in denen die Teile auch ohne das Aggregat existieren können, hier handelt es sich dann aber nicht mehr um eine Aggregations-Beziehung, sondern um eine sogenannte Assoziations-Beziehung.

Fahrrad

Seriennummer	Bezeichnung	Farbe

Rahmen

Herstellernummer	Bauweise	Höhe	*Seriennummer*

Schraube

Herstellernummer	Gewinde	Größe	*Seriennummer*

Abbildung 3.82: Relationales Datenmodell - Abbildung einer Aggregationsbeziehung

In einem relationalen Datenmodell wird eine Aggregation gemäß der Abbildungsvorschriften für 1:1- und 1:N-Beziehungen aufgelöst. Für jede Entität wird dabei zuerst eine Tabelle angelegt. Bei einer 1:1-Beziehung wird der Primärschlüssel von einer der beiden Entitäten als Fremdschlüssel in die andere übernommen. Welche Entität dabei den Primärschlüssel liefert, bleibt dem Designer überlassen. Im Fahrrad-Beispiel wird der Primärschlüssel des Aggregats, also der Tabelle “Fahrrad“ in die Tabelle des entsprechenden Teils, die Tabelle “Rahmen“, übernommen. Für 1:N-Beziehungen ist vorgegeben, wie die Beziehung aufgelöst wird: Die Tabelle, die der N-Seite der Beziehung entspricht, muß den Primärschlüssel des Aggregats als Fremdschlüssel

übernehmen. Die Tabelle “Schrauben“ enthält also ein Attribut für den Fremdschlüssel, der aus der Tabelle “Fahrrad“ übernommen wird. Dieses Prinzip ist in Abbildung 3.82 veranschaulicht.

In einem objekt-relationalen Datenmodell hat man nun drei Möglichkeiten, eine Aggregationsbeziehung abzubilden.

Variante I - Teile referenzieren auf das Aggregat: Für das Aggregat und für jede Entität, die einen Teil repräsentiert, wird ein zusammengesetzter Datentypen angelegt, der die Attribute der jeweiligen Entität enthält. Für Entitäten, die an einer 1:1-Beziehung teilnehmen, muß eine ausgewählt werden, die eine Referenz auf die andere Entität enthält.

Abbildung 3.83: Objekt-relationales Modell - Aggregation aufgelöst nach Variante I

Welche hier herangezogen wird, bleibt, wie auch schon bei der relationalen Abbildung, dem Designer überlassen. Im Beispiel enthält wieder der zusammengesetzte Datentyp “Rahmen“ eine Referenz auf den Datentypen “Fahrrad“. Bei Auflösung der 1:N-Beziehung muß der zusammengesetzte Datentyp “Schraube“ das referenzierende Attribut enthalten. Diese Variante hat sehr große Ähnlichkeit mit der relationalen Abbildungsmöglichkeit, anstelle des Primär- und Fremdschlüssel-Konzepts werden aber Referenzen verwendet. In Abbildung 3.83 ist diese Variante graphisch dargestellt.

Ist eine N:M-Beziehung in die Aggregation involviert, wird diese gemäß Abschnitt 3.5 aufgelöst. N:M-Beziehungen sind in einer Aggregation aber sehr selten und werden daher in dieser Arbeit nicht weiter behandelt.

Variante I ist vor allem dann vorteilhaft, wenn Abfragen auf Teile des Aggregats nicht immer gemeinsam mit Abfragen auf das Aggregat selbst gestellt werden. Wenn es also vorkommen kann, daß häufig Abfragen auf Rahmen von Fahrrädern gestellt werden, bei denen die Information, zu welchem Fahrrad der Rahmen gehört, unwichtig ist, ist Variante I sehr gut geeignet.

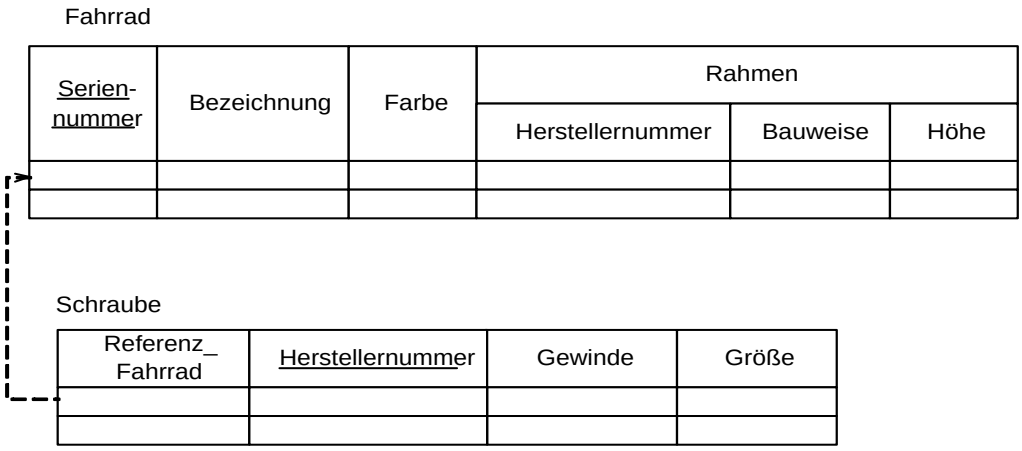


Abbildung 3.84: Objekt-relationales Modell - Aggregation aufgelöst nach Variante II

Variante II - Aggregats-Tabelle enthält Teile einer 1:1-Beziehung: Bei Variante II wird zunächst für jede Teile-Entität, die an einer 1:1-Beziehung teilnimmt, ein zusammengesetzer Datentyp definiert. Danach wird ein zusammengesetzter Datentyp für das Aggregat angelegt, der neben den Attributen des Aggregats ein Attribut für jeden zusammengesetzten Datentypen einer 1:1-Beziehung enthält, der davor für eine Teile-Entität angelegt wurde. Auf diesem Datentyp wird eine Tabelle für das Aggregat definiert. Teile-Entitäten, die durch eine 1:N-Beziehung mit dem Aggregat verbunden sind, referenzieren auf das Aggregat. Dementsprechend werden für diese Entitäten zusammengesetzte Datentypen definiert, die ein Referenz-Attribut auf das Aggregat enthalten. Die Auflösung nach Variante II ist in Abbildung 3.84 zu sehen.

Variante II ist dann vorteilhaft, wenn Teile immer in Verbindung mit dem Aggregat benötigt werden. Wird die Tabelle der Teile in die Aggregation hineingezogen, kann der Join, den man bei Variante I benötigt, gespart werden, zumindest für 1:1-Aggregationsbeziehungen. 1:N-Aggregationsbeziehungen werden weiterhin durch eine Referenz mit dem Aggregat verbunden.

Variante III - Einzelrelation mit mehrwertigen Attributen: Variante III verwendet zur Auflösung der 1:N-Beziehungen zwischen dem Aggregat und seinen Teilen ein mehrwertiges Attribut. Dadurch bleibt Information über die Struktur der Daten erhalten, obwohl man nur

eine einzige Tabelle für das Aggregat definieren muß. Das mehrwertige Attribut kann selbst wieder auf einem zusammengesetzten Datentyp definiert sein. Die resultierende Tabelle bei Anwendung von Variante III ist in Abbildung 3.85 zu sehen.

Fahrrad

Serien-nummer	Bezeichnung	Farbe	Rahmen			Schraube		
			Hersteller-nummer	Bauweise	Hoehe	Hersteller-nummer	Gewinde	Groesse

Abbildung 3.85: Objekt-relationales Modell - Aggregation aufgelöst nach Variante III

Variante III ist gut geeignet, wenn Informationen über Teile immer in Verbindung mit dem Aggregat benötigt werden. Die Zusammengehörigkeit der Daten ist klar erkenntlich, Information über die Struktur der Daten bleibt erhalten.

Variante III benötigt prinzipiell weniger logische Zugriffe als Variante II und ist daher zu bevorzugen, wenn Teile einer 1:N-Beziehung in Verbindung mit dem Aggregat benötigt werden. Da die physische Speicherung mehrwertiger Attribute aber von den DBMS ganz unterschiedlich umgesetzt wird, was zur Folge hat, daß trotz geringer logischer Zugriffsanzahl ein schlechtes Performance-Verhalten auftreten kann, sollte auf jeden Fall zur Entscheidungshilfe eine Performance-Analyse herangezogen werden.

3.8.1 Aggregation in Oracle

In Oracle können alle drei Varianten realisiert werden, da Oracle sowohl Referenzen und zusammengesetzte Datentypen als auch mehrwertige Attribute unterstützt. In diesem Abschnitt soll gezeigt werden, wie die drei Varianten in Oracle in ein logisches Schema abgebildet werden.

Varianten I

Um Aggregation mit Variante I in Oracle abzubilden, müssen zunächst Datentypen für die später zu definierenden Object Tables angelegt werden:

```
CREATE OR REPLACE TYPE varianteI_fahrrad_typ AS OBJECT (
            seriennummer      CHAR(11),
            bezeichnung       VARCHAR(30),
```

```
            farbe              VARCHAR(20)
);
/
CREATE OR REPLACE TYPE varianteI_rahmen_typ AS OBJECT (
            herstellernummer      CHAR(11),
            bauweise              VARCHAR(20),
            hoehe                 NUMBER(5,2),
            ref_fahrrad           REF varianteI_fahrrad_typ
);
/
CREATE OR REPLACE TYPE varianteI_schraube_typ AS OBJECT (
           herstellernummer    CHAR(11),
           gewinde             CHAR(1),
           groesse             NUMBER(5,2),
           ref_fahrrad         REF varianteI_fahrrad_typ
);
/
```

Auf jedem dieser Datentypen wird eine Object Table definiert, wobei jeweils ein Primärschlüssel anzugeben ist. Für die referenzierenden Attribute muß angegeben werden, auf welche Tabelle referenziert wird.

```
CREATE TABLE varianteI_fahrrad OF varianteI_fahrrad_typ (
                       PRIMARY KEY (seriennummer)
);

CREATE TABLE varianteI_rahmen OF varianteI_rahmen_typ (
                       PRIMARY KEY (herstellernummer),
                       ref_fahrrad REFERENCES varianteI_fahrrad
);

CREATE TABLE varianteI_schraube OF varianteI_schraube_typ (
                       PRIMARY KEY (herstellernummer),
                       ref_fahrrad REFERENCES varianteI_fahrrad
);
```

Variante I ist dann gut geeignet, wenn die Teile sehr oft unabhängig vom Aggregat oder das Aggregat unabhängig von den Teilen gebraucht wird. Es ist erkenntlich, daß zwischen dem Ag-

gregat und den Teilen eine Beziehung vorhanden ist, trotzdem kann unabhängig von dieser Beziehung sowohl auf das Aggregat als auch die Teile zugegriffen werden. Wird das Aggregat allerdings in Verbindung mit den Teilen gebraucht, weist diese Variante eine schlechtere Performanz auf, da die Referenz implizit in einen Join umgewandelt werden muß, wenn zu den Teilen auch Informationen über das Aggregat gebraucht werden. In diesem Fall ist Variante II besser geeignet.

Variante II

Bei Variante II wird zunächst ein zusammengesetzter Datentyp für den Rahmen eines Fahrrades angelegt. Dieser wird in weiterer Folge als zugrundeliegender Datentyp für ein Attribut im zusammengesetzten Datentyp verwendet, der das Fahrrad repräsentiert. Ein zusammengesetzter Datentyp für die Schrauben mit einer Referenz auf eine Instanz eines Fahrrads wird ebenfalls benötigt.

```
CREATE OR REPLACE TYPE varianteII_rahmen_typ AS OBJECT (
        herstellernummer     CHAR(11),
        bauweise             VARCHAR(20),
        hoehe                NUMBER(5,2)
);
/
CREATE OR REPLACE TYPE varianteII_fahrrad_typ AS OBJECT (
         seriennummer         CHAR(11),
         bezeichnung          VARCHAR(50),
         farbe                VARCHAR(20),
         rahmen               varianteII_rahmen_typ
);
/
CREATE OR REPLACE TYPE varianteII_schraube_typ AS OBJECT (
        herstellernummer      CHAR(11),
        gewinde               CHAR(1),
        groesse               NUMBER(5,2),
        ref_fahrrad           REF varianteII_fahrrad_typ
);
/
```

Auf dem zusammengesetzten Datentyp *varianteII_fahrrad_typ* wird eine Object Table definiert, ebenso wie für den zusammengesetzten Datentyp *varianteII_schraube_typ*. Dabei wird je-

weils der Primärschlüssel angegeben. Das Attribut *rahmen.herstellernummer*, das in Variante I Primärschlüssel der Tabelle "rahmen" war, muß hier als UNIQUE deklariert werden, damit es eindeutig bleibt. Für das referenzierende Attribut der Tabelle *VarianteII_schraube* muß angegeben werden, auf welche Tabelle referenziert wird.

```
CREATE TABLE varianteII_fahrrad OF varianteII_fahrrad_typ (
                        PRIMARY KEY (seriennummer),
                        CONSTRAINT rahmen_herstellernummer UNIQUE (rahmen.herstellernummer)
);

CREATE TABLE varianteII_schraube OF varianteII_schraube_typ (
                        PRIMARY KEY (herstellernummer),
                        ref_fahrrad REFERENCES varianteII_fahrrad
);
```

Variante II ist vor allem dann gut geeignet, wenn Teile immer in Verbindung mit dem Aggregat benötigt werden. Der Join wird sozusagen in die Tabelle eingebaut, was Zeit und Kosten spart. Möglich ist diese Auflösung allerdings nur für 1:1-Beziehungen. Nur diese können als zusammengesetzter Datentyp in die Fahrrad-Tabelle übernommen werden. Für 1:N-Beziehungen muß weiterhin eine eigene Object Table definiert werden, die auf die Aggregatstabelle referenziert.

Variante III

Der Nachteil von Variante II - die Auflösung der 1:N-Beziehung als eigene Object Table - kann in Oracle mit Hilfe von Nested Tables umgangen werden. Dabei wird die Tabelle auf der N-Seite der Aggregationsbeziehung als Nested Table in die Aggregats-Tabelle eingebettet. Was man dadurch erreicht ist, daß die Zusammengehörigkeit der Daten auch für mehrwertige Attribute erhalten bleibt. Zuerst müssen wiederum Datentypen definiert werden, die in späterer Folge beim Erstellen der Tabellen verwendet werden können:

```
CREATE OR REPLACE TYPE varianteIII_rahmen_typ AS OBJECT (
        herstellernummer     CHAR(11),
        bauweise             VARCHAR(20),
        hoehe                NUMBER(5,2)
);
/
CREATE OR REPLACE TYPE varianteIII_schraube_typ AS OBJECT (
        herstellernummer      CHAR(11),
        gewinde               CHAR(1),
```

```
        groesse                 NUMBER(5,2)
);
/
CREATE OR REPLACE TYPE varianteIII_schraube_table AS TABLE OF varianteIII_schraube_typ;
/
```

Es reicht nicht, einen zusammengesetzten Datentyp zu definieren, der die Schrauben repräsentiert. Dem DBMS muß auch mitgeteilt werden, daß es sich hier eigentlich um eine Tabelle handelt, die mehrere Instanzen dieses Datentyps enthält. Eine Object Table kann dafür nicht eingesetzt werden, da diese nicht als zugrundeliegender Datentyp für ein Attribut einer Tabelle verwendet werden kann. Was man benötigt ist also ein Datentyp, der eine Tabelle repräsentiert. Im Beispiel entspricht dieser Datentyp dem zusammengesetzten Datentyp *varianteIII_schraube_table*. Der Datentyp kann nun als zugrundeliegender Datentyp von Tabellen-Attributen, Object Tables oder Attributen anderer zusammengesetzter Datentypen verwendet werden.

```
CREATE OR REPLACE TYPE varianteIII_fahrrad_typ AS OBJECT (
         seriennummer           CHAR(11),
         bezeichnung            VARCHAR(50),
         farbe                  VARCHAR(20),
         rahmen                 varianteIII_rahmen_typ,
         schrauben              varianteIII_schraube_table
);
```

Nun wird noch eine Object Table benötigt, die auf dem Typ *varianteIII_fahrrad_typ* basiert.

```
CREATE TABLE varianteIII_fahrrad OF varianteIII_fahrrad_typ (
             PRIMARY KEY (seriennummer),
             CONSTRAINT varIII_rahmen_herstnr UNIQUE (rahmen.herstellernummer)
) NESTED TABLE schrauben STORE AS schrauben_table;
```

Bei der Erstellung der Tabelle, die die Nested Table enthält, muß angegeben werden, in welcher physischen Tabelle die Daten der Nested Table gespeichert werden sollen. Das DBMS legt diese Tabelle an und verwaltet sie intern wie jede andere Tabelle auch. Allerdings kann auf diese Tabelle nicht direkt in Abfragen zugegriffen werden, Zugriffe müssen immer über die Elterntabelle erfolgen. Wenn ein Datensatz eingefügt werden soll, muß ein Statement folgender Art verwendet werden:

```
INSERT INTO varianteIII_fahrrad VALUES (
            '234',
            'Rennrad',
            'gruen',
             varianteIII_rahmen_typ(
                        '456',
                        'ALU leicht',
                        102
             ),
             varianteIII_schraube_table(
                varianteIII_schraube_typ(
                                  '789',
                                  'M',
                                   2)
                        )
);
```

Dieses Statement sieht genauso aus, wie das Insert-Statement einer Tabelle, deren Attribute auf zusammengesetzten Datentypen definiert sind. Das Attribut, das die Nested Table repräsentiert, wird genauso angesprochen. Zuerst wird der Datentyp des Attributs angegeben, hier *varianteIII_schraube_table*. Innerhalb dieses Datentyps muß der Datentyp angegeben werden, auf dem die Nested Table basiert. Danach können Werte definiert werden. Beim Insert-Statement muß sozusagen der Aufbau der Typ-Hierarchie noch einmal nachgebaut werden. Zusätzlich hat man die Möglichkeit, nachträglich Datensätze bei bestehender Instanz der Elterntabelle nur in die Nested Table dieser Instanz einzufügen, ohne daß die restlichen Attribute der Instanz davon betroffen sind. Dazu muß folgendes Statement verwendet werden:

```
INSERT INTO TABLE(
          SELECT f.schrauben
          FROM varianteIII_fahrrad f
          WHERE f.seriennummer = '234')
       VALUES ('888', 'Z', 4);
```

Mit Hilfe der TABLE-Klausel wird die Nested Table einer bestimmten Instanz ausgewählt, in die Werte eingefügt werden sollen. Ein Update-Statement wird genau gleich aufgebaut.

Abfragen auf eine Nested Table erfolgen mit einem ähnlichen Statement, wie es in Abschnitt 3.6.3 bei Erläuterung von Variante V für das Unnesting eines VARRAYs verwendet wurde:

```
SELECT p.*
FROM varianteIII_fahrrad f, TABLE(f.schrauben) p
WHERE p.herstellernummer = '789';
```

Bei der Verwendung von Nested Tables ist allerdings eine Einschränkung vorhanden: Table Nesting kann nicht in beliebig vielen Stufen erfolgen, das heißt Attribute einer Nested Table dürfen als Datentyp keinen weiteren Nested-Table- oder VARRAY-Datentyp verwenden. Die Schachtelung darf nur eine Stufe tief sein.

Variante III ist nicht geeignet wenn Teile oft unabhängig vom Aggregat gebraucht werden. Bei Abfragen auf die Nested Table führt das DBMS jedesmal implizit einen Join zwischen der Elterntabelle und der physischen Tabelle durch, in der die Daten der Nested Table gespeichert sind. Die Durchführung eines Joins wirkt sich immer negativ auf Performanz und Kosten einer Abfrage aus, daher sollte Variante III nur verwendet werden, wenn Abfragen ohnehin immer gemeinsam auf das Aggegrat und seine Teile gestellt werden.

Damit sind alle Varianten, die Oracle zur Abbildung von Aggregation anbietet, besprochen. Im nächsten Abschnitt wird das Performanz-Verhalten der drei Varianten untersucht.

3.8.2 Aggregation in Oracle - Performanceanalyse

Name/Kurzbeschreibung	Häufigkeit	Typ (Online/Batch)
O1: Abfragen von Rahmen (ohne Fahrrad)	5x/Tag	OL
O2: Abfragen von Fahrrädern und den dazugehörigen Schrauben	6x/Tag	OL
O3: Abfragen von Fahrrädern und dem dazugehörigen Rahmen	7x/Tag	OL
O4: Abfragen von Fahrrädern mit Gesamtzubehör	15x/Tag	OL
O5: Anlegen eines Fahrrades mit Zubehör	6x/Tag	OL
O6: Update einer Schraube	1x/Tag	OL
O7: Update eines Rahmens	2x/Tag	OL
O8: Löschen von Fahrrädern und ihrem Zubehör	1x/Woche	B

Abbildung 3.86: Operationshäufigkeitstabelle einer Aggregation

Die gerade vorgestellten Varianten zur Abbildung einer Aggregation wurden im Hinblick auf ihre Kosten und ihre Laufzeit getestet. Dabei wurde die Operationshäufigkeitstabelle aus

Abbildung 3.86 zur Bestimmung der Operationen herangezogen.

Getestet wurden die Online-Transaktionen, also O1 bis O7. Das verwendete Mengengerüst ist in Abbildung 3.87 zu sehen.

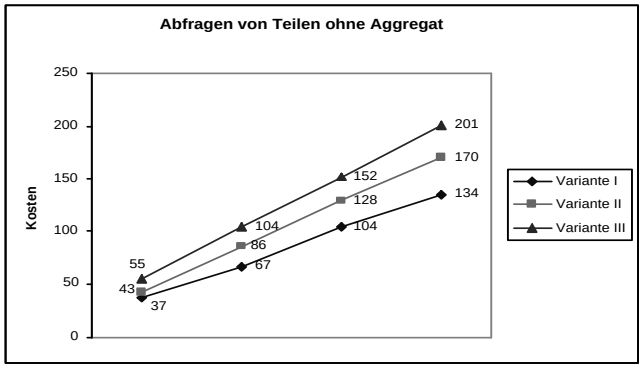
Konzept	Typ	Anzahl
Fahrrad	E	40.000
Schraube	E	120.000
Rahmen	E	40.000
seriennummer	A	40.000
bezeichnung	A	40.000
farbe	A	40.000
herstellernummer (rahmen)	A	40.000
bauweise	A	40.000
hoehe	A	40.000
herstellernummer (schraube)	A	120.000
gewinde	A	120.000
groesse	A	120.000

Abbildung 3.87: Mengengerüst einer Aggregation

Ausgegangen wird von einer Aggregatsanzahl von 40.000 Datensätzen, das entspricht im Beispiel 40.000 Fahrrädern. Zu jedem Fahrrad gehört ein Rahmen, damit wird eine 1:1-Beziehung zwischen dem Aggregat und seinen Teilen realisiert. Jedem Fahrrad sind außerdem drei Schrauben zugeordnet, was die 1:N-Beziehung zwischen Aggregat und Teil darstellt. Das Mengengerüst wurde in vier Schritten um jeweils 40.000 Aggregate erweitert, bis eine Obergrenze von 160.000 Datensätzen erreicht war. Die Abfragen O1 bis O4 wurden auf den verschiedenen Mengengerüsten durchgeführt, die benötigten Kosten und die Laufzeit wurde aufgezeichnet. Danach wurden Kosten und Laufzeiten für alle drei Varianten miteinander verglichen. Die Analyse wurde auf einem Athlon XP 1800+ mit 256 MB RAM und Windows 2000 durchgeführt.

Für Abfrage O1, die ein Teil, das durch eine 1:1-Beziehung mit dem Aggregat verbunden ist, unabhängig von der Zugehörigkeit zum Aggregat zurückliefert, ergibt sich das Kosten- und Laufzeitverhalten aus Abbildung 3.88. Im Diagramm ist erkennbar, daß Variante I die niedrigsten Kosten benötigt. Diese Tatsache erklärt sich dadurch, daß in Variante I nur die Tabelle, die die Datensätze des Teils beinhaltet, abgefragt werden muß. Diese Tabelle hat ein eigenes Schlüsselattribut zur Identifizierung der Datensätze. In Variante II und III sind die Datensätze des Teils als zusammengesetztes Attribut in die Tabelle des Aggregats eingebettet, der Primärschlüssel der Aggregatstabelle liegt zudem auf einem anderen Attribut, als der Primärschlüssel der Teile-

Tabelle von Variante I. Bei der Abfrage von Daten, die nur Teile betreffen, unabhängig vom Aggregat wird daher ein Full-Table-Scan durchgeführt, der die hohen Kosten erklärt. Hier kann ein Index Abhilfe schaffen.



Laufzeit

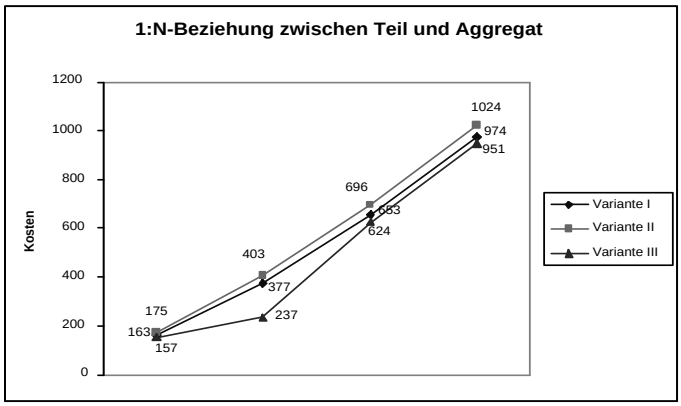
Anzahl Datensätze	Variante I	Variante II	Variante III
40.000	0,04 sec	0,04 sec	0,02 sec
80.000	0,06 sec	0,07 sec	0,07 sec
120.000	1,0 sec	1,01 sec	1,02 sec
160.000	1,07 sec	1,02 sec	1,03 sec

Abbildung 3.88: Kosten und Laufzeiten - Abfrage von Teilen ohne Aggregat

Die Laufzeiten sind für alle drei Varianten nahezu identisch, es handelt sich hier um Unterschiede im Bereich von ein bis vier Millisekunden. Bei den Laufzeiten ist also für eine Abfrage, die ein Teil unabhängig vom Aggregat benötigt, keine eindeutige Entscheidung zugunsten einer der Varianten zu treffen. Sollen allerdings die Kosten minimiert werden, ist auf jeden Fall Variante I zu empfehlen.

Eine Abfrage, die Teile einer 1:N-Beziehung in Verbindung mit dem Aggregat benötigt, also O2, ergibt das Kosten- und Laufzeitverhalten, das in Abbildung 3.89 zu sehen ist. Variante III benötigt hier die niedrigsten Kosten, der Unterschied zu den anderen beiden Varianten wird aber mit der Anzahl der Datensätze geringer. Variante II hat die höchsten Kosten. Das DBMS kann offensichtlich die physische Speichertabelle der Nested Table effizient mit der Aggregatstabelle joinen, während es für die Auflösung der Referenzen etwas höhere Kosten benötigt. Die Laufzeiten unterscheiden sich auch bei O2 nur minimal, wobei allerdings Variante I ab einer Größenordnung von 160.000 Datensätze eine Sekunde länger benötigt als Variante II und III. Bei Variante III ist zu beobachten, daß die Laufzeit bei den Größenordnungen 80.000 und 120.000 Datensätze nahezu gleich bleibt. Bei 160.000 Datensätzen ist die Laufzeit wieder dieselbe wie bei Variante II. Auch hier kann das DBMS offensichtlich bis zu einer bestimmten Größenordnung die Daten effizient aufbereiten und bereitstellen. Wenn die Laufzeit minimiert werden soll, ist bis zu einer Größenordnung von 160.000 Datensätzen Variante III, die Abbildung der 1:N-Beziehung durch eine Nested Table, zu empfehlen. Variante II und I haben zwar nahezu dieselben Laufzeiten, aber dafür ein schlechteres Kostenverhalten. Wird diese Größenordnung überschritten, ist die Auswahl zwischen Variante II und Variante III zu treffen, wobei auch hier

Variante III wegen der niedrigeren Kosten zu bevorzugen ist. Ist ausschließlich die Minimierung der Kosten relevant ist in jedem Fall Variante III einzusetzen.

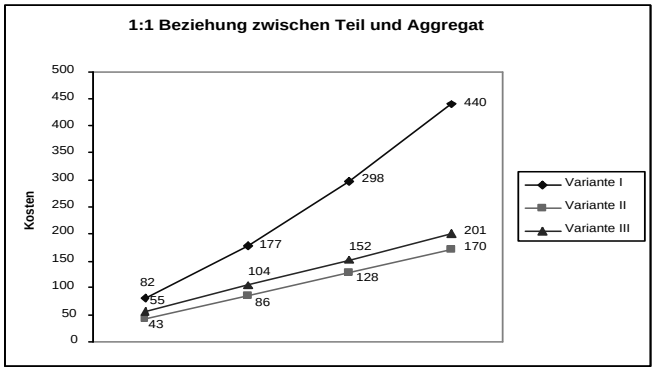


Laufzeit

Anzahl Datensätze	Variante I	Variante II	Variante III
40.000	1,03 sec	1,05 sec	1,03 sec
80.000	2,03 sec	2,0 sec	2,03 sec
120.000	3,03 sec	3,05 sec	2,07 sec
160.000	4,04 sec	3,07 sec	3,05 sec

Abbildung 3.89: Kosten und Laufzeiten - Abfragen von Teilen und Aggregat, die durch eine 1:N-Beziehung verbunden sind

Abfragen, die Teile einer 1:1-Beziehung in Verbindung mit dem Aggregat zurückliefern, wie O3, ergeben im Vergleich zu Teilen einer 1:N-Beziehung ein sehr unterschiedliches Kosten- und Laufzeitverhalten, wie in Abbildung 3.90 erkennbar ist.



Laufzeit

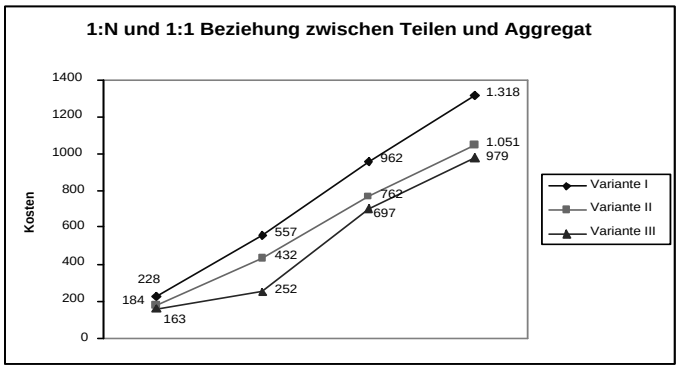
Anzahl Datensätze	Variante I	Variante II	Variante III
40.000	0,03 sec	0,05 sec	0,04 sec
80.000	0,08 sec	0,09 sec	0,08 sec
120.000	1,06 sec	1,01 sec	1,02 sec
160.000	1,04 sec	1,03 sec	1,04 sec

Abbildung 3.90: Kosten und Laufzeiten - Abfragen von Teilen und Aggregat, die durch eine 1:1-Beziehung verbunden sind

Hier benötigt, wie dem Diagramm entnommen werden kann, Variante II die niedrigsten Kosten. Variante III hat etwas höhere Kosten, während Variante I deutlich mehr Kosten benötigt. Die Erklärung ergibt sich durch die Aufnahme des Teils als zusammengesetzter Datentyp in die Aggregatstabelle, wie es bei Variante II und Variante III der Fall ist. Dadurch ist der Join, beziehungsweise die Auflösung einer Referenz, die bei Variante I die hohen Kosten verursacht,

nicht mehr nötig. Die etwas höheren Kosten von Variante III ergeben sich dadurch, daß die Aggregats-Tabelle hier noch das Attribut für die Nested Table beinhaltet, es müssen also etwas mehr Daten pro Datensatz gelesen werden, auch wenn die Nested Table nicht verwendet wird. Die Kostenunterschiede, die sich daraus ergeben sind allerdings sehr gering. Die Laufzeiten sind bei einer Abfrage, die Teile und Aggregat einer 1:1-Beziehung benötigt, nahezu identisch. Die Unterschiede liegen im Bereich von ein bis vier Millisekunden, wie der Abbildung entnommen werden kann und sind damit vernachlässigbar. Sowohl bei Laufzeitminimierung als auch bei Kostenminimierung ist daher Variante II zu empfehlen, da diese die geringsten Kosten benötigt.

O4 repräsentiert eine Abfrage, bei der sowohl Teile einer 1:N-Beziehung, als auch Teile einer 1:1-Beziehung in Verbindung mit dem Aggregat benötigt werden. Das Kosten- und Laufzeitverhalten von O4 ist in Abbildung 3.91 zu sehen.



Laufzeit

Anzahl Datensätze	Variante I	Variante II	Variante III
40.000	1,04 sec	1,05 sec	1,02 sec
80.000	2,05 sec	2,04 sec	2,0 sec
120.000	3,09 sec	3,04 sec	2,04 sec
160.000	4,0 sec	3,08 sec	3,0 sec

Abbildung 3.91: Kosten und Laufzeiten - Abfragen von Teilen und Aggregat, die sowohl durch eine 1:N- als auch durch eine 1:1-Beziehung verbunden sind

Hier benötigt Variante III die niedrigsten Kosten, dicht gefolgt von Variante II, während Variante I sich wieder deutlich abhebt. Die Gründe dafür sind dieselben, die bei O2 und O3 angeführt wurden. In Variante I muß für alle Teile eine Referenz aufgelöst werden, was eine sehr teuere Operation ist. Bei Variante II muß die Auflösung der Referenz nur für die 1:N-Beziehung aufgelöst werden, die Teile der 1:1-Beziehung sind durch ein zusammengesetztes Attribut in der Aggregatstabelle enthalten. Auch Variante III enthält die Teile der 1:1-Beziehung als zusammengesetztes Attribut, die Teile der 1:N-Beziehung sind hier als Nested Table eingebettet. Diese kann im Vergleich zur Auflösung einer Referenz effizienter mit der Aggregatstabelle verbunden werden, was zu den niedrigeren Kosten führt. Auch bei den Laufzeiten ist Variante III die beste Wahl, vor allem bei Größenordnung ab 120.000 Datensätzen. Darunter sind die Laufzeiten der einzelnen Varianten nahe identisch, mit vernachlässigbaren Unterschieden. Ab einer Größenordnung von 160.000 Datensätzen weist Variante I die höchsten Laufzeiten vor, Variante

II und Variante III liegen nahezu gleichauf, mit etwas geringeren Laufzeiten bei Variante III. Es wird empfohlen, sowohl bei Kosten- als auch bei Laufzeitminimierung Variante III zu verwenden.

Ein Überblick darüber welche Variante bei Kosten- und Laufzeitminimierung im Hinblick auf verschiedenen Größenordnungen gewählt werden soll, ist in Abbildung 3.92 zu sehen.

Variante I, II und III	Teile ohne Aggregat		Teile einer 1:N-Beziehung in Verbindung mit Aggregat		Teile einer 1:1-Beziehung in Verbindung mit dem Aggregat		Teile einer 1:1- und einer 1:N-Beziehung in Verbindung mit dem Aggregat	
	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze
Minimale Kosten	I	I	III	III	II	II	III	III
Minimale Laufzeit	I, II, III	I,II,III	III	II, III	I, II, III	I, II, III	III	II, III

Abbildung 3.92: Entscheidungshilfe zur Wahl zwischen Variante I, II und III

Erzeugen und Modifizieren von Datensätzen

Wenn nicht Abfragen auf das Aggregat und seine Teile den Großteil der Operationen ausmachen, sondern die Neuanlage und die Modfikation von Datensätzen, muß die Auswahl anhand der Kosten für Insert- und Update-Statements getroffen werden. Das Kostenverhalten der drei Varianten ist in Abbildung 3.93 zu sehen. Die Laufzeiten unterscheiden sich für die drei Varianten nicht.

Variante	Insert	Update (Rahmen)	Update (Schraube)
I	128	4	4
II	179	3	4
III	966	3	255

Abbildung 3.93: Kosten von Insert- und Update-Statements

Beim Erzeugen von Datensätzen wurde ein Fahrrad mit einem Rahmen und drei Schrauben angelegt. Dazu werden mehrere Insert-Statements benötigt, deren Kosten addiert wurden. Variante I benötigt zur Neuanlage eines Datensatzes eindeutig am wenigsten Kosten und ist daher zu bevorzugen, wenn hauptsächlich neue Datensätze angelegt werden. Variante III hat hier sehr viel schlechtere Kosten, als die anderen beiden Varianten, was daher rührt, daß erst die passende Nested Table für den Aggregatsdatensatz gesucht werden muß, in die die Teile eingefügt werden können. Bei einer Modifikation von Teilen einer 1:1-Beziehung benötigt Variante I um eine Kosteneinheit mehr als Variante II und III, dieser Unterschied ist eigentlich vernachlässigbar. Hier spielt keine Rolle, welche Variante gewählt wird. Erst bei der

Modifikation von Datensätzen einer 1:N-Beziehung ist Variante III wieder klar im Nachteil. Auch hier gilt, daß für das Update der Teile erst die passende Nested Table des Aggregats gesucht werden muß, während bei Variante I und II das spezielle Teil, das modifiziert werden soll, aus der Teile-Tabelle gefiltert wird, was nur die Auflösung einer einzigen Referenz nach sich zieht. Wenn also hauptsächlich Datensätze modifiziert werden, ist die Wahl zwischen Variante I und II zu treffen, Variante III ist hier auf gar keinen Fall einzusetzen.

Die Performance-Studie hat gezeigt, daß es eine wesentliche Rolle spielt, welche Art von Abfragen an die Tabellen gestellt werden. Dabei ist zunächst wichtig, ob die Teile in Verbindung mit dem Aggregat benötigt werden oder nicht, es ist aber auch entscheidend, ob die Teile mittels einer 1:1-Beziehung oder einer 1:N-Beziehung mit dem Aggregat verbunden sind. Es hängt damit primär von der Struktur der Daten ab, welche der drei Varianten gewählt wird. Entscheidend ist auch, ob lesend oder schreibend auf die Datensätze zugegriffen wird. Bei überwiegenden Schreibzugriffen, ist eine andere Entscheidung zu treffen, als bei Lesezugriffen. Letztendlich spielt es auch eine Rolle ob Kosten oder Laufzeit miniert werden sollen.

Auch hier ist wie schon in Abschnitt 3.6.4 hinzuzufügen, daß bei ausreichendem Budget und Zeitrahmen eine derartige Performance-Analyse für jedes Projekt mit dem genauen Mengengerüst und dem Transaktionsprofil des Projektes durchgeführt werden sollte, um ein möglichst optimales Ergebnis zu erzielen. Auch physischer Entwurf und Tuning können die Ergebnisse einer Performance-Analyse erheblich beeinflußen, worauf in dieser Arbeit keine Rücksicht genommen wurde.

3.8.3 Aggregation in DB2

In DB2 können nur zwei der Varianten zur Aggregationsauflösung verwendet werden, da DB2 keine Unterstützung für mehrwertige Attribute anbietet. Variante III kann damit in DB2 nicht realisiert werden. Zusammengesetzte Datentypen und Referenzen werden allerdings unterstützt, Variante I und Variante II können daher verwendet werden.

Variante I

Um Aggregation nach Variante I aufzulösen, müssen zunächst zusammengesetzte Datentypen für alle an der Aggregationsbeziehung beteiligten Klassen angelegt werden.

```
CREATE TYPE varI_fahrrad_typ AS (
            seriennummer       CHAR(11),
            bezeichnung        VARCHAR(30),
```

```
            farbe               VARCHAR(20)
)
REF USING INT
MODE DB2SQL;

CREATE TYPE varI_rahmen_typ AS (
            herstellernummer    CHAR(11),
            bauweise            VARCHAR(20),
            hoehe               DECIMAL(5,2),
            ref_fahrrad         REF(varI_fahrrad_typ)
)
REF USING INT
MODE DB2SQL;

CREATE TYPE varI_schraube_typ AS (
            herstellernummer    CHAR(11),
            gewinde             CHAR(1),
            groesse             DECIMAL(5,2),
            ref_fahrrad         REF(varI_fahrrad_typ)
)
REF USING INT
MODE DB2SQL;
```

Da in DB2 nur 18 Zeichen für einen Objekt-Name erlaubt sind (das betrifft sowohl Tabellen als auch Constraints, Trigger und Datentypen), müssen die Typnamen entsprechend gewählt werden. Weiters wird für jeden Datentyp angegeben, mit welchem Basisdatentyp eine Referenz dargestellt werden soll. Im Beispiel wird dafür immer Integer verwendet. Auf diesen Datentypen werden nun Typed Tables angelegt.

```
CREATE TABLE varianteI_fahrrad OF varI_fahrrad_typ (
             REF IS oid USER GENERATED,
             oid WITH OPTIONS UNIQUE,
             seriennummer WITH OPTIONS NOT NULL PRIMARY KEY
     );

CREATE TABLE varianteI_rahmen OF varI_rahmen_typ (
             REF IS oid USER GENERATED,
             herstellernummer WITH OPTIONS NOT NULL PRIMARY KEY,
```

```
            ref_fahrrad WITH OPTIONS SCOPE varianteI_fahrrad
                        REFERENCES varianteI_fahrrad(oid)
    );

CREATE TABLE varianteI_schraube OF varI_schraube_typ (
            REF IS oid USER GENERATED,
            herstellernummer WITH OPTIONS NOT NULL PRIMARY KEY,
            ref_fahrrad WITH OPTIONS SCOPE varianteI_fahrrad
                        REFERENCES varianteI_fahrrad(oid)
    );
```

Für jede Typed Table muß die Spalte angegeben werden, die den OID enthalten soll. Diese darf keine der Spalten sein, die durch den zusammengesetzten Datentypen definiert werden. Wenn von anderen Tabellen aus auf diese Spalte referenziert werden soll, muß außerdem ein UNIQUE-Constraint angegeben werden. Zusätzlich wird für jede Tabelle ein Primärschlüssel definiert. Den referenzierenden Attributen in den Tabellen "varianteI_rahmen" und "varianteI_schraube" muß ein Scope zugewiesen werden, weiters muß die Spalte der referenzierten Tabelle bekanntgegeben werden, die den OID enthält.

Auch in DB2 ist Variante I vor allem dann gut geeignet, wenn das Aggregat nicht in Abfragen miteinbezogen wird, die Teile der Aggregationsbeziehung betreffen.

Variante II

Um sich bei Abfragen den Join zwischen den teilnehmenden Entitäten einer 1:1-Aggregationsbeziehung zu sparen, kann in DB2 Variante II eingesetzt werden. Dabei wird zunächst ein zusammengesetzter Datentyp für jede Teile-Entität angelegt, die Teil einer 1:1-Aggregationsbeziehung ist. In den zusammengesetzten Datentypen, der das Aggregat repräsentiert, wird für jeden definierten Datentypen der Teile-Entitäten ein Attribut aufgenommen, das auf diesem Datentyp aufbaut. Für 1:N-Beziehungen wird zur Abbildung Variante I verwendet, das heißt, für jede Teile-Entität, die die N-Seite einer Aggregationsbeziehung darstellt, wird ein zusammengesetzer Datentyp definiert, der eine Referenz auf das Aggregat enthält.

```
CREATE TYPE varII_rahmen_typ AS (
            herstellernummer    CHAR(11),
            bauweise            VARCHAR(20),
            hoehe               DECIMAL(5,2)
)
```

```
REF USING INT
MODE DB2SQL;

CREATE TYPE varII_fahrrad_typ AS (
            seriennummer        CHAR(11),
            bezeichnung         VARCHAR(50),
            farbe               VARCHAR(20),
            rahmen              varII_rahmen_typ
)
REF USING INT
MODE DB2SQL;

CREATE TYPE varII_schraub_typ AS (
            herstellernummer    CHAR(11),
            gewinde             CHAR(1),
            groesse             DECIMAL(5,2),
            ref_fahrrad         REF(varII_fahrrad_typ)
)
REF USING INT
MODE DB2SQL;
```

Die Entität “Rahmen“ ist im Beispiel an einer 1:1-Beziehung beteiligt und wird daher in die Aggregats-Tabelle aufgenommen. Die Entität “Schraube“ stellt die N-Seite einer 1:N-Beziehung dar und wird daher als eigene Typed Table angelegt, die auf die Aggregats-Tabelle referenziert.

```
CREATE TABLE varII_fahrrad OF varII_fahrrad_typ (
             REF IS oid USER GENERATED,
             oid WITH OPTIONS UNIQUE,
             seriennummer WITH OPTIONS NOT NULL PRIMARY KEY
);

CREATE TABLE varII_schraube OF varII_schraub_typ (
             REF IS oid USER GENERATED,
             ref_fahrrad WITH OPTIONS SCOPE varII_fahrrad
                          REFERENCES varII_fahrrad(oid)
     );
```

Variante II wird dann eingesetzt, wenn Abfragen auf Teile immer auch das Aggregat miteinbeziehen. Zwar wird eine 1:N-Beziehung immer noch mit Hilfe einer eigenen Tabelle abgebildet, aber zumindest die 1:1-Beziehung kann in die Aggregatstabelle mitaufgenommen und so ein Join eingespart werden.

Weitere Varianten können in DB2 nicht zur Aggregationsabbildung eingesetzt werden, da keine Unterstützung für mehrwertige Attribute vorhanden ist. Auch die sonstigen von DB2 angebotenen objekt-relationalen Erweiterungen können nicht sinnvoll zur Abbildung von Aggregation verwendet werden, deshalb sind mit den zwei diskutierten Varianten alle Abbildungsmöglichkeiten erschöpft.

3.8.4 Aggregation in DB2 - Performanceanalyse

Konzept	Typ	Anzahl
Fahrrad	E	40.000
Schraube	E	120.000
Rahmen	E	40.000
seriennummer	A	40.000
bezeichnung	A	40.000
farbe	A	40.000
herstellernummer (rahmen)	A	40.000
bauweise	A	40.000
hoehe	A	40.000
herstellernummer (schraube)	A	120.000
gewinde	A	120.000
groesse	A	120.000

Abbildung 3.94: Mengengerüst für Aggregationsabbildung

Die beiden in DB2 möglichen Varianten wurden auf ihr Kosten- und Laufzeitverhalten untersucht. Das zugrundeliegende Mengenprofil ist dasselbe, das in der Performance-Analyse für Oracle verwendet wurde. Das Mengengerüst ist in Abbildung 3.94 zu sehen.

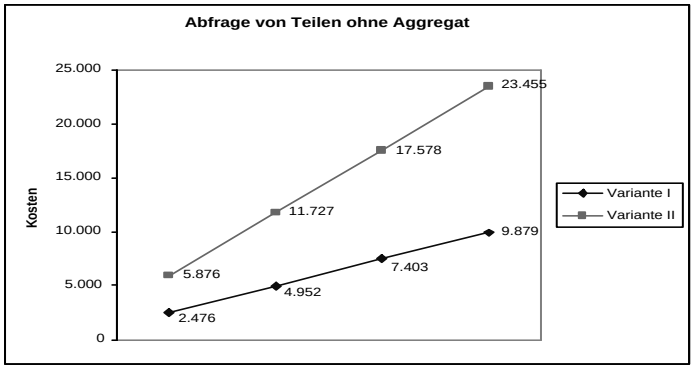
Begonnen wurde mit einer Größenordnung von 40.000 Aggregaten. In vier Schritten wurde diese Anzahl jeweils um 40.000 Datensätze erhöht, bis eine Obergrenze von 160.000 Datensätzen erreicht war. Das Aggregat war mit einem seiner Teile durch eine 1:1-Beziehung verbunden, eine 1:N-Beziehung des Aggregats zu einem Teil wurde ebenfalls realisiert, wobei eine Kardinalität

von drei Teilen angenommen wurde. Die Operationen wurden der Operationshäufigkeitstabelle aus Abbildung 3.95 entnommen.

Name/Kurzbeschreibung	Häufigkeit	Typ (Online/Batch)
O1: Abfragen von Rahmen (ohne Fahrrad)	5x/Tag	OL
O2: Abfragen von Fahrrädern und den dazugehörigen Schrauben	6x/Tag	OL
O3: Abfragen von Fahrrädern und dem dazugehörigen Rahmen	7x/Tag	OL
O4: Abfragen von Fahrrädern mit Gesamtzubehör	15x/Tag	OL
O5: Anlegen eines Fahrrades mit Zubehör	6x/Tag	OL
O6: Update einer Schraube	1x/Tag	OL
O7: Update eines Rahmens	2x/Tag	OL
O8: Löschen von Fahrrädern und ihrem Zubehör	1x/Woche	B

Abbildung 3.95: Operationshäufigkeitstabelle einer Aggregation

Untersucht wurden die Operationen O1 bis O7, da es sich hier um Online-Transaktionen handelt. O8 wird im Batch-Modus durchgeführt und wird daher für weitere Analysen nicht berücksichtigt. Bei den Operationen O1 bis O4 handelt es sich um Abfragen, während O5, O6 und O7 Datensätze erzeugen, beziehungsweise modifizieren. Die benötigten Kosten und Laufzeiten wurden für jede der Varianten für die jeweiligen Operationen festgehalten und miteinander verglichen. Die Analyse selbst wurde auf einem Athlon XP 1800+ mit 256 MB RAM und Windows 2000 durchgeführt.



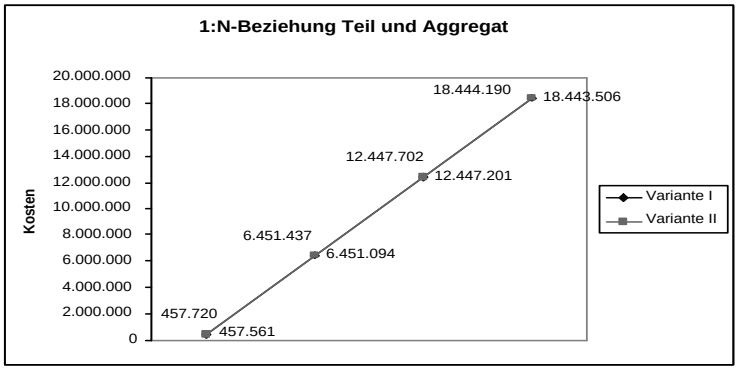
Laufzeit

Anzahl Datensätze	Variante I	Variante II
40.000	0,07 sec	0,10 sec
80.000	0,12 sec	0,19 sec
120.000	0,16 sec	0,26 sec
160.000	0,22 sec	0,39 sec

Abbildung 3.96: Kosten und Laufzeiten - Abfragen von Teilen ohne Aggregat

O1, eine Abfrage, die Teile unabhängig von ihrer Zuordnung zu einem Aggregat zurückliefert,

wurde als erstes gestestet und ergab das Kosten- und Laufzeitverhalten, das in Abbildung 3.96 zu sehen ist. Im Kostendiagramm ist zu erkennen, daß Variante I ganz klar niedrigere Kosten benötigt als Variante II. Das erklärt sich dadurch, daß die Datensätze des Teils in einer eigenen getypten Tabelle gespeichert sind, die auch ein Schlüsselattribut besitzt. Die Datensätze des Aggregats sind nicht von Bedeutung und müssen daher auch nicht betrachtet werden. Bei Variante II werden die Datensätze des Teils gemeinsam mit den Daten des Aggregats in einer Tabelle gespeichert. Da der Primärschlüssel dieser Tabelle auf einem anderem Attribut liegt, als der Primärschlüssel der Tabelle aus Variante I, benötigt der Zugriff auf die Daten des Teils höhere Kosten, da ein Full Table Scan durchgeführt wird. Ein Index kann hier allerdings Abhilfe schaffen. Bei den Laufzeiten ist Variante I ebenfalls zu bevorzugen. Die Unterschiede zwischen den Laufzeiten sind zwar nicht gravierend, da Variante I aber ohnehin auch das bessere Kostenverhalten aufweist, ist bei Abfragen, die nur Teile unabhängig vom Aggregat zurückliefern, in jedem Fall Variante I einzusetzen.



Laufzeit

Anzahl Datensätze	Variante I	Variante II
40.000	1,43 sec	1,46 sec
80.000	2,96 sec	3,17 sec
120.000	4,57 sec	5,07 sec
160.000	5,93 sec	6,18 sec

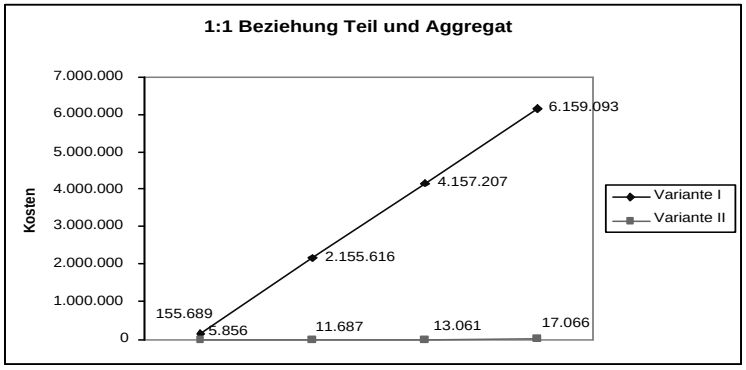
Abbildung 3.97: Kosten und Laufzeiten - Abfragen von Teilen einer 1:N-Beziehung in Verbindung mit dem Aggregat

Abfragen, die Teile einer 1:N-Beziehung in Verbindung mit dem Aggregat benötigen, wie es bei O2 der Fall ist, ergeben ein ungewöhnliches Kosten- und Laufzeitverhalten, wie Abbildung 3.97 zu entnehmen ist.

Das Kostendiagramm beinhaltet auf den ersten Blick nur eine Kostenkurve, der aber verschiedene Kostenwerte zugeordnet sind. Tatsächlich handelt es sich hier aber um zwei verschiedene Kostenkurven, die sich im Vergleich zur Größenordnung der Kosten allerdings kaum voneinander unterscheiden, sodaß der Eindruck einer einzigen Kostenkurve entsteht. Im Diagramm ist zu sehen, daß die Werte der Kosten einen Bereich von ungefähr 18,5 Millionen erreichen. Die Unterschiede zwischen Variante I und Variante II liegen im Bereich von 200 bis 700 Kosteneinheiten, was bei benötigen Kosten in dieser Größenordnung vernachlässigbar ist. Die etwas

niedrigeren Kosten sind Variante I zuzuordnen. Variante I weist auch die besseren Laufzeiten auf, deshalb ist auch bei Abfragen in der Art von O2 Variante I, die getypten Tabellen mit Referenzen, zu empfehlen.

Wenn Teile einer 1:1-Beziehung mit dem Aggregat verbunden werden sollen (O3), ist auf jeden Fall Variante II zu bevorzugen. Das Kosten- und Laufzeitverhalten ist in Abbildung 3.98 zu sehen.



Laufzeit

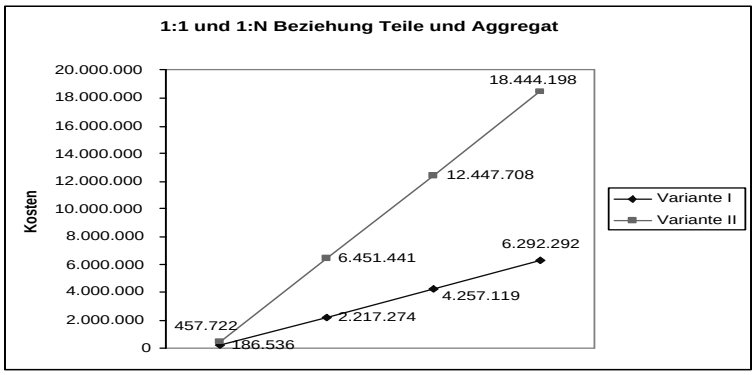
Anzahl Datensätze	Variante I	Variante II
40.000	0,51 sec	0,11 sec
80.000	1,06 sec	0,17 sec
120.000	1,57 sec	0,17 sec
160.000	2,07 sec	0,21 sec

Abbildung 3.98: Kosten und Laufzeiten - Abfragen von Teilen einer 1:1-Beziehung in Verbindung mit dem Aggregat

Variante II benötigt eindeutig die niedrigeren Kosten, was dadurch zu erklären ist, daß ein Join durch die Einbettung des Teils als zusammengesetztes Attribut in die Tabelle des Aggregats nicht mehr nötig ist. Bei Variante I hingegen muß eine Referenz aufgelöst werden, was signifikant höhere Kosten verursacht und sich auch negativ auf die Laufzeiten auswirkt. Während Variante II nahezu konstante Laufzeiten aufweist, die sich auch mit der Anzahl der Datensätze nur geringfügig ändern, hat Variante I einen linearen Zeitverlauf, der mit der Anzahl der Datensätze zunimmt. Wenn also hauptsächlich Abfragen an das Datenbankschema gestellt werden, die Teile einer 1:1-Beziehung in Verbindung mit dem Aggregat zurückliefern, empfiehlt sich der Einsatz von Variante II.

Werden Abfragen in der Art von O4 gestellt, die sowohl Teile einer 1:1-Beziehung als auch Teile einer 1:N-Beziehung in Verbindung mit dem Aggregat zurückliefern, ergibt sich das Kosten- und Laufzeitverhalten aus Abbildung 3.99. Variante I weist hier eindeutig das bessere Kostenverhalten auf, als Variante II, die mehr als doppelt soviele Kosteneinheiten benötigt. Bei Variante I müssen zwar alle Teile durch eine Referenz mit dem Aggregat verbunden werden, während der Abfrage joint allerdings das DBMS zunächst die beiden Tabellen, die die Teile beinhalten, miteinander, filtert Teile mit denselben Referenzen heraus und löst erst dann die

verbleibenden Referenzen zum Aggregat auf. Bei Variante II ist zwar das Teil der 1:1-Beziehung in der Aggregatstabelle enthalten, die Auflösung der Referenzen der 1:N-Beziehung verursacht aber immer noch sehr hohe Kosten. Zudem müssen mehr Referenzen aufgelöst werden, als es bei Variante I der Fall ist. Dadurch ergeben sich für Variante II auch höhere Laufzeiten, was dazu führt, daß bei Abfragen, die Teile von 1:1-Beziehungen und Teile von 1:N-Beziehungen zurücklieferen, in jedem Fall Variante I eingesetzt werden sollte, die auch kürzere Laufzeiten hat.



Laufzeit

Anzahl Datensätze	Variante I	Variante II
40.000	0,89 sec	1,56 sec
80.000	2,06 sec	3,57 sec
120.000	3,08 sec	4,88 sec
160.000	4,98 sec	6,82 sec

Abbildung 3.99: Kosten und Laufzeiten - Abfragen von Teilen einer 1:1 und 1:N-Beziehung in Verbindung mit dem Aggregat

Ein Überblick darüber, welche Variante im Hinblick auf Kosten- und Laufzeitminimierung bei verschiedenen Abfragekategorien die vorteilhafteste ist, ist in Abbildung 3.100 zu sehen.

Variante I und II	Teile ohne Aggregat		Teile einer 1:N-Beziehung in Verbindung mit Aggregat		Teile einer 1:1-Beziehung in Verbindung mit dem Aggregat		Teile einer 1:1- und einer 1:N-Beziehung in Verbindung mit dem Aggregat	
	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze	Wenig Datensätze	Viel Datensätze
Minimale Kosten	I	I	I, II	I,II	II	II	I	I
Minimale Laufzeit	I	I	I	I	II	II	I	I

Abbildung 3.100: Entscheidungshilfe zur Auswahl von Variante I oder Variante II

Erzeugen und Modifizieren von Datensätzen

Wenn auf das Datenbankschema nur wenig Abfragen gestellt, dafür aber viele Datensätze erzeugt oder verändert werden, darf nicht das Kosten- und Laufzeitverhalten der Abfragen, das im vorherigen Abschnitt untersucht wurde, ausschlaggebend für die Auswahl sein. In diesem Fall

muß das Kostenverhalten der Insert- und Update-Statements, die auf dem Datenbankschema eingesetzt werden, als Entscheidungshilfe herangezogen werden.

Zum Testen der Neuanlage eines Datensatzes wurde ein Aggregat, also ein Fahrrad, angelegt, dazu ein Rahmen und drei Schrauben. Es wurden also sowohl eine 1:1 als auch eine 1:N-Beziehung erzeugt. Die Kosten der einzelnen Insert-Statements wurden addiert und ergeben zusammen die benötigten Gesamtkosten einer Variante, die in Abbildung 3.101 zu sehen sind.

Variante	Insert	Update
I	145	10.082
II	116	100

Abbildung 3.101: Kosten für Insert- und Update-Statements

Variante I benötigt hier etwas höhere Kosten, da die Daten der Teile einer 1:1-Beziehung in einer eigenen Tabelle gespeichert werden. Bei Variante II werden diese Teile gemeinsam mit dem Aggregat in eine Tabelle eingefügt, was niedrigere Kosten verursacht. Beim Erzeugen von Datensätzen ist damit Variante II zu bevorzugen. Werden Datensätze modifiziert, spielt es eine Rolle, ob es sich beim zu ändernden Datensatz um den Teil einer 1:1-Beziehung oder um den Teil einer 1:N-Beziehung handelt. Die Modifikation eines Teils, der mittels 1:N-Beziehung mit dem Aggregat verbunden ist, verursacht bei beiden Varianten dieselben Kosten, die etwa 25.000 Kosteneinheiten betragen. Wird hingegen ein Teil geändert, der mit einer 1:1-Beziehung mit dem Aggregat verbunden ist, benötigt Variante II eindeutig niedrigere Kosten, da hier zum Suchen des Teils keine Referenzen aufgelöst werden müssen. Daher ist auch bei häufigen Modifikationen von Datensätzen der Einsatz von Variante II zu empfehlen.

Die Performance-Analyse hat ergeben, daß im Bereich der Abfragen sehr oft der Einsatz von Variante I deutliche Vorteile in der Kosten- und Laufzeitminimierung bringt. Eine Ausnahme sind Abfragen, die Teile in Verbindung mit dem Aggregat benötigen und mit diesem durch eine 1:1-Beziehung verbunden sind. Hier war Variante II im Vorteil, wie auch beim Erzeugen und Modifizieren von Datensätzen. Welche Variante also schlußendlich gewählt wird, hängt also primär von der Kategorie der Abfragen ab.

3.9 Mehrfachvererbung

Mehrfachvererbung tritt beim Erstellen komplexer Datenmodelle relativ häufig auf und ist auch ein durchaus geeignetes Mittel um Einschränkungen, die von der Realwelt vorgegeben werden,

darzustellen. Bei der Abbildung von Mehrfachvererbung in ein Datenbankschema ergeben sich aber einige Probleme, die berücksichtigt werden wollen und deren Lösung nicht oder nur mit extremen Aufwand automatisiert werden kann. Deshalb liegt es immer in der Verantwortung des Datenbank-Designers eine Mehrfachvererbung bei der Erstellung des Datenbankschemas aufzulösen. Obwohl [SBM99] fordert, daß ein objekt-relationales DBMS Mehrfachvererbung unterstützen muß, ist diese Forderung in der Praxis bis jetzt nicht umgesetzt worden. Aus der Arbeit mit objekt-orientierten DBMS hat sich gezeigt, daß die automatisierte Auflösung von Mehrfachvererbung mit Berücksichtigung aller Attribute und Methoden einer Klasse derart komplex ist, daß es in den meisten Fällen effizienter ist, diese Auflösung dem Designer zu überlassen. Deshalb bietet keines der sich im Moment am Markt befindlichen objekt-relationalen DBMS Unterstützung für Mehrfachvererbung an.

Mehrfachvererbung zeichnet sich dadurch aus, daß eine Subklasse nicht nur eine, sonderen mehrere Superklassen haben kann, die in der Vererbungshierarchie nebeneinander liegen. Dabei müssen zwei Fälle berücksichtigt werden. Es gibt

- geteilte Subklassen (Shared Subclasses) und
- Vereinigung (Union Types oder Categories).

3.9.1 Geteilte Subklassen

Eine geteilte Subklasse befindet sich in einer Vererbungshierarchie unter zwei oder mehreren Superklassen, die alle eine gemeinsames Schlüsselattribut haben. Die geteilte Subklasse erbt die Attribute aller Superklassen, wobei zu berücksichtigen ist, daß doppelt vererbte Attribute in den Superklassen nur einmal an die Subklasse vererbt werden dürfen. Schon bei der Erstellung des konzeptuellen Schemas muß darauf geachtet werden, daß in den Superklassen einer geteilten Subklasse keine Attribute vorhanden sind, die einen gemeinsamen Namen aber eine unterschiedliche Bedeutung haben. In diesem Fall handelt es sich um Strukturkonflikte, die nur vom Designer aufgelöst werden können. Die Superklassen selbst können ebenfalls Subklassen weiterer Superklassen sein. Ein Beispiel für eine geteilte Subklasse ist in Abbildung 3.102 zu sehen.

Kunde und Angesteller haben deshalb den gleichen Primärschlüssel, weil sie diesen von der gemeinsamen Superklasse “Person“ erben. Daher handelt es sich bei der Klasse “Kontoinhaber“ um eine geteilte Subklasse. Diese erbt alle Attribute von ihren Superklassen “Kunde“ und “Angestellter“, wobei zu beachten ist, daß die Attribute, die die beiden Superklassen von der Klasse “Person“ erben, nur einmal an die geteilte Subklasse vererbt werden. Attribute mit glei-

chen Namen und unterschiedlichen Bedeutungen sind in den Superklasse nicht vorhanden, hier gibt es keine Probleme mit der Mehrfachvererbung.

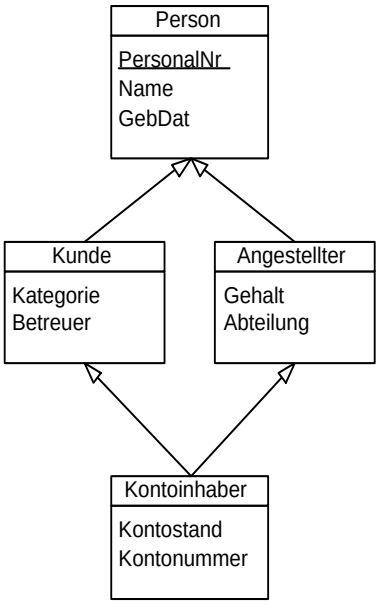


Abbildung 3.102: Vererbungshierarchie mit geteilter Subklasse

Die Abbildung einer geteilten Subklasse in ein objekt-relationales Datenbankschema kann durch Auswahl einer der in Abschnitt 3.6.2 vorgestellten Varianten zur Vererbungsabbildung erfolgen. Die einzigen Varianten, die hier nicht eingesetzt werden können, sind die Varianten VI und VII, die die vom DBMS zur Verfügung gestellte Vererbung ausnutzen, was daher rührt, daß die meisten DBMS keine Mehrfachvererbung unterstützen und es daher verbieten, Subtypen mit mehr als einem Supertypen zu definieren. Ansonsten kann entlang der Vererbungshierarchie beliebig zwischen den Varianten gewechselt werden, wobei auf doppelt vererbte Attribute geachtet werden muß.

3.9.2 Vereinigung

Ein Vereinigung ist - im Gegensatz zu geteilten Subklassen - Superklassen mit unterschiedlichen Primärschlüsseln untergeordnet. Die Vereinigung repräsentiert sozusagen eine UNION-Operation der verschiedenen Superklassen. Für jede Instanz werden dabei die Attribute der Superklasse vererbt, zu der die jeweilige Instanz gehört. Dabei kann immer nur von einer Superklasse gleichzeitig geerbt werden, es handelt sich also um eine Art “Oder-Verknüpfung“ zwischen den Superklassen, die ähnlich einer umgekehrten Generalisierung ist. Der Unterschied zu einer Generalisierung liegt darin, daß eine Generalisierung total ist. Bei einer Generalisierung kann nie der Fall eintreten, daß eine Instanz nur einer Subklasse, aber nicht der Generalisierungsklasse zu-

geordnet ist. Bei einer Vereinigung kann es durchaus sein, daß eine Instanz nur den Superklassen, nicht aber der Vereinigungsklasse zugeordnet ist. Aus dieser Logik ist erkennbar, daß eine totale Vereinigung und eine Generalisierung semantisch equivalent sind. Daher kann eine totale Vereinigung auch als Generalisierung abgebildet werden, und umgekehrt. Eine Vereinigungsklasse erbt zwar Attribute von mehreren Superklassen, trotzdem handelt es sich genaugenommen nicht um Mehrfachvererbung im herkömmlichen Sinn, da nie von mehreren Superklassen gleichzeitig geerbt wird. Ein Beispiel für Vereinigung ist in Abbildung 3.103 zu sehen.

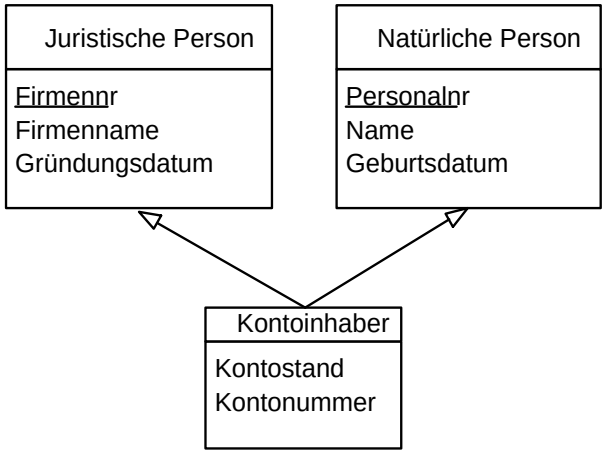


Abbildung 3.103: Vererbung durch Vereinigung

Geht man davon aus, daß eine juristische Person nie gleichzeitig eine natürliche Person sein kann, was auch durch die unterschiedlichen Primärschlüssel ausgedrückt wird, handelt es sich bei dieser Vererbung um eine Vereinigung. Die Subklasse “Kunde“ erbt für jede Instanz entweder die Attribute der juristischen Person oder die Attribute der natürlichen Person.

3.10 Erkenntnisse

Das Kapitel über objekt-relationales Design hat gezeigt, daß es tatsächlich eine Vielzahl von Möglichkeiten gibt, wie Elemente des konzeptuellen Modells unter Verwendung von objekt-relationalen Erweiterungen in ein logisches Modell übergeführt werden können. Die objekt-relationalen Erweiterungen, die dafür eingesetzt werden, sind

- zusammengesetzte Datentypen
- Mengen
- Referenzen

- Vererbung des DBMS

Für die Grundelemente des konzeptuellen Modells, wie Entitäten, Attribute und Beziehungen, gibt es eindeutige Vorschriften, mit denen eine Überführung in ein objekt-relationales logisches Modell realisiert werden kann. Diese sind in Abschnitt 3.5 ausführlich erläutert. Komplexere konzeptuelle Konzepte, wie Vererbung und Aggregation, können durch eine Reihe verschiedener Varianten in ein logisches Schema eingebettet werden und verlangen deshalb nach genauerer Betrachtung.

Für das Konzept der Vererbung gibt es im relationalen Modell vier verschiedene Möglichkeiten, mit denen Vererbung logisch abgebildet werden kann. Im objekt-relationalen Design erweitert sich diese Anzahl auf sieben Möglichkeiten, wobei die DBMS-spezifischen Varianten noch nicht berücksichtigt sind.

- **Variante I:** Hier wird für die Superklasse und für jede Subklasse ein zusammengesetzter Datentyp definiert, der die Attribute der Klassen enthält. In jeder Subklasse ist zusätzlich noch ein Attribut definiert, das auf die Tabelle der Superklasse referenziert. Totale, partielle, disjunkte und überlappende Vererbung ist möglich.
- **Variante II:** Für jede Subklasse wird eine Tabelle angelegt, die die Attribute der Superklasse als zusammengesetzten Datentypen enthält. Die speziellen Attribute der Subklasse werden als relationale Attribute aufgenommen. Partielle Vererbung ist nicht möglich.
- **Variante III:** Eine Einzelrelation wird definiert, die sowohl die Attribute der Superklasse als auch die Attribute der verschiedenen Subklassen als zusammengesetzten Datentypen enthält. Weiters wird ein relationales Attribut in die Tabelle mitaufgenommen, anhand dessen die Zuordnung einer Instanz zu einer bestimmten Subklasse erfolgt. Überlappende Vererbung ist nicht möglich.
- **Variante IV:** Auch hier wird eine Einzelrelation mit zusammengesetzten Datentypen für die Attribute der Super- und Subklassen angelegt, die Zuordnung einer Instanz zu einer bestimmten Subklasse erfolgt aber ebenfalls durch einen zusammengesetzten Datentypen. Totale, partielle, disjunkte und überlappende Vererbung ist möglich.
- **Variante V:** Eine Einzelrelation, wie in Variante III und IV wird verwendet. Die Zuordnung einer Instanz zu einer Subklasse erfolgt mit Hilfe eines mehrwertigen Attributs. Totale, partielle, disjunkte und überlappende Vererbung ist möglich.
- **Variante VI:** Die Vererbung des DBMS wird ausgenutzt um eine Hierachie von zusammengesetzten Datentypen zu definieren. Der Subtyp erbt dabei jeweils die Attribut der

Supertypen. Auf jedem der Datentypen wird eine getypte Tabelle definiert. Totale, partielle, disjunkte und überlappende Vererbung ist möglich.

- **Variante VII:** Es wird ebenfalls die Vererbung des DBMS verwendet um eine Typhierarchie aufzubauen, allerdings ist der Root-Typ nicht instanzierbar, was bedeutet, daß auch keine Tabellen auf ihm definiert werden können. Getypte Tabellen werden nur auf den Subtypen angelegt. Partielle Vererbung ist nicht möglich.

Aus allgemeiner Sicht sind diese sieben Varianten zur Abbildung einer Vererbungshierarchie möglich. Jedes objekt-relationale DBMS sollte in der der Lage sein, diese Varianten umzusetzen. Diese Theorie wurde mit den DBMS Oracle 9.0.2 und IBM DB2 UDB 7.2 getestet. Es hat sich gezeigt, das das DBMS DB2 UDB 7.2 keine mehrwertigen Attribute unterstützt, was bedeutet, daß Variante V nicht einsetzbar ist. Oracle 9.0.2 hingegen kann alle sieben Möglichkeiten realisieren. Generell gilt, daß bei Einsatz von Variante II, VI und VII für überlappende Vererbung Redundanz in Kauf genommen werden muß, da die Attribute der Superklasse in jeder betroffenen Subklasse gespeichert werden müssen.

Zusätzlich zu den sieben Varianten bietet jedes der beiden DBMS durch seine speziellen objekt-relationalen Erweiterungen noch eine weitere Möglichkeit an, um Vererbung abzubilden.
In Oracle ist es möglich eine getypte Tabelle auf dem Supertyp einer Hierarchie von zusammengesetzten Datentypen zu definieren. In diese Tabelle können in weiterer Folge auch Datensätze eingefügt werden, die auf einem Subtyp basieren. Damit reicht eine einzige Tabelle um sämtliche Datensätze der Vererbung zu speichern. DB2 bietet die Möglichkeit an, die Typhierarchie mit Tabellen nachzubauen. Für jeden zusammengesetzten Datentyp wird eine Tabelle definiert, wobei die Tabellen der Subtypen hierarchisch unter der Tabelle des Supertyps liegen. Datensätze werden je nach Zuordnung zu einer Subklasse in die entsprechende Subklassen-Tabelle eingefügt. Bei Abfragen wird allerdings die gesamte Tabellenhierarchie durchlaufen, das bedeutet, eine Selektion auf Attribute der Superklassen-Tabelle liefert immer auch die Instanzen der darunterliegenden Subklassen zurück. Bei beiden dieser Varianten ist keine überlappende Vererbung möglich.
Berücksichtigt man diese beiden speziellen Varianten, hat man in Oracle 9.0.2 insgesamt acht Möglichkeiten um Vererbung in ein objekt-relationales logisches Schema abzubilden, also doppelt soviele wie es ein relationales Schema erlaubt. DB2 UDB 7.2 bietet sieben Möglichkeiten an.

Durch die große Auswahl erschwert sich aber auch die Entscheidung für den Designer, welche Variante er wählen soll. Wichtig für eine erste Einschränkung ist das Mengen- und Transaktionsprofil. Aus der Struktur der Operationen, die auf dem Schema ausgeführt werden sollen und

aus der Anzahl der logischen Zugriffe wird genügend Information gewonnen, um die Auswahl auf zwei bis vier Varianten einzuschränken. Zwischen den verbleibenden Varianten entscheiden die Ergebnisse der Performance-Analyse.

Ist eine Einschränkung zumindest soweit getroffen, daß klar ist, ob totale, partielle, disjunkte und/oder überlappende Vererbung unterstützt werden soll, bleiben maximal vier Varianten übrig.

Wenn alle vier Vererbungsarten eingesetzt werden sollen, muß die Wahl zwischen den Varianten I, IV, V und VI getroffen werden. Für jedes der beiden DBMS wurden Abfragen erstellt, die eine möglichst repräsentative Auswahl von Operationen darstellen sollen. Die Einteilung erfolgte in Abfragen, die genau eine Subklasse in Verbindung mit der Superklasse benötigen, in Abfragen die mehrere Subklassen in Verbindung mit der Superklasse benötigen, in Abfragen, die nur Daten der Superklasse zurückliefern und in Abfragen, die eine Aggregationsfunktion beinhalten.

Für Oracle hat sich dabei gezeigt, daß je nach Kategorie der Abfrage eine andere Variante zu bevorzugen ist. Eindeutig auszuschließen war nur Variante V, die immer die höchsten Kosten und die längsten Laufzeiten benötigte. Variante V verwendet ein VARRAY zur Realisierung des mehrwertigen Attributs. Zugriffe auf ein VARRAY sind mit hohen Kosten verbunden und verwenden keine Indizes. Der Einsatz von VARRAYS ist daher nicht empfehlenswert. Variante I, die Referenzen zur Realisierung von Beziehungen verwendet, weist so gut wie immer die kürzesten Laufzeiten auf und bei Abfragen, die Aggregationsfunktionen beinhalten, auch die niedrigsten Kosten. Bei den anderen Kategorien von Abfragen, sowie beim Erzeugen und Modifzieren von Datensätzen kann Variante I nicht überzeugen. Die Kosten für die Auflösung und die Anlage einer Referenz sind sehr hoch, Indizes werden beim Auflösen einer Referenz nicht verwendet, es erfolgt immer ein Full-Table-Scan. Wenn also die Kosten eine wesentliche Rolle spielen, sollte der Einsatz von Referenzen auf jeden Fall vermieden werden. Wenn genau eine Subklasse in Verbindung mit der Superklasse benötigt wird, kann im Hinblick auf die Kosten Variante VI überzeugen, die die Vererbung des DBMS mittels zusammengesetzter Datentypen einsetzt. Bei Verbindung mehrerer Subklassen mit der Superklasse liegen Variante IV und VI gleichauf, hier bleibt es dem Designer überlassen, welche Variante er wählt. Bei Abfragen, die nur auf die Superklasse zugreifen, kann zwischen Variante I und IV gewählt werden, Variante VI verursacht hier sehr hohe Kosten. Allerdings ist Variante VI beim Erzeugen und Modifizieren von Datensätzen eindeutig zu bevorzugen, da sie dort sehr niedrige Kosten benötigt. Aus der Performance-Analyse ist damit ersichtlich, daß keine Variante gegenüber den anderen eindeutig

zu bevorzugen ist. Ganz klar auszuschließen ist nur der Einsatz eines VARRAYS, ansonsten gilt, daß jede der Varianten, die besonders gute Ergebnisse auf einem Gebiet liefert, auf einem anderen dafür meistens extrem schlechte Werte aufweist. Vor allem Variante I und Variante VI haben das bestätigt. Variante IV liegt immer im Mittelfeld, liefert weder besonders gute, noch besonders schlechte Werte. Sie ist dann zu bevorzugen, wenn ein Kompromiß zwischen Kosten- und Laufzeitminimierung gefunden werden muß.

In DB2 ist Variante V nicht realisierbar, da keine mehrwertigen Attribute unterstützt werden. Die Entscheidung muß hier zwischen den Varianten I, IV und VI getroffen werden. Variante I liefert hier im Gegensatz zu Oracle schlechte Ergebnisse bei den Laufzeiten. Nur wenn Abfragen mit Aggregationsfunktionen gestellt werden, weist Variante I kurze Laufzeiten auf, ansonsten ist auf jeden Fall für Laufzeitminimierung Variante IV oder VI zu bevorzugen. Variante IV liefert dabei gute Ergebnisse, wenn nur Daten der Superklasse benötigt werden. Für Abfragen die Subklassen in Verbindung mit der Superklasse benötigen kann keine eindeutige Entscheidung zugunsten von Variante IV oder VI getroffen werden. Klar ist nur, daß Variante I zu vermeiden ist. Für Kostenminimierung ist die Entscheidung je nach Abfragekategorie sehr eindeutig. Abfragen, die genau eine Subklasse in Verbindung mit der Superklasse benötigen, werden am besten durch Variante IV unterstützt, der Einzelrelation, in der Instanzen mit Hilfe eines zusammengesetzten Datentypen zu einer Subklasse zugeordnet werden. Für mehrere Subklassen in Verbindung mit der Superklasse liefert Variante VI, die Vererbung des DBMS mittels Typhierarchie, die niedrigsten Kosten. Und Variante I, die Referenzen einsetzt, ergibt die niedrigsten Kosten, wenn nur die Daten der Superklasse benötigt werden, oder wenn die Abfrage eine Aggregationsfunktion beinhaltet. Beim Erzeugen von Datensätzen muß die Wahl zwischen Variante IV und VI getroffen werden, da diese nur jeweils ein Insert-Statement benötigen, im Gegensatz zu Variante I, bei der mehr Tabellen betroffen sind. Müssen viele Datensätze modifiziert werden, ist ganz klar Variante IV zu bevorzugen, Variante VI, weist die schlechtesten Ergebnisse auf. Aus der Performance-Analyse war also klar erkenntlich, daß in DB2 UDB 7.2 Referenzen zwar unterstützt werden, es aber nicht gelungen ist, den Zugriff auf die Referenzen zu optimieren. Sobald eine Referenz aufgelöst oder erzeugt werden muß, ergibt sich ein sehr schlechtes Kosten- und Laufzeitverhalten. Daher ist der Einsatz von Referenzen zu vermeiden. Die Vererbung, die vom DBMS angeboten wird (Variante VI), bietet ein sehr akzeptables Kosten- und Laufzeitverhalten, wenn genau eine Subklasse in Verbindung mit der Superklasse benötigt wird. Auch im Bereich der Laufzeitminimierung bietet Variante VI durchaus gute Ergebnisse. Sobald aber mehrere Subklassen in Verbindung mit der Superklasse benötigt werden, oder hauptsächlich Datensätze modifziert werden, kann Variante VI nicht mehr überzeugen. Auch in DB2 UDB 7.2 wird also Variante IV, die Einzelrelation mit dem

zusamengesetzten Datentypen eine gute Lösung sein, wenn ein Kompromiß zwischen Kosten- und Laufzeitminimierung gefunden werden muß.

Wenn partielle Vererbung nicht erlaubt ist, muß eine Entscheidung zwischen Variante II, den Subklassenrelationen, die die Attribute der Superklasse als zusammengesetzten Datentypen enthalten, und Variante VII, der Vererbung des DBMS mittels Typhierarchie mit einem nicht instanzierbaren Root-Typ, getroffen werden.

In Oracle hat sich dabei herausgestellt, daß unabhängig von der Kategorie der Abfragen und unabhängig von der Art des Zugriffs (lesend oder schreibend) Variante II in jedem Fall die geringsten Kosten verursacht. Bei den Laufzeiten liegen die beiden Varianten in fast allen Abfrage-Kategorien gleichauf, einzig und allein bei Abfragen, die nur Daten der Superklasse benötigen, kann Variante VII punkten. Hier liefert Variante II schlechte Ergebnisse, da die Superklassen-Attribute aller Tabellen durch eine UNION-Operation verbunden werden müssen. In Oracle ist also bei nicht erlaubter partieller Vererbung in jedem Fall Variante II zu empfehlen, die in fast allen Bereichen die besten Ergebnisse liefert.

In DB2 schneidet Variante VII etwas besser ab, als bei Oracle. Bei den Laufzeiten weist sie fast immer die besseren Werte auf, mit Ausnahmen von Abfragen, die genau eine Subklasse in Verbindung mit der Superklasse benötigen, und außerdem viele Datensätze in den Tabellen gespeichert sind. Bei den Kosten ist Variante VII zu bevorzugen, wenn Abfragen eingesetzt werden, die Aggregationsfunktionen beinhalten, oder wenn nur Daten der Superklasse benötigt werden und außerdem wenig Datensätze in den Tabellen gespeichert sind. In den anderen Kategorien kann Variante II mit niedrigen Kosten punkten, was auch für das Modifizieren von Datensätzen gilt. Die Kosten für das Erzeugen eines Datensatzes sind für beide Varianten gleich hoch. Generell ist für Laufzeitminimierung Variante VII zu empfehlen und für Kostenminimierung Variante II, wobei es aber für bestimmte Größenordnungen der Datensatzanzahl und für Abfragen mit Aggregationsfunktionen zu Ausnahmen kommt.

Darf nur disjunkte Vererbung eingesetzt werden, ist also überlappende Vererbung nicht erlaubt, wird die Entscheidung zwischen Variante III und VIII gefällt. Variante III verwendet eine Einzelrelation mit zusammengesetzten Datentypen für Attribute der Super- und Subklassen. Die Zuordnung einer Instanz zu einer bestimmten Subklasse erfolgt mit einem relationalen Attribut. Variante VIII ist jeweils die spezielle Abbildungsmöglichkeit, die das DBMS zusätzlich zu den allgemein gültigen Varianten anbietet. In Oracle ist das die getypte Tabelle, die auf dem Root-Typ einer Vererbungshierarchie definiert ist und die Instanzen aller Subtypen aufnehmen

kann. In DB2 ist es die Vererbungshierarchie von Tabellen, die auf der Vererbungshierarchie der Datentypen aufbaut. Bei Abfragen auf die Superklasse werden dabei auch immer die Subklassen miteinbezogen.

In Oracle hat sich aufgrund der Performance-Analyse gezeigt, daß Variante III immer die niedrigeren Kosten aufweist, als Variante VIII, unabhängig von der Kategorie der Abfragen und unabhängig von der Art des Zugriffs (lesend oder schreibend). Dabei ist anzumerken, daß Abfragen, die mehrere Subklassen in Verbindung mit der Superklasse benötigen hier nicht eingesetzt werden können. Bei Abfragen, die genau eine Subklasse in Verbindung mit der Superklasse benötigen, kann Variante III im Hinblick auf die Laufzeiten überzeugen. Wenn allerdings Abfragen gestellt werden, die nur Daten der Superklasse benötigen, hat Variante VIII die kürzeren Laufzeiten. Abfragen, die Aggregationsfunktionen beinhalten, liefern für beide Varianten gleichwertige Ergebnisse. Soll also die Laufzeit minimiert werden, muß eine genaue Auswertung des Transaktionprofils vorgenommen werden, um festzustellen, welche Abfragekategorie dominiert. Für Kostenminimierung ist in jedem Fall Variante III zu empfehlen. Zusätzlich muß an dieser Stelle erwähnt werden, daß Abfragen, die auf der Tabelle von Variante VIII definiert werden, sehr komplex zu formulieren sind, da dem DBMS immer mitgeteilt werden muß, welchen Datentyp es auf die zurückgelieferten Datensätze anwenden muß.

In DB2 bietet sich ein etwas anderes Bild. Wenn hier nur Daten der Superklasse benötigt werden, benötigt Variante VIII die niedrigsten Kosten, Variante III aber die geringsten Laufzeiten. Hier hängt die Entscheidung also primär davon ab, ob Kosten oder Laufzeit minimiert werden sollen. Bei Abfragen mit Aggregationsfunktionen ist Variante VIII, die Tabellenhierarchie, sowohl bei den Kosten als auch bei den Laufzeiten klar im Vorteil. Bei Abfragen, die genau eine Subklasse in Verbindung mit der Superklasse benötigen, ergibt sich ein genau umgekehrtes Bild, hier kann Variante III bei Kosten und Laufzeiten überzeugen. Die Kosten zur Erzeugung eines Datensatzes sind für beide Varianten gleich hoch, bei Modifikationen von Datensätzen ist allerdings eindeutig Variante III zu bevorzugen.

Generell hat sich aus der Analyse für Vererbungsabbildung ergeben, daß der Einsatz von Referenzen und der Vererbung des DBMS, wohlüberlegt sein will, es sei denn, ein bestimmter Bereich von Abfragen ist in der Operationstabelle eindeutig dominant. Beide objekt-relationalen Erweiterungen von beiden DBMS haben Bereiche, in denen sie sehr gut abschneiden, dafür weisen sie in anderen Bereichen aber extrem schlechte Werte auf. Ganz abzuraten ist bei Vererbungsabbildung in Oracle vom Einsatz von VARRAY und in DB2 UDB vom Einsatz von Referenzen.

Das zweite Konzept, das genauer untersucht wurde, ist die Abbildung von Aggregation in ein objekt-relationales logisches Modell. Hier gibt es allgemein gesehen drei Möglichkeiten, mit denen eine Aggregationsbeziehung aufgelöst werden kann.

- **Variante I:** Für das Aggregat und jeden der Teile wird eine getypte Tabelle angelegt. Die Tabellen der Teile enthalten ein Attribut, das auf das Aggregat referenziert. Eingesetzt wird diese Variante vor allem dann, wenn Teile nicht in Verbindung mit dem Aggregat benötigt werden.

- **Variante II:** Für das Aggregat und jedes Teil, das durch eine 1:N-Beziehung mit dem Aggregat verbunden ist, wird eine getypte Tabelle angelegt. Die Tabelle der Teile enthält wie schon bei Variante I eine Referenz auf das Aggregat. Teile, die durch eine 1:1-Beziehung mit dem Aggregat verbunden sind, werden als zusammengesetzter Datentyp in die Tabelle des Aggregats mitaufgenommen. Variante II ist gut geeignet, wenn Teile immer in Verbindung mit dem Aggregat benötigt werden.

- **Variante III:** Für das Aggregat wird eine Tabelle angelegt. Teile, die durch eine 1:1-Beziehung mit dem Aggregat verbunden sind, werden als zusammengesetze Datentypen in die Aggregatstabelle mitaufgenommen. Teile einer 1:N-Beziehung werden durch ein mehrwertiges Attribut in der Aggregatstabelle realisiert. Information über logische Zusammengehörigkeit bleibt erhalten.

In Oracle ist die Realisierung aller drei Variante möglich. Das mehrwertige Attribut von Variante III wird mit Hilfe einer Nested Table dargestellt. Ein VARRAY ist hier unpassend, da ein Teil, der an einer 1:N-Beziehung teilnimmt, selbst wieder durch Attribute beschrieben wird, was mit einem VARRAY nicht dargestellt werden kann. In DB2 werden keine mehrwertigen Attribute unterstützt, daher können nur Variante I und II realisiert werden.

Auch bei der Aggregation wird zunächst anhand des Transaktionsprofils bestimmt, welche Kategorie von Abfragen die auszuführenden Operationen dominiert. Abfragen liefern entweder Teile ohne dazugehöriges Aggregat zurück oder benötigen Teile in Verbindung mit dem Aggregat. Weiters spielt es eine Rolle, ob die Teile durch eine 1:1 oder eine 1:N-Beziehung mit dem Aggregat verbunden sind und es gibt Abfragen, die sowohl Teile einer 1:1 als auch Teile einer 1:N-Beziehung zurückliefern.

In Oracle hat die Kosten- und Laufzeitanalyse ergeben, daß für Abfragen der ersten Kategorie, die Teile unabhängig vom Aggregat benötigen, für Kostenminimierung auf jeden Fall Variante

I einzusetzen ist. Die Laufzeiten unterscheiden sich für alle drei Variante nur sehr minimal, hier ist keine eindeutige Entscheidung zugunsten einer Variante zu treffen. Für die anderen Kategorien von Abfragen, die Teile in Verbindung mit dem Aggregat benötigen, ist Variante I auf keinen Fall zu empfehlen, da sie durchwegs höhere Kosten benötigt. Auch bei den Laufzeiten ist Variante I entweder schlechter oder bestenfalls gleich gut, wie die anderen Varianten. Für Abfragen, die Teile einer 1:N-Beziehung benötigen ist in Oracle Variante III, der Einsatz einer Nested Table als mehrwertiges Attribut zu empfehlen. Werden dagegen hauptsächlich Teile benötigt, die mit einer 1:1-Beziehung mit dem Aggregat verbunden sind, kommt Variante II zum Zug. Variante II und III liefern gleichwertige Laufzeiten, wobei Variante III bei Tabellen mit wenig Datensätzen etwas kürzere Laufzeiten benötigt. Auch für die Aggregation gilt, daß Varianten, die in einem Bereich gute Ergebnisse liefern, in anderen Bereichen extrem schlechte Werte aufweisen. So benötigt Variante III, die bei Abfragen ein sehr gutes Kosten- und Laufzeitverhalten hat, beim Erzeugen und Modifizieren von Datensätzen sehr hohe Kosten, was mit der Auswahl der passenden Nested Table für einen bestimmten Datensatz zusammenhängt. Nested Tables wurden offensichtlich für Abfragen optimiert, bei schreibenden Zugriffen ist das Performanz-Verhalten sehr schlecht. Hier dominiert Variante I, die die geringsten Kosten aufweist, der Unterschied zu Variante II ist allerdings nicht sehr hoch. Variante II liefert damit einen Kompromiß, wenn in allen Operationskategorien möglichst gute Ergebnisse erzielt werden sollen.

In DB2, wo nur Variante I und II realisiert werden können, bietet sich ein etwas anderes Bild. Hier ist Variante I in fast allen Abfragekategorien zu bevorzugen, sowohl bei Kosten- als auch bei Laufzeitminimierung. Nur bei Abfragen, die Teile einer 1:1-Beziehung in Verbindung mit dem Aggregat benötigen, liefert Variante II bessere Kosten und bessere Laufzeiten. Bei Abfragen, die Teile einer 1:N-Beziehung in Verbindung mit dem Aggregat benötigen, liegt Variante II beim Kostenverhalten mit Variante I gleichauf, allerdings liefert Variante I hier die besseren Laufzeiten und ist damit zu bevorzugen. Werden Datensätze erzeugt, erfordert Variante II niedrigere Kosten, der Unterschied zu Variante I ist allerdings relativ gering. Erst bei der Modifikation von Datensätzen weist Variante I ein deutlich schlechteres Kostenverhalten auf, es wird die hundertfache Anzahl an Kosteneinheiten benötigt. Die hohen Kosten eines Update-Statements ergibt sich durch die notwendige Auflösung der Referenzen, wie es auch im Bereich der Vererbungsabbildung schon ersichtlich war. Da bei Abfragen, die nur Teile ohne das Aggregat benötigen, keine Referenz aufgelöst werden muß, bleiben hier die Kosten niedrig. Bei Abfragen, die eine 1:N-Beziehung beinhalten, muß die Referenz in beiden Varianten aufgelöst werden, damit ergibt sich auch dasselbe Kostenverhalten. Abfragen, die sowohl Teile einer 1:1 als auch einer 1:N-Beziehung benötigen, filtern bei Variante I zunächst die Teile heraus, die zum gleichen Aggregat gehören (vergleichen also die Werte der Referenzattribute) und

lösen erst dann die Referenzen zum Aggregat auf. Dadurch ergeben sich ingesamt niedrigere Kosten, als bei Variante II, bei der nur die Referenzen des Teils, das an einer 1:N-Beziehung teilnimmt, aufgelöst werden müssen. Da hier allerdings keine vorherige Filterung der Teile erfolgt, müssen insgesamt mehr Referenzen verfolgt werden, als bei Variante I. Generell gilt damit in DB2, daß Referenzen, sobald sie aufgelöst werden, ein sehr schlechtes Kostenverhalten liefern, da aber DB2 keine Unterstützung für mehrwertige Attribute anbietet, gibt es für 1:N-Aggregationsbeziehungen keine objekt-relationale Alternative, es kann ansonsten nur die relationale Abbildungsvorschrift verwendet werden.

Die Analyse für Vererbungs- und Aggregationsabbildung hat generell auch ergeben, daß die Hersteller der beiden DBMS ganz unterschiedliche Preferenzen in der Realisierung von objekt-relationalen Erweiterungen zeigen. Während bei Oracle viel Wert auf die Optimierung von Referenzen gelegt wird und die Vererbung erst seit Version 9i wirklich möglich ist, unterstützt DB2 UDB 7i Vererbung besser, das aber offenbar zu Lasten der Referenzen. Oracle bietet auch Möglichkeiten zur Unterstützung mehrwertiger Attribute, auch wenn diese noch nicht optimal eingesetzt werden, DB2 UDB hingegen sieht hier noch keine Möglichkeiten vor. Für beide DBMS gilt aber, daß unter Berücksichtigung aller Kategorien von Abfragen, Insert- und Update-Statements, die Varianten am Besten abschneiden, die den relationalen Lösungen sehr ähnlich sind. Dazu zählen die Einzelrelationen, deren Attribute teilweise oder ganz auf zusammengesetzten Datentypen aufbauen und auch getypte Tabellen, diese allerdings nur, wenn sie nicht in die vom DBMS angebotene Vererbung eingebunden sind und wenn sie nicht referenziert werden. Für die typisch objekt-relationalen Erweiterungen wie, Referenzen, Vererbung und mehrwertige Attribute, muß festgestellt werden, daß sie nur Hinblick auf ganz bestimmte Kategorien von Operationen des Mengen- und Transaktionsprofils wirklich überzeugen können, aber weit entfernt von einem optimalen Gesamtergebnis sind.

Kapitel 4

Schlußbemerkungen

Obwohl der Wunsch nach einen DBMS, das die Vorteile relationaler und objekt-orientierter Systeme in sich vereint, vorhanden ist und es bereits erste Ansätze für die Entwicklung objekt-relationaler DBMS gibt, ist man noch weit von der Vorstellung von [SBM99] entfernt, daß diese Systeme die relationalen Datenbanken ablösen werden.

Die Vorgehensweise der Hersteller ist bisher jene, daß relationale Systeme um objekt-relationale Konzepte wie Referenzen, zusammengesetzte Datentype, getypte Tabellen und Vererbung, erweitert werden. Die Performanz der relationalen Komponenten darf dabei auf keinen Fall negativ beeinflußt werden, die Performanz der objekt-relationalen Erweiterungen sollte zumindest an die eines relationales Systems herankommen. Das ist der Wunsch, doch in der Praxis werden diese Anforderungen noch nicht erfüllt. Hersteller wie Oracle und IBM bieten die wichtigsten objekt-relationalen Erweiterungen zwar schon an, es gibt aber noch kein DBMS, das alle Anforderungen, die an ein objekt-relationales System gestellt werden, vollständig enthält. So fehlt im DBMS des Herstellers IBM, DB2 UDB, jegliche Unterstützung für mehrwertige Attribute. Referenzen und zusammengesetzte Datentypen (inklusive Vererbung) werden zwar angeboten, sind aber weit von den Performanz-Ergebnissen eines relationalen Systems entfernt. Bei der angebotenen Vererbung handelt es sich derzeit um reine Schema-Vererbung, sprich Attribute eines Supertyps werden an seine Subtypen vererbt, Instanzenvererbung wird allerdings noch nicht angeboten. Es liegt immer noch in der Verantwortung der Applikation beziehungsweise des SQL-Programmierers, eine Tabelle, die auf einer Typhierachie von zusammengesetzten Datentypen aufbaut, im Hinblick auf die Vererbung oder auch Aggregation mit den korrekten Datensätzen zu befüllen und sich dabei um die Konsistenz der Daten zu kümmern. In Oracle wurde bei der Entwicklung in den letzten Versionen viel Wert auf die Unterstützung von Referenzen gelegt, zusammengesetzte Datentypen und mehrwertige Attribute werden ebenfalls unterstützt, dafür wurde offensichtlich die Entwicklung der Vererbungsunterstützung vernachlässigt. Diese wurde

erst mit dem letzten großen Versionssprung (von 8i auf 9i) realisiert und ist noch nicht ausgereift.

Entscheidet man sich trotz allem die objekt-relationalen Erweiterungen eines DBMS zu nutzen, steht man zunächst vor der Aufgabe, logische Datenmodelle für eine objekt-relationale Datenbank zu entwerfen. Durch den Einsatz objekt-relationaler Konzepte ergibt sich im logischen Schritt des Design-Prozesses eine Vielzahl von Möglichkeiten, mit denen ein konzeptuelles Modell auf ein logisches abgebildet werden kann, dabei sollte man aber nicht vergessen, daß die Arbeit eines Designers durch die große Auswahl auch erschwert wird. In einem relationalen logischen Modell gibt es vier Möglichkeiten, wie Vererbung abgebildet werden kann, in einem objekt-relationalen logischen Modell hingegen acht. Es ist schwieriger aus acht Varianten die passende auszuwählen, als aus vieren. Die Wahl der geeigneten Varianten hängt sehr stark vom Mengen- und Transaktionsprofil ab, wobei vor allem die Navigationspfade der Operationen eine Rolle spielen. Das Mengen- und Transaktionsprofil alleine reicht aber nicht, um eine Entscheidung so zu treffen, daß das entstehende logische Modell im Hinblick auf Kosten und Laufzeit möglichst optimal ist. Die Bewertung einer logischen Zugriffsoperation auf eine objekt-relationale Erweiterung ist schwer zu treffen, da man auf der logischen Ebene von der tatsächlichen Zugriffsrealisierung durch ein konkretes DBMS zu weit entfernt ist. Man kann deshalb auch nicht davon ausgehen, daß die logische Zugriffsberechnung einen Annäherungswert an die tatsächliche Zugriffszahl eines objekt-relationalen Datenbankschemas darstellt. Als zusätzliche Entscheidungshilfe sollte daher auf jeden Fall eine Performanz-Analyse herangezogen werden, wie sie in dieser Arbeit durchgeführt wurde.

Die Ergebnisse einer solchen Performanz-Analyse sind teilweise sehr ernüchternd. Obwohl in den Handbüchern der Hersteller immer nur positive Bemerkungen zu den verschiedenen objekt-relationalen Erweiterungen zu finden sind (was allerdings nicht weiter erstaunlich ist), können diese großteils die in sie gesetzten Erwartungen nicht erfüllen. Vor allem bei den "wirklichen" objekt-relationalen Erweiterungen wie Referenzen, Vererbung und mehrwertige Attribute (wenn diese unterstützt wurden), hat sich gezeigt, daß sie zwar in einigen Bereichen, wie zum Beispiel bei Abfragen, sehr gute Kosten- oder Laufzeitergebnisse liefern, dafür aber in anderen Bereichen, wie beim Erzeugen oder Modifizieren von Datensätzen, extrem schlechte Werte aufweisen. Die Unterschiede sind nicht nur im Vergleich von Abfragen zu Insert- und Update-Statements gegeben, auch innerhalb verschiedener Abfragekategorien können die objekt-relationalen Erweiterungen entweder niedrige Kosten oder niedrige Laufzeiten liefern, wobei für gute Laufzeiten meistens hohe Kosten notwendig waren, oder umgekehrt. Dazu kommt, daß jede der objekt-relationalen Erweiterungen nur für bestimmte Abfrage-Kategorien gute Ergebnisse liefern konnte, in anderen dafür aber wieder schlechte Werte hatte. Einzig

und allein die Verwendung zusammengesetzter Datentypen in relationalen Tabellen konnte überzeugen und lieferte gute Werte bei Abfragen und bei der Modifikation von Datensätzen und zwar im Hinblick auf Kosten- und Laufzeitminimierung. Eine solche Tabelle ist aber wenn man es genau nimmt, einer relationalen Lösung sehr ähnlich, man kann nicht wirklich von einer objekt-relationalen Lösung sprechen.

Prinzipiell ist also der Einsatz objekt-relationaler Erweiterungen nur unter Vorbehalt zu empfehlen. Der Designer muß dabei die speziellen Eigenschaften eines bestimmten DBMS sehr gut kennen, ebenso das Mengen- und Transaktionsprofil. Der Erfolg einer objekt-relationalen Erweiterung hängt sehr stark von der Struktur der Operationen ab, die auf dem logischen Schema ausgeführt werden, und in den meisten Fällen kann der Einsatz einer bestimmten Erweiterung, wie einer Referenz, eines mehrwertigen Attributes oder der Vererbung des DBMS, nur für ganz bestimmte Kategorien von Abfragen überzeugen. Wenn also wirklich genau bestimmt werden kann, welche Operationen das Mengen- und Transaktionsprofil eindeutig dominieren und Performanz-Einbußen bei den weniger wichtigen Operationen in Kauf genommen werden können, kann der Einsatz objekt-relationaler Erweiterungen wirklich die Vorteile bringen, die von den Herstellern versprochen werden. Wenn aber ein guter Mittelweg im Hinblick auf das Kosten- und Laufzeitverhalten aller Operationen, nicht nur der dominierenden, gewünscht ist, sollte auf relationale Konzepte zurückgegriffen werden, wobei auch relationale Tabellen, deren Attribute auf zusammengesetzten Datentypen aufbauen, als relationale Konzepte zu verstehen sind. Ebenso können getypte Tabellen bedenkenlos eingesetzt werden, solange sie nicht referenziert oder in die vom DBMS angebotenen speziellen Konzepte für Vererbung miteingebunden werden. Allerdings sollte man an dieser Stelle nicht vergessen, daß die Erweiterung einer getypten Tabelle oder eines Attributes, das auf einem zusammengesetzten Datentyp aufbaut, Probleme mit sich bringt, die in einer relationalen Tabelle nicht vorkommen. Während eine relationale Tabelle ohne Probleme um ein nullierbares Attribut erweitert werden kann, ist es nicht möglich einen zusammengesetzten Datentyp zu erweitern, auf dem bereits Tabellen oder Attribute definiert sind. Um eine Erweiterung oder Änderung eines zusammengesetzten Datentyps vorzunehmen, müssen zunächst alle getypten Tabellen und alle Spalten von Tabellen, die auf dem zusammengesetzten Datentyp definiert sind, gelöscht werden, dann kann der Typ geändert werden. Alle betroffenen Spalten und Tabellen müssen im Anschluß daran neu angelegt werden. Daten sollten während einer solchen Vorgehensweise natürlich nicht verloren gehen.

Meine Schlußfolgerung aus den Untersuchungen, mit denen sich diese Arbeit beschäftigt hat, ist jene, daß bei der Entwicklugn objekt-relationaler Schemata auf jeden Fall eine Performanz-Analyse für das eingesetzte DBMS durchgeführt werden sollte, die die speziellen Operationskate-

gorien des Transaktionsprofils berücksichtigt. Anhand der Performanz-Studie kann entschieden werden, welche objekt-relationalen Erweiterungen und Abbildungsvarianten eingesetzt werden sollen, um besonders gute Kosten- und Laufzeitergebnisse für das Datenbankschema zu erhalten. Zusätzlich sollte auf jeden Fall noch das physische Datenbank-Design berücksichtigt werden, ebenso wie die Tuning-Maßnahmen, die von den Herstellern für objekt-relationale Erweiterungen empfohlen werden, um ein optimales Ergebnis zu erzielen.

Literaturverzeichnis

[Alh98] Sinan Si Alhir. *UML in a Nutshell: A Desktop Quick Reference*. O'Reilly & Associates, Inc., 1998.

[BCN92] Carlo Batini, Stefano Ceri, and Shamkant B. Navathe. *Conceptual Database Design - An Entity-Relationship Approach*. The Benjamin/Cummings Publishing Company, 1992.

[BDT83] Dina Bitton, David J. DeWitt, and Carolyn Turbyfill. Benchmarking Database Systems: A Systematic Approach. In *VLDB 1983*, pages 8–19, 1983.

[CBB+97] R.G.G Cattell, Douglas Barry, Dirk Bartels, Mark Berler, Jeff Eastman, Sophie Gamerman, David Jordan, Adam Springer, Henry Strickland, and Drew Wade. *The Object Database Standard: ODMG 2.0*. Morgan Kaufmann Publishers, Inc., 1997.

[CCN+99] Michael Carey, Don Chamberlin, Srinivasa Narayanan, Bennet Vance, Doug Doole, Serge Rielau, Richard Swagerman, and Nelson Mattos. O-O, What Have They Done to DB2? In *Proceedings of the 25th VLDB Conference*, Edinburgh, Scotland, 1999.

[CD96] Michael J. Carey and David J. DeWitt. Of Objects and Databases: A Decade of Turmoil. In *Proceedings of the 22nd VLDB Conference*, Mumbai(Bombay), India, 1996.

[CDN+97] Michael J. Carey, David J. DeWitt, Jeffrey F. Naughton, Mohammad Asgarian, Paul Brown, Johannes E. Gehrke, and Dhaval N. Shah. The BUCKY object-relational benchmark. In *Proceedings VLDB Conference*, pages 135–146, Arizona, US, 1997.

[Cha97] Donald D. Chamberlin. Evolution of Object-Relational Database Technology in DB2. In *Proceedings of COMPCON '97*, San Jose, California, February 1997.

[Che76] Peter Chen. The Entity-Relationship Model - Toward a Unified View of Data. In *ACM Transactions on Database Systems*, June 1976.

[DD95] Hugh Darwen and C.J. Date. The third manifesto. *ACM SIGMOD Record*, 24(1):39–49, March 1995.

[Dev01] Ramakanth Subrahmanya Devarakonda. Object-Relational Database Systems - The Road ahead. *Crossroads*, 7(3):15–18, 2001.

[EN00] Ramez Elmasri and Shamkant B. Navathe. *Fundamentals of Database Systems (Third Edition)*. Addison-Wesley, 2000.

[Mir97] Serge Miranda. Object Relational Datamodels of the Future. In *3rd Basque International Workshop on Information Technology (BIWIT '97)*, Biarritz, France, June 1997.

[MP01] Wai Yin Mok and David P.Paper. On Transformations from UML Models to Object-Relational Databases. In *Proceedings of the 34th Hawaii International Conference on System Sciences*, Maui, Hawaii, January 2001.

[MRS99] W. Mahnke, N. Ritter, and H. P. Steiert. Towards Generating Object-Relational Software Engineering Repositories. In *Tagungsband der GI-Fachtagung 'Datenbanksysteme in Büro, Technik und Wissenschaft' (BTW '99)*, Freiburg, Germany, March 1999.

[MS00] Wolfgang Mahnke and Hans-Peter Steiert. Zum Einsatzpotential von ORDBMS in Entwurfsumgebungen. In *Tagungsband der Fachtagung CAD 2000*, pages 219–239, Berlin, Germany, March 2000.

[Obj02] Object Management Group - OMG. The UML Ressource Page. 2002. `http://www.omg.org/uml/` - zuletzt besucht am 08.07.2002.

[RBP+91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[SBM99] Michael Stonebraker, Paul Brown, and Dorothy Moore. *Object-relational DBMSs - Tracking the Next Great Wave*. Kaufmann, 1999.

[Sch00] Celia Schahczenski. Object-oriented databases in our curricula. *The Journal of Computing in Small Colleges*, 16(1), October 2000.

[Sto97] Michael Stonebraker. Architectural Options for Object-Relational DBMS. *Informix Software, CA*, February 1997.

[Vos99] Gottfried Vossen. *Datenbankmodelle, Datenbanksprachen und Datenbankmanagement-Systeme*. R. Oldenbourg Verlag München Wien, 1999.

[ZR00] Weiping Zahng and Norbert Ritter. Measuring the Contributions of (O)RDBMS to Object-Oriented Software Development. In *2000 International Database Engineering and Applications Symposium (IDEAS'00)*, Yokohama, Japan, September 2000.

[ZR01] Weiping Zhang and Norbert Ritter. The Real Benefits of Object-Relational DB-Technology for Object-Oriented Software Development. In *Proc. 18th British National Conference on Databases (BNCOD 2001)*, Oxford, UK, July 2001.

[ZRH01] Nan Zhang, Norbert Ritter, and Theo Härder. Enriched Relationship Processing in Object-Relational Database Management Systems. In *Proceedings of the Third International Symposium on Cooperative Database Systems for Advanced Applications (CODAS '01)*, Beijing, China, April 2001.