Einbettung von Interpolationsfunktionen in die Datenbanksprache SQL - Datenbankunterstützung für die Umweltforschung

Von der Fakultät Informatik der Universität Stuttgart
zu Erlangung der Würde eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

Vorgelegt von
Leonore Zink
aus Hameln an der Weser

Hauptberichter:	Prof. Dr. Andreas Reuter
Mitberichter:	Prof. Dr. Bernhard Mitschang
Tag der mündlichen Prüfung:	16. Juni 2000

Institut für Parallele und Verteilte Höchstleistungsrechner
der Universität Stuttgart

2000

Vorwort

Das Verbundprojekt ‘Naturmeßfeld “Horkheimer Insel”’ innerhalb dessen die vorliegende Arbeit entstanden ist, begann vor mehr als zehn Jahren als interdisziplinäres Forschungsprojekt innerhalb einer größeren Gruppe von Umweltforschungsprojekten, die als ‘Projekt “Wasser, Abfall, Boden”’ (PWAB) vom Land Baden-Württemberg finanziert wurden. Als das Informatikprojekt im Frühjahr 1988 zu arbeiten begann, lief das Verbundprojekt bereits seit einiger Zeit und auch eine für das Informatikprojekt wichtige Entscheidung, die Auswahl des zu verwendenden Datenbanksystems, war bereits getroffen. Die Projektlaufzeit war zunächst auf ca. drei Jahre für alle Teilprojekte begrenzt; aufbauende Projekte für die nächsten zehn Jahre wurden aber von Anfang an angestrebt. Aufgrund verschiedener nicht von den Mitarbeitern verschuldeter Verzögerungen wie Überschwemmungen der “Horkheimer Insel” wurde an allen Projekten ein bis zwei Jahre länger gearbeitet als zunächst vorgesehen.

Zunächst arbeitete ich innerhalb des Informatikprojekts sowohl an Konzepten für die verteilte Datenhaltung an den verschiedenen Projektstandorten als auch an der in dieser Arbeit beschriebenen Thematik. Dann zeichnete sich aber ab, daß verteilte homogene und heterogene Datenbanksysteme schon bald kommerziell verfügbar sein würden und mit der globalen wurde auch die lokale Vernetzung zwischen den Projektstandorten immer besser, so daß sich das Informatik-Teilprojekt ganz auf die Problematik der Umweltdaten und Meßwerte sowie der dafür notwendigen Erweiterungen der Datenbanksysteme konzentrieren konnte.

Für einen Teil der Umweltdaten, die sogenannten räumlichen Daten, wie sie jeder Landkarte zugrunde liegen, bieten kommerzielle Datenbanksysteme heute geeignete Datentypen mit passenden Datenstrukturen und Operationen darauf an. Für die Meßwerte selbst jedoch mit ihren besonderen Eigenschaften gibt es noch keine fertigen Produkte zu kaufen. Die in dieser Arbeit vorgestellten Konzepte und Lösungen sind damit noch immer aktuell. Mit den heute verfügbaren Datenbanksystemen, die Erweiterungen durch die Benutzer zulassen, ließen sie sich jedoch etwas schneller und einfacher implementieren als hier beschrieben.

Zusammenfassung

Diese Arbeit begann im Rahmen des interdisziplinären Umweltforschungsprojekts “Naturmeßfeld ‘Horkheimer Insel’”, das sich aus fünf Anwenderprojekten und einem Informatik-Teilprojekt zusammensetzte. Innerhalb dieses Projekts wurden von allen Anwendern unterschiedlichste Meßwerte erhoben. Die wesentlichen Aufgaben des Informatik-Teilprojekts bestanden darin, die erhobenen Daten zu strukturieren, zu sichern, aufzubereiten und für die Weiterverarbeitung - auch anderen Projektpartnern als denen, die sie erhoben hatten - bereitzustellen.

Zur Speicherung der Daten wurde ein relationales Datenbanksystem gewählt, weil die Strukturierung, die Sicherung und inzwischen auch die Verteilung der Daten durch ein verteiltes Datenbanksystem zufriedenstellend abgedeckt werden. Nur für die Aufbereitung speziell von Umweltdaten und Meßwerten werden noch immer keine geeigneten Werkzeuge oder Ergänzungen zu den Datenbanksystemen angeboten.

Die wichtigsten Anwendungen auf Umweltmeßwerten sind die Visualisierung der Messungen, der flexible Vergleich mehrerer Meßreihen sowie Modellbildung und Simulation auf der Grundlage dieser Meßwerte. Nun sind Meßreihen aus der natürlichen Umwelt oft unvollständig, weil die Ergebnisse einzelner Messungen fehlen, oder die zeitlichen bzw. räumlichen Abstände der Meßwerte können nicht in der vom weiterverarbeitenden Programm gewünschten Form geliefert werden. Hier setzt diese Arbeit an: Durch in das Datenbanksystem integrierte Interpolationsverfahren wird es möglich, Werte an beliebigen Punkten des Meßintervalls innerhalb einer ‘normalen’ Datenbankanfrage abzufragen. Die Werte werden dann aus den umliegenden Meßwerten mit einem geeigneten Verfahren berechnet.

In dieser Arbeit wird beschrieben, wie eine Anfragesprache erweitert werden muß, damit solche Anfragen möglich sind, und wie dazu das Datenbanksystem ergänzt werden muß. Verschiedene mögliche Ansätze werden aufgezeigt. Ein Ansatz wurde als Prototyp implementiert, um die Benutzerfreundlichkeit dieses Konzepts zu zeigen. Die Leistungsmessungen an einigen Beispielanfragen belegen ein akzeptables Antwortzeitverhalten.

Danksagung

Mein besonderer Dank gilt

- Herrn Prof. Dr. Reuter für die Übernahme des Hauptberichts und für die wissenschaftliche Begleitung während der gesamten Zeit, in der diese Arbeit entstanden ist,
- Herrn Prof. Dr. Mitschang für die Übernahme des Mitberichts,
- Herrn Prof. Dr. Ehrich von der TU Braunschweig, dafür, daß er mein Interesse an der wissenschaftlichen Arbeit und am Gebiet der Datenbanken geweckt hat,
- meinen Kolleginnen und Kollegen für die freundliche und kreative Arbeitsatmosphäre am IPVR und insbesondere Franz Haberhauer für die Unterstützung bei vielen kleinen und größeren technischen Problemen,
- den Mitarbeitern der übrigen PWAB-Teilprojekte für die Einführung in ihre Problemstellungen und besonders Thomas Entenmann für die Bereitstellung der Wetterdaten,
- allen Studentinnen und Studenten, die innerhalb von studentischen Arbeiten oder als wissenschaftliche Hilfskräfte an diesem Teilprojekt mitgearbeitet haben und ohne deren engagiertem Einsatz der erste Prototyp nicht in der vorliegenden Form hätte fertiggestellt werden können,
- der Frauenförderung der Universität Stuttgart, die mir mit einem Stipendium des Hochschulsonderprogramms II 1995 die Fortführung und damit die Vollendung dieser Arbeit nach der Geburt meiner ersten Tochter ermöglichte,
- meinem Mann Roland, der mich unterstützt hat, wo er nur konnte, und meinen Töchtern Sarina und Evelyn, die zugunsten dieser Arbeit immer wieder einmal auf mich verzichten mußten
- und nicht zuletzt ganz herzlich meinen Eltern, die immer für mich da waren, wenn ich sie brauchte.

INHALT

1. Einleitung 1

2. Die Teilprojekte 5
2.1 Das Gesamt-Projekt 5
2.2 Das Teilprojekt “Geohydrologie” 6
2.3 Das Teilprojekt “Bodenkunde” 6
2.4 Das Teilprojekt “Bodenchemie” 7
2.5 Das Teilprojekt “Pflanzenbau” 7
2.6 Das Teilprojekt “Wasserchemie” 8
2.7 Das Teilprojekt “Datenbankunterstützung” 8
2.8 Hard- und Software-Austattung des Projekts 9
3. Meßwerte, Umweltdaten und Umweltprojekte 11
3.1 Eigenschaften von Umwelt-Forschungsprojekten 11
3.2 Die Weiterverarbeitung von Meßwerten 12
3.2.1 Visualisierung von Meßreihen 12
3.2.2 Modellbildung und Simulation 13
3.3 Eigenschaften von Umweltdaten und Meßwerten 13
3.3.1 Der Kontext von Meßwerten 14
3.3.2 Wertebereiche für Meßwerte 15
3.3.3 Beziehungen zwischen Meßwerten 17
3.3.4 Komprimierung und Archivierung 18
3.4 Aufbereitung der Daten: Interpolation 19
3.4.1 Aufbereitung für spontane interaktive Anfragen 20
3.4.2 Der Vergleich von zwei Meßreihen 21
3.4.3 Aufbereitung für Modellrechnung und Simulation 23
3.4.4 Die Ergänzung fehlender Werte 23
3.5 Schnittstellen zum Anwender 24
3.5.1 Dateneingabe 24
3.5.2 Bereitstellung von Daten 25
3.5.3 Kooperation mit vorhandener Hard- und Software 26
3.5.4 Automatische Datensicherung 26
3.6 Resultierende Anforderungen an das “Datenhaltungssystem” 27
3.6.1 Datentypen 27
3.6.2 Definition von Relationenschemata 28
3.6.3 Standard-DBS-Funktionen 29
3.6.4 Archivierung und Komprimierung 29
3.6.5 Verteilte Datenhaltung 30
3.6.6 Schnittstellen 30
3.6.7 Zusammenstellung der Anforderungen 31
4. Technische Möglichkeiten zur Verarbeitung von Umweltdaten 35
4.1 Existierende Systeme 35

4.2 Datenmodelle für Meßwerte und Umweltdaten 36
4.3 Weiterentwicklungen von Datenbanksystemen 38
4.3.1 Geographische Informationssysteme 38
4.3.2 Multimedia-Datenbanksysteme 39
4.3.3 Deduktive Datenbanksysteme 41
4.3.4 Objekt-orientierte Datenbanksysteme 42
4.3.5 Erweiterbare Datenbanksysteme 44
4.4 Formen der Erweiterbarkeit 48
4.4.1 Neue Datentypen 48
4.4.2 Verschiedene Funktions- und Prozedurerweiterungen in DBS 50
4.4.3 Spracherweiterungen 56
4.4.4 Andere Erweiterungen 58
4.5 DB-Erweiterungen zur Unterstützung von Umweltprojekten 60
4.6 Aufwand zur Erweiterung des DBS und der Anfragesprache 61
5. Interpolation von Meßwerten 65
5.1 Die Art der Einbettung der Interpolationsfunktionen 65
5.2 Die Schnittstelle zum Benutzer 68
5.3 Die Schnittstelle zum DBS 69
6. Die SQL-Erweiterung CSQL 73
6.1 Allgemeine Anforderungen an die Spracherweiterung CSQL 73
6.2 Syntax und Semantik von CSQL 74
6.2.1 Differenzierte Relationentypen 75
6.2.2 Neue Operationen 76
6.2.3 DDL-Erweiterungen 77
6.2.4 Erweiterungen der Anfragesprache 78
6.3 Verträglichkeit mit Standard-SQL 81
6.3.1 Anfrageteil von SQL 82
6.3.2 Datenmanipulationsteil von SQL 89
6.4 Zusätzliche Systemrelationen 90
7. Die Schnittstelle zwischen Interpolationsmodul und DBS 93
7.1 Standard-Zugriffe auf die DB 94
7.2 Auswirkungen auf die Leistung des DBS 95
7.3 Implementierung der Standardzugriffe für Interpolationsfunktionen 96
8. Verarbeitung von CSQL 97
8.1 CSQL im DBS INGRES 97
8.2 CSQL-Benutzerschnittstellen 100
8.3 Analyse der CSQL-Anweisungen 101
8.4 Ablaufsteuerung bei der Interpolation 104
8.5 Entschachtelung und Optimierung von Interpolationsanfragen 107
8.6 Generierung der SQL-Anfragen 110
8.7 Nachbehandlung 111
8.8 Stand der Prototyp-Implementierung 111
8.9 Mögliche Verbesserungen bei direkter Einbettung 114

9. Alternative Syntax und Optimierung 117
9.1 Gründe für einen zweiten neuen Ansatz 117
9.2 Der zweite Ansatz 119
9.3 Auswertung und Optimierungsmöglichkeiten bei Interpolationsanfragen 123
9.3.1 Innere Optimierung 124
9.3.2 Äußere Optimierung 127
9.3.3 Optimierung in CSQL 128
9.3.4 Optimierung im zweiten Ansatz 129
9.4 Vergleich und Bewertung beider Ansätze 130
10. Fallstudie 133
10.1 Die Beispiel-Datenbank 133
10.2 Die Beispiel-Interpolationsverfahren 136
10.2.1 Die Interpolation von Grundwasserständen 136
10.2.2 Die Interpolation von Niederschlag 137
10.2.3 Die Interpolation von Lufttemperatur 139
10.3 Anwendungen auf der Beispiel-Datenbank 141
10.4 Das INGRES-Datenbanksystem 148
10.5 Leistungsmessungen 149
10.5.1 Leistungsmessung im INGRES-System 149
10.5.2 Antwortzeitverhalten im CSQL-System 151
10.5.3 Messung der Beispielanfragen 153
10.6 Bewertung der Ergebnisse 155
11. Zusammenfassung und Ausblick 157

12. Literatur 161

Anhang: Testanfragen und Meßergebnisse 173

1. Einleitung

Seit gut 30 Jahren werden *Datenbanksysteme* (DBS) erfolgreich zur Verwaltung großer Datenmengen eingesetzt, wie sie etwa im Bankwesen, in der Verwaltung großer Unternehmen oder bei weltweit arbeitenden (Flug-) Reservierungssystemen anfallen. Diese konventionellen bzw. kommerziellen Anwendungen von DBS arbeiten alle mit Daten, die relativ einfach strukturiert sind. Durch die Erfolge im konventionellen Bereich ermutigt, wird immer mehr versucht, DBS auch in anderen Bereichen einzusetzen, in denen große Datenmengen anfallen, die aber komplexer strukturiert sind. Gerade bei der Verwaltung und Verarbeitung komplex strukturierter Daten wird maschinelle Unterstützung besonders dringend benötigt.

Einer dieser Bereiche ist die Erforschung von Abläufen in unserer Umwelt sowie die Einwirkungen des Menschen darauf z.B. durch Landwirtschaft, Industrie und Verkehr. Das ist heute eine wichtige Aufgabe, um Ansätze für wirksamen Umweltschutz und die Renaturierung belasteter Gebiete zu finden. Die komplexen Zusammenhänge in unserer natürlichen Umwelt können nur interdisziplinär erfaßt und erklärt werden. Natürlich fallen dabei große Mengen unterschiedlicher Daten an. Zur Verwaltung dieser Daten bietet sich der Einsatz von DBS an. Jedoch sind DBS heute noch überwiegend für die oben erwähnten sogenannten kommerziellen Anwendungen konzipiert. Für einen erfolgreichen Einsatz im Umweltbereich sind Ergänzungen und Erweiterungen erforderlich.

Diese Arbeit begann im Rahmen des interdisziplinären Umwelt-Forschungsprojekts “Naturmeßfeld ‘Horkheimer Insel’”, das von Februar 1988 bis Ende 1990 durchgeführt wurde. (Die ‘Horkheimer Insel’ liegt im Neckar bei Heilbronn.) Dieses Projekt kann als repräsentativ für viele ähnliche Projekte angesehen werden. Das Forschungsprojekt bestand aus sechs Teilprojekten, fünf Anwender-Projekten - aus der Sicht des Informatikers - und dem Datenbank-Teilprojekt. Die Anwender-Projekte waren in den Fachgebieten Geohydrologie, Bodenkunde, Bodenchemie, Pflanzenbau (Landwirtschaft) und Wasserchemie angesiedelt. Das Hauptziel des Gesamtprojekts war die Erforschung des Bodens und Grundwassers unter landwirtschaftlich genutzten Flächen. Basierend auf diesen Erkenntnissen sollten Anbaumethoden gefunden werden, die nicht den schädlichen Einfluß heute üblicher Bewirtschaftung auf die Umwelt haben, und trotzdem qualitativ und quantitativ gute Erträge bringen.

Um diese Ziele zu erreichen, mußten zunächst der Zustand des Testfelds und die natürlichen Vorgänge aufgezeichnet werden. Dazu wurden unterschiedliche Meßgeräte installiert und dann

verschiedene Daten erhoben. Diese Daten wurden später ausgewertet, d.h. es wurden z.B. verschiedene Meßreihen verglichen oder Simulationsmodelle damit durchgerechnet. Dabei mußte in allen Teilprojekten auch oft auf Daten zugegriffen werden, die von anderen Teilprojekten erhoben wurden.

Die Haltung aller im Projekt anfallenden Daten in einer gemeinsamen *Datenbank* (DB) oder in miteinander kooperierenden Datenbanken bot sich also an. Das zu verwendende DBS sollte laut Spezifikation der Anwender mindestens folgende Funktionen erfüllen:

- Kontrollierter Zugriff auf die Daten durch verschiedene Projektpartner ist gewährleistet, jeder Teilnehmer verwaltet aber seine ‘eigenen’ Daten selbst. Es besteht also eine verteilte Datenhaltung.
- Die Konsistenzerhaltung des Datenbestands sowie die Sicherung gegen Hardware- und Software-Fehler ist gewährleistet.
- Die Speicherung erfolgt in tabellenartigen Strukturen (Meßreihen).
- Eine Aggregation der gespeicherten Daten (Auswertung / Statistik) ist einfach möglich.
- Die Archivierung, insbesondere die Auslagerung größerer Datenbestände erfolgt auf offline-Media wie Bänder.
- Die Daten werden für verschiedene Anwendungen in unterschiedlichen Formaten (z.B. fehlende Werte werden berechnet) bereitgestellt.
- Mehrere Meßreihen können flexibel verglichen werden.

Die ersten vier Punkte werden von allen heute kommerziell erhältlichen relationalen DBS erfüllt. Archivierung im oben beschriebenen Sinn wird mit der Integration neuer Speichermedien wie optische Platten und mit der zunehmenden Verwendung von DBS in Aufgabengebieten wie Bürokommunikation in die DB-Technologie immer mehr Eingang finden. Verteilte DBS mit transparenter bzw. funktionaler Verteilung im Sinne der Client-Server-Architektur sind jetzt Stand der Technik.

Defizite bestehen jedoch bei der Bereitstellung der Daten für weiterführende Auswertungen und Anwendungen. Beispiele hierfür sind der Ersatz fehlender Werte, die Berechnung äquidistanter Wertefolgen oder der flexible Vergleich mehrerer Meßreihen. Dafür müssen immer noch spezielle, auf den Einzelfall zugeschnittene Anwendungen und Konvertierungsprogramme geschrieben werden. Nur eine spezielle Weiterentwicklung der DBS, die sogenannten *Erweiterbaren DBS*, zeigen hier Ansätze zu akzeptablen Lösungen. Diese DBS waren aber zur Laufzeit des Projekts nur als Forschungsprototypen verfügbar und wiesen die damit verbundenen üblichen Nachteile auf: Sie waren in ihrer sonstigen Funktionalität nicht ausreichend, nicht auf der gewünschten Hardware verfügbar und es erfolgte keine Wartung. Außerdem sind die Erweiterungsmöglichkeiten kommerzieller erweiterbarer DBS sehr allgemein gehalten und das Anpassen an eine bestimmte Anwendung bzw. Gruppe von Anwendungen erfordert auch einigen Aufwand und Kenntnisse der Funktionalität erweiterbarer DBS. Deshalb sollte sich das Informatik-Teilprojekt auf die o.g. Defizite konzentrieren, ein kommerzielles Standard-DBS verwenden und den Anwendern eine für viele Anwendungen ausreichende Lösung anbieten.

Im Kapitel 2 werden zuerst die Teilprojekte kurz beschrieben, auch das DB-Projekt, und zwar aus der Sicht der Umweltforschung. Die Austattung des Gesamtprojekts, soweit für das DB-Teilprojekt relevant, wird aufgezählt. Im dritten Kapitel werden dann, aus der Sicht des Informatikers oder Datenbankspezialisten, die besonderen Eigenschaften von Umweltdaten und Umweltforschungsprojekten herausgestellt. Daraus werden grobe Anforderungen an ein für Umweltdaten geeignetes DBS aufgestellt.

In Kapitel 4 erfolgt eine Bestandsaufnahme über bestehende Möglichkeiten zur Umweltdaten- und Meßwertverarbeitung. Dabei werden sowohl konventionelle relationale DBS als auch objektorientierte und erweiterbare DBS betrachtet. Verschiedene Systeme und ihre Funktionen werden miteinander verglichen, und ihre Eignung für Umweltdatenbanken und Meßwertverarbeitung wird geprüft.

Im nächsten Kapitel wird dann eine Erweiterung der Funktionalität von DBS, die Interpolation von Meßwerten eingebettet in Datenbankanfragen, begründet und eingeführt. Es werden Syntax und Semantik der Erweiterungen der Anfragesprache festgelegt, sowie andere erforderliche Erweiterungen des DBS, wie zusätzliche Systeminformation, definiert. Das Zusammenwirken mit der DB-Sprache SQL [ANSI86, ANSI89] wird beschrieben. Im siebten Kapitel werden die Schnittstellen der Interpolationsroutinen zum DBS näher spezifiziert, und in Kapitel 8 wird die Verarbeitung der um Interpolation erweiterten Anfragen erläutert. Darauf aufbauend wird die Implementierung der Erweiterungen grob beschrieben.

In Kapitel 9 schließlich werden die Implementierung und mögliche Optimierungen diskutiert. Ein Alternativ-Vorschlag für Syntax und Implementierung zu den in Kapitel 6 bis 8 beschriebenen Erweiterungen wird gegeben, der sich auch einfacher in die im Kapitel 4 beschriebenen Erweiterbaren DBS integrieren ließe, aber vom Endbenutzer mehr Aufwand und Verständnis verlangt. Zur Abrundung wird in Kapitel 10 eine Fallstudie mit Leistungsuntersuchungen präsentiert. Die Ergebnisse einer größeren Anzahl von Leistungsmessungen, die durchgeführt wurden, um herauszufinden, welche Parameter einen deutlichen Einfluß auf das Antwortzeitverhalten haben, sind im Anhang aufgelistet. In Kapitel 11 werden eine Zusammenfassung und ein Ausblick auf weitere mögliche Verbesserungen des Gesamtsystems gegeben.

2. Die Teilprojekte

In diesem Abschnitt werden die Randbedingungen für das Informatik-Teilprojekt im Rahmen eines interdisziplinären Verbundprojekts und damit auch für diese Arbeit beschrieben. Dazu werden kurz alle Teilprojekte vorgestellt, auch das DB-Projekt aus der Sicht der Anwender. Zu jedem Teilprojekt werden die Projektziele genannt. Verschiedene Meßwerte, die zum Erreichen dieser Ziele erhoben werden müssen, werden aufgezählt, so daß ein Überblick über das Gesamtprojekt entsteht und die Vielfalt der Daten, die vom Informatik-Teilprojekt zu verwalten sind, erkennbar wird. Detailliertere Beschreibungen der Projekte finden sich in [Kobu89, BEHL90, PlKa89, KTCP89 und Teut90]. Die Hardware-Austattung der Teilprojekte wird aufgezählt.

2.1 Das Gesamt-Projekt

Das Gesamt-Projekt wurde innerhalb des Schwerpunktprogramms PWAB, *P*rojekt *W*asser-*A*bfall-*B*oden des Landes Baden-Württemberg durchgeführt, das vom Kernforschungszentrum Karlsruhe koordiniert wurde. In allen PWAB-Forschungsvorhaben wurde in interdisziplinärer Arbeit der Einfluß des Menschen auf die Bereiche Wasser und Boden bzw. durch Abfall auf die gesamte Umwelt untersucht.

Das Projekt ‘Naturmeßfeld Horkheimer Insel’ beschränkte sich auf die Bereiche Wasser, insbesondere Grund- und Sickerwasser, und Boden, und zwar unter landwirtschaftlich genutzter Fläche. Auf kleiner Fläche wurden möglichst viele Daten gesammelt, um Boden, Wasser und die Vorgänge darin möglichst genau zu erfassen

Hierbei standen die Auswirkungen von Düngemitteln, Herbiziden und Pestiziden auf die Qualität des Bodens, des Sickerwassers und des Grundwassers im Vordergrund. Der Abbau und Transport dieser künstlich aufgebrachten Stoffe wurden dazu untersucht. Ein Vergleich und die Eichung verschiedener Meßmethoden und -geräte wurden angestrebt. Erforscht wurden vor allem Wechselwirkungen zwischen verschiedenen Pflanzen, Anbaumethoden und den aufgebrachten Stoffen (Dünger, Herbizide, Pestizide). Für vergleichende Untersuchungen wurde das Testfeld in konventionell und in alternativ (umweltschonend) bewirtschaftete Parzellen aufgeteilt.

Auf lange Sicht wird so eine ökologisch und ökonomisch verträgliche Art der Landwirtschaft gesucht.

Bei der großen Anzahl von Projektpartnern, die auf einem relativ kleinen Versuchsfeld zusammenarbeiten, ist eine gute Koordination besonders wichtig. Z.B. muß genau aufgezeichnet werden, wer wo Bodenproben genommen und den Untergrund mit Granulat aufgefüllt hat, damit spätere Proben nicht verfälscht werden. Auch solche Daten müssen gesammelt und schnell und sicher an alle Projektpartner verteilt werden.

2.2 Das Teilprojekt “Geohydrologie”

Die Hauptziele dieses Projekts sind die Schadstoffverfolgung im Untergrund und die Entwicklung eines Grundwasserströmungsmodells für das Gebiet, in dem das Testfeld liegt. Als wesentlicher neuer Aspekt dieses Modells wird dabei der in diesem Gebiet vorherrschende Boden einbezogen, der aus einem Gemisch aus Körnern unterschiedlicher Größe besteht.

Dazu wurde zunächst eine Bestandsaufnahme des Geländes vorgenommen: In einem äquidistanten Raster wurde die Geländehöhe vermessen. An denselben Punkten wurden geoelektrische Messungen durchgeführt. Zusätzlich ließ man Meßbrunnen bohren, in denen seitdem regelmäßig der Grundwassserstand gemessen wird. Die beim Brunnenbau entstandenen Bohrkerne wurden ins Labor gebracht und dort genau untersucht. Zum einen haben die Projektmitarbeiter durch Siebanalysen die Zusammensetzung des Bodens aus Körnern unterschiedlicher Größe in verschiedenen Schichten bestimmt. Zum anderen führten sie sogenannte Permeameter-Versuche durch, um die hydraulischen Eigenschaften des Aquifers (grundwasserführende Schichten des Bodens) zu bestimmen. Es wurden dabei Parameter wie die Porosität (Durchlässigkeit) einschließlich der Durchlässigkeitsbeiwerte und die Turtuosität (Umflußeigenschaften) erhoben.

Auf dem Feld wurden Pumpversuche durchgeführt. Dabei wurde aus einem Brunnen Wasser abgepumpt und gleichzeitig in den umliegenden Brunnen in sehr kurzen Zeitabständen die Absenkung des Grundwasserspiegels gemessen. Zur selben Zeit wurden sogenannte Tracer-Versuche durchgeführt: An einer Stelle wird ein (Farb-) Stoff in das Grundwasser gegeben, und es wird untersucht, wo und nach welcher Zeit dieser Stoff wieder gefunden wird. Genaue Angaben zum Meßprogramm und Beschreibungen der Meßgeräte sind in [KTCP89] zu finden.

2.3 Das Teilprojekt “Bodenkunde”

In diesem Teilprojekt wird nach einer umweltschonenden aber ökonomisch vertretbaren Landwirtschaft gesucht. Dazu wurde das Testfeld in eine konventionell bewirtschaftete und eine umweltschonend bearbeitete Parzelle aufgeteilt. Es wird vor allem der Einfluß der Landwirtschaft auf die Qualität und Quantität des Sickerwassers (Wasser auf dem Weg von der Oberfläche

zum Grundwasser) untersucht mit Schwerpunkt auf dem Düngestoff Nitrat und einigen ausgewählten Herbiziden. Es wurde eine Fruchtfolge für das Testfeld festgelegt, und für beide Parzellen wurden eine unterschiedliche Art der Bearbeitung (z.B. hacken, pflügen, Unkraut jäten) sowie die einzusetzenden Mengen an Dünger und Pflanzenschutzmitteln unterschiedlich bestimmt.

Zur Bestimmung des Nitrat-Gehalts im Boden werden in regelmäßigen Abständen (4 Wochen) Bodenproben auf der gesamten Fläche in drei verschiedenen Tiefen genommen. Die Proben müssen vorschriftsmäßig zwischengelagert und dann analysiert werden. Auf den beiden unterschiedlich bewirtschafteten Parzellen wurde je eine Intensivmeßparzelle eingerichtet, auf der kontinuierlich die Sickerwassermenge, der Niederschlag und die Verdunstung, sowie Wasserspannungen und Saugspannungen (wöchentlich in 6 bis 8 verschiedenen Tiefen) gemessen werden und Bodenwasserproben (14-tägig in 5 verschiedenen Tiefen) genommen werden. Detaillierte Beschreibungen der Meßvorrichtungen und Meßgrößen finden sich in [PlKa89].

2.4 Das Teilprojekt “Bodenchemie”

Das Bodenchemie-Projekt, das erst später in das Verbundprojekt eingegliedert wurde, untersucht die Wechselwirkung zwischen dem Herbizid Triazin und der Bodenflora. Auch hier werden in regelmäßigen Abständen (6 Wochen) Bodenproben auf dem ganzen Feld an insgesamt 104 Rasterpunkten pro Parzelle gezogen, die auf Herbizide hin analysiert werden [PlKa89]. Um die große Menge an anfallenden Proben bewältigen zu können, müssen diese nach den Vorschriften der DFG an der Luft getrocknet, homogenisiert und bis zur Analyse in der Tiefkühltruhe (-20^0C) zwischengelagert werden.

2.5 Das Teilprojekt “Pflanzenbau”

Der Wasserverbrauch einer Pflanze an einem Standort steht im Mittelpunkt der Arbeit im Pflanzenbau-Projekt. Es wird die Fragestellung untersucht, ob durch unterschiedliche Bewirtschaftungsformen - umweltschonend oder konventionell - eine Veränderung des Bodenwasserhaushalts hervorgerufen wird. Dabei werden auch Pflanzenwachstumsmodelle, die als Eingabeparameter im wesentlichen Wetterdaten wie Temperatur, Sonneneinstrahlung und Niederschlag benötigen, validiert [PlKa89].

Für dieses Teilprojekt werden zunächst einmal Wetterdaten und Bodentemperaturen erhoben, teilweise im 10-Minuten-Abstand, teilweise in stündlichen Intervallen. Außerdem werden während der Vegetationsperiode unterschiedliche Parameter an den Pflanzen (in größeren Abständen, etwa 14-tägig) erhoben, wie die Atmungsaktivität (CO_2-Austausch) der Blätter der Pflanzen, ihre Trockenmasse, ihre Gesamtwurzellänge und die Größe ihrer Früchte.

2.6 Das Teilprojekt “Wasserchemie”

Das Hydrochemie-Teilprojekt befaßt sich mit dem Zustandekommen der natürlichen Grundwasserbeschaffenheit in Abhängigkeit von den in der ungesättigten Zone stattfindenden chemischen Prozesse. Dazu werden Bodenproben aus den verschiedenen Schichten im Untergrund entnommen. Das Wasser aus diesen Proben wird abzentrifugiert und dann auf verschiedene Inhaltsstoffe wie z.B. Nitrat untersucht. U.a. wird die Auswirkung des Pflügens auf die Konzentration der verschiedenen Stoffe in der Bodenlösung untersucht.

2.7 Das Teilprojekt “Datenbankunterstützung”

Die Aufgabe des Datenbankprojekts bestand darin, die unterschiedlichen anfallenden Daten aus den Messungen zu strukturieren, zu verwalten, zu sichern und bei deren Verarbeitung Unterstützung anzubieten. Da das Gesamtprojekt als Langzeitprojekt angelegt war, war zu erwarten, daß nach ersten Messungen und Ergebnissen die Meßparameter verändert wurden, neue Meßgeräte, Versuche und sogar Teilprojekte hinzukamen. Es mußte auf einfache Art, also ohne Veränderung bestehender Anwendungen, möglich sein, die hier anfallenden Daten einzugliedern.

Innerhalb dieses Projekts sollte ein Gesamtsystem entstehen, zusammengesetzt aus einer Datenbank-Komponente und Erweiterungen bzw. Anwendungen darauf, das zugeschnitten ist auf

- die Verwaltung und Verarbeitung von Meßwerten und Umweltdaten und ihre Bereitstellung für die weitere Auswertung;
- die Unterstützung von Forschungsprojekten, d.h. Projekten mit Zielsetzungen, die sich im Laufe der Zeit ändern;
- die Unterstützung von verteilter Meßwerterhebung und -verarbeitung [Neug89a].

Die Verwirklichung dieser Ziele erforderte eine Datenmodellierung, die auf die speziellen Eigenschaften der Meßwerte und anderer Umweltdaten eingeht. Eine Archivierung von Daten, die zur Zeit nicht benötigt werden, war geplant. Um auf Projekterweiterungen und -änderungen schnell und flexibel reagieren zu können, wurde erwogen, Versionen von Datenstrukturen einzuführen.

Es wurde ein Konzept für die verteilte Datenhaltung erarbeitet, das es erlaubt, daß jeder Projektpartner seine Daten selbst verwaltet, aber (auf Anforderung) allen anderen lesenden Zugriff erlaubt.

Bei der Bereitstellung der Daten für die Weiterverarbeitung mußte zunächst herausgefunden werden, wie die Daten weiterverarbeitet werden sollten, damit geeignete Erweiterungen des DBS dafür definiert und implementiert werden konnten.

2.8 Hard- und Software-Austattung des Projekts

Die Rechner-Ausstattung jedes Teilprojekts bestand aus einem Rechner der Workstation-Klasse unter einem UNIX-artigen Betriebssystem und dem relationalen DBS INGRES darauf. Die Workstations in den Teilprojekten stammten von drei verschiedenen Herstellern. Zusätzlich waren noch einige PCs vorhanden, die zum Auslesen der Meßgeräte und als zusätzliche Terminals an den Workstations dienten. Bis auf ein Teilprojekt, das über öffentliche und Universitätsnetze nicht zu erreichen war (- hier mußten die Daten auf Bändern, bzw. Streamer-Cartridges ausgetauscht werden -), waren alle Workstations über Ethernet miteinander verbunden. Das Testfeld selbst war für die Dauer dieses Projekts nicht in das Netz integriert. So mußten alle Meßergebnisse auf digitalen Datenaufzeichnungsgeräten mit Speicherkapazität (sog. Datenlogger), Disketten oder Papier aufgezeichnet zu den Projektrechnern transportiert werden.

Da Ergebnisse und Anwendungen aus früheren und anderen Projekten weiterverwendet werden sollten, mußten noch einige zusätzliche Hard- und Softwarekomponenten in die Projektumgebung integriert werden. Hierbei handelte es sich um Auswertungs-, Statistik- und Plotprogramme sowie entsprechende Ausgabegeräte. Zu diesen Programmen mußten Schnittstellen zum DBS geschaffen werden, d.h., die Daten müssen geeignet aus der DB ausgewählt werden können. Das Meßfeld und die Projektdatenbank wurden neu eingerichtet. Auf alte Strukturen brauchte also relativ wenig Rücksicht genommen zu werden. Die Hardware-Ausstattung des Projekts ist in Bild 2.1 dargestellt.

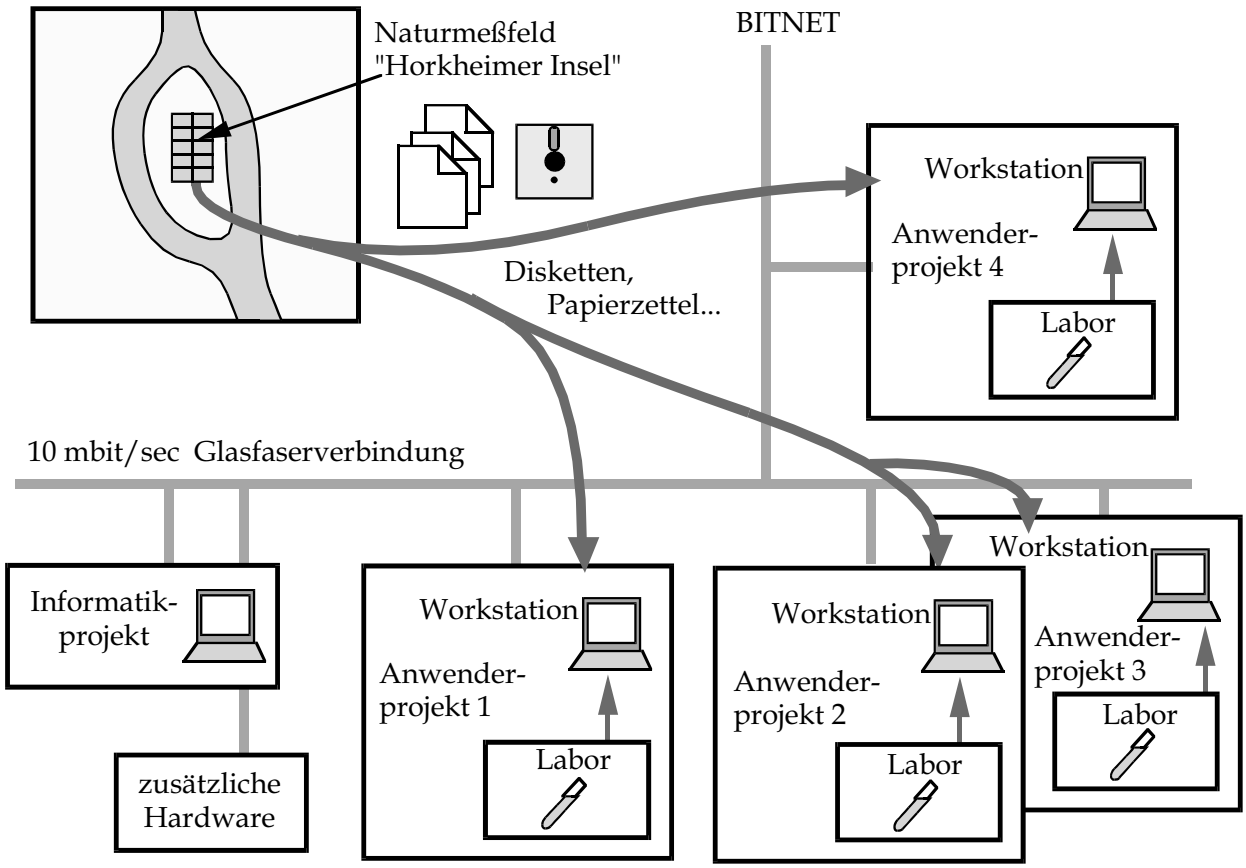


Bild 2.1: Rechner-Ausstattung und Vernetzung des Projekts Naturmeßfeld ‘Horkheimer Insel’

Entscheidend für dieses Projekt war, daß alle Teilprojekte ein ähnliches Betriebssystem und das gleiche DBS verwendeten, damit der Datenaustausch vereinfacht wurde und die Benutzerschnittstelle für alle Teilprojekte gleich aussah. Es mußte außerdem ein kommerziell erhältliches DBS gewählt werden, für das Anpassungen an neue Betriebssystemversionen vom Hersteller auf allen Rechnern garantiert werden konnten, weil schon bald nach Beginn des Projekts ein ständig funktionsfähiges DBS bereitstehen sollte.

3. Meßwerte, Umweltdaten und Umweltprojekte

In diesem Kapitel wird definiert, welche Daten als Umweltdaten bezeichnet werden können. Es werden typische Merkmale und Eigenschaften von Umweltdaten, insbesondere von Umwelt-Meßwerten, aufgezählt und erläutert. Die Charakteristika von Projekten, die im Bereich Umweltschutz angesiedelt sind, werden beschrieben. Alle Eigenschaften werden aus dem Projekt ‘Naturmeßfeld Horkheimer Insel’ abgeleitet und anhand von Beispielen erklärt. Sie treffen aber auch auf viele andere interdisziplinäre natur- und ingenieurwissenschaftliche Projekte zu. Die Schwächen von konventionellen DBS bei der Bearbeitung von Daten aus Umweltprojekten werden aufgedeckt. Am Ende des Kapitels werden Anforderungen an ein System zur Speicherung und Bereitstellung der Meßwerte und Umweltdaten aus dem vorher Gesagten abgeleitet. Dabei wird beschrieben, wie die Eigenschaften der Meßwerte das System beeinflussen können.

3.1 Eigenschaften von Umwelt-Forschungsprojekten

Umweltforschungsprojekte befassen sich mit den Auswirkungen der Eingriffe des Menschen auf die natürliche Umwelt. Die Ziele dienen dem Umweltschutz, das heißt, es werden Wege aufgezeigt, wie der schädliche Einfluß des Menschen verringert werden kann bzw., wie Gegenmaßnahmen ergriffen werden können (s. a. [BaZi95]). Umweltprobleme sind i.a. nur interdisziplinär vollständig erfaßbar und müssen deshalb institutsübergreifend bearbeitet werden. So sind Umweltforschungsprojekte dann meistens interdisziplinäre Projekte, an denen Partner aus verschiedenen Fachrichtungen häufig in räumlich voneinander entfernten Institutionen mitwirken. Die Hardware-Ausstattung, der Kenntnisstand und die Erfahrung mit Datenverarbeitung der verschiedenen Projektpartner sind meistens sehr unterschiedlich. Jedoch möchten alle bereits vorhandene Hard- und Software weiterverwenden bzw. gleichartigen Ersatz bekommen.

Umweltforschungsprojekte sind so strukturiert, daß man das Endziel bzw. die Endziele des Projekts über mehrere Zwischenziele erreicht. Häufig werden in Umweltforschungsprojekten neue Meßverfahren und -geräte erprobt, von denen man noch nicht genau weiß, ob und in welcher Qualität sie die gewünschten Werte liefern. Je innovativer also so ein Forschungsprojekt ist, desto weniger genau weiß man zu Anfang, wie alle Zwischenziele und Endziele erreicht werden

können. Nachdem man erste Messungen durchgeführt hat und erste Teilergebnisse gewonnen hat, werden die Projekt- (Zwischen-) Ziele redefiniert, es kommen neue Versuche, Projektziele und sogar Teilprojekte hinzu. Fast alle diese Änderungen haben Einfluß auf das Datenschema, also die Strukturen, in denen Meßwerte und andere Projektdaten in der Datenbank abgelegt werden.

Nicht nur neue Teilprojekte und Projektziele können Gründe für Änderungen des Datenschemas sein. Wenn Meßgeräte ausfallen, müssen sie sofort durch neue ersetzt werden, damit die Aufzeichnungen möglichst wenig Lücken aufweisen. Häufig können nur Geräte neuerer Bauart beschafft werden, die dann die Daten mit anderer Genauigkeit, Dimension oder sogar etwas andere Meßparameter aufzeichnen. Auf diese unvorhergesehenen Strukturänderungen müssen die Datenbankadministation und die Datenauswertung flexibel und schnell reagieren können.

Kennzeichnend für Umweltforschungsprojekte aus dem natur- und ingenieurwissenschaftlichen Bereich sind also:

- heterogene Hard- und Softwareumgebung,
- neue und revidierte Versuchsumgebungen, Projektziele und Teilprojekte,
- sich fortentwickelnde Datenschemata.

Es ist ein Vorteil, wenn ein Projekt neu eingerichtet wird und nicht die Fortsetzung eines oder mehrerer früherer Projekte ist. Dann gibt es weniger gewachsene Strukturen (organisatorisch und datenverarbeitungstechnisch), die berücksichtigt werden müssen.

3.2 Die Weiterverarbeitung von Meßwerten

Reihen von Meßwerten sind auf den ersten Blick nur Zahlenkolonnen oder Tabellen mit wenig Aussagekraft. Damit man etwas aus ihnen ablesen kann, müssen sie aufbereitet oder ausgewertet werden. Die typischen und häufigsten Arten der Weiterverarbeitung sind die Visualisierung von Meßreihen und die Modellbildung oder Simulation.

3.2.1 Visualisierung von Meßreihen

Wenn die Ergebnisse von Messungen übersichtlich dargestellt werden sollen, insbesondere auch fachfremden Personen, ist die graphische Darstellung der Meßwerte wichtig. Dies kann z.B in Karten für räumliche Verteilungen oder in Diagrammen für zeitliche Veränderungen geschehen. Ein Beispiel für eine Karte ist eine Karte zur Bodenbeschaffenheit; ein Beispiel für ein Diagramm ist der Jahresniederschlag aufgetragen als Balkendiagramm für die Monate an einem Ort. Solche Karten und Diagramme dienen auch als Argumentationshilfen, wenn Ergebnisse aus Umweltprojekten Entscheidungsträgern vorgelegt werden, um umweltfreundliche politische Entscheidungen zu bewirken. Oder wenn die Ergebnisse einer breiten Öffentlichkeit zugänglich gemacht werden, um ein umweltfreundlicheres Verhalten zu motivieren.

Eine andere häufige Art der Weiterverarbeitung von Meßwerten sind Statistiken. Im Unterschied zum Diagramm wird hier bereits ein Zusammenhang zwischen mindestens zwei Meßgrößen hergestellt, z.B. die Atmungsaktivität der Grünteile von Pflanzen im Verhältnis zur Sonneneinstrahlung. Statistiken werden für die gleichen Zwecke wie Karten und Diagramme benötigt.

Zur einfachen Erzeugung von Statistiken und einfachen Diagrammen gibt es bereits einige kommerzielle Softwareprodukte. Auch relationale DBS bieten meist ein geeignetes Werkzeug dazu an. Allen diesen Programmen ist aber gemein, daß sie korrekte und vollständige Eingabedaten erwarten und auch keine Vollständigkeits- oder Konsistenzprüfungen übernehmen.

3.2.2 Modellbildung und Simulation

Anspruchsvolle Anwendungen auf Meßwerten sind Simulationsprogramme und Modellrechnungen unterschiedlichster Art. Diese Anwendungen berechnen für einen Ausgangszustand und für einige über die Dauer der Simulation gegebene Parameter das Verhalten von einem oder mehreren anderen Parametern. Beispiele sind ein Grundwasserströmungsmodell oder ein Pflanzenwachstumsmodell. Ein Grundwasserströmungsmodell berechnet aus den Bodengegebenheiten und dem aktuellen Niederschlag den Fluß des Grundwassers. So läßt sich z.B. die Verbreitung von Schadstoffen im Grundwasser vorausberechnen. Ein Pflanzenwachstumsmodell berechnet die Entwicklung einer Pflanze abhängig von verschiedenen Wetterparametern wie Temperatur, Niederschlag und Sonneneinstrahlung. Diese Programme können aber häufig nicht direkt mit den Meßwerten arbeiten, sondern benötigen sie in speziell aufbereiteten Formaten.

Alle Meßwerte und andere Umweltdaten sollen so gespeichert bzw. aufbereitet werden, daß die Weiterverarbeitung möglichst effizient unterstützt wird.

3.3 Eigenschaften von Umweltdaten und Meßwerten

An dieser Stelle soll zunächst kurz definiert werden, welche Daten als *Umweltdaten* bezeichnet werden. Der Begriff Umweltdaten wird hier als Gegensatz zu kommerziellen oder konventionellen Daten verstanden. Umweltdaten sind alle Daten, die in irgendeiner Weise die (natürliche) Umwelt beschreiben wie z.B. Wetterdaten. Kommerzielle Daten dagegen sind Daten wie Kontostände, Lagerbestände oder Gehalt von Personen (s.a. [BaZi95]). Die Grenzen zwischen konventionellen und Umweltdaten sind z.T. fließend und hängen auch von den Anwendungen ab: Landkartendaten, die als Grundlage zu Stadtplänen und Straßenkarten nur zur Orientierung dienen, sind konventionelle Daten; werden sie aber zur Berechnung und Veranschaulichung der Flächenversiegelung herangezogen, sind es Umweltdaten. Meßwerte gehören genau dann zu den Umweltdaten, wenn sie aus einem natülichen Vorgang stammen (s.u.).

Praxisorientierte natur- und ingenieurwissenschaftliche Untersuchungen gelten in der Regel den Abhängigkeiten bestimmter (Umgebungs-) Parameter natürlicher Vorgänge untereinander. Nachdem die Abhängigkeiten erkannt wurden, wird versucht, sie möglichst exakt mathematisch zu beschreiben. Zunächst werden der Vorgang bzw. seine Parameter, genauer Folgen seiner Parameter erhoben, also gemessen. Um einen natürlichen Vorgang vollständig und genau zu erfassen, sind eine Reihe von Versuchen und eine größere Anzahl von Meßgeräten notwendig. Dabei entstehen Daten in vielen unterschiedlichen Formaten, von unterschiedlicher Qualität und Menge. Die Anzahl der Werte ist bei einigen Messungen, z.B. Grundwasserpegeln, relativ gering, bei anderen, wie z.B. einer Wetterstation aber außerordentlich hoch. Trotz der Vielfalt gelten für die gewonnenen Meßwerte und Meßreihen bestimmte gemeinsame Eigenschaften, die näher betrachtet werden sollen.

In [LSBM83] wurde zum ersten Mal beschrieben, daß Meßwerte besondere Eigenschaften aufweisen und daß Raum- und Zeitattribute im Zusammenhang mit Meßwerten sich stark von solchen Attributen in kommerziellen Anwendungen unterscheiden. In kommerziellen Anwendungen werden nur diskrete Zeitpunkte, Orte oder Oberflächen benötigt; und nur solche Attribute können in konventionellen DBS modelliert werden. Für sich über die Zeit kontinuierlich ändernde Daten wird kein Modellierungskonzept angeboten. Deshalb sind konventionelle DBS bisher für die Verarbeitung von Umweltdaten oder Meßwerten und Anwendungen darauf auch nur bedingt geeignet.

Auch in anderen Projekten und Arbeitsgruppen wurde schon auf die mangelnde Eignung konventioneller DBS für umweltbezogene Daten hingewiesen und Lösungen für Teilgebiete werden angeboten. Bezüglich temporaler Aspekte wurden die Anforderungen und die Mängel konventioneller DBS schon ausführlich in [JaMa86], [SnAh85] und [Gadi88] beschrieben. Eine besondere Behandlung von räumlichen Daten wird u.a. im PROBE Projekt [Oren86] und in dem ‘Geo-Kernel-System’ des DASDBS [ScWa86, SPSW90, Wolf90] angeboten. Auf das System DASDBS wird im Kapitel 4.3 im Zusammenhang mit erweiterbaren DBS noch näher eingegangen werden. Im Zusammenhang mit geographischen Datenbanken und einem Geo-Objektmodell wird auch in [LiNe86] die Forderung nach einer besonderen Behandlung von Raum- und Zeit-Attributen aufgestellt. Speziell für die Speicherung geographischer Daten werden die sogenannten *Geographischen Informationssysteme* (GIS) angeboten. Die GIS werden ebenfalls in Kap. 4 näher beschrieben. Inzwischen bieten auch einige kommerzielle relationale DBS Erweiterungen für raumbezogene Daten an, etwa die sog. “DataBlades” in Informix [Info96] oder ein entsprechender “Spatial Extender” in DB2 [Davi98]. Diese Systeme werden ebenfalls in Kap. 4.3 mit betrachtet.

3.3.1 Der Kontext von Meßwerten

Zu jedem Meßwert gehören Angaben darüber, wann, wo, wie und unter welchen Bedingungen er erhoben wurde, denn nur so hat er eine Bedeutung. Diese Beschreibung, der *Kontext*, besteht, genauer betrachtet, aus der Beschreibung des gemessenen Parameters selbst und aus der Beschreibung des Versuchs, innerhalb dessen der Wert aufgezeichnet wurde. Der einzelne Para-

meter wird mindestens durch seine Dimension, Meßgenauigkeit und die Basis, gegen die gemessen wird, beschrieben. Es können noch weitere Parameter hinzukommen, etwa sich ändernde Randbedingungen. Zur Meßbasis bei Umweltmeßwerten gehören immer der Ort und die Zeit, zu der gemessen wird. Orts- und Zeitgrößen können in Punkten oder über Intervalle aufgezeichnet werden. Punktförmige Basiswerte hat z.B. die Lufttemperatur, die zu bestimmten Zeitpunkten gemessen wird und haben Grundwasserstände, die an bestimmten Punkten auf einer Fläche gemessen werden. Intervalle als Basis gibt es beim Niederschlag, der über einen Zeitraum aufsummiert wird, oder bei Bodenproben, die zylinderförmig sind.

Manchmal kann die explizite Aufzeichnung einer oder mehrerer Dimensionen entfallen, weil sie sich während des Versuchs oder Projekts nicht ändern. So ist z.B. der Ort bei Laborversuchen oder bei Wetterdaten, die für ein ganzes Gebiet gelten, nicht relevant. Ebenso muß die Zeit bei geologischen Formationen nicht aufgezeichnet werden, die sich während einer Versuchslaufzeit von beispielsweise 10 Jahren nicht ändern. Wenn in Laborversuchen Ort und Zeit nicht relevant sind, kann es sein, daß die Meßbasis nur aus Randbedingungen wie Luftdruck oder Umgebungstemperatur besteht.

Die Versuchsbeschreibung besteht meistens aus der Beschreibung der Versuchsanlage mit den einzelnen Geräten. Außerdem gehören Angaben dazu, die den Versuch als Ganzes betreffen, wie die Gesamtlaufzeit des Versuchs und der Versuchsleiter. Zusätzlich müssen alle Parameter, die erhoben werden, und schon bekannte Abhängigkeiten zwischen ihnen gespeichert werden.

Dieselben Meßparameter können unter idealisierten Bedingungen im Labor, im kontrollierten Feldversuch und in der freien Natur erhoben werden. Im Idealfall werden die Parameter in allen drei Umgebungen gleichzeitig oder nacheinander gemessen. Dabei entstehen Werte von sehr unterschiedlicher Qualität. Die Meßgenauigkeit schwankt abhängig von den verwendeten Meßgeräten und dem Einfluß von Störparametern. Auch das muß alles im Kontext vermerkt werden.

3.3.2 Wertebereiche für Meßwerte

Um die Besonderheiten von Wertebereichen von Meßwerten zu verdeutlichen, werden hier zunächst einige Beispiele genannt. Ein Beispiel ist der Grundwasserstand, der in einer Anzahl von Meßpegeln auf dem Naturmeßfeld in wöchentlichen Abständen ermittelt wird. Zwischen den Meßpegeln ist der Grundwasserstand ähnlich dem in den Meßpegeln aber nicht genau gleich. Er kann durch Interpolation mit den umliegenden Meßwerten abgeschätzt werden. Außerdem oder zusätzlich kann auch der Grundwasserstand zwischen den Meßzeitpunkten berechnet werden. Andere Beispiele für Meßwerte sind Bodenproben zur Bestimmung der Zusammensetzung des Untergrunds, die in einem Raster in verschiedenen festgelegten Tiefen gezogen werden, oder Wetterdaten wie Außentemperatur, die in diskreten Zeitintervallen aufgezeichnet werden.

Allen diesen Meßparametern und ihren Basisgrößen sind die folgenden Eigenschaften gemein:

- Sowohl Meßdaten als auch ihren Basiswerten liegen immer numerische Wertebereiche wie Gleitpunktzahlen, Prozentangaben oder Zeitangaben (Sekunden und Bruchteile davon) zu Grunde. Ihr eigentlicher Wertebereich ist eingeschränkt auf ein Intervall daraus, etwa [Meßbeginn, Meßende]. Zwar kann als Basis in manchen Fällen auch eine Bezeichnung angegeben werden, wie Parzelle “P11", aber hier sollten dann in derselben Datenbank auch die Koordinaten der Parzelle, etwa als Gleitpunktzahlen vermerkt sein. Alle diese Wertebereiche weisen die Eigenschaften dichte Belegung und Stetigkeit auf. Es gibt also keine undefinierten Stellen. (Dies gilt zumindest für den praktischen Fall, wenn Gleitpunktzahlen mit einer begrenzten Anzahl Dezimalstellen hinter dem Komma verwendet werden.) Die Wertebereiche (*domains*) für Meßparameter und Basisgrößen dazu lassen sich also ähnlich wie die rationalen Zahlen (**Q**) definieren:

  $D_i \subset \mathbf{Q}$ bzw. $D_i \subset \{ \frac{x}{y} \mid x \in \mathbf{Z}, y \in \mathbf{N} \}, i = 1, ..., n$

- Umweltbezogene Meßwerte sind i.a. Stichproben eines kontinuierlichen technischen Vorgangs oder eines kontinuierlich laufenden natürlichen Prozesses. Im Idealfall sind die Meßstellen punktförmig für jede Basisgröße, also “Meßpunkte”. Gelegentlich müssen aus meßtechnischen Gründen kleinere Intervalle gebildet werden: Eine Bodenprobe muß ein gewisses Volumen haben, damit man sie analysieren kann, und der Niederschlag muß ein paar Minuten lang auf einer gewissen Fläche gesammelt werden, damit Aussagen bezüglich der Menge gemacht werden können. Hier lassen sich die Meßparameter dann aber auf punktförmige Basiswerte zurückrechnen. Das heißt z.B. für das Niederschlagsbeispiel, daß der Niederschlag zwar auf einer mehrere cm^2-großen Fläche gesammelt wird, daß der ermittelte Wert z.B. in der Einheit “Liter pro cm^2 pro Stunde (oder Minute)” aber für jeden Punkt dieser Fläche gilt. Ebenso läßt sich für jeden Zeitpunkt des (zeitlichen) Meßintervalls sagen, daß der Niederschlag mit einer Stärke fiel, die einer bestimmten Anzahl “Liter pro cm^2 pro Stunde” entspricht.

- Für jede mögliche Kombination des Zeitattributs, der räumlichen Koordinaten oder anderer Basisgrößen innerhalb des Versuchsrahmens gibt es genau einen Wert für die Meßgröße, den kontinuierlichen Prozeß. Sind die Meßwerte z.B. Wetterdaten, so gibt es für jeden Zeitpunkt und Ort Werte für Lufttemperatur, Niederschlag, Luftdruck etc., unabhängig davon, ob diese Werte aufgezeichnet werden. Soll die Festigkeit eines Metalls bestimmt werden, gibt es für jede Kombination aus Temperatur und Druck einen Wert, ebenfalls unabhängig davon, ob dieser Wert auch im Versuch ermittelt wird. (Wenn es undefinierte Werte gibt, heißt das i.a., daß ein ungeeigneter Wertebereich bzw. ein ungeeignetes Intervall aus dem Wertebereich gewählt wurde.) Meßgrößen behalten keinen konstanten Wert über ein längeres zeitliches oder räumliches Intervall. Es gibt aber auch keine sprunghaften Änderungen. Also gibt es nur geringfügige Änderungen der Meßgröße zwischen nahe beieinander liegenden Werten (bezüglich der Basisgrößen). Dieser Vorgang läßt sich als eine stetige Funktion f, die die *Basis*, das sind Ort und/oder Zeit und evtl. weiterer Kontext, in den Meßwertbereich abbildet, beschreiben:

f: $b_1 \times b_2 \times ... \times b_n \rightarrow m$, b_i Basisgrößen, m Meßwert; $b_i \in D_i$, $m \in D_m$.

Die zugehörigen Basiswerte identifizieren jeden Wert bezüglich dieser Funktion eindeutig. Das entspricht genau Relationen mit Schlüsseln in der relationalen Algebra und in den relationalen Datenbanken.

Einzelne Kombinationen von b_i mit den dazugehörigen m werden explizit erfaßt bzw. gemessen und in den *Relationen* R (b_1, b_2, ... b_n, m) als Tupel $r_j = (b_{1j}, b_{2j}, ..., b_{nj}, m_j) \in R$ der Datenbank gespeichert. Die b_i, die m und damit das ganze Tupel eindeutig identifizieren, sind der *Schlüssel* der Relation R. Es können natürlich immer nur eine begrenzte Anzahl Meßwerte m_j erfaßt und eine begrenzte Anzahl Tupel r_j in der Datenbank gespeichert werden. Für viele Meßgrößen sind jedoch Algorithmen oder Verfahren bekannt, die nicht explizt erfaßte Zwischenwerte mit hinreichender Genauigkeit aus den umliegenden Meßpunkten berechnen können. Dies ist eine ähnliche Eigenschaft wie die Kontinuität für das Repräsentationsschema räumlicher Objekte, das von [Guen88] gefordert wird. Deshalb werden die Meßgrößen im folgenden auch als *kontinuierlich* bezeichnet.

3.3.3 Beziehungen zwischen Meßwerten

Wie erwähnt, sind zum vollständigen Erfassen eines natürlichen Vorgangs eine Reihe von Versuchen mit vielen Meßreihen notwendig. Bei den Beziehungen zwischen all diesen Meßreihen und Meßwerten gibt es zwei Arten: einfache Beziehungen über den Kontext und komplexe Beziehungen, die oft noch nicht bekannt oder erforscht sind. Die einfachen Beziehungen sind entweder hierarchisch, etwa Projekt - Versuch - Messung. Oder es sind Querbeziehungen über Orts- und/oder Zeitangaben oder über andere Attribute aus dem Kontext.

Hierarchische Beziehungen gibt es nicht nur bei Meßwerten sondern in sehr vielen DB-Anwendungen. In relationalen Datenbanken werden sie mit dem sog. *Fremd-* oder *Sekundärschlüssel*-konzept modelliert. Dabei enthält die übergeordnete Relation auch die Schlüsselattribute der untergeordneten Relation (nicht unbedingt als Schlüssel für die eigenen Tupel), um damit auf die ganzen Tupel der untergeordneten Relation zu verweisen. Dabei müssen übergeordnete Relationen nicht unbedingt Meßwerte enthalten. Für zwei Relationen R_1 und R_2 mit den Tupeln

$r_1 = (a_{11}, a_{12}, ..., a_{1n}) \in R_1$
$r_2 = (a_{21}, ..., a_{2m}, b_1, ..., b_k, m_2) \in R_2$

muß für eine hierarchische Beziehung gelten: $a_{2j} \subseteq a_{1j} \subset D_j$ für alle a_{ij}, aus denen sich der Schlüssel bzw. Fremdschlüssel zusammensetzt. Dabei befindet sich a_{1j} in der übergeordneten Relation. Die Relation R_1 könnte zum Beispiel einen Versuch (Grundwassestandsmessungen an verschiedenen Meßstellen) beschreiben, während R_2 eine einzelne Meßreihe (Messungen an einem Pegel) zu diesem Versuch enthält.

Unter einer Querbeziehung soll hier verstanden werden, daß beide Meßreihen im Sinne der o.g. Hierarchie auf einer Ebene liegen und damit gleichwertig sind. Messungen, die dieselben Basisattribute haben, z.B. Weltkoordinaten, können über diese Attribute miteinander in Beziehung gebracht werden. Bei einer Querbeziehung zwischen zwei Meßreihen als Relationen R_1 und R_2 mit den Tupeln

$r_1 = (b_{11}, b_{12}, ..., b_{1n}, m_1) \in R_1$
$r_2 = (b_{21}, b_{22}, ..., b_{2k}, m_2) \in R_2$

muß gelten: $\left.\begin{matrix} b_{2i} \in D_i \\ b_{1i} \in D_i \end{matrix}\right\} \Rightarrow R_1 \mathit{rel} R_2 , r_1 \times r_2$

Das heißt, wenn zwei Meßrelationen R_1 und R_2 Basisattribute haben, die aus demselben Wertebereich stammen (z.B. Weltkoordinaten), dann lassen sich R_1 und R_2 über diese Basis in Beziehung setzen und die einzelnen Tupel r_1 und r_2 über die Werte dieser Basis verknüpfen. Im Beispiel “Horkheimer Insel” können Wasserstände und Bodenproben über die Koordinaten verknüpft werden.

Manchmal hängen Meßparameter auch auf eine komplexere Weise voneinander ab, z.B. ist ein Parameter eine Funktion eines anderen Parameters, häufig mit Basiswerten als weitere Funktionsparameter:

$m_2 = f_i (m_1)$ bzw. $f_i: m_1 \times b_{11} \times ... \times b_{1j} \rightarrow m_2$

Ein Beispiel hierfür ist der Grundwasserstand auf der Horkheimer Insel an verschiedenen Punkten: Er ist abhängig vom Wasserstand der Oberflächengewässer (Neckar und Kanal), dem aktuellen Niederschlag und der Zusammensetzung des Bodens.

Das Herausfinden weiterer komplexer Beziehungen ist häufig Gegenstand der Forschung. Es wird untersucht, welche Parameter sich in Abhängigkeit von anderen Meßparametern ändern, also aus diesen anderen Parametern abgeleitet werden können. So können die Auswirkungen künstlicher Veränderungen der Umwelt vorausgesagt werden oder Gegenmaßnahmen gegen unerwünschte Folgen bereits erfolgter Veränderungen ergriffen werden.

Bei der Beschreibung von Meßreihen und Umweltdaten bietet sich die Darstellung im Relationenmodell (und die spätere Speicherung in einem relationalen DBS) an. Auch die von Anfang an bekannten Beziehungen zwischen den Meßwerten lassen sich so darstellen. Es müssen allerdings zusätzliche Informationen vorhanden sein, was Meßwert und was Basis ist.

3.3.4 Komprimierung und Archivierung

Gemessene Werte ändern sich nicht und veralten auch nicht. In vielen Fällen gewinnen Meßwerte im größeren (zeitlichen) Zusammenhang, etwa über eine Versuchslaufzeit von zehn Jah-

ren, an Bedeutung. Deshalb werden auch in Datenbank- oder Datenhaltungssysteme für Meßdaten stets nur Werte eingefügt. Änderungen und Löschvorgänge kommen praktisch nicht vor. Das Gesamtvolumen der Daten wächst dabei ständig. Die Komprimierung und die Archivierung von zur Zeit nicht (mehr) benötigten Daten sind daher wichtige Aufgaben. Dabei muß auch der Kontext der Daten mit archiviert werden, damit man sie später noch richtig interpretieren kann.

Zu Anfang einer Meßreihe ist oft noch nicht bekannt, in wie großen Abständen Meßwerte erhoben werden müssen, um Änderungen gut erkennen zu können. Deshalb werden zu Anfang einer Meßreihe die Werte oft sehr dicht erhoben. Weiß man später, in welchen Abständen die Werte erhoben werden müssen, um Änderungen hinreichend früh zu erkennen, müssen die anfänglich erhobenen Werte komprimiert werden, i.a. durch Aggregationsoperationen wie z.B. Tagessummen oder -durchschnittswerte. Trotzdem werden die Originalwerte i.a. aufbewahrt, d.h. archiviert: Man könnte sie noch einmal für andere Zwecke betrachten wollen oder, bei Zweifeln an der Korrektheit der Komprimierung, diese nachvollziehen wollen.

Häufig müssen alte Meßwerte aus früheren Versuchen einbezogen werden bzw., man kann davon ausgehen, daß die Werte, die jetzt erhoben werden, auch in späteren Projekten mit herangezogen werden. Deshalb sollte eine systemunabhängige Art der Archivierung gewählt werden, die es auch nach längerer Zeit, also nach zehn, 20 oder noch mehr Jahren mit neuer Software erlaubt, die Daten wieder einzulesen. Systemunabhängig heißt hier unabhängig von einem bestimmten Betriebssystem und speziellen Komprimierungsmethoden, unabhängig von speziellen Datenträgern und auch unabhängig von einem bestimmten Datenbanksystem. Z.B. bieten DBS i.a. die Möglichkeit, Daten entweder in ASCII-Dateien zu entladen oder in einer internen DBS-Repräsentation. Hier sind ASCII-Dateien Daten der internen Repräsentation des DBS vorzuziehen, auch wenn die letzteren wesentlich weniger Speicherplatz benötigen. Die ASCII-Dateien lassen sich mit relativ wenig Aufwand in das DBS eines anderen Herstellers laden oder in ein anderes Programm, z.B. ein Statistik-Paket. Notfalls lassen sie sich auch direkt mit Hilfe eines Editors lesen. Für die Anwender ist das oft sehr wichtig, denn das Erheben der Meßwerte ist oft sehr arbeitsaufwendig, und sie wollen nicht die Kontrolle über ihre Daten verlieren.

Werte aus Messungen, die im zeitlichen Zusammenhang unter natürlichen Bedingungen (im Feld) durchgeführt wurden, dürfen nicht verloren gehen, denn sie können nicht wiederholt oder nachträglich erhoben werden. Sie benötigen eine besonders sorgfältige Sicherung. Und bei jedem Wechsel von Betriebssystem oder Hardware (Speichermedien) sollte man sicherstellen, daß alle Daten später noch gelesen werden können. Im Zweifelsfall müssen sie umformatiert oder umgespeichert werden.

3.4 Aufbereitung der Daten: Interpolation

Von der Speicherung in einem DBS erwartet man nicht nur sicheres ‘Aufbewahren’ der Daten, sondern auch eine geeignete Bereitstellung bzw. Aufbereitung. In diesem Abschnitt wird ge-

zeigt, daß diese Erwartungen für die zuvor in Kap. 3.3 geschilderten Fälle nicht ganz erfüllt werden. Dazu werden (Standard-) SQL-Anfragen konstruiert, die dem Gewünschten möglichst nahe kommen. Danach wird beschrieben, wie Abhilfe geschaffen werden kann, nämlich durch die Integration von Interpolation in das DBS und in die Anfragesprache.

3.4.1 Aufbereitung für spontane interaktive Anfragen

In konventionellen DBS ist es nicht möglich, einen Wert - auch aus einem kontinuierlichen Vorgang - an einem bestimmten Meßpunkt abzufragen, wenn dieser Wert nicht genau an diesem Punkt aufgezeichnet wurde. Z.B. kann man man aus einer Temperaturmeßreihe, die halbstündliche Werte um xx.10 Uhr und xx.40 Uhr enthält, niemals die Werte zur vollen Stunde abfragen. Meßwerte werden in der Praxis fast nie genau an den Orten und zu den Zeitpunkten aufgezeichnet, an denen sie später benötigt werden, siehe Bild 3.1.

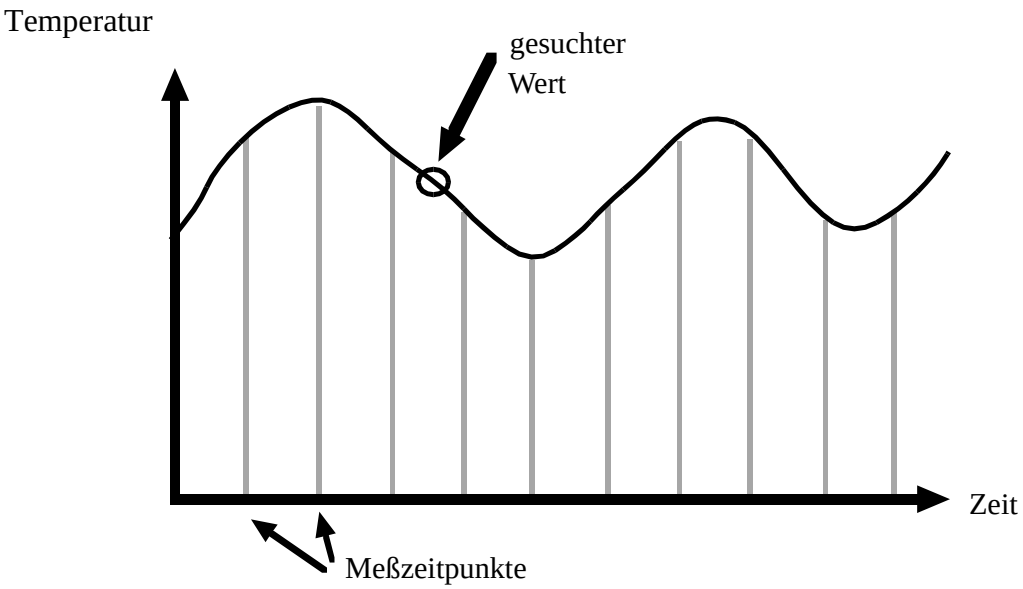


Bild 3.1: Beispiel für eindimensionale Interpolation (zeitlich)

Dies bedeutet, daß der (interaktiv arbeitende) Benutzer immer genau wissen muß, an welchen Orts- oder Zeitpunkten Meßwerte aufgezeichnet wurden. Weiß er dies nicht, muß er sich mit umständlichen Intervallabfragen behelfen. Er kann nicht einfach den nächsten (oder vorhergehenden) Wert nach 10.00 Uhr abfragen, denn das Relationenmodell und damit SQL kennen keine Tupelordnung und also auch keinen ´nächsten´ Wert. Intervallabfragen können in manchen Fällen bei ungeeignet gewählten Intervallbreiten sehr verzerrte Ergebnisse liefern. Weiß der Benutzer zur Temperaturmeßreihe, gespeichert als temperatur (zeitpunkt, meßwert) nur, daß halbstündlich aufgezeichnet wird, könnte er, wenn er den Wert um 10.00 Uhr sucht, die umliegende halbe Stunde in SQL so abfragen:

```
SELECT meßwert
FROM   temperatur
WHERE  zeitpunkt BETWEEN '01.05.91 09:45:00' AND '01.05.91 10:14:59'
```

In diesem Fall wird der um $10.\underline{10}$ Uhr aufgezeichnete Wert geliefert. Je nachdem, wie schnell sich die Temperatur zwischen $10.\underline{00}$ Uhr und $10.\underline{10}$ Uhr geändert hat, weicht der gelieferte Wert von der Realität ab. Soll jetzt die Temperaturschwankung mit einbezogen werden, könnten die Werte aus der ganzen Stunde um $10.\underline{00}$ Uhr betrachtet werden. Wenn der Benutzer das Ergebnis weiterverarbeiten will, möchte er jedoch weiterhin nur einen Ergebniswert. Das könnte in SQL so formuliert werden:

```
SELECT  AVG (meßwert)
FROM    temperatur
WHERE   zeitpunkt BETWEEN '01.05.91 09:30:00' AND '01.05.91 10:29:59'
```

Weil hier nur die Standard-AVG-Funktion (arithmetisches Mittel) zur Verfügung steht, wird jedoch der um $9.\underline{40}$ Uhr gemessene Wert zu stark gewichtet. Dieses Problem verstärkt sich bei unregelmäßig gemessenen Größen, bei großen Meßintervallen, bei mehrdimensionalen Meßbasen und wenn die Anfrageintervalle zu groß oder zu klein im Verhältnis zu den Meßintervallen gewählt werden.

3.4.2 Der Vergleich von zwei Meßreihen

Zwei Meßreihen, die - im Versuchskontext gesehen - in demselben Zeitraum an demselben Ort stattfinden, können i.a. nicht mit Hilfe der für diesen Zweck geeigneten relationalen Operatoren, dem natürlichen oder Equi-Verbund, verglichen werden, weil ihre genauen Meßorte und -zeitpunkte fast nie übereinstimmen bzw. gar nicht übereinstimmen können. Möchte der Betreiber eines Meßfeldes z.B die Grundwasserstände zu einer bestimmten Serie von Bodenproben ermitteln, so kann er dies nur über benachbarte, fest installierte Meßpegel tun. Die Ortskoordinaten werden also nie genau übereinstimmen (siehe Bild 3.2).

Die Meßwerte von Bodenproben und Grundwasserständen könnten für das oben gezeigte Beispiel in Relationen mit den folgenden Schemata gespeichert werden:

```
bodenproben (probe_id, stoff, meßwert, x_koord, y_koord, zeitpunkt)
gw_stände (pegel_id, meßwert, x_koord, y_koord, zeitpunkt).
```

Der Benutzer möchte nun zu jeder Bodenprobe den passenden Grundwasserstand an der Stelle der Bodenprobe ermitteln. Mit SQL könnte man hier nur mit der folgenden Anfrage zu einem evtl. brauchbaren Ergebnis kommen:

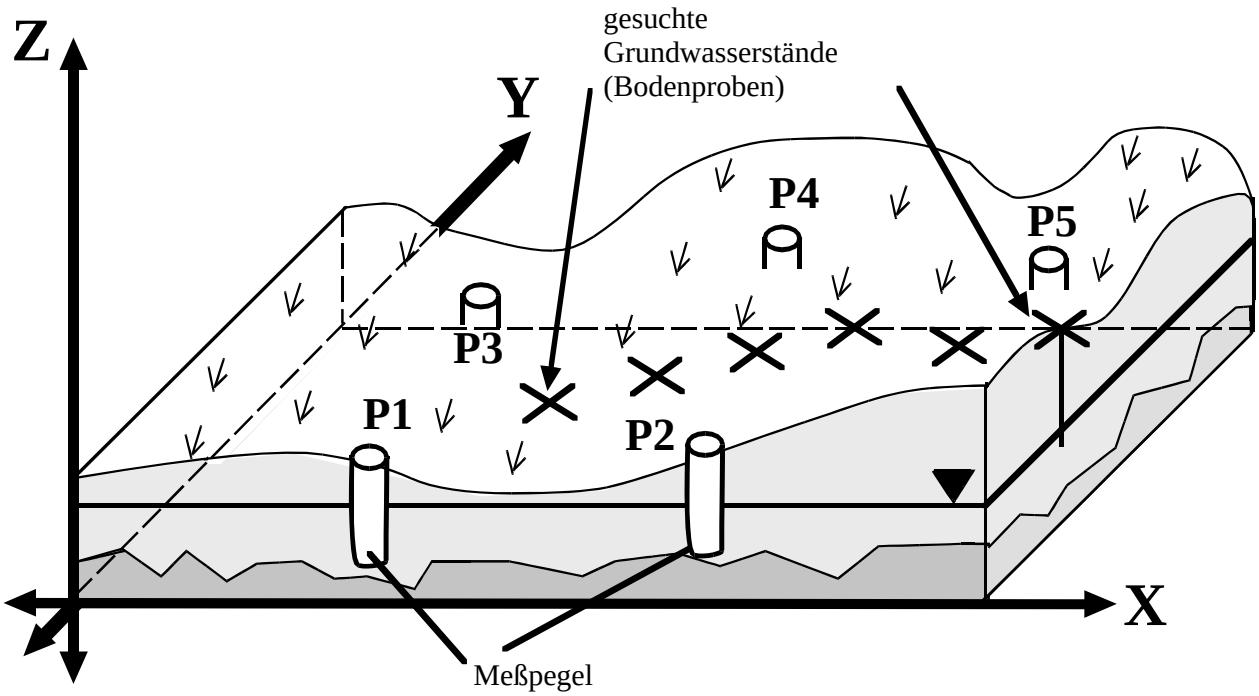


Bild 3.2: Beispiel für zwei-/ mehrdimensionale Interpolation (räumlich)

```
SELECT AVG (g.meßwert), b.meßwert, b.x_koord, b.y_koord, b.zeitpunkt
FROM   bodenproben b, gw_stände g
WHERE  g.x_koord    BETWEEN b.x_koord - 5     AND b.x_koord + 5     AND
       g.y_koord    BETWEEN b.y_koord - 5     AND b.y_koord + 5     AND
       g.zeitpunkt  BETWEEN b.zeitpunkt - 24  AND b.zeitpunkt + 24  AND
       b.stoff = 'Nitrat'
GROUP BY b.meßwert, b.x_koord, b.y_koord, b.zeitpunkt
ORDER BY b.x_koord, b.y_koord, b.zeitpunkt
```

Der Benutzer läßt hier den Durchschnitt aller der Grundwasserstandsmessungen bilden, die in einem 10*10 Maßeinheiten großen Quadrat um die Bodenprobe liegen und in den 24 Stunden vor oder nach dem Ziehen der Bodenprobe gemessen wurden. Um diese Anfrage stellen zu können, muß der Benutzer aber wieder die räumlichen und zeitlichen Meßintervalle kennen. Wenn die Intervalle sehr unterschiedlich sind, wie z.B. die Abstände der Grundwasser-Meßpegel im in Kap. 2 beschriebenen Projekt, gibt es Probleme, etwa zu viele leere Intervalle. Weitere Probleme entstehen hier, weil räumlich und zeitlich nahe Werte gemischt werden. Wurde z.B der dichteste Pegel P5 für die Bodenprobe mit der größten x-Koordinate in Bild 3.2 als einziger am Tag der Bodenprobe nicht abgelesen, sondern jeweils zwei Tage vorher und nachher, werden diese Werte nicht berücksichtigt, obwohl sie vielleicht bessere Ergebnisse liefern würden als entferntere Pegel (P2 und P4 in Bild 3.2), die am Tag der Bodenprobe abgelesen wurden. Sinnvoller ist es, zuerst über den Raum und dann über die Zeit (oder umgekehrt) zu mitteln. Räumlich werden quadratische Umgebungen gebildet. So werden nicht immer die wirklich

dichtesten Werte herangezogen. So kann ein Pegel, der genau in der Ecke des umgebenden Quadrats liegt, berücksichtigt werden, während einer mit geringerem Abstand, der aber genau in x- oder y-Richtung liegt, schon außerhalb des Quadrats bleibt. Auch hier ist die AVG-Funktion, das arithmetische Mittel, nicht immer die am besten geeignete Funktion zur Bestimmung der Zwischenwerte, weil sie dichter liegende Pegel nicht stärker gewichtet.

3.4.3 Aufbereitung für Modellrechnung und Simulation

Für Modellrechnungen und Simulationsprogramme werden häufig äquidistante Meßreihen mit vorgegebenen festen Schrittweiten benötigt. Wenn diese Schrittweiten bei der Installation des Meßgeräts noch nicht bekannt sind oder das Meßgerät die gewünschten Meßabstände nicht liefern kann, müssen auch hier Werte zu den benötigten Meßpunkten nachträglich berechnet werden. Das kann nicht direkt in einer SQL-Anfrage geschehen, weil es in SQL nicht möglich ist, Zählvariablen für die Schrittweiten zu definieren.

Sollen z.B aus der im Beispiel oben beschriebenen Grundwasserstandsrelation Grundwasserstände in einem (räumlich) äquidistanten Raster (zu den vorhandenen Meßzeitpunkten) geliefert werden, kann dies nicht mit einer einzigen SQL-Anfrage geschehen. Hier bleibt dem Benutzer nur die Möglichkeit, zuerst eine Relation anzulegen, die die Rasterpunkte enthält: raster (x_koord, y_koord). Die SQL-Anfrage lautet dann ähnlich wie die Anfrage in Abschnitt 3.4.2:

```
SELECT AVG (g.meßwert), r.x_koord, r.y_koord, g.zeitpunkt
FROM   raster r, gw_stände g
WHERE  g.x_koord   BETWEEN r.x_koord - 5    AND r.x_koord + 5    AND
       g.y_koord   BETWEEN r.y_koord - 5    AND r.y_koord + 5
GROUP BY r.x_koord, r.y_koord, g.zeitpunkt
ORDER BY g.zeitpunkt, r.x_koord, r.y_koord
```

Es treten hier auch die gleichen Probleme wie die in Abschnitt 3.4.2 beschriebenen auf.

3.4.4 Die Ergänzung fehlender Werte

Wenn Meßgeräte unerwartet ausfallen, sind die Werte für die ausgefallenen Zeiträume unwiederbringlich verloren. Sollen mit solchen Meßreihen dennoch Modellrechnungen laufen, müssen die fehlenden Werte abgeleitet werden, entweder räumlich aus benachbarten Meßstationen oder zeitlich aus den vorher und nachher gemessenen Werten. Welche Methode hier bevorzugt wird, hängt stark von der Art der Messung ab, von der Dauer des Ausfalls und von den Gegebenheiten, z.B. ob es überhaupt benachbarte Stationen gibt.

Wenn mehrere Werte hintereinander ausgefallenen sind, läßt sich der Ersatz auf keinen Fall mehr durch Standard-SQL-Anfragen berechnen. In den meisten Fällen ist die AVG-Funktion nicht ausreichend; etwa die Lufttemperatur (über die Zeit) ähnelt einer Sinuskurve und fehlende Werte sollten durch Unterlegen einer Sinusfunktion durch die vorhandenen Werte bestimmt werden. Solche Funktionen werden aber von konventionellen DBS nicht angeboten und sind

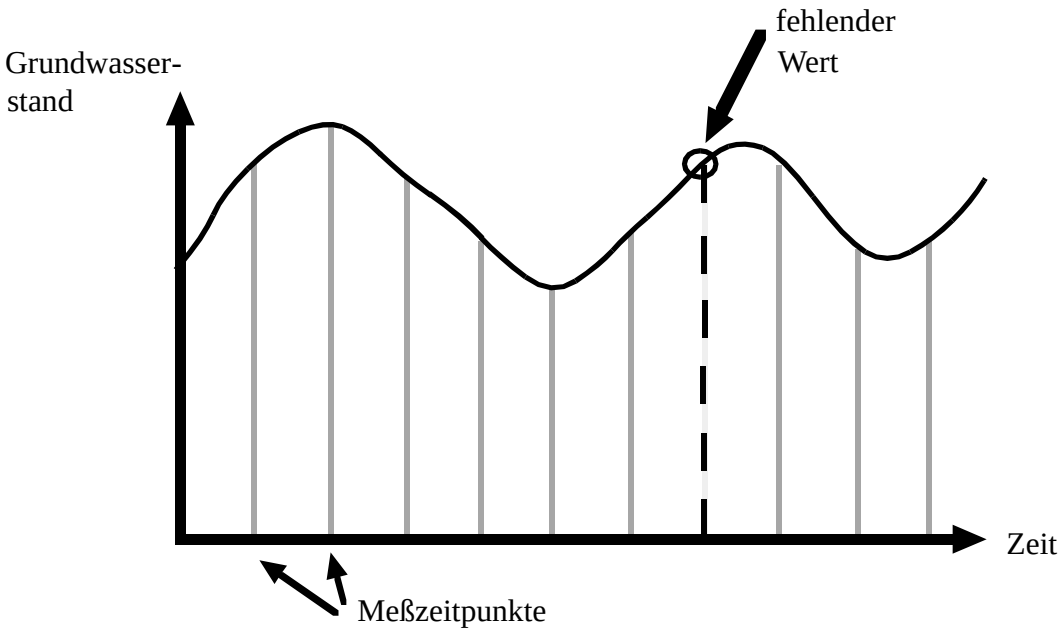


Bild 3.3: Beispiel für die Interpolation eines fehlenden Wertes

auch in Standard-SQL für die nähere Zukunft nicht vorgesehen. Auch das Fehlen von Zählvariablen führt bei mehreren hintereinander ausgefallenen Werten wieder zu Problemen.

3.5 Schnittstellen zum Anwender

Neben den Anforderungen, die genau die typischen Funktionen und Eigenschaften eines DBS betreffen, stellen Anwender noch weitere Anforderungen insbesondere bezüglich der Schnittstellen an die neue DB-Software. Diese Anforderungen würden sie auch an jede andere Software stellen, mit der sie ihre Meßdaten verwalten und aufbereiten. Sie müssen von jeder (DB-) Software erfüllt werden, wenn diese vom Anwender akzeptiert und sinnvoll genutzt werden soll.

Die Personen, die Meßwerte erheben und auch diejenigen, die sie weiterverarbeiten und auswerten, sind i.a. keine Informatiker sondern Wissenschaftler aus den Anwendungsgebieten. Deshalb muß die Benutzerfreundlichkeit und Handhabbarkeit von Werkzeugen zur Unterstützung der Verarbeitung von Meßwerten besonders hoch sein.

3.5.1 Dateneingabe

Meßwerte und Umweltdaten können prinzipiell auf drei verschiedene Arten in das DBS gelangen, nämlich durch (1) manuelle Eingabe, (2) durch Übertragung von aufgezeichneten Daten von Disketten oder aus Datenloggern oder (3) durch direkte Übertragung vom Meßgerät.

Die Unterstützung der manuellen Dateneingabe ist besonders wichtig. Diese Tätigkeit wird oft von Hilfspersonal durchgeführt, das die genauen Sachverhalte nicht kennt und deshalb die Schlüssigkeit der eingegebenen Daten nicht überprüfen kann. Deshalb sind hier besonders genaue Anweisungen und automatische Konsistenzprüfungen der eingegebenen Daten notwendig. Wenn bei der manuellen Eingabe effizient gearbeitet werden soll, müssen Eingabemasken ergonomisch gestaltet werden können und die mehrfache Eingabe derselben Sachverhalte muß vermieden werden. Heute bieten alle bekannten kommerziellen DB-Hersteller zu ihren Systemen passende Werkzeuge zur Gestaltung von Benutzerschnittstellen an, mit denen die oben genannten Anforderungen erfüllt werden können. Die Implementierung einer wirklich gut gestalteten und funktionellen Eingabeschnittstelle macht zwar immer noch einige Mühe, ist aber in einer Zeit zu bewerkstelligen, die zu dem erreichten Nutzen in einem guten Verhältnis steht. Zu Beginn des Projekts war das noch nicht so. Die Implementierung einer guten Eingabeschnittstelle erforderte einen deutlich höheren Aufwand, z.B. mit Embedded SQL/C, und konnte dann nach der Fertigstellung auch nur mit größerem Aufwand an neue Wünsche der Anwender angepaßt werden.

Bei der Übertragung von Daten von Disketten oder aus Datenloggern auf den Rechner müssen die Meßwerte häufig ergänzt werden, etwa um Versuchsnummern oder Jahreszahlen beim Datum. Auch hier müssen während des Einlesens automatische Überprüfungen stattfinden, um Inkonsistenzen oder Abweichungen von der Regel aufzudecken, die z.B. durch einen Defekt des Meßgeräts oder mangelnde Stromversorgung verursacht sein können. Passende Eingaberoutinen lassen sich relativ einfach mit Embedded bzw.- Dynamic SQL programmieren. In einfachen Fällen reichen Massenlader-Befehle zusammen mit Datenbankprozeduren zur Konsistenzprüfung.

Die direkte Übertragung von Daten aus dem Meßgerät in die Datenbank soll hier nur kurz erwähnt werden. In dem in Kapitel 2 beschriebenen Projekt kam sie nicht vor, weil keine geeignete Leitung zum Testfeld hin bestand. Prinzipiell bestehen hier die gleichen Anforderungen wie bei der Übertragung von Diskette oder Datenlogger. Nur die sofortige Sicherung der Daten ist noch wichtiger, weil die ‘Sicherungskopie’, die sonst auf der Diskette oder im Datenlogger (zumindest temporär) noch besteht, wegfällt.

3.5.2 Bereitstellung von Daten

Die Daten müssen sofort nach der Eingabe für interaktive Anfragen und zur Weiterverarbeitung zur Verfügung stehen. Auch Mitarbeiter aus anderen Teilprojekten sollen möglichst bald lesenden Zugriff darauf erhalten. Es darf davon ausgegangen werden, daß sich die Projektmitarbeiter für diesen Zweck in eine genormte SQL- oder maskenorientierte Benutzeroberfläche einarbeiten.

Im dieser Arbeit zugrundeliegenden Projekt durften bei der Bereitstellung von Meßdaten für andere Teilprojekte einige Tage vergehen. In so einem Fall reicht es aus, wenn die Daten entladen und mit den passenden Ladebefehlen über ein Netzwerk an andere Teilprojekte verschickt wer-

den. Wenn jedes Teilprojekt seine eigene DB hält, kann es jedoch bei Unachtsamkeiten der Benutzer zu inkonsistenten Daten kommen in dem Sinn, daß die Daten zu derselben Messung sich in den verschiedenen Projektdatenbanken unterscheiden. Diese Probleme können beim Einsatz von netzwerkfähigen oder verteilten DBS vermieden werden. In einem verteilten DBS stehen die Daten auch sofort nach ihrer Eingabe allen Teilprojekten zur Verfügung.

3.5.3 Kooperation mit vorhandener Hard- und Software

Solange die neue (Datenbank-) Software ältere Software und ihre Funktionen nicht vollständig ersetzt, muß sie mit den noch benötigten Teilen älterer Software gut kooperieren. Natürlich muß sie auch überwiegend mit vorhandener Hardware arbeiten, die zum Teil nicht auf dem neuesten Stand der Technik ist. Z.B. muß die Unterstützung älterer Ausgabegeräte gewährleistet sein. DBS bieten aber immer auch eine zeilenorientierte Sprach-Schnittstelle (z.B. SQL) und die Ausgabe von Daten und Ergebnismengen in ASCII-Dateien. Damit können alle Werkzeuge und Ausgabegeräte arbeiten.

Für die Auswertung der erhobenen Meßwerte werden auch weiterhin (teilweise) Software-Pakete eingesetzt werden, in die DB-Zugriffe nicht direkt integriert werden können. Es muß möglich sein, auf einfache Weise die gespeicherten Meßwerte für diese Software bereitzustellen.

Die finanziellen Mittel für Hardware sind in Umweltprojekten oft begrenzt. Deshalb ist es wichtig, ein DBS zu verwenden, das auf vorhandener unterschiedlicher Hardware verfügbar ist und dessen Installationen auf unterschiedlicher Hardware miteinander kommunizieren können. Das bedeutet konkret, daß auf allen Knoten bzw. bei allen Teilprojekten dasselbe DBS laufen sollte, um zusätzliche Komplikationen zu vermeiden.

3.5.4 Automatische Datensicherung

Wie bereits in Kapitel 3.1.4 beschrieben, müssen Meßwerte besonders sorgfältig gesichert werden. Sowie die Eingabe abgeschlossen ist, erfolgt deshalb die automatische Sicherung der Daten. Bei manueller Eingabe und beim Einlesen von Meßwerten direkt aus einem Meßgerät muß noch häufiger gesichert werden, weil die ‘Sicherungskopie’ auf der Originaldiskette entfällt.

Bei manueller Dateneingabe ist es wichtig, daß die eingegebenen Werte bereits nach wenigen Eingaben (zwischen-) gesichert werden, damit bei Systemunterbrechungen möglichst wenig dieser zeitintensiven Arbeit verloren geht. Auch dürfen bei der Korrektur von Eingabefehlern zuvor richtig eingegebene Werte nicht verloren gehen. Für beide Fälle muß es möglich sein, den Eingabevorgang in sehr kurze Transaktionen zu unterteilen. Nur so läßt sich der Wiederherstellungsaufwand nach System- oder Eingabefehlern gering halten. Normalerweise brauchen die Daten nach der Eingabe nicht mehr häufig gesichert zu werden, weil sich Meßwerte nicht ändern. Laufen jedoch aufwendige Simulationsrechnungen, kann es wichtig werden auch - oft umfangreiche - (Zwischen-) Ergebnisse zu sichern. Hier werden dann lange oder sogar sehr lange Transaktionen (Stunden, Tage) benötigt.

3.6 Resultierende Anforderungen an das “Datenhaltungssystem”

Wie schon in den einzelnen Abschnitten gezeigt, lassen sich die bisher aufgelisteten Anforderungen für das Informatik-Teilprojekt beim heutigen Stand der Technik am besten durch den Einsatz eines (relationalen) DBS realisieren. DBS garantieren dem Benutzer viele Eigenschaften, die im Projekt benötigt werden. In einigen Fällen reichen diese Eigenschaften aber noch nicht aus. Dann müssen Erweiterungen an dem DBS vorgenommen werden. In diesem Abschnitt werden die Anforderungen des Projekts an ein geeignetes Datenhaltungssystem mit den Eigenschaften von DBS, wie sie heute kommerziell erhältlich sind, verglichen und die Bereiche herausgestellt, an denen das DBS ergänzt muß.

Bei allen Werkzeugen und Hilfsmitteln, um die Umwelt-DBS ergänzt werden, sollten die besonderen Eigenschaften von Meßwerten und Umweltprojekten berücksichtigt bzw. ausgenutzt werden. Berücksichtigung dieser Eigenschaften erfolgt etwa durch die Bereitstellung komfortabler und umfangreicher Möglichkeiten zur Datensicherung. Ausgenutzt werden kann z.B. die Tatsache, daß wenig Änderungen vorkommen, bei der Auswahl geeigneter Datenstrukturen.

3.6.1 Datentypen

Klassische relationale DBS verfügen nur über Grunddatentypen wie Zeichenketten, Integer- und Gleitpunktzahlen sowie Datums- und Zeitangaben. Diese Werte können nur auf eine einzige Art strukturiert werden, nämlich als Relationen (zweidimensionale Tabellen) mit Attributen (Spalten) und Tupeln (Zeilen). Zur Laufzeit des Projekts ´Horkheimer Insel´ gab es kein DBS, das Beziehungen zwischen den Relationen unterstützte. Beziehungen mußten vom Benutzer mit Hilfe passender Werte über das Fremdschlüsselkonzept eingeführt und gepflegt werden. (Heute kann bei den meisten DBS zumindest die Pflege der Fremdschlüssel durch die Definition referentieller Integritätsbedingungen mit anhängenden kaskadierenden Operationen automatisiert werden.)

Diese Art der Datenhaltung eignet sich gut für das Speichern einzelner Meßreihen. Für die einzelnen Meßwerte sind die Grunddatentypen geeignet und ausreichend. Der Unterschied zwischen diskreten und stetigen Wertebereichen läßt sich damit jedoch nicht ausdrücken. Wenn komplexere Zusammenhänge wie z.B. die Zuordnung verschiedener Meßreihen zu einem größeren Versuch dargestellt werden müssen, werden mächtigere Konzepte zur Strukturierung benötigt. Hier ist eine aufwendige Modellierung durch übergeordnete Relationen notwendig, die z.B. enthalten, welche Meßreihe welchem Versuch zuzuordnen ist. Erweiterungen wie geschachtelte Relationen würden Abhilfe schaffen. Auch mit einem Konzept zum Ausdrücken vom Vererben von Eigenschaften, wie es etwa seit einiger Zeit zunächst vom DBS Illustra [Illu94] und dann von dessen Nachfolger Informix [Info96] mit den benutzerdefinierten Datentypen in den sog. “DataBlades” angeboten wird, läßt sich so ein Sachverhalt gut modellieren. Der Aufwand zum Definieren der verschiedenen über- und untergeordneten Relationen (-typen) ist aber auch hier nicht unerheblich.

In den gängigen DBS werden alle Daten gleich bearbeitet. Eine besondere Behandlung insbesondere räumlicher Daten ist nicht möglich, weder bei der Modellierung noch bei der Speicherung. Räumliche Daten können dem DBS nicht als solche bekannt gemacht werden, um z.B. Beziehungen darüber zu definieren. Die Eigenschaften von Meßwerten können auch nicht durch spezielle Speicherstrukturen oder durch spezielle Zugriffspfade, die einen effizienteren Zugriff auf diese Daten erlauben, ausgenutzt werden. Es ist nicht möglich, zwischen punktuellen Werten und Intervallen zu unterscheiden (Ausnahme: Datentyp ´Zeit´ mit z.B. '03:00:00' für 3.$\underline{00}$ Uhr bzw. '3 hours'). Es können keine zusammengesetzten Attribute definiert werden, etwa um mehrdimensionale Meßbasen zu modellieren. Daten lassen sich auch nicht mit bestimmter Semantik, wie z.B. "Meßwerte" oder "(mehrdimensionale) Meßbasis" auszeichnen. Abhängig von dieser Semantik könnte man aber bestimmte Operationen zulassen oder verbieten. Auch die wichtige Eigenschaft Kontinuität (siehe Kap. 3.1.2) läßt sich so den Meßwerten bzw. Meßreihen nicht zuordnen.

In den neusten, kommerziell erhältlichen DBS wie Informix [Info96] und dessen Vorgänger Illustra [Illu94] sind zumindest einige dieser Mängel behoben oder abgemildert. Es lassen sich verschiedene Relationentypen definieren. Damit ist es möglich, bestimmte Relationen gegenüber anderen auszuzeichnen z.B. als Meßreihen für Pegelstände. Allerdings hat ein bestimmter Relationentyp ein festes Schema. Man kann also nicht einfach beliebige Meßwerterelationen mit ganz unterschiedlichen Basisattributen direkt zusammenfassen. Über eine Hierarchie läßt sich aber hier etwas erreichen, was dem Gewünschten schon recht nahe kommt. Das Definieren der Relationenschemata wird aber wieder nicht ganz einfach, inbesondere, wenn auch noch Hierarchien für die oben erwähnten Versuchsbeschreibungen bestehen. Zusammengesetzte Attribute können definiert werden. Attribute können im Prinzip als Meß- oder Basiswerte ausgezeichnet weden, jedoch muß der Benutzer (DB-Programmierer) dies noch mit entsprechender Semantik füllen, indem die passenden Funktionen geschrieben und diesen Datentypen zugeordnet werden.

Zusammenfassend läßt sich sagen, daß die wichtigsten Forderungen an ein zur Bearbeitung von Meßwerten und Umweltdaten geeignetes DBS die folgenden sind: Es muß möglich sein, bestimmte Daten (Attribute) mit einer besonderen Semantik auszuzeichnen und abhängig davon auf diesen Daten neue Operationen zu definieren. Zu Beginn des Projekts ´Horkheimer Insel´ waren solche DBS nicht kommerziell verfügbar. Neue DBS wie das neue Informix ermöglichen dies und nehmen dem Benutzer auch schon einen kleinen Teil der Arbeit ab. Der Aufwand für den DB-Designer und den DB-Programmierer bleibt aber nicht unerheblich.

3.6.2 Definition von Relationenschemata

Alle kommerziellen DBS erlauben es, daß das Datenschema bzw. die einzelnen Relationenschemata nach der Inbetriebnahme der Datenbank verändert und ergänzt werden. Bei manchen DBS können Ergänzungen einzelner Relationenschemata direkt vorgenommen werden, bei manchen müsssen dazu die alten Relationen umgeladen, gelöscht und neu erzeugt werden. Es ist jedoch in keinem DBS möglich, die Geschichte dieser Änderungen direkt zu speichern und bei Bedarf

auch auf ein älteres Schema zuzugreifen. Über die sogenannten Sichten (Views) ist es zwar möglich, eine Relation in verschiedenen Formaten dem Endbenutzer bereitzustellen und dabei auch einfache Umrechungen vorzunehmen, aber nur unter unterschiedlichen Relationenamen. Seit wann eine neue Relation oder Sicht gültig ist, ist ebenfalls nur mühsam oder auch gar nicht herauszufinden. Hier fehlt ein Werkzeug, mit dem die “Geschichte” der einzelnen Relationenschemata nachvollzogen werden kann.

3.6.3 Standard-DBS-Funktionen

DBS bieten einen kontrollierten Mehrbenutzerbetrieb, d.h. es können mehrere Benutzer gleichzeitig Daten in verschiedenen Teilen (Relationen, Speicherseiten) der DB ändern oder auch dieselben Daten gleichzeitig lesen. Dies ist für die Arbeit in einem interdisziplinären Umwelt-Projekt wichtig und ausreichend.

DBS bieten dem Benutzer die Möglichkeit, Daten bestimmten Benutzern praktisch als Eigentum zuzuordnen und erlauben diesen Benutzern dann, Zugriffsrechte wie Lesen, Ändern, Einfügen und Löschen auf diesen Daten an andere Benutzer weiterzugeben. Auch diese Funktion entspricht genau der in einem interdisziplinären Projekt benötigten.

Um jederzeit die Richtigkeit und Konsistenz des Datenbestands zu gewährleisten, werden Änderungen auf Datenbanken nur innerhalb von Transaktionen durchgeführt, das sind unteilbare (Gruppen von) DB-Operationen, die entweder ganz oder gar nicht ausgeführt werden. Um die Dauerhaftigkeit der durch Transaktionen hervorgerufenen Änderungen im Datenbestand zu gewährleisten, werden alle Operationen protokolliert. Diese Protokolle können verwendet werden, um die DB nach einer Beschädigung wiederherzustellen, oder auch, um sie nach Anwendungsfehlern auf einen früheren konsistenten Zustand zurückzusetzen. Da jedes DBS dem Benutzer die Möglichkeit bietet, Transaktionsklammern selbst zu setzen, lassen sich sowohl sehr kurze Transaktionen für (manuelle) Eingabevorgänge definieren als auch sehr lange für Auswertungsprogramme.

In den Bereichen Mehrbenutzerbetrieb, Autorisierungen, Transaktionsverarbeitung und Protokollierung bieten die heute erhältlichen kommerziellen relationalen DBS ausreichende Möglichkeiten. Es müssen also keine Erweiterungen für Umweltdaten vorgenommen werden.

3.6.4 Archivierung und Komprimierung

Heutige DBS sind nur auf die Korrektheit und hohe Verfügbarkeit eines online-Datenbestands ausgerichtet. Archivierungen, also Auslagerungen auf nicht direkt verfügbare Speichermedien wie Bänder, von älteren, nur noch selten benötigten Daten sind nicht vorgesehen. Dies wird aber bei der großen Anzahl der erhobenen Werte in einem Umweltprojekt unumgänglich, wenn es über eine längere Zeit läuft. Hier müssen noch Regeln gefunden werden, nach welchen Kriterien Daten archiviert werden sollen und wie diese Archive verwaltet und in das DBS integriert werden können.

Die Archive können im Lauf der Jahre sehr umfangreich werden, deshalb sollte ihre Verwaltung auf jeden Fall in dasselbe DBS integriert werden, das zur Verwaltung der Online-Daten verwendet wird. Dafür gibt es zur Zeit noch keine Standard-Werkzeuge.

Bei einigen DBS wie Starburst, POSTGRES (siehe Kap. 4.4) und dessen kommerziellen und konzeptionellen Nachfolgern Illustra bzw. Informix wird die Speicherung auf optischen Platten, sog. WORM-Devices (*write once read many*; nur einmal beschreibbar) angeboten. Prinzipiell eignen sich solche Speichermedien für Meßdaten, weil diese ja kaum geändert werden. Außerdem muß so ein DBS gewisse Archivierungsfunktionen für den ständig wachsenden Datenbestand anbieten. Ob sich diese allerdings auch für Meßwerte eignen, von denen ja keine durch Löschen oder Änderungen ungültig werden, muß sich in der Praxis erst noch zeigen.

Eine Alternative zur Archivierung ist die Komprimierung von Daten. Das kann z.B. durch die Durchschnitts- oder Summenbildung über deutlich größere Intervalle als die Meßintervalle geschehen. Die Originaldaten müssen dann aber trotzdem archiviert werden, um später bei Verdacht auf fehlerhafte Komprimierung diese nachvollziehen zu können.

3.6.5 Verteilte Datenhaltung

Für ein Verbundprojekt eignet sich am besten ein verteiltes DBS, das aus unabhängigen, vernetzten DBS-Servern besteht, die verteilte Transaktionen verarbeiten können. In diesem System hält jeder Projektpartner seine Daten lokal und gewährt den anderen Teilprojekten (lesenden) Zugriff darauf. Zugriffstransparenz ist wünschenswert; es wird also ein verteiltes DBS benötigt. Für die im Projekt verwendete Hardware war so ein DBS zur Laufzeit des Informatik-Teilprojekts noch nicht kommerziell erhältlich. Von Seiten der Hersteller war aber bereits angekündigt, daß es in absehbarer Zeit verfügbar sei. Deshalb wurden die Daten zunächst mit einfacher Netzwerk-Software ausgetauscht und repliziert. Dies läßt sich für Meßwerte, die sich normalerweise nicht ändern, mit relativ einfachen Mitteln weitgehend automatisieren [Neug89], ohne die Konsistenz der Gesamt-DB zu gefährden. Zwischenzeitlich sind geeignete kommerzielle Datenbanksysteme verfügbar.

3.6.6 Schnittstellen

Die Eingabe der Daten in das DBS kann je nachdem, wie die Daten vorliegen, durch Ladebefehle, spezielle Ladeprogramme oder manuelle Eingabe durch den Benutzer erfolgen. Ladebefehle sind meist nicht mächtig genug, um die Daten in dem Format, in dem sie vom Meßgerät geliefert werden, in die Struktur der DB zu übertragen. Für die manuelle Eingabe können Bildschirmmasken definiert werden. Sowohl Ladeprogramme als auch Programme zur Unterstützung der manuellen Eingabe können Integritätsprüfungen auf den Daten vornehmen. Beide Methoden haben den Nachteil, daß sie sehr unflexibel sind gegenüber dem Format, in dem die Daten vorliegen. Das Format ändert sich jedoch in der Realität häufig, weil versucht wird, Meßgeräte immer wieder optimal einzusetzen. Programmierbare Massenlader können hier bis zu ei-

nem gewissen Grad Abhilfe schaffen, indem sie z.B. Daten aus multiplen Dateien in multiple Relationen laden. Ganz läßt sich das Problem damit aber nicht in den Griff bekommen.

Wünschenswert sind hier Bildschirmmasken und Ladebefehle bzw. Programme, die leicht änderbar sind. D.h., sie sollten auf hoher Ebene programmierbar sein (z.B. in einer Sprache der 4. Generation) und gut dokumentierbar sein.

Damit die Meßwerte weiterverarbeitet werden können, müssen sie vom DBS wieder bereitgestellt werden. Dies kann grundsätzlich auf zwei Arten geschehen: (1) Über eine interaktive Schnittstelle werden Anfragen gestellt, die die gewünschten Daten genau beschreiben. Die Ergebnisse dieser Anfragen werden aus der DB in Dateien geladen, die dann von Auswertungsprogrammen wie gewohnt verarbeitet werden.

(2) Anweisungen der Anfragesprache (z.B. Embedded SQL) werden direkt in das Auswertungsprogramm eingebettet. Dies setzt voraus, daß die Quelle des Auswertungsprogramms verfügbar ist und daß für die Programmiersprache, in der es geschrieben ist, ein Precompiler existiert, der die Anweisungen der DB-Anfragesprache in Prozeduraufrufe übersetzt. Wenn das Auswertungsprogramm noch andere Prozedurpakete, z.B. für bestimmte Ausgabeformen oder -geräte benutzt, die auch einen Precompiler benötigen, kann es zu Konflikten zwischen den Precompilern kommen.

Beide Möglichkeiten sind in vielen Fällen unbefriedigend, weil sie zu unflexibel sind. Die erste Methode ist außerdem dann ungeeignet, wenn die Daten vor der Auswertung noch aufbereitet werden müssen. Alles, was über eine einfache Umrechnung der Meßwerte, etwa Multiplikation mit einer Konstanten hinausgeht, ist in interaktiven Anfragen nicht möglich. Die zweite Methode kann bei gekaufter Software, für die keine Quellprogramme vorliegen, ohnehin nicht angewendet werden. Für eine kompliziertere Aufbereitung, etwa das Ergänzen fehlender Werte, müssen dann Programme in einer höheren Programmiersprache oder in einer sog. Sprache der vierten Generation (4GL, die meist zum DBS mitgeliefert wird) geschrieben werden, die diese Aufbereitung vornehmen.

3.6.7 Zusammenstellung der Anforderungen

Zur Unterstützung eines interdisziplinären, räumlich auf verschiedene Standorte verteilten Projekts, das überwiegend Meßwerte und andere Umweltdaten verarbeitet, wird ein System benötigt, das

- alle wesentlichen Eigenschaften eines heute kommerziell verfügbaren relationalen DBS aufweist,
- auf sehr unterschiedlicher Hardware verfügbar ist und sich zu einem heterogenen Gesamtsystem koppeln läßt,
- Möglichkeiten zur Archivierung und Komprimierung von Daten anbietet,
- vielfältige Strukturierungsmöglichkeiten für Daten anbietet und es ermöglicht, bestimmte Klassen von Daten auszuzeichnen,

- auf besondere Eigenschaften von räumlichen und zeitlichen Daten eingeht, sowohl in der Möglichkeit, sie zu modellieren, als auch mit geeigneten Speicherstrukturen und Zugriffspfaden zur effizienten Bereitstellung großer Mengen dieser Daten,
- komfortable, leicht programmierbare Eingabeschnittstellen anbietet,
- neuartige Operationen anbietet, um die Aufbereitung der Daten für Auswertungen und die Kopplung mit vorhandener Software vereinfacht und flexibler gestaltet.

Die Ergänzung von DBS um bessere Strukturierungsmöglichkeiten und Archivierungs- bzw. Speicherungsmöglichkeiten, wird auch in vielen anderen Zusammenhängen untersucht. Bei Speicherung und Archivierung großer Datenmengen werden derzeit im Multimediabereich große Fortschritte gemacht. Z.B. wird in [ORSS96] ein fehlertolerantes System vorgestellt, das die Speicherung (und das Lesen) großer Objekte bzw. Datenmengen mit bestimmten Übertragungsraten garantiert. So ein System könnte man z.B. als Grundlage für ein System zum Einlesen und Weiterverarbeiten von Umwelt-Meßdaten direkt vom Meßgerät nehmen.

Zur Strukturierung großer Datenmengen bzw. zum Finden von Informationen in großen Mengen von Daten wurden in letzter Zeit unter dem Stichwort “data mining” interessante Ansätze veröffentlicht [ACM96]. Z.B. wird in [SeSW96] das IDEA-System beschrieben, das auf kommerzielle Daten zugeschnitten ist. Große Mengen von (Umwelt-) Meßwerten ließen sich damit aber auch strukturieren und bearbeiten. Eine Erweiterung für Umweltdaten so eines Systems könnte für Umweltforscher sinnvoll und nützlich sein.

In kleineren Projekten, wie dem in dieser Arbeit beschriebenen, konnte man sich zunächst behelfen, indem man zusätzliche Relationen einrichtete, die Informationen über Beziehungen zwischen verschiedenen Meßparametern (in unterschiedlichen Relationen) enthalten. Ebenso kann in Dokumentationsrelationen gespeichert werden, welche Daten es noch in - vom Benutzer manuell angelegten - Archiven gibt.

DBS, die die besonderen Eigenschaften von Meßwerten, sowie räumlichen und zeitlichen Daten kennen und berücksichtigen, gibt es bisher noch nicht. Lediglich bei der Erforschung mehrdimensionaler Speicherstrukturen, die für die Speicherung von Umweltdaten auch geeignet wären, wurden bereits Erfolge erzielt (siehe Kap. 7.2). Neue Speicherstrukturen lassen sich aber in ein kommerzielles DBS durch den Benutzer nicht einfügen. Auch für die Aufbereitung von Meßdaten für die Weiterverarbeitung bieten kommerzielle DBS praktisch keine Unterstützung. Hier bestehen aber gute Möglichkeiten, kommerzielle DBS um solche Komponenten zu ergänzen, weil sie auf ein vorhandenes System aufgesetzt werden können. In dieser Arbeit soll nun gezeigt werden, wie Interpolationsfunktionen, die bei der Aufbereitung von Meßwerten eine entscheidende Rolle spielen, in ein kommerzielles DBS integriert werden können. In diesem Zusammenhang werden neben den Meßwerten hauptsächlich räumliche und zeitliche Daten benötigt, deshalb werden solche Daten in dieser Arbeit an vielen Stellen besonders berücksichtigt.

Zuvor werden bestehende Forschungsprototypen und in der Entwicklung befindliche DBS untersucht, in wieweit sie bei der Aufbereitung von Meßdaten zur Weiterverarbeitung durch Inter-

polation eingesetzt werden könnten, wenn sie bereits auf dem Markt erhältlich wären. Daraus werden Anregungen für die eigenen Erweiterungen genommen. Außerdem soll bei der Gestaltung der eigenen Erweiterungen berücksichtigt werden, daß sie später an ein erweiterbares DBS angepaßt werden könnten.

4. Technische Möglichkeiten zur Verarbeitung von Umweltdaten

In diesem Kapitel wird aufgezeigt, welche Möglichkeiten zur Verwaltung und Verarbeitung von Umweltdaten zu Beginn und zur Laufzeit des Projekts ´Horkheimer Insel´ bestanden, und welche davon hauptsächlich genutzt wurden. Anschließend werden neuere Entwicklungen von Datenmodellen (DM) und DBS kurz beschrieben und auf ihre Eignung zur Strukturierung, Verwaltung und Verarbeitung von Umweltdaten hin untersucht. Bei den DBS werden insbesondere *Objekt-Orientierte DBS* und *Erweiterbare DBS* näher betrachtet. Einige neue DBS-Entwicklungen, die zur Projektlaufzeit noch nicht verfügbar waren, werden kurz vorgestellt. Zum Schluß werden Anforderungen an ein auf Umweltdaten zugeschnittenes System aufgestellt.

4.1 Existierende Systeme

In allen natur- und ingenieurwissenschaftlichen Projekten liegt die Verwaltung der anfallenden Daten zunächst bei dem, der sie erhebt. Die Verwaltung war deshalb bisher meist abhängig vom Kenntnisstand und von der verfügbaren Zeit des Anwenders, von verfügbaren Hilfsmitteln und vom Umfang der Daten. Entweder wurden manuelle Aufzeichnungen auf Papier eingesetzt, oder es wurden einfache sequentielle Dateien auf PCs oder auf Großrechnern in Rechenzentren verwendet. Der Umfang umweltrelevanter Daten wächst jedoch ständig, einerseits mit der Zeitspanne, in der die Messungen durchgeführt werden und andererseits durch immer bessere und genauere Meßgeräte. Zusätzlich wird eine immer stärkere Kooperation zwischen verschiedenen Fachgebieten notwendig, um komplexe Umweltprobleme erfassen zu können. Durch den großen Umfang der Daten und durch die Kooperation verschiedener Institute stoßen die herkömmlichen Arten der Datenverwaltung immer häufiger auf Schwierigkeiten. Als Beispiele seien hier die Inkompatibilität zwischen verschiedenen Rechnern, Betriebssystemen und Anwenderprogrammen genannt und die dadurch notwendigen Konvertierungsprogramme, sowie Fehler, die sich beim Kopieren und Konvertieren einschleichen können.

Es gibt immer Probleme, wenn verschiedene Personen auf demselben Datenbestand arbeiten müssen. Jeder Mitarbeiter konvertiert dazu i.a. die Daten zunächst in das für seine Auswertungsprogramme oder Statistiken geeignete Eingabeformat. So entstehen unkontrolliert mehrere

Versionen der Daten. Bei der Verarbeitung oder Ergänzung der Daten wird häufig nicht geklärt, ob wirklich die aktuellste Version verwendet wird.

(Relationale) DBS wurden bisher nur vereinzelt genutzt, was dann die Probleme nicht beseitigt, sondern noch verstärkt, weil eine weitere Version der Daten hinzukommt. Einzelne Benutzer sind i.a. auch weder zeitlich noch aufgrund ihrer Kenntnisse in der Lage, ein DBS an ihre eigenen Bedürfnisse anzupassen. Zu diesen Anpassungen gehören Laderoutinen, die es ermöglichen, die Daten, so wie sie vom Meßgerät aufgezeichnet wurden, automatisch umzuformatieren und in die DB zu laden, sowie Entladeroutinen oder andere Schnittstellen zur DB, die die Daten im geeigneten Format für die weiteren Auswertungsprogramme aufbereiten, und auch Unterstützung bei online-Anfragen. Ohne diese speziellen Erweiterungen und geschickte Datenmodellierung sind DBS aber unflexibel. Sie werden häufig auch zu langsam für effizientes Arbeiten, weil auf ungeeigneten logischen und physischen Datenstrukturen komplizierte Anfragen gestellt werden müssen. Eine geringe Verarbeitungsgeschwindigkeit steht wieder einer höheren Akzeptanz entgegen.

4.2 Datenmodelle für Meßwerte und Umweltdaten

Eine wichtige Voraussetzung für ein effizientes Verarbeiten von Umweltdaten in Datenbanken ist eine angemessene Modellierung der Umweltdaten. Es wäre günstig, wenn sich Meßwerte und Umweltdaten in einem der Datenmodelle darstellen ließen, für die es bereits kommerzielle DBS gibt oder in Kürze geben wird, weil damit eine einfache Wartung des DBS und die Kompatibilität zu anderen Datensammlungen gewährleistet wäre. Von den drei ‘klassischen’ Datenmodellen, Hierarchisches, Netzwerk- und Relationales Datenmodell, kann für den Einsatz bei Anwendern nur das Relationale Modell gewählt werden, weil nur für dieses Datenmodell und seine Erweiterungen deskriptive und damit hinreichend einfach zu erlernende Datensprachen (Quel, SQL und Erweiterungen davon wie ObjectSQL) kommerziell angeboten werden.

Das Relationale Datenmodell erscheint zunächst gut geeignet zur Modellierung von Meßwerten, da sich Meßreihen gut als Tabellen (entspricht Relationen) abspeichern lassen: So fallen etwa im Versuch x nur Tupel der Art an: Meßwert, Meßort, Meßzeitpunkt, Meßgerät (Sensortyp).

Meßwert	Meßort	Meßzeitpunkt	Sensortyp
83.5	F1	13.01.94 17:10:00	XYZ
71.8	F1	13.01.94 18:10:00	XYZ

Es fehlt jedoch die Möglichkeit, Hierarchien auszudrücken, wie sie benötigt werden, wenn Versuchsbeschreibungen zusammengefaßt werden sollen, oder wenn Versuche mit mehreren Meßreihen modelliert werden müssen. Sollen diese Werte, mit ‘Meßparameter’ (z.B. Lufttemperatur, Feuchtigkeit) in derselben Relation gespeichert werden? Speichert man Meßwerte des Versuchs y mit entsprechender Kennzeichnung auch in derselben Relation? Muß der Sensortyp jedes Mal

mitgespeichert werden? (Als Beispiel siehe Tab. 4.1.) Diese oder ähnliche Probleme sind auch bei konventionellen Daten bekannt und in der Literatur (z.B. [Date86]) unter dem Stichwort Normalisierung zu finden. In Umweltdatenbanken treten sie aber wegen der hierarchischen Versuchsstruktur zwangsläufig auf und sind bei großen Datenmengen besonders gravierend. Für diese Probleme könnten DBS, die auf dem NF^2-Modell (Non-First Normal Form) basieren [ScSc83, ScWe87, SPSW90], also relationenwertige Attributtypen zulassen, eine gewisse Abhilfe schaffen.

Versuch	Leiter	Parameter	Sensor	Meßort	Meßwert	Zeitpunkt
x	Afrau	Temperatur	XYZ	F1	3.1	13.01.94 17:10:00
					5.9	13.01.94 18:10:00
					...	...
		Feuchte	ABC	F2	83.5	13.01.94 17:10:00
					71.8	13.01.94 18:10:00
					...	...
y	Bmann	Temperatur	XYZ	F1	23.1	13.08.94 17:10:00
					15.9	13.08.94 17:30:00
					...	...
		Feuchte	ABM	F2	65.9	13.08.94 17:10:00
					60.7	13.08.94 18:10:00
					..	..

Tabelle 4.1: Nicht normalisierte “Relation” für Meßwerte

Außer Meßwerten gibt es aber auch noch andere Umweltdaten, die komplexer strukturiert sind als flache Tabellen. Für geographische Daten fehlen geeignete Modellierungskonzepte etwa bei dreidimensionalen Raumpunkten. Das Gleiche gilt für mehrdimensionale Meßbasen. In einigen Systemen fehlte sogar noch ein geeigneter Zeitdatentyp. Werden chemische Stoffdaten gespeichert, muß für einige Anwendungen der Aufbau von Molekülen erkennbar bleiben. Es gibt aber noch keine geeigneten Datenstrukturen, die Suchalgorithmen nach Teilstrukturen darauf unterstützen, wie z.B. nach beliebigen funktionalen Gruppen in Molekülen. Hier werden keine Hierarchien sondern allgemeine Netze benötigt. Sowohl Hierarchien als auch Netze werden bei der Darstellung von Produktionsverfahren und von Stoff-/ Produktströmen benötigt.

Es fehlt die Möglichkeit, dem DBS mitzuteilen, daß einige Relationen zusätzlich zu den Eigenschaften, die alle Relationen haben, noch einige weitere bestimmte Eigenschaften aufweisen müssen, etwa eine bestimmte Mindestanzahl von Attributen oder Attribute mit bestimmten Datentypen (siehe auch Kapitel 3.6.1) und damit diese Relationen gegenüber den anderen *auszuzeichnen*. Diese ausgezeichneten Relationen bilden dann verschiedene Untertypen vom übergeordneten Typ Relation. Es könnten dann Relationen z.B. als Meßwertrelation, als Stoffdatenrelati-

on, oder als Geo-Relation ausgezeichnet werden. Das Sicherstellen solcher zusätzlicher Eigenschaften kann als Voraussetzung für neue, benutzerdefinierte Operationen dienen. Es könnten dann Operationen zur Weiterverarbeitung der Attributwerte definiert werden wie geometrische Operationen, Plausibilitätsprüfungen oder Interpolation.

Heute besteht in vielen neuen DBS, wie z.B. Illustra [Illu94] und dem Nachfolger Informix [Info96] zumindest die Möglichkeit, Relationentypen zu deklarieren. Diese Relationentypen haben dann aber ein festes Schema, etwa “bodenprobe”. Verschiedene Relationentypen können zwar über Hierarchien zusammengefaßt werden, jedoch muß jeder neue Typ von Meßwertrelation einzeln deklariert werden. Es ist nicht möglich, eine allgemeine Deklaration zu machen in der Art: Eine Meßrelation ist eine Relation mit mindestens zwei numerischen Attributen und beliebigen weiteren Attributen.

4.3 Weiterentwicklungen von Datenbanksystemen

In diesem Abschnitt werden verschiedene Ansätze für DBS beschrieben und verglichen. Die Ansätze, die vom Prinzip her geeignet erscheinen, werden näher vorgestellt. Ihre veränderte bzw. erweiterte Funktionalität gegenüber ´einfachen´ relationalen DBS wird erläutert, damit sie später auf ihre Eignung für Umwelt-DBS hin untersucht werden können.

4.3.1 Geographische Informationssysteme

Ein *geographisches Informationssystem*, auch *Geo-Informationssystem*, kurz *GIS* genannt, ist nach [BiFr91, Bill95] ein rechnergestütztes System, das aus Hardware, Software, Daten und den Anwendungen besteht. Mit ihm können raumbezogene Daten digital erfaßt und redigiert, gespeichert und reorganisiert, modelliert und analysiert sowie alphanumerisch und graphisch präsentiert werden.

Aus dieser Definition geht schon hervor, daß GIS für die Bearbeitung räumlicher Daten konzipiert wurden. Sowohl Meßwerte als auch Modelle und Simulationsverfahren können nicht direkt in ein GIS integriert werden. Verschiedene Beispiele für kleinere GIS-Anwendungen sind in [PiJa90] im einem Kapitel über GIS zu finden, u.a. ein Abwasserinformationssystem, ein System zur Deponiestandortanalyse und ein System zur Erfassung der organischen Anreicherung des Meeresbodens. Diese Beispiele zeigen, daß GIS immer dann sinnvoll eingesetzt werden können, wenn topographische oder kartographische Daten häufig verarbeitet werden müssen. Auch in [Bill95] werden GIS als eine - wenn auch wichtige - Komponente von Umweltinformationssystemen dargestellt, die die Bearbeitung der räumlichen Daten übernimmt. Für die in dieser Arbeit und im PWAB-Projekt gestellte Aufgabe, die Bearbeitung von unterschiedlichen Meßwerten erscheinen GIS deshalb eher ungeeignet.

Wenn GIS als Teil eines Gesamtsystems gesehen werden, ist es wichtig zu wissen, wie und wie gut GIS mit anderen Komponenten zusammen arbeiten. In [Buis89] werden Betrachtungen angestellt, wie räumliche Daten mit wissensbasierten Systemen bearbeitet werden können. Bei räumlichen Daten bzw. räumlichem Wissen, ist es zunächst wichtig zu definieren, in welchem Modell (z.B. euklidisch, Vektor, Raster, metrisch, topologisch) das Wissen dargestellt wird. Das wissensbasierte System, kann dann bei der Umwandlung von einem in ein anderes Modell behilflich sein. Es kann also bei der Aufbereitung von Daten für ein GIS, z.B. der Ableitung von topologischen Daten aus digitalisierten eingesetzt werden. Andere sinnvolle Hilfen können wissensbasierte Systeme bei der Entscheidungsfindung aufgrund von in GIS gespeicherten Informationen anbieten. “Neue Daten”, etwa Zwischenwerte zu den gespeicherten können jedoch nicht abgeleitet werden (siehe auch unten Kap. 4.3.3).

In [Bill90] wird auf die Einsatzmöglichkeiten von GIS in Umweltinformationssystemen eingegangen. Auch hier wird klar, daß GIS hier nur einen Teil der benötigten Funktionen abdecken können. I.a. gibt es in GIS kaum Versionenverwaltung und keinen Zeitbezug. Auch arbeiten GIS häufig noch nicht vollständig auf der Basis eines relationalen DBS. Oft werden nur die Sachdaten in einem DBS gehalten, nicht aber die räumlichen Daten.

Alle diese Punkte verstärken den Eindruck, daß GIS für das hier beschriebene Projekt nicht geeignet sind. Bei einem so kleinen Meßfeld sind Kartenaspekte nicht so wichtig und für die Verarbeitung von Meßwerten werden keine Hilfen angeboten.

4.3.2 Multimedia-Datenbanksysteme

“Multimedia” bzw. “Multimedia-System” sind Schlagworte, die in der Informatik in den letzten Jahren immer häufiger zu hören sind. *Multimedia-Systeme* sind Systeme, denen verschiedenen (Programm-) Systeme, Datenträger und Endgeräte zugeordnet sind, die sie kombinieren und integrieren. Typische Multimedia-Daten sind Bilder (z.B. Photographien, Satellitenaufnahmen aber auch Graphiken), Audio-Sequenzen (Musik, gesprochener Text), Videofilme aber auch freier (Schrift-) Text. Natürlich gibt es auch konventionelle Daten, allein schon, um die Multimedia-Daten zu beschreiben. Typische Anwendungen von Multimedia-Systemen sind (nach [Wege91]: (a) Archivierung, (b) Unterrichtung (z.B. Hypertext-Systeme), Werbung (Animation), Unterhaltung (Trickfilme), (c) Entwurf und Publikation (Edition) und (d) Überwachung (z.B. Wetterüberwachung).

Natürlich ließ der Wunsch nicht lange auf sich warten, die Multimedia-Daten in einem DBS abzuspeichern, so daß dessen Vorteile genutzt werden könnten. So entstanden die Schlagwörter *Multimedia-Datenbanken* bzw. *Multimedia-Datenbanksysteme* (MMDBS). Das bedeutet aber zunächst nicht, daß es MMDBS schon fertig zu kaufen gibt.

Bei der Suche nach einem geeigneten System für eine Umweltdatenbank bzw. für Anwendungen auf einer Umweltdatenbank könnte man also prüfen, ob ein MMDBS auch für eine Umweltdatenbank geeignet ist.

Umweltdatenbanken enthalten ähnliche Datentypen: Bilder, oft auch freier Text und natürlich konventionelle Daten. Bei den Anwendungen ist die Übereinstimmung noch besser: Archivierung, Unterrichtung (Präsentation) und Überwachung zählen zu den Hauptaufgaben von Umwelt-DBS neben Simulation, die von MMDBS nicht direkt unterstützt wird.

Bevor nun die Möglichkeiten und Hindernisse, die der Aufbau einer Umwelt-Anwendung auf der Grundlage eines MMDBS bietet, im einzelnen diskutiert werden können, müssen zunächst folgende Fragen geklärt werden:

(1) Gibt es schon ausgereifte MMDBS, die erworben und benutzt werden können?

(2) Bietet ein MMDBS alle Funktionen, die die Umwelt-Anwendung unbedingt benötigt, bzw. lassen sich einzelne fehlende Funktionen durch einfache Erweiterungen ergänzen?

Schon die Antwort auf die erste Frage fiel zur Projektlaufzeit unbefriedigend aus. In der damals aktuellen Literatur [Wege91, RLMN93] wurden zwar einige bereits existierende MMDBS vorgestellt. Das waren aber alles Forschungsprototypen mit den damit verbundenen Nachteilen: Sie liefen nur auf bestimmter Hardware bzw. mit bestimmter anderer Software zusammen und wurden nur für eine gewisse Zeit oder gar nicht gewartet.

Zudem waren diese Systeme alle Erweiterungen von objektorientierten DBS. Hier stellt sich jedoch sofort die Frage, ob es nicht günstiger ist, genau zugeschnitten auf die Bedürfnisse von der Umwelt-Anwendung zu erweitern, damit die Leistungsfähigkeit nicht zu sehr leidet (siehe auch Kapitel 4.3.4). MMDBS bieten aus Sicht der Umwelt-Anwendung auch viel “Ballast”, viele Funktionen, die voraussichtlich nie wirklich gebraucht werden, etwa das Abspielen von Videofilmen in Realzeit oder die Synchronisation von Audio und Video. Durch das Bereitstellen dieser Funktionen wird das Gesamtsystem dann unnötig groß und komplex.

Es werden auch einige Funktionen, die für MMDBS gefordert werden und in Umwelt-DBS wichtig wären, wie die inhaltsorientierte Suche, noch von keinem der existierenden Systeme angeboten. Das liegt u.a. daran, daß sich diese Funktion innerhalb eines objektorientierten Systems nicht einfach implementieren läßt. [Wege91] schlägt hier andere Techniken vor wie Schlagworte oder Wissensrepräsentation.

[Wege91] stellt auch prinzipiell die Frage, ob die Erweiterung und Ergänzung eines objektorientierten DBS der richtige Ansatz ist. Auch die Erweiterung eines relationalen DBS wird erwogen. Es werden dafür ähnliche Gründe genannt, wie sie auch in dieser Arbeit zu einer SQL-Erweiterung führten (siehe Kapitel 5.2).

Abschließend läßt sich feststellen, daß MMDBS zur Projektlaufzeit noch nicht weit genug entwickelt waren, um in einem so praxisnahen Projekt eingesetzt zu werden. Sie sind jedoch eine interessante Entwicklung und könnten jetzt, da einige MMDBS produktreif sind, für ähnliche Projekte als Basis-System eine Alternative darstellen. Es müßte dann geprüft werden, ob

die große Funktionalität wirklich benötigt wird oder ob sie das Arbeiten mit dem System eher kompliziert.

4.3.3 Deduktive Datenbanksysteme

Nach [Reut85] umfaßt eine *Deduktive Datenbank* (DDB)
1. Axiome für Fakten und Ableitungsregeln,
2. eine Menge von Integritätsbedingungen und
3. eine Meta-Regel "negation as failure". ("Was nicht in der Datenbank gefunden wird, gibt es nicht.") Ein ganz einfaches Beispiel ist eine Ableitungsregel der Form

if condition A ***then*** conclusion C

nach [Baye85]. Ein konkretes Beispiel gibt [Baye85] mit

if Vater von x ist z ***and*** Bruder von z ist y ***then*** Onkel von x ist y

Diese Regel besagt, daß y der Onkel von x sein muß, wenn z der Vater von x ist und y der Bruder von z ist. Formaler, als *Horn-Klausel*, kann diese Regel wie folgt geschrieben werden:

Onkel (x, y) ← Vater (x, z) Bruder (z, y)

Fakten sind dann Regeln ohne rechte Seite, etwa

Vater (Sarina, Roland) ←
Bruder (Roland, Jürgen) ←

In einer relationalen DB werden Fakten in Relationen dargestellt, hier etwa die Relationen 'Vater' und 'Bruder' mit je zwei Attributen. Regeln entsprechen dann der Definition von Anfragen oder Sichten, hier im konkreten Beispiel:

```
CREATE VIEW Onkel AS
        SELECT    Vater.x, Bruder.y
        FROM      Vater, Bruder
        WHERE     Vater.z = Bruder.z
```

Wenn die Relationen 'Vater' und 'Bruder' u.a. die o.g. Fakten enthalten, wird die Ergebnisrelation das Tupel (Sarina, Jürgen) umfassen. (Probleme, die durch die Symmetrie der 'Bruder'-Relation und durch eine etwaige Rekursion in der Anfrage bei ganzen Ketten von Brüdern entstehen, seien hier nicht betrachtet.) Integritätsbedingen lassen sich formal auf die gleiche Art darstellen, z.B., daß Personen, die in die 'Vater'-Relation aufgenommen werden dürfen, männlichen Geschlechts sein müssen:

Geschlecht (z, 'männlich') ← Vater (x, z)

In der Praxis relationaler DBS ist die Definition von Integritätsbedingungen aufwendiger, weil zusätzlich gesagt werden muß, wann die Integritätsbedingung überprüft werden soll und was bei einer Verletzung der Integritätsbedingung geschehen soll.

Bei der Betrachtung der neuen Fakten, die durch Anwendung von Regeln entstehen, fällt auf, daß es sich hier um Fakten handelt, die etwa in einem Entity-Relationship-DBS als *Beziehungen* bezeichnet würden, nicht als *Objekte*. Oder, anders gesagt, es werden keine neuen Konstanten oder Werte durch die Anwendung von Regeln geschaffen. Und aus einer endlichen Menge von Fakten und einer endlichen Menge von Regeln lassen sich (wenn man z.B. von dem Problem der Generierung von Zyklen bei der Erzeugung von Pfaden aus Wegsegmenten einmal absieht,) auch nur eine endliche Menge von neuen Fakten ableiten.

Bei der Interpolation von Meßwerten liegt eine andere Aufgabe vor. Hier sollen tatsächlich neue Werte oder Objekte aus den vorliegenden berechnet oder geschätzt werden. Es soll etwa die Lufttemperatur zu einem Zeitpunkt interpoliert werden, der als solcher noch in keinem Zusammenhang in der DB gespeichert ist, und es kann dabei ein Temperaturwert berechnet werden, der bisher noch nie genau so gemessen wurde. Das einzige, das sicher über diesen Wert gesagt werden kann ist, daß zu diesem Zeitpunkt eine Lufttemperatur geherrscht hat und daß diese mit sehr großer Wahrscheinlichkeit in einer δ-Umgebung der vorher und nachher gemessenen Werte liegt. Auch lassen sich (wenn man von der in der Praxis begrenzten Genauigkeit der Rechner einmal absieht,) beliebig viele neue Temperaturwerte interpolieren, weil sich die Temperatur kontinuierlich über die Zeit, also über kleinste Sekundenbruchteile ändert.

Die Art, in der in Meßwertdatenbanken Information aus vorhandenen Fakten abgeleitet werden soll, nämlich durch evtl. aufwendige Interpolation an beliebigen, bisher nicht in der DB gespeicherten Punkten, verbietet den Einsatz von DDBS für dieses Problem. Obwohl es, wie [Ullm91] aufzeigt, auch gute Argumente für den Einsatz von DDBS gäbe, nämlich ihre gute Eignung für spontane, ungeplante Anfragen. Deshalb empfiehlt [Ullm91, p.270] auch den Einsatz von DDBS für wissenschaftliche DB, um bisher unbekannte Beziehungen zwischen Meßparametern zu suchen und zu finden. Die in Kap. 3 und hier aufgezeigten Probleme lassen den Einsatz von DDBS für nicht aufbereitete Meßwerte jedoch als ungeeignet erscheinen.

Ein weiteres Argument gegen den Einsatz von DDBS für die in dieser Arbeit behandelten Probleme zeigt [Ullm91] selbst noch auf: Um die Möglichkeit spontaner, ungeplanter Anfragen zu erhalten, möchte er den Einsatz neuer Datentypen und benutzerdefinierter Funktionen verbieten. Genau dies geschieht aber bei der Einbindung von Interpolation.

4.3.4 Objekt-orientierte Datenbanksysteme

Mit neuen, nicht-konventionellen Anwendungen wie Umwelt-, Ingenieurs- oder CAD-Datenbanken entstand das Bedürfnis nach reichhaltigeren Strukturen für die Datenmodellierung als relationale DBS sie anbieten. Deshalb wurde an mehreren Stellen die Idee formuliert, die Vorteile wie objekt-orientierte Programmiersprachen (Simula, C++) sie bieten, in DBS zu integrieren. Daraus enstanden die sog. *Objekt-orientierten DBS* (OODBS). Natürlich wiesen zunächst die verschiedenen als OODBS bezeichneten Systeme noch durchaus unterschiedliche Eigenschaften auf. Um einen gewissen ‘Qualitätsanspruch’ für die Bezeichnung OODBS zu sichern, wurde von einer Forschergruppe der Begriff OODBS mit dem ‘Object-Oriented Database Sys-

tem Manifesto' [ABDD90] genau definiert. Dabei wurde nicht von konkreten Systemen ausgegangen, sondern die Anforderungen wurden systemunabhängig zusammengestellt. Im nächsten Kapitel (4.3.5) werden fünf experimentielle DBS daraufhin untersucht, inwieweit ihre heutigen Versionen dem Manifesto genügen. In [ACM91] sind Beschreibungen weiterer Systeme mit Diskussion dieser Punkte zu finden. Nach dem Manifesto müssen Systeme, die sich OODBS nennen dürfen, mindestens die folgenden Anforderungen erfüllen:

(1) Das OODBS muß auf einem echten DBS aufgebaut sein, wobei unter einem 'echten' DBS ein DBS verstanden wird mit gemeinsamer, persistenter Datenhaltung aller Benutzer auf Sekundärspeicher, sowie mit Mehrbenutzerbetrieb, Transaktionskontrolle, Protokollierung aller Aktionen, Sicherungs- und Wiederherstellungsverfahren und interaktiven Benutzerschnittstellen zu den Datenbeständen.

(2) Komplexe Objekte müssen definiert, erzeugt und verarbeitet werden können. Komplexe Objekte werden mit Hilfe von sog. *Konstruktoren* aus anderen Objekten aufgebaut. Die einfachsten Objekte sind ganze Zahlen, Zeichen, Zeichenketten beliebiger Länge, Wahrheitswerte und Gleitpunktzahlen. Diese Grundobjekte müssen vom DBS bereitgehalten werden. Konstruktoren sind Tupel, Mengen, Listen, Felder u.ä. Die ersten drei der aufgezählten Konstruktoren müssen angeboten werden. Orthogonalität der Konstruktoren wird gefordert, d.h., überall, wo ein skalares Objekt stehen kann, kann auch ein komplexes stehen und umgekehrt. Dazu müssen geeignete Operatoren bereitgestellt werden wie z.B. "tiefe Kopie" (Kopie eines ganzen Objekts einschließlich aller Teil- und Unterobjekte).

(3) Alle Objekte müssen eindeutig identifizierbar sein und zwar unabhängig von ihrem (derzeitigen) Wert. Zwischen Gleichheit (derselbe Wert) und Identität (dasselbe Objekt) muß ein Unterschied bestehen. Diese Unterscheidung wird notwendig, wenn Objekte Verweise auf andere Objekten enthalten dürfen und so gemeinsame Unterobjekte haben können.

(4) Das Prinzip der *Kapselung* muß angeboten werden, d.h. die Implementierung von einem neuen Datentyp und der Operationen darauf muß unsichtbar für den Endbenutzer durchgeführt werden können. Es dürfen aber nebenbei auch andere, für den Benutzer nicht transparente Mechanismen zur Implementierung von Datentypen und Operationen angeboten werden.

(5) Ein Konzept für *Typen* oder *Klassen* muß angeboten werden. Für eine Klasse oder einen Typ von Objekten müssen bestimmte Eigenschaften garantiert werden können.

(6) Es muß möglich sein, *Hierarchien* der Typen oder Klassen zu bilden mit Hilfe von Spezialisierungen und Generalisierungen. *Vererbung* von Eigenschaften wie z.B. Wertebereichsbeschränkungen in beiden Richtungen muß möglich sein.

Eine Schnittstelle ähnlich den *Abstrakten Datentypen* (ADT) eignet sich für die Konstruktion von Klassen oder Typen, evtl. in Hierarchien mit Vererbung von Eigenschaften ebenso wie für eine gekapselte Implementierung (siehe z.B. [LWWM79, Maib85]). Die Definition neuer Da-

tentypen und Operationen darauf erfolgt im ADT-Ansatz ähnlich wie die Definition algebraischer Strukturen. Neue Datentypen (in der Literatur zu ADT meist *Sorten* genannt) werden definiert durch Aufzählung der möglichen Werte, durch Definition von Mengen und Listen aus vorhandenen Datentypen, durch Einschränkung oder durch Kombination vorhandener Datentypen ähnlich der Typdefinitionen in höheren Programmiersprachen wie C oder PASCAL. Über die Konstruktion von neuen Datentypen aus vorhandenen können deren Eigenschaften vererbt werden. Funktionen werden durch Abbildungen der Art

$$\pi: B_1 \times ... \times B_m \rightarrow B_n,\ n \in IN,\ B_i \text{ Datentypen bzw. Sorten}$$

definiert. Das Verhalten dieser Funktionen wird durch Axiome oder Gleichungen spezifiziert. Es kann eine Funktion ohne linke Seite zur Erzeugung neuer Elemente der Typen definiert werden. Eigenschaften der Elemente können dann durch Axiome, die diese Funktion enthalten, spezifiziert werden. Jede Implementierung der neuen Typen und Funktionen muß die Gültigkeit der Axiome sicherstellen.

(7) Das *Überladen* von Operatoren muß möglich sein. Dabei wird derselbe Operator für ähnliche Operationen auf verschiedenen Typen verwendet, etwa "+" für Integer- und Float-Werte, bei Kompatibilität auch zwischen den beiden Datentypen. Er kann auch noch für ganz andere Operationen wie "+" für die Konkatenation von Zeichenketten eingesetzt werden. Das System muß spät binden und selbst finden, welche Implementierung eines Operators eingesetzt werden muß.

(8) Die Anfragesprache muß berechnungsuniversell sein.

(9) Das System muß erweiterbar sein. Benutzer können immer wieder neue Objekte (Typen bzw. Klassen) und Operationen darauf definieren.

Es gibt noch weitere wünschenswerte Funktionen, die einige der verfügbaren OODBS schon aufweisen. Diese Funktionen sind nach dem Manifesto von [ADBB90] aber optional. Weitere Funktionen können u.a. Versionenkontrolle von Objekten sein oder zusätzliche Transaktionsmodelle, wie etwa lange Transaktionen für Design-Datenbanken.

4.3.5 Erweiterbare Datenbanksysteme

Für den Begriff *Erweiterbare DBS* (EDBS) gibt es keine so präzise Definition wie für die OODBS. In verschiedenen Forschergruppen sind als Forschungsprototypen mehrere sog. EDBS entstanden. Bei diesen Systemen handelt es sich - grob gesagt - um Relationale DBS, die darauf vorbereitet sind, um verschiedene Komponenten durch den Benutzer erweitert zu werden. Einige Systeme wie z.B. EXODUS [CDFG86] oder GENESIS [BBGS88], bestehen aus einer Reihe von Moduln für die DB-Funktionen, die zunächst ergänzt werden müssen, bevor sie zusammengesetzt und eingesetzt werden können. Das ist der sog. *Toolkit*-Ansatz. Andere EDBS bestehen wie DASDBS [SPSW90] nur aus einen "Datenbank-Kern", der die Grundlage für das eigentliche

DBS bildet; um den Kern herum muß die “Schale” mit anwendungsspezifischen Benutzerschnittstellen erst noch programmiert werden. Wieder andere Systeme wie Starburst [HCLM90] oder POSTGRES [StRH90] können auch ohne Erweiterungen durch den Benutzer eingesetzt werden. Dann arbeiten sie wie konventionelle (relationale) DBS. Zu den Erweiterungen, die ein Benutzer in einem EDBS ergänzen kann, gehören neue Datentypen, neue Operationen oder direkt in des DBS integrierte Prozeduren, neue Speicherstrukturen und Zugriffspfade oder die Verwendung neuartiger Speichermedien wie nur einmal beschreibbare optische Platten.

Für die Betrachtungen in dieser Arbeit wurden fünf Systeme ausgewählt, bei denen die Implementierung zur Laufzeit des Projekts ´Horkheimer Insel´ schon fortgeschritten war. Außerdem waren über diese Systeme genug Literatur und Beschreibungen verfügbar, um ihre Funktionalität verstehen, beschreiben und auf den möglichen Einsatz im Projekt hin untersuchen zu können. Es handelt sich um die Systeme EXODUS [CDFG86, CaDV88], POSTGRES [StRH90, StRo86], Starburst [HCLM90, LiHa90, SCFL86], GENESIS [BBGS88] und DASDBS [ScWe87, SPSW90]. Das System GENESIS beruht auf einem funktionalen Datenmodell, EXODUS, POSTGRES und Starburst basieren auf dem relationalen Datenmodell, während DASDBS ein Datenmodell mit geschachtelten Relationen, das sog. NF^2-Modell (Non-First-Normal-Form) zugrunde liegt. EXCESS, die Sprache von EXODUS enthält auch Konstrukte, die es möglich machen, geschachtelte Relationen zu bearbeiten [CaDV88]. Außerdem wird in [GrDe87, p. 161] behauptet, daß in EXODUS auch andere als das relationale Datenmodell unterstützt werden können. Es gibt aber bisher keine Implementierungen in dieser Richtung. Die Tabelle 4.2 faßt diese Systeme, ihre Datenmodelle, Datensprachen und Konzepte zusammen.

	EXODUS	POSTGRES	Starburst	GENESIS	DASDBS
Datenmodell	relational u.a.	relational	relational	funktional	NF^2
Datensprache	relational und Erweiterungen	relational, QUEL-artig	relational, SQL-artig	funktional, vermischte Konzepte	relational, QUEL-artig
Name d. Sprache	EXCESS	POSTQUEL	Hydrogen	—	e.g. GEO-QUEL
Systemkonzepte	Toolkit	relational-erweiterbar	relational-erweiterbar	Toolkit	Kern-Architektur

Tabelle 4.2: Überblick über die diskutierten EDBS

Natürlich sind die hier beschriebenen Projekte heute abgeschlossen und die Systeme so wie beschrieben nicht mehr verfügbar. Aber die in diesen Projekten erarbeiteten Konzepte und gewonnenen Erkenntnisse sind in viele heute erhältlichen objektorientierten Erweiterungen von relationalen DBS, die sog. *objekt-relationalen DBS*, kurz ORDBS eingeflossen. Diese ORDBS, auch *Universal Server* genannt, werden heute von allen großen DBS-Herstellern angeboten (z.B. von INGRES mit der Object Management Extension, Oracle, Informix, DB2, Sybase). Insbesondere von POSTGRES kam die kommerzielle und konzeptionelle Weiterentwicklung Illustra [Illu94] heraus, wobei der DB-Server komplett neu codiert wurde [Ston97]. Illustra wurde dann

von Informix übernommen. Hier wurden die neuen Konzepte in den vorhandenen DB-Server eingearbeitet [Info96]. Viele Konzepte von Starburst sind in das kommerzielle System DB2 eingegangen [Davi98]. Die Erweiterungen werden dort *Extender* genannt; z.B. gibt es einen Spatial Extender für die räumliche Datenverarbeitung.

Interessant dabei ist, daß POSTGRES und Starburst, die schon in der sehr groben Tabelle 4.2 am ähnlichsten sind, am direktesten in kommerzielle Produkte umgesetzt wurden also auch von den Benutzern am besten akzeptiert wurden (mit der Ausnahme, daß seit Illustra auch SQL verwendet wird). Die Idee aus EXODUS und GENESIS, dem Benutzer, der die Erweiterungen programmieren muß, Hilfen, ein sog. *Toolkit* mitzuliefern, wurde jetzt von Informix wieder aufgenommen. Außerdem sind einige Konzepte aus GENESIS auch in die DB-Schnittstelle von Microsoft für Daten aus Datenbanken und anderen Datensammlungen *OLE DB* (Object Linking and Embedding for Databases) innerhalb von *COM* (Component Object Model) [COM97] eingeflossen (s. [Blak97]).

Die folgende Klassifizierung der Funktionen und die Anforderungsliste bleiben auch heute gültig. Für ein ähnliches Projekt können heute verfügbare Systeme auf der Grundlage dieses Schemas analysiert werden und auf ihre Eignung für ein Projekt hin überprüft werden.

Viele der für OODBS im ‘Object-Oriented Database System Manifesto’ [ABDD90] aufgestellten Mindestanforderungen werden auch von den EDBS erfüllt, bzw. diese Systeme lassen sich so erweitern, daß sie diesen Anforderungen genügen. Das soll im folgenden Punkt für Punkt der Anforderungsliste aus Kap. 4.3.4 für die oben genannten Beispiel-EDBS gezeigt werden.

(1) Die erste Forderung, alle Eigenschaften der bisherigen (relationalen) DBS auch zu besitzen, ist natürlich immer erfüllt, da EDBS Erweiterungen konventioneller DBS sind. Auch bei den Systemen, die nur als sog. Toolkit geliefert werden, die der Benutzer also ergänzen muß, sind diese Eigenschaften auf jeden Fall vorhanden, weil sie in den bereits vorgefertigten Moduln enthalten sind. So hat der Datenbankimplementierer zumindest immer die Möglichkeit, sie in das DBS hinein zu konfigurieren.

(2) Die Definition neuer Datentypen ist in allen betrachteten EDBS möglich. Und in allen diesen EDBS ist es auch möglich, reichhaltiger strukturierte Typen zu definieren als sie von konventionellen DBS angeboten werden. EXODUS [CaDV88] kennt die Konstruktoren Tupel, Menge, Feld und Referenz, wobei Felder variabel lange Sequenzen oder Listen sind. Alle Konstruktoren sind orthogonal, können also auch benutzerdefinierte Typen verarbeiten. In POSTGRES können neue Basistypen und zusammengesetzte Typen definiert werden. Die Konstruktoren Menge und Feld gibt es nicht direkt, auf Umwegen kann jedoch der gleiche Effekt erreicht werden: Es gibt den Attributtyp PROCEDURE, der besagt, daß ein Attribut den Namen einer Prozedur enthält, die jedesmal ausgeführt wird, wenn der Wert des Attributs gefragt ist. Es ist auch zulässig, daß hier eine Menge von Werten geliefert wird. Auf vollständige Orthogonalität aller Konstruktoren wurde in POSTGRES noch zugunsten von besserer Performance verzichtet. Im Starburst-System war die Komponente zur Definition neuer Datenty-

pen zur Projektlaufzeit noch in der Entwicklung [LiHa90], deshalb ließ sich über die angebotenen Konstruktoren noch nichts Genaues sagen. Die in [WSSH88] aufgestellten Anforderungen lassen aber auf ein mächtiges Werkzeug schließen. In DASDBS [SPSW90] stehen mit dem NF^2-Modell die Konstruktoren Tupel und Menge orthogonal zur Verfügung. Im GENESIS-System gibt es die Konstruktoren Tupel, Menge, Link und endliche Menge. Über deren Orthogonalität wird in der zitierten Literatur keine Aussage gemacht.

(3) Zu den Punkten der wertunabhängigen Identifikation und der Surrogat-Schlüssel bzw. systemgenerierten Schlüssel wird in der Literatur zu keinem der Systeme eine präzise Aussage gemacht. Es scheint in allen fünf Systemen nicht möglich zu sein oder nur auf explizite Angabe des Benutzers hin.

(4) Die Systeme EXODUS und POSTGRES bieten zur Definition der benutzerdefinierten Datentypen und der Operationen darauf eine Schnittstelle nach der Syntax der ADT. Damit ist eine Trennung von Definition und Implementierung der Datentypen und Operationen gegeben und die Kapselung wird ermöglicht. In Starburst werden Operationen und später auch Datentypen ebenfalls gekapselt implementiert. Die Literatur zu DASDBS und GENESIS enthält hierzu keine konkreten Aussagen.

(5) Wie bereits in (4) gesagt, werden ADT-Schnittstellen von den Systemen EXODUS und POSTGRES angeboten, und so auch das damit verbundene Typkonzept. Für Starburst ist auch ein Typkonzept geplant, bei DASDBS und GENESIS gibt es hierzu keine Aussage.

(6) Zu allen betrachteten Systemen außer GENESIS wird behauptet, daß Hierarchien von Typen konstruiert werden können und Eigenschaften durch diese Hierarchien vererbt werden können. In der Literatur zu keinem der Systeme wird aber Genaueres zu Vererbung der Eigenschaften angegeben. Wie die dafür geeignete Schnittstelle aussieht, z.B. nach Art der ADT, wird auch nicht genauer beschrieben.

(7) In EXODUS, POSTGRES und Starburst ist das Überladen von Operatoren möglich. In der Literatur über DASDBS und GENESIS wird nichts dazu gesagt.

(8) Bei Systemen wie POSTGRES und Starburst, die die Implementierung neuer Operatoren und Funktionen in einer höheren berechnungsuniversellen Programmiersprache erlauben, kann auch damit erreicht werden, daß die Anfragesprache berechnungsuniversell wird. Für DASDBS könnte auch eine Schale um den Kern mit berechnungsuniverseller Anfragesprache implementiert werden.

(9) Die Erweiterbarkeit dieser Systeme ist selbstverständlich gewährleistet. Allerdings kann dies beim Toolkit- und Kernansatz nicht durch den Endbenutzer geschehen.

Die EDBS erfüllen also im wesentlichen die Anforderungen, die an OODBS gestellt werden ebenso gut wie heutige OODBS, die vor Veröffentlichung des OODBS Manifesto implemen-

tiert wurden. Die Systeme POSTGRES und EXODUS werden häufig auch unter den OODBS aufgeführt. Dort, wo die Forderungen in heutigen Systemen noch nicht ganz erfüllt werden, geschieht dies im wesentlichen aus Gründen der Performance: Je allgemeiner neue Datentypen und Funktionen vom Benutzer definiert werden können, desto aufwendiger wird die Optimierung von Anfragen später. OODBS und EDBS unterscheiden sich eigentlich nur durch die Motivation, aus der heraus sie entwickelt wurden. Bei den OODBS stand die Definition komplexer Objekte im Vordergrund. Daraus ergab sich dann aber auch die Notwendigkeit, neue Funktionen zu definieren, um mit diesen Objekten richtig arbeiten zu können. Bei den EDBS war die Möglichkeit zur die Definition neuer Funktionen und Operationen durch den Benutzer die wichtigste Neuerung. Dies ist aber in vielen Fällen nicht mächtig genug. Erst durch die Definition neuer komplexer Datentypen lassen sich auch geeignete Bild- und Urbildmengen für diese Funktionen definieren. In dieser Arbeit soll im folgenden nur noch von EDBS gesprochen werden, weil in den folgenden Betrachtungen die Erweiterbarkeit vorhandener Systeme im Vordergrund steht. In dem System, das in den in späteren Kapiteln vorgestellt wird, liegt der Schwerpunkt auf der Definition neuer Funktionen.

4.4 Formen der Erweiterbarkeit

Es wird nun zunächst darauf eingegangen, welche Erweiterungen in den verschiedenen EDBS selbst bereits enthalten sind und welche Erweiterungsmöglichkeiten dem Benutzer angeboten werden. Danach wird untersucht, mit welchem Aufwand die Einbettung verschiedener neuartiger Funktionen und Operationen erfolgen muß und wie gut die neuen Funktionen dann genutzt werden können. Dies ist ein wichtiges Kriterium dafür, ob sich der Aufwand für den Einsatz eines EDBS für eine kleinere Anwendung überhaupt lohnt.

4.4.1 Neue Datentypen

Wenn im Zusammenhang mit EDBS von neuen Datentypen gesprochen wird, sind meist reicher strukturierte Objekte gemeint, die sich aus Grunddatentypen (i.a. integer, real, double, character, strings und evtl. bool) zusammensetzen. Dazu müssen vom DBS die in Punkt (2) des OODBMS Manifesto beschriebenen Konstruktoren angeboten werden. Beispiele für solche Konstruktoren sind Menge, Liste, Feld oder Struktur. Mit Hilfe dieser Konstruktoren kann der Benutzer selbst aus den vom System angebotenen Standarddatentypen und den von ihm schon konstruierten zusammengesetzten Datentypen neue Datentypen konstruieren.

Mit den Konstruktoren muß der Benutzer immer auf die Standarddatentypen zurückgreifen. In manchen Fällen ist es aber wünschenswert, noch andere Grunddatentypen zu haben, die auch ganz andere Eigenschaften aufweisen als die bisher angebotenen Typen. Deshalb werden von einigen DBS, auch als Erweiterungen bezeichnet, neue Grunddatentypen angeboten. Von DASDBS werden z.B. im Grundsystem einige über die bekannten hinausgehende Datentypen,

speziell zur Beschreibung geometrischer und geographischer Objekte angeboten. Zu diesen Datentypen werden auch die passenden Operationen darauf bereitgestellt.

Zu den neuen Grunddatentypen, die von einigen DBS unterstützt werden, gehören die sog. *Binary Large Objekts* (BLOBs), das sind große unstrukturierte Objekte, etwa gerasterte Photographien, Satellitenaufnahmen oder digitalisierte Geräusche. Diese Objekte haben ganz spezielle Eigenschaften, die bei der Verarbeitung unbedingt berücksichtigt werden müssen, z.B. passen sie gewöhnlich nicht auf eine Speicherseite. Bei der Anfrageoptimierung müssen sie deshalb anders behandelt werden, und es werden ganz andere Operationen, Speicherstrukturen und Zugriffsmechanismen benötigt. Auch als Teil- oder Unterobjekte innerhalb komplexer Objekte müssen diese Datentypen gesondert behandelt werden. Andere Beispiele sind die sog. *File*-Datentypen, bei denen eine ganze Datei einen Attributwert bildet oder *text*-Datentypen die mehrseitige Texte enthalten können.

Soweit aus der gängigen Literatur bekannt, gibt es bisher kein System, das wirklich benutzerdefinierte neue Grunddatentypen zuläßt und alle notwendigen Erweiterungen dazu anbietet, angefangen bei der Definition und Implementierung neuer Speicherstrukturen (durch den Benutzer) bis hin zur Bereitstellung von geeigneten Optimierungsverfahren (benutzerdefiniert). (Informix erlaubt mit den DataBlades praktisch alles bis auf die Definition neuer Optimierungsverfahren. Genaueres dazu siehe Kap. 4.6.) Am Beispiel der BLOBs und drei der fünf verschiedenen EDBS, die in dieser Arbeit näher betrachtet werden - POSTGRES, EXODUS und Starburst - soll nun gezeigt werden, wie verschieden die Ansätze zur Implementierung sein können. (In der Literatur zu GENESIS und DASDBS wurden BLOBs nicht näher betrachtet.) Zusätzlich wird ersichtlich, wie kompliziert es sein könnte und welche Folgen es haben könnte, dem (End-) Benutzer solche Erweiterungen zu ermöglichen.

Auch in verschiedenen kommerziellen DBS werden die BLOBs schon angeboten, jedoch (nach [StOl93]) ohne die passenden Operationen darauf oder die Möglichkeit, solche Operationen zu definieren. Ein Grund für die fehlenden Operationen ist, daß es nur einen Datentyp “BLOB” gibt, ein stärkeres Typisieren wäre notwendig: Ein Überlagern von Bildern muß anders erfolgen als ein Überlagern von Geräusch- oder Tonsequenzen.

In POSTGRES [StOl93] wurden vier Ansätze gemacht, BLOBs zu integrieren und zu implementieren. Die Schnittstelle zum Benutzer ist jedoch in allen vier Fällen gleich: Es müssen Ein- und Ausgabeprozeduren geschrieben werden, die die großen Objekte aus Dateien in die Datenbank einlesen und umgekehrt aus der Datenbank in Dateien schreiben.

Die erste und einfachste Möglichkeit zur Implementierung besteht darin, Benutzerdateien als große ADT zuzulassen. Das hat verschiedene, große Nachteile: Die Sicherheit ist ungenügend, Transaktionsverwaltung und die in POSTGRES üblichen Versionen sind dann auf BLOBs nicht möglich. Der zweite Ansatz bestand darin, nur POSTGRES-eigene Dateien als ADT zuzulassen, um die Sicherheit zu verbessern. Die Probleme mit den Transaktionen und Versionen werden dadurch aber nicht gelöst. Im dritten Ansatz werden die großen Objekte in Datenblöcke fester

Länge aufgeteilt. Diese Datenblöcke können wie andere Datenbankobjekte behandelt werden. So gibt es keine Probleme mehr mit der Transaktionsverwaltung und der Versionenbildung. Im vierten Ansatz wurden Datenblöcke variabler Länge verwendet, was bei der Komprimierung der großen Objekte Vorteile haben kann. Aus diesen Ansätzen wurde dann die DataBlade-Technologie entwickelt, wie sie heute in Informix verfügbar ist [Info96].

Im System EXODUS gibt es Speicherobjekte von virtuell beliebiger Länge [CDRS86]. Große Objekte werden auch hier in Unterobjekte der Größe “normaler” DB-Objekte zerteilt, die dann in einer B^+-Baum-Struktur gespeichert werden. Dieses Verfahren hat den besonderen Vorteil, daß Änderungen innerhalb eines großen Objekts leicht vorgenommen werden können. Über die Ein-/ Ausgabe von BLOBs wird in [CDRS86] nichts gesagt. Da EXODUS ein modulares System ist, das durch den Datenbankimplementierer erst noch zusammengesetzt werden muß, wäre es nur konsequent, diesem auch, ähnlich wie dem Benutzer in POSTGRES, die Implementierung entsprechender Ein-/ Ausgabeprozeduren zu überlassen.

In Starburst heißen die BLOBs “lange Felder” (long fields) [LeLi89]. Sie dürfen bis zu maximal 120 MBytes groß sein. Auch hier werden die langen Felder in Segmente zerteilt, die dann nach dem Prinzip den Buddy-Systems [Knut75] gespeichert werden. Zu jedem langen Feld gibt es einen Deskriptor, der an Stelle des eigentlichen Objekts in der Relation gespeichert wird. Wie die End-Benutzerschnittstelle zu den langen Feldern aussieht, darüber gibt es in [LeLi89] keine Angaben. Beim Entwurf des Speichersystems von Starburst wurde besonderer Wert auf eine gute Verarbeitungsgeschwindigkeit von langen Feldern gelegt. Und es wurde darauf geachtet, daß paralleler Zugriff und eine schnelle Wiederherstellung nach Systemfehlern und dergleichen möglich sind.

An diesen drei Beispielen kann man sehen, daß es erhebliche Probleme bei der Bereitstellung von BLOBs für den Benutzer gibt: Da muß zunächst eine Benutzerschnittstelle geschaffen werden. Bilder z.B. können nicht einfach wie Zahlen oder eine Zeichenkette vom Endbenutzer am Bildschirm per INSERT eingefügt werden oder in eine Bildschirmmaske eingetippt werden. Bei der Ausgabe gibt es ähnliche Probleme. Auch alle anderen Standard-Funktionen eines DBS wie Transaktionsverwaltung, Mehrbenutzerbetrieb und Wiederherstellungsmöglichkeiten müssen für BLOBs neu durchdacht und evtl. auch implementiert werden. Man sieht am Beispiel der BLOBs, daß es aus diesen Gründen es kaum möglich ist, dem Benutzer die Möglichkeit zur Erweiterung seines DBS durch beliebige neue Datentypen offen zu halten.

4.4.2 Verschiedene Funktions- und Prozedurerweiterungen in DBS

Unter benutzerdefinierten Operationen oder Funktionen werden in den verschiedenen DBS ganz unterschiedliche Angebote des Systems an den Benutzer verstanden. In diesem Kapitel werden die verschiedenen Funktionen zusammengestellt und systematisch klassifiziert. Dabei sollen auch Funktionen, genauer Funktionsschemata, bestimmt werden, die noch in keinem System angeboten werden, aber dennoch nützlich wären.

Die bisher in DBS und EDBS angebotenen Funktionen können nach zwei Gesichtspunkten klassifiziert werden:

1) wo sie im DBS eingesetzt werden und wie weit ihr Gültigkeitsbereich geht,

2) welcher Art ihre Ein- und Ausgabeparameter sind (Skalar, Menge Tupel, Relation) und wieviele Eingabeparameter sie haben.

Zunächst werden die verschiedenen benutzerdefinierbaren Funktionen beschrieben. Dabei werden auch, soweit zutreffend, die Art der Ein- und Ausgabeparameter mit angegeben.

- Berechnung abgeleiteter Attribute. Anstatt konkreter Werte werden bei einigen Attributen (skalare) Funktionen mit Eingabeparametern gespeichert. Die verwendeten Operationen müssen bereits im DBS (built-in oder benutzer-definiert) vorhanden sein. Der oder die Eingabeparameter dieser Funktionen können skalar oder auch relationenwertig sein (POSTGRES-Datentyp POSTQUEL). Ebenso kann die Ausgabe skalar, ein Tupel oder eine Menge bzw. Relation sein. So lassen sich z.B. auch nicht-normalisierte Relationen darstellen. Die Eingabeparameter müssen immer in irgendeiner Beziehung zum berechneten Attribut stehen (dasselbe Tupel, Fremdschlüssel-Beziehungen). In konventionellen DBS ist so etwas nur in View-Definitionen möglich und hat dann immer die Nicht-Änderbarkeit des gesamten Views zur Folge.

- Trigger und Regeln. Das sind Mechanismen, die bei Einfügungen oder Änderungen im Datenbestand ohne weiteres Zutun des Benutzers ausgeführt werden und z.B. ein Einfügen von Tupeln an mehreren Stellen bewirken oder bestimmte Änderungs- oder Löschwünsche zurückweisen. Trigger und Regeln bestehen immer aus zwei Teilen: im ersten Teil wird eine Bedingung ausgewertet, d.h. beliebige Eingabeparameter sind erlaubt, und es wird ein bool'scher Wert zurückgeliefert. Der zweite Teil sieht aus wie die unten beschriebenen Datenbankprozeduren.

- Datenbankprozeduren oder ´stored procedures´. Das sind Befehlsfolgen der Anfragesprache, wie sie auch interaktiv oder in einem Anwendungsprogramm möglich wären, evtl. erweitert um Kontrollstrukturen wie IF-THEN-ELSE. Sie werden permanent in der DB gespeichert, um dem Benutzer die wiederholte Eingabe zu ersparen. Wenn diese DB-Prozeduren mit Zugriffsrechten gekoppelt werden können, lassen sich damit, auch ohne Trigger, umfangreiche Integritätsbedingungen realisieren (POSTGRES). Wenn z.B. Änderungsrechte ausschließlich an Programme, also Anwendungen vergeben werden können und diese dann stets die entsprechende DB-Prozedur verwenden - beides ist leicht zu kontrollieren - , ist die Einhaltung der dort spezifizierten Integritätsbedingingen sichergestellt.
  Durch die zusätzlichen Kontrollstrukturen sind Datenbankprozeduren mächtiger als reine SQL-Anweisungen. Die Anzahl und Art der Ein- und Ausgabeparameter sind nicht festgelegt.

- Einfache skalare Funktionen, die aus einem oder mehreren skalaren Werten (einzelnes Tupel oder Konstanten) einen neuen Wert berechnen (ähnlich den Standard-Operationen wie +, -, *, >, sqrt). Sie haben immer die Form:

$$f_s: \text{skalar } \{ \times \text{skalar } \} \rightarrow \text{skalar}$$

- Funktionen bzw. Operatoren auf benutzerdefinierten Datentypen. Dabei können sowohl vorhandene Operatoren um die neuen Datentypen erweitert werden als auch neue, typspezifische Funktionen definiert werden. Diese Operatoren und Funktionen haben als Argument(e) immer nur einzelne Instanzen eines (Attributs eines) bestimmten Datentyps und liefern auch nur skalare Ergebnisse:

$$f_{bd}: \text{skalar } \{ \times \text{skalar } \} \rightarrow \text{skalar}$$

- Mengenprädikate (Starburst). Standard-DBS bieten nur die Quantoren ALL und EXISTS bzw. ANY. Es sind aber noch andere Mengenprädikate denkbar etwa MAJORITY (Mehrheit) oder “mindestens ein Drittel” bzw. “mindestens 10". Mengenprädikate haben immer die Form:

$$\text{MP: relation} \rightarrow \text{skalar (bool)}$$

- Neue Aggregationsfunktionen (Starburst), wie z.B. der kleinste positive Wert einer Menge von Werten. Die in Standard-DBS angebotenen Aggregationsfunktionen haben die Form:

$$A_s: \text{menge} \rightarrow \text{skalar}$$

Es sind aber noch andere, weitere Eingabeparameter für Aggregationsfunktionen denkbar:

$$A_{bd1}: \text{menge } [\times \text{skalar }]^* \rightarrow \text{skalar}$$
$$A_{bd2}: \text{relation } [\times \text{skalar }]^* \rightarrow \text{skalar} \mid \text{tupel}$$
$$A_{bd3}: \text{relation} \times \text{tupel} \rightarrow \text{skalar} \mid \text{tupel}$$

- Funktionen auf ganzen Relationen, *Table Functions* (Starburst). Diese Funktionen erhalten mindestens eine Relation als Eingabeparameter und bei Bedarf noch weitere skalare oder tupelförmige Eingabeparameter. Sie liefern eine neue Relation, z.B. eine zufällige Auswahl von 100 Tupeln oder eine Projektion auf die Schlüsselattribute als Ausgabe. Sie haben die Form:

$$T: \text{relation } \{ \times \text{skalar } \}^* \{ \times \text{tupel } \} \rightarrow \text{relation}$$

Die verschiedenen Arten von Funktionen können zunächst nach ihrem *Gültigkeitsbereich* klassifiziert werden. Unter Gültigkeitsbereich wird hier verstanden, wie weit die Funktion bekanntgemacht wird. Eine Funktion kann entweder einer oder einzelnen Relationen bekannt sein oder

in der gesamten Datenbank oder installations- bzw. netzwerkweit im DBS. Die oben zusammengestellten Funktionen haben folgende Gültigkeitsbereiche:

Art der Funktion	Gültigkeitsbereich
Berechnung abgeleiteter Attribute	eine Relation
Trigger, Regeln	eine oder mehrere Relationen
Datenbankprozeduren	datenbankweit
ben.-def. skalare Funktionen ben.-def. Mengenprädikate ben.-def. Aggregationsfunktionen ben.-def. Table Functions	installations- oder netzwerkweit

Tabelle 4.3: Gültigkeitsbereiche benutzerdefinierter Funktionen

Die Berechnung abgeleiteter Attribute, die Defintion von Triggern, Regeln und Datenbankprozeduren können sich ganz aus den vom DBS angebotenen Funktionen und Operationen zusammensetzen. Um sie nutzen zu können, sind aber Erweiterungen der Anfragesprache, der DML und der DDL notwendig (siehe Kap. 4.4.3). Zu bemerken ist außerdem, daß nur Trigger bzw. Regeln und Datenbankprozeduren direkt den Inhalt der Datenbank verändern können. Alle anderen benutzerdefinierten Funktionen können dies nur, wenn sie in Verbindung mit INSERT-, UPDATE- oder DELETE-Anweisungen angewendet werden.

Die zweite Möglichkeit, benutzerdefinierte Funktionen zu klassifizieren, ist die Unterteilung nach Typ und Anzahl der Ein- und Ausgabeparameter. Hier bilden die Datenbankprozeduren, die abgeleiteten Attribute und die Trigger/Regeln Ausnahmen: Datenbankprozeduren können eine beliebige Zahl von Eingabeparametern beliebigen Typs und auch eine beliebige Anzahl von Ausgabeparametern beliebigen Typs haben. Abgeleitete Attribute und Trigger/Regeln können eine beliebige Zahl Eingabeparameter beliebigen Typs haben. Abgeleitete Attribute liefern immer einen skalaren Wert, den Attributwert zurück; Trigger/Regeln nur einen Statuscode.

In der folgenden Tabelle 4.4 werden der Übersichtlichkeit halber in der oberen Hälfte die in SQL vorhandenen Operationen und Funktionen aufgeführt, in der unteren Hälfte dann die benutzerdefinierten Prozeduren und Funktionen:

Funktion/ Operation	#Ein.-parm.	Eingabeparameter	Ausgabeparameter
Selektion (WHERE)	1	Relation	Relation (Menge von bool)
Projektion (SELECT)	mind. 2	Relation [x Skalar]*	Relation
Kart. Produkt (FROM)	2	Relation x Relation	Relation
Vereinigung (UNION)	2	Relation x Relation	Relation
Built-in-Funktion	2	Skalar x Skalar	Skalar
Aggregationsfunktion	1	Menge	Skalar
Mengenprädikat	2	Relation x Skalar Relation x Tupel	bool bool
Gruppierung	mind. 2	Relation x [Skalar]*	Menge von Relationen
Sortierung	mind 2	Relation x [Skalar]*	Relation (Liste)
ben.-def. skalare Funktion	mind. 2	Skalar x Skalar Tupel x [Skalar]*	Skalar Tupel
ben.-def. Mengenprädikat	mind. 2	Relation x [Skalar | Tupel] x [Skalar]*	bool
ben.-def. Aggr.-Funkt.	mind. 1	Menge Menge x [Skalar]* Relation x [Skalar]* Relation x Tupel	Skalar Skalar Skalar | Tupel Skalar | Tupel
ben.-def. Table Function	mind. 1	Relation {x Skalar}* {x Tupel}	Relation
Abgeleitetes Attribut	beliebig	beliebig	Skalar
Trigger/Regel	beliebig	beliebig	— (Return-Code)
Datenbankprozedur	beliebig	beliebig	beliebig (+ Return-Code)

Tabelle 4.4: Klassifikation von Operationen/Funktionen nach Ein-/Ausgabeparametern

Unter einem Skalar kann hier auch immer ein strukturiertes, aber endlich langes Objekt (benutzerdefiniert) verstanden werden. Unter den skalaren Funktionen ist u.a. auch das Ansprechen von Komponenten strukturierter Objekte zu finden.

In einigen EDBS, z.B. POSTGRES, wird zusätzlich zwischen Funktionen und Operatoren unterschieden. Hier steht aber nicht der syntaktische Unterschied (f (x, y) vs. x op y) im Vordergrund sondern die Implementierung bzw. Integration in das DBS. Als Funktionen bezeichnet man hier die ‘normalen’ benutzerdefinierten Funktionen, die dem System bekanntgegeben wer-

den, über deren Vorgehensweise die Anfragebearbeitung aber nichts weiß und deren Ausführung deshalb nicht optimiert werden kann. Operatoren werden anders als Funktionen voll eingebunden und ihre Arbeitsweise wird bei der Optimierung berücksichtigt. Daher werden bei ihrer Definition vom Datenbank-Programmierer mehr Kenntnisse von DBS-Interna und Programmieraufwand verlangt: Es müssen Hilfsfunktionen für den Optimierer geschrieben werden. Auf benutzerdefinierten Datentypen läßt POSTGRES nur benutzerdefinierte Funktionen zu, keine Operatoren.

Der Einfluß benutzerdefinierter Operatoren auf die Anfrageoptimierung soll noch einmal an einem einfachen Beispiel erläutert werden. Auch Benutzern des INGRES-Systems ist es möglich, eigene Datentypen sowie neue Funktionen und Operatoren auf vorhandenen sowie eigenen Datentypen zu definieren und zu implementieren, wenn ihre Installation die INGRES Object Management Extension (INGRES Erweiterung zur Objekt-Verwaltung) umfaßt [INGR91c]. Hier ist u.a. die Definition neuer Vergleichsoperatoren (=, !=, >, >=, <, <=) möglich. Bei der Anfrageoptimierung, insbesondere bei der Entschachtelung geschachtelter Anfragen ist es gelegentlich notwendig, einen Vergleichsoperator durch sein Komplement zu ersetzen (s.a. Kap. 8.5). Deshalb verlangt INGRES nicht nur, daß dem System mitgeteilt wird, daß es sich bei dem neuen Operator um einen Vergleichsoperator handelt, sondern auch, daß der jeweils komplementäre Operator auch bereitgestellt, also definiert und implementiert wird (!= zu =, > zu <=, >= zu < und jeweils umgekehrt). In der Operator-Definition muß außerdem gesagt werden, welches der komplementäre Operator ist (alle Operatoren erhalten eindeutige Nummern). INGRES läßt neben benutzerdefinierten Funktionen nur benutzerdefinierte Vergleichsoperatoren, arithmetische Operatoren und automatische Typanpassungen (coercion) zu [INGR91c]. Die notwendigen komplementären Vergleichsoperatoren sind hier die einzige Voraussetzung für den Optimierer und der Benutzer wird hier im Handbuch deutlich darauf hingewiesen. Je mehr benutzerdefinierte Operatoren ein System zuläßt und je aufwendiger die Anfrageoptimierung des Systems ist, desto umfangreicher und komplizierter werden die Aufgaben des Datenbank-Programmierers bei der Definition und Implementierung neuer Operatoren.

In Tabelle 4.5 werden noch einmal die verschiedenen Möglichkeiten der Benutzer, neue Funktionen und Prozeduren zu definieren zusammengestellt, soweit dies aus der zitierten Literatur hervorgeht.

	EXODUS	POSTGRES	Starburst	GENESIS	DASDBS
Scalar, Simple Functions	[CaDV88 p.421]	[StRH90 p.127]	[HFLP88 p.3]	[BaLW88 p.258]	[SPSW90 p.37]
Operatoren für ben.-def. Typen	[CaDV88 p.420]	[StRH90 p.128]	[LiHa90 p.231]	[BBGS88 p.1720]	[SPSW90 p.38]
Mengen-prädikate	—*)	—	[HFLP88 p.4]	—	—
Aggregations-funktionen	[CaDV88 p.422]	[StAH87a p.80]	[HFLP88 p.3]	—	—
Table Functions	—*)	—**)	[LiHa90 p.229] [HFLP88 p.4]	—	—
Berechnung abgel. Attribute	[CaDV88 p.421]	[StRH90 p.128]	—	—	—
Trigger, Regeln	—	[StRH90 p.134]	[LiHa90 p.237]	—	—
Datenbank-Prozeduren	[CaDV88 p.421]	[StAH87 p.352] [StAH87a p.76] [StRH90 p.128]	—	—	—
Operatoren	[CaDV88 p.420]	[StRH90 p.127]	—	—	—

*) Etwas Gleichwertiges läßt sich auf Umwegen mit EXCESS programmieren.

**) POSTGRES erlaubt auch die Definition von Funktionen auf ganzen Relationen. Sie lassen sich innerhalb der POSTQUEL-Anfragesprache aber nicht voll ausnutzen.

Tabelle 4.5: Übersicht, welche EDBS welche Typen von Funktionserweiterungen unterstützen

4.4.3 Spracherweiterungen

In den Kapiteln 4.4.1 und 4.4.2 wurden *Erweiterungen des DBS* und damit der Datensprache um Datentyp- und Funktionsnamen vorgestellt, die der Benutzer selbst vornehmen kann. Um diese Erweiterungen aber voll ausnutzen zu können, sind einige *Spracherweiterungen innerhalb der Anfragesprache* notwendig, die neue Arten von Klauseln [LiHa90] zulassen. Während POSTGRES zwar (in der Endausbau-Stufe) die umfangreichsten Möglichkeiten bietet, Funktionen zu definieren, ist die Spanne der Anwendungen, die mit Starburst implementiert werden können, größer. Starburst bietet wichtige Spracherweiterungen, um diese neuen, selbst implementierten Funktionen, wie z.B. Table Functions auch voll einsetzen zu können.

Bei den Spracherweiterungen kann wieder in DDL und DML bzw. Anfragesprache unterschieden werden. Zunächst sollen hier die für die am Anfang dieses Kapitels beschriebenen Datentyp- und Funktionsdefinitionen notwendigen Erweiterungen aufgezählt werden.

Für die Handhabung von benutzerdefinierten Datentypen sollten noch mindestens folgende Spracherweiterungen eingeführt werden:

- Syntax zur Definition von neuen Datentypen, etwas wie CREATE DATATYPE, DROP DATATYPE und Schlüsselwörter für die möglichen Konstruktoren (DDL),
- neue, benutzerdefinierte Namen an jeder Stelle (DML),
- Ansprechen von Komponenten in strukturierten Objekten, etwa die Punktnotation in POSTGRES [StRH90] (DML),

Die Definition neuer Datentypen wird sich i.a. nicht auf eine Anweisung beschränken können, wie sich der neue Datentyp aus vorhandenen zusammensetzt bzw., wie er aus ihnen abgeleitet wird. Das DBS benötigt eine Reihe von Funktionen für die Konvertierung bei der Ein- / Ausgabe von Werten des neuen Datentyps, für die Anfrageoptimierung, wie Statistiken zu sammeln sind und wie eine totale Ordnung auf den Werten definiert ist, damit die üblichen Speicherstrukturen angelegt werden können.

Für das Arbeiten mit benutzerdefinierten Prozeduren, Funktionen u.ä. müssen folgende Voraussetzungen geschaffen werden:

- Definition von Datenbankprozeduren (CREATE PROCEDURE, DROP PROCEDURE), gibt es schon in einigen kommerziellen DBS wie z.B. INGRES,
- Erweiterung der Attributdefinition in der CREATE TABLE-Anweisung zur Beschreibung abgeleiteter Attribute
- (interaktive) Definition von Triggern, in konventionellen DBS bisher meist nur gebunden an Anwendungen oder bestimmte Tools, Anwendungsgeneratoren, bzw. Definition von Regeln (POSTGRES, Starburst), etwa:
  CREATE RULE ON <Anlaß> <Prädikat/Trigger>
  <Anweisungsliste bzw. DB-Prozedur>;
  Löschen von Reglen wie DROP RULE
- Definition von Funktionen, Operatoren, Mengenprädikaten und Table Functions,
- Syntax für Aggregationsfunktionen mit zusätzlichen Parametern (DML),
- Anwendung der Table Functions, also Funktionsaufrufe in der FROM-Klausel.
- Für abgeleitete Attribute, die Anwendung von Triggern und Regeln sowie von DB-Prozeduren innerhalb von Triggern werden keine Erweiterungen der DML benötigt. Benutzerdefinierte Funktionen und Operatoren werden wie andere built in-Funktionen bzw. Operatoren auch angewendet.

Im POSTGRES-System wird POSTQUEL, eine Erweiterung der Datensprache QUEL verwendet. Alle anderen Systeme verwenden Erweiterungen von SQL, der einzigen Datensprache für relationale Datenbanken, für die bisher ein Standard definiert wurde [ANSI86, ANSI89, ANSI89a], und für die an weiteren Standards gearbeitet wird [ISO90].

Für einige der oben genannten Erweiterungen existieren Vorschläge im SQL-Standard (SQL2, SQL3 [ISO90, Shaw90, Matt96]), so daß es voraussichtlich in einigen Jahren eine einheitliche

Syntax geben wird. Der Standard SQL2 kann in mehreren Stufen verwirklicht werden. In der letzten Stufe werden noch viele andere nützliche Erweiterungen vorgeschlagen. Einige, die für die in dieser Arbeit verfolgten Ziele nützlich sein können, werden hier aufgezählt. (In Kapitel 9 wird näher darauf eingegangen:)

- Wertebereichsbeschränkungen (domain constraints), auch in konventionellen DBS schon teilweise verwirklicht,
- Schlüsseldeklarationen, in konventionellen DBS meist nur im Umweg über eine entsprechende Speicherstruktur,
- referentielle Integrität oder Fremdschlüsselbeziehungen,
- rekursive Anfragen, Modellierung von Bäumen oder Netzen,
- Orthogonalität in SQL [Date84, LiHa90],
- Beschreibung von Integritätsbedingungen.

Zur Orthogonalität gehören Erweiterungen wie

- Schachtelung in der SELECT-Klausel,
- Schachtelung in der FROM-Klausel,
- Aggregationsfunktionen u.ä. an beliebiger Stelle in allen Anfragen.

4.4.4 Andere Erweiterungen

In diesem Abschnitt werden Erweiterungen von DBS vorgestellt, die sich in die vorhergehenden Kapitel nicht direkt einordnen lassen und die nicht nur die Erweiterung der Datensprache betreffen.

Insbesondere bei interaktiven Anfragen an das DBS ist es wichtig, daß nicht nur neue Datentypen und komplexe Objekte definiert werden können, sondern daß diese auch dem Benutzer in angemessener Form vom DBS präsentiert werden können. Es besteht zwar die in Abschnitt 4.4.3 angedeutetete Möglichkeit, sie vom Benutzer bei der Datentypdefinition spezifizieren zu lassen, aber für die vom System angebotenen Konstruktoren, können auch schon Mechanismen im System existieren (POSTGRES, Object Management von INGRES).

Zusätzlich können neuartige Speicherstrukturen und / oder Zugriffspfade angelegt werden für folgende Zwecke:

- Die Eigenschaften der neuen Datentypen müssen berücksichtigt werden wie variabel lange Tupel und BLOBs.
- Mehrdimensionalität (geometrisch, geographisch) muß berücksichtigt werden.
- Neue Speichermedien wie optische Platten können einbezogen werden.
- Archive können einbezogen werden.

Die Möglichkeit, neue Speichermedien verwenden zu können, ist wichtig, wenn große Mengen Daten anfallen, die nicht mehr verändert werden, wie z.B. Bilder aller Art. Neue Speichermedien können u.U. auch dann effizient genutzt werden, wenn dies nicht explizit vorgesehen ist,

aber umfangreiche Erweiterungen bzw. Änderungen bei der Implementierung physischer Speicherstrukturen und Zugriffspfade möglich sind.

Der Bedarf für Versionenverwaltung und für eine (fast) für den Benutzer transparente Verteilung der Daten wurde bereits in Kap. 3 motiviert. Eine aufwendigere Transaktionsverwaltung kann z.B. bei der Bearbeitung von BLOBs nützlich sein, ist aber i.a. bei Umwelt-DB und Meßwerten nicht unbedingt erforderlich.

Wissensrepräsentation nach den Konzepten der Künstlichen Intelligenz kann ebenfalls nützlich sein. So eine Komponente wurde nur in den Konzepten von DASDBS [ScWe87] erwähnt, allerdings keine entsprechende Deduktionskomponente. Wenn in anderen Systemen wie Starburst und POSTGRES von Regeln und Regelauswertung (-skomponente) gesprochen wird, sind immer DBS-Interna, wie etwa der Optimierer gemeint, für die der Benutzer bei der Definition von Erweiterungen weitere Regeln angeben kann.

Je höher die Modularität des Systems ist und dies auch für den Benutzer sichtbar ist, desto eher ist es vom demjenigen, der die Erweiterungen programmiert, zu durchschauen und zu ergänzen. I.a. ist aber trotzdem ein Spezialist, der *Datenbankimplementierer* (DBI) erforderlich. Wenn jedoch, wie bei POSTGRES genaue Anweisungen zur Erstellung und Einbindung der Erweiterungen gegeben werden, braucht Programmierer werder das Gesamtsystem zu verstehen noch Spezialist zu sein, wenn nur kleinere Erweiterungen angebracht werden sollen. Bei DASDBS war schon die Zielsetzung anders: Hier wurde davon ausgegangen das ganze anwendungsorientierte ´front-ends´ implementiert werden.

Eine Zusammenstellung der in diesem Abschnitt besprochenen Punkte ist der Tabelle 4.6 zu entnehmen, die mit Hilfe der angegebenen Literatur aufgestellt wurde. Im oberen Teil der Tabelle sind die Erweiterungen aufgelistet, die über die Datensprache hinausgehen. Im unteren Teil werden Angaben zur Implementierung der Erweiterungen gemacht.

(Diskutierte Punkte, die hier in Tab. 4.6 nicht aufgeführt sind, wurden von keinem der betrachteten EDBS angeboten.)

	EXODUS	POSTGRES	Starburst	GENESIS	DASDBS
Ausgabe / Präsentation von Datentypen		x			
Zugriffspfade	x	x	x		x
Speichermedien: Optische Platten		x	x		
Versionenverwaltung	x	x			
auch als verteiltes DBS		(x)	(x)		
geschachtelte Transaktionen	x				x
Wissensrepräsentation, Regeln		x	x		x**
System modular aufgebaut	x		x*	x	
DBI erforderlich	x		x	x	x

(x): eingeschränkt möglich / nur im Entwurf
* : für Erweiterungen
** : nach den Konzepten der Künstlichen Intelligenz

Tabelle 4.6: Möglichkeiten verschiedener EDBS

4.5 DB-Erweiterungen zur Unterstützung von Umweltprojekten

Mit den bisher in diesem Kapitel vorgestellten Erweiterungen lassen sich noch nicht alle in Kap. 3.6 aufgestellten Forderungen an ein Umwelt- / Meßwert-DBS erfüllen. Alle Erweiterungen, die die EDBS bzw. OODBS anbieten, zielen darauf, die **vorhandenen** Daten anders zu sehen, zu repräsentieren, zu strukturieren und zu bearbeiten. In Umwelt- und Meßwert-DB besteht aber ein ganz anderes Problem: Hier sollen (auf Anfrage) Daten geliefert werden, die gar nicht explizit in der Datenbank gespeichert sind, sondern die aus den vorhandenen Daten abgeleitet werden müssen. Ein erster Ansatz, alle Werte, die gebraucht werden könnten, auch explizit zu speichern, muß - abgesehen vom inakzeptablen Speicherplatzbedarf - schon deshalb scheitern, weil kontinuierliche Vorgänge (siehe Kap. 3.3.2) abgelegt werden sollen, bei denen sich immer noch ein Zwischenwert finden läßt. Andererseits sind Algorithmen bekannt, wie sich diese beliebigen Zwischenwerte aus den umliegenden berechnen lassen. Dieses Verfahren ist ähnlich dem Vorgehen bei *Expertensystemen* (XPS), genauer deren Deduktionskomponente, die mit Hilfe von Regeln (nicht zu verwechseln mit den Integritätsregeln in den vorhergehenden Abschnitten!) neue Fakten aus den vorhandenen ableiten. Solche Deduktionskomponenten sind in

keinem der betrachteten EDBS vorhanden oder geplant, lediglich in DASDBS war die Darstellung von Wissen, mit Hilfe von Ableitungsregeln zumindest vorgesehen. Andererseits ist die Verbindung von DBS und XPS, also DBS mit Wissensdarstellung und integrierter Deduktionskomponente bisher selten, und wenn es sie gibt, ist das DBS nicht im beschriebenen Sinn erweiterbar (siehe Kap. 4.3.3).

Wenn im System ein ADT-Mechanismus vorhandenen ist (EXODUS, POSTGRES) oder der CREATE RELATION-Befehl entsprechend erweitert wurde (DBS Illustra [Illu94]), lassen sich verschiedene Arten von Relationen, “Relationentypen”, auszeichnen. Das ist im Prinzip eine Wertebereichsbeschränkung auf dem Datentyp ‘Relationenschema’. Es bleiben alle Eigenschaften einer Relation erhalten, es kommen noch neue hinzu. (Man kann allerdings bei den sonst immer erlaubten Operationen aus diesem eingeschränkten Wertebereich herauskommen.) Die überall angebotenen Definitionen von Objekten bzw. Objektklassen eignen sich, um komplexere Zusammenhänge zwischen Versuchsreihen gleich im DB-Schema auszudrücken. Besonders die benutzerdefinierten Funktionen können verwendet werden, um Daten für die Weiterverarbeitung auszuwählen und aufzubereiten. Hier sind insbesondere aggregierende Funktionen, Table Functions und Funktionen auf benutzerdefinierten Datentypen wichtig (siehe Kap. 5).

Diese Mittel erleichtern das Einbinden zusätzlicher Funktionen wie die oben genannte Inter- und Extrapolation, die in Kapitel 3.4 erwähnte räumliche Suche oder Verfahren für die Suche auf chemischen Strukturen. Es ist i.a. nicht notwendig, daß ein System alle Anforderungen erfüllt. Hier ist die Kernel-Architektur von DASDBS [SPSW90, Wolf90] mit verschiedenen Ergänzungen für unterschiedliche Anwendungen eine gute Alternative. Auch die Modularität der meisten EDBS kommt uns hier entgegen; so könnte man mit vertretbarem Aufwand eine Art “Schablonen” für Klassen von Anwendungen wie z.B. chemische Stoffdatenbanken, Datenbanken für wenige Meßreihen oder für einfache (Schad-) Stoffverfolgung herstellen.

4.6 Aufwand zur Erweiterung des DBS und der Anfragesprache

Neben der Feststellung, ob die für eine bestimmte Umweltanwendung benötigten Erweiterungen überhaupt mit den angebotenen Mitteln erreicht werden können, spielen vor dem tatsächlichen Einsatz noch andere Betrachtungen eine wichtige Rolle:

- Wie hoch ist der Aufwand im Einzelfall? Wie aufwendig ist die Definition, Implementierung und Einbettung einer benutzerdefinierten Funktion einer bestimmten Art? Kann dies von einem DB-geschulten Anwender geleistet werden oder ist ein Spezialist zur DB-Implementierung erforderlich?

- Werden die Erweiterungen das Antwortzeitverhalten und die Performance des EDBS entscheidend beeinflussen? Wieviel Aufwand ist u.U. erforderlich, um dies zu vermeiden?

- Wieviel zusätzlicher Aufwand wird durch die Erweiterungen bei der bei DBS ohnehin notwendigen regelmäßigen Wartung erforderlich?

- Wie stark spezialisieren die Erweiterungen das DBS? Wird es weiterhin möglich sein, mit anderen DBS (des gleichen Herstellers) problemlos zu kommunizieren?

Zur Beurteilung des Aufwands, um neue Funktionen in ein EDBS einzubetten, muß man unterscheiden zwischen dem Toolkit-Ansatz von EXODUS, GENESIS oder DASDBS und dem Ansatz eines (relationalen) DBS mit Erweiterungsmöglichkeiten wie POSTGRES und Starburst.

Um z.B. das System POSTGRES um neue Funktionen zu erweitern, ist es notwendig

1) diese Funktionen als gewöhnliche C-Moduln zu implementieren, also als Funktion, die Eingabeparameter erhält und einen Wert zurückliefert,

2) diese Funktionen mit einer Definitionsanweisung dem DBS bekanntzugeben.

Diese beiden Punkte können sicher von einem Programmierer, der nicht notwendigerweise DB-Spezialist ist, oder einem programmiererfahrenen Anwender geleistet werden.

Komplizierter wird es, wenn die sog. Operationen implementiert werden und Hinweise für den Optimierer gegeben werden müssen. Dies ist sicher von einem Anwendungsprogrammierer nicht ohne weiteres zu leisten, aber bei einigen Anwendungen bezüglich der Performance unbedingt erforderlich.

Für die Implementierung eines neuen Datentyps reicht es nicht, eine Struktur oder ein Feld zu definieren wie in einer höheren Programmiersprache. In POSTGRES ist es notwendig, die Länge der internen und externen Darstellung anzugeben, evtl. variabel, und zwei C-Funktionen zu implementieren, die die Exemplare vom neuen Datentyp von der externen zur internen Repräsentation konvertieren und umgekehrt. Soll der Datentyp auf verschiedenen Maschinen verwendet werden, müssen bei Bedarf noch Funktionen zur Konvertierung der verschiedenen Maschinendarstellungen angegeben werden [POST90]. Sollen in der Erweiterung des kommerziellen DBS INGRES, der INGRES Object Management Extension, neue Datentypen implementiert werden, ist der Aufwand noch größer. Hier werden u.a. Funktionen benötigt zur Konvertierung von der internen zur externen Darstellung und umgekehrt, zum Vergleich zweier beliebiger Werte dieses Datentyps auf größer/kleiner/gleich, zum Erzeugen eines Default-Werts, zur Berechnung der aktuellen Länge eines Werts, zur Unterstützung von Speicherstrukturen (Hash-Funktionen) und zur Erstellung von Statistiken [INGR89c]. Hier ist sicher ein geschulter Datenbank-Programmierer erforderlich.

Um in Starburst neue Speicher- und Zugriffsmethoden in das DBS einzubinden, ist ein tieferes Verständnis vom Prinzip des Zugriffssystems eines DBS erforderlich. Es muß nicht nur die Speichermethode selbst implementiert werden, sondern auch die Operationen ‘Direktzugriff’

über Schlüssel oder TID (**T**upel-**Id**entifikator), INSERT, UPDATE, DELETE und Relationen-Scan. (Einfache Funktionen lassen sich aber hier genauso einfach implementieren wie bei POSTGRES.)

Die Systeme EXODUS und GENESIS bieten nur ein sog. Toolkit an, das heißt eine Auswahl von Komponenten, die nach den Anforderungen des Anwenders zusammengesetzt und ergänzt werden können, was später sehr aufwendig ist und u.U. zu ganz unterschiedlichen DBS führt, die nicht mehr ohne weiteres miteinander kommunizieren und Daten austauschen können. POSTGRES und Starburst hingegen können für eher konventionelle Anwendungen sofort ohne Erweiterungen benutzt werden. Hier ist die Kompatibilität zwischen verschiedenen DB, die mit Hilfe eines dieser Systeme aufgebaut wurden, auf jeden Fall gewährleistet (der Argumentation von [LiHa90, p. 219] folgend).

Als Beispiel für die heute kommerziell verfügbaren ORDBS soll hier kurz beschrieben werden, wie der INFORMIX-Universal Server mit den sog. DataBlade-Modulen erweitert werden kann. Hier wurde berücksichtigt, daß für den Benutzer die Erweiterungen mit der Definition der neuen Datentypen und der Operationen darauf noch lange nicht beendet sind, und ein entsprechendes Werkzeug, das “DataBlade Developers Kit” sowie entsprechende Schulungen [Info96] werden angeboten. Wenn der Benutzer nicht nur abgeleitete Datentypen spezifiziert, sondern auch ganz neue, sog. ´opaque types´, muß er auch Funktionen definieren und schreiben, mit denen der Server auf diesen Datentypen arbeiten kann (z.B. Vergleichsoperatoren). Für diese Aufgabe stehen verschiedenen Programmiersprachen (C, C++, Java) sowie die Sprache der stored procedures zur Verfügung. Informix erlaubt auch das Erstellen neuer Zugriffsmethoden wie z.B. R-trees für räumliche Daten. Das Bereitstellen passender Zugriffsmethoden für neue Datentypen wird dringend empfohlen, um ein gutes Antwortzeitverhalten zu behalten. Jedes DataBlade-Modul darf sich eigene Systemrelationen anlegen, z.B. für verschiedene Formate und Konvertierungsverfahren etwa auf Bilddaten. Sollen mehrere DataBlade-Module zusammen arbeiten, müssen Schnittstellen spezifiziert werden. Mit dem “DataBlade Developers Kit” kann nun ein neues DataBlade-Modul dem Server zur Verfügung gestellt werden. Mit diesem Toolkit können u.a. C-Quellen, Header, Makefiles, SQL scripts, sowie ausführbarer Code erstellt werden und die Einbindung des Codes in den Server durchgeführt werden.

Zusammenfassend läßt sich sagen, daß für große Systeme wie [BKLM90] die Implementierung eines angepaßten DBS mit Hilfe eines EDBS sicher eine gute Lösung ist, bei kleinen unabhängigen Systemen ist wahrscheinlich der Erstellungsaufwand recht hoch, insbesondere bei solchen EDBS, die von vorn herein einen DBI benötigen. Für kleinere Systeme ist sicherlich eine abgemagerte Version - auch mit an einigen Stellen eingeschränkter Funktionalität -, die aber einfacher zu bedienen ist, besser geeignet. Am ehesten wären noch Systeme wie POSTGRES [StRo86, StRH90] oder Starburst [LiHa90] geeignet, die auch ohne oder mit wenigen Erweiterungen voll funktionsfähig sind. Es ist aber immer fraglich, ob sich ein aufwendiges System für eine kleine Anwendung lohnt. Der Aufwand der Anpassung und Erweiterung lohnt nur für große Projekte [BKLM90] oder z.B. Systeme in Behörden, die an vielen Stellen eingesetzt werden.

Es scheint so, daß der Anwender sich eine gute, schnelle Datenverwaltung, -bereitstellung und -wartung mit dem Aufgeben seiner Selbständigkeit und der alleinigen Verfügbarkeit über seine Daten und sein System erkaufen muß. Ein Spezialist zur Erweiterung, Anpassung und Wartung des eingesetzten DBS wird gebraucht. Vergleichbar ist dies mit der Tatsache, daß die Workstation auf dem eigenen Schreibtisch auch ohne das In-House-Netz nicht mehr (oder nur sehr eingeschränkt) funktionsfähig ist, man dadurch aber hohe und vielfältige Funktionalität und viel Plattenplatz gewinnt.

EDBS stehen zur Projektlaufzeit mit vollem Funktionsumfang nur als Forschungsprototypen zur Verfügung, was bedeutete, daß sie auf heterogener Hardware i.a. nicht verfügbar waren, bzw. selbst angepaßt werden mußten. Auch Aspekte wie verteilte DB und Archivierung standen bei ihnen nicht im Vordergrund. Kommerziell erhältliche DBS dagegen sind in diesen Aspekten führend, dafür fehlten hier noch viele Erweiterungsmöglichkeiten.

Für das Projekt, innerhalb dessen diese Arbeit entstanden ist, bedeutete das, daß ein kommerzielles System gewählt werden mußte und die gewünschten Erweiterungen als Präprozessor implementiert werden mußten (Kapitel 6-8). Ein Vorschlag, wie die Erweiterungen auf einem EDBS implementiert werden könnten, erfolgt in Kapitel 9.

5. Interpolation von Meßwerten

In diesem Kapitel wird allgemein die Integration von Interpolationsverfahren in ein DBS und in die Anfragesprache motiviert und vorgestellt. Die verschiedenen Optionen, die dabei vorhanden waren, werden erläutert und die Entscheidungen begründet. Dabei wird vorausgesetzt, daß die Interpolationsverfahren bekannt (und richtig) sind und daß sie vom Anwender oder einem mit der Problematik vertrauten Programmierer implementiert sind bzw. werden können. Die konkrete Syntax zur Integration wird dann im Kap. 6 beschrieben. Die Unterstützung durch das Datenbanksystem kann immer nur darin bestehen, Interpolationsanforderungen des Benutzers anzunehmen, die Selektion der für die Interpolation jeweils benötigten Daten vorzunehmen und die Interpolationsmoduln selbst in das DBS zu integrieren.

In diesem Kapitel wird damit der ‘Rahmen’ beschrieben, der sich aus den konkreten Wünschen der Anwender und Benutzer im PWAB-Projekt ergab und der schon ziemlich zu Beginn des Projekts festgelegt war.

Wie in Kap. 3.5 beschrieben, kann Interpolation integriert in ein DBS, genauer in die Anfragesprache und in die Anfrageauswertung für den Benutzer sehr nützlich sein. Dies ist insbesondere der Fall bei spontanen, interaktiven Anfragen, wenn Meßpunkte und -intervalle nicht bekannt sind, beim Vergleich verschiedener Meßreihen oder -parameter, bei der Bereitstellung von Daten für Modellrechnungen und bei der Ergänzung fehlender Werte. Von solchen Anwendungen soll bei den folgenden Entscheidungen ausgegangen werden.

5.1 Die Art der Einbettung der Interpolationsfunktionen

Vom Standpunkt des Benutzers aus betrachtet bestehen bei der Einbettung von Interpolation in das DBS zwei Alternativen: Entweder wird immer dann interpoliert, wenn ein Wert abgefragt wird, der nicht in der Datenbank vorhanden ist, der aber interpoliert werden könnte. Oder es wird nur dann interpoliert, wenn dies explizit vom Benutzer gefordert ist. Die erste Alternative erforderte zwar keinerlei zusätzlichen Aufwand vom Benutzer, hätte aber zur Folge,

- daß der Benutzer nicht weiß, wann er einen interpolierten Wert erhält und wann einen Original-Meßwert,
- daß er nicht zwischen verschiedenen Interpolationsverfahren wählen kann

- und daß er keine Abstände für äquidistante Folgen angeben kann.

Um diese Nachteile zu vermeiden, wird hier ein Ansatz gewählt, bei dem der Benutzer immer explizit sagen muß, daß er interpolierte Werte wünscht, welche Abstände mehrere Werte voneinander haben sollen und nach welchem Verfahren interpoliert werden soll. Damit sollte sich der Benutzer auch der Mängel, die interpolierten Werten immer anhaften, bewußt sein. Obwohl Meßwerte sich in der Regtel nie mehr ändern, muß der Benutzer z.B. damit rechnen, daß dieselbe Anfrage mit Interpolation zu zwei verschiedenen Zeitpunkten unterschiedliche (aber trotzdem jeweils korrekte) Ergebnisse liefert, weil etwa in der Zwischenzeit neue Meßwerte eingefügt wurden, die das Interpolationsergebnis verbessern. Die Interpolation muß also so in das DBS eingebettet werden, daß sie eine weitere Option für den Benutzer darstellt, d.h. Interpolation wird nur durchgeführt, wenn der Benutzer dies explizit angibt. Der Benutzer muß außerdem gewisse Parameter angeben, z.B. den Namen des Interpolationsverfahrens.

Wenn ein DBS die Einbettung von Interpolationsfunktionen anbietet, sollten diese ähnlich wie die üblichen Built In-Funktionen (Aggregations- / Konvertierungsfunktionen) aufgerufen werden und arbeiten. Im Unterschied zu Built In-Funktionen sind Interpolationsfunktionen aber immer nur in einzelnen Datenbanken verfügbar und an einzelne Relationen mit bestimmten Relationenschemata gebunden (s.u.). Interpolationsfunktionen müssen also auf der Ebene der Relationen dem DBS deklariert werden. Damit ähneln sie abgeleiteten Attributen (siehe Kap. 4.4.2). Im Gegensatz zu abgeleiteten Attributen werden aber nicht nur einzelne Werte sondern ganze Tupel neu berechnet.

Interpolation ist nur bei Meßwerten oder numerischen Werten mit ähnlicher Semantik sinnvoll, deshalb werden zusätzliche Systeminformationen benötigt, die besagen, welche Relationen solche Werte enthalten, und so diese Meßrelationen gegenüber “normalen” auszeichnen. Wie bereits in Kap. 3.1 erwähnt, gehört zu jedem Meßwert, damit er sinnvoll interpretiert werden kann, die Meßbasis und der Kontext. Das bedeutet für Meßrelationen, daß neben dem eigentlichen Meßwert noch mindestens ein Attribut, das die Basis oder einen Verweis auf die Basis enthält, existieren muß. Der Kontext kann durch die Bezeichnung der Relation und der Attribute oder durch weitere Attribute oder beides beschrieben werden.

Damit enthält eine Meßrelation, auf der Interpolation ausgeführt werden kann, mindestens ein Meßwertattribut, im folgenden als *kontinuierliches Attribut* bezeichnet, weil Interpolation nur bei kontinuierlichen Vorgängen sinnvoll ist, und mindestens ein sog. *Basisattribut*. Außerdem kann eine Meßrelation noch beliebig viele *zusätzliche Attribute* enthalten. Eine Meßrelation kann mehrere kontinuierliche Attribute enthalten, wenn diese gegen dieselbe Basis aufgezeichnet wurden. Die Basis besteht entweder aus so vielen Attributen wie Dimensionen oder, bei einem Verweis auf die Basis, aus den Schlüsselattributen der referenzierten Relation.

Wenn interpoliert wird, müssen nicht immer alle Attribute der Basis, genauer Dimensionen, einbezogen werden. Es kann z.B. bei Wasserstandsmessungen in Meßbrunnen wahlweise über den (Stand-) Ort, die Zeit oder beides interpoliert werden. Die Attribute, über die interpoliert

wird, werden im folgenden als *variierende* Basisattribute bezeichnet, die übrigen als *feste* Basisattribute.

Prinzipiell kann jede Relation, die mindestens zwei Attribute mit kontinuierlichem Wertebereich besitzt, Meßwert und Basis, als Meßrelation behandelt werden. Interpolation kann auf dem als Meßwert betrachteten Attribut ausgeführt werden.

Als Ergebnis einer Interpolation entsteht - wie bei jeder SQL-Anfrage auch - eine Relation. Die Ergebnisrelation ist auf das interpolierte kontinuierliche Attribut und die Basisattribute der Ausgangsrelation projiziert worden. Bei den variierenden Basisattributen sind dann keine Verweise, sondern die wirkliche Basis in der Relation enthalten. Zusätzliche Attribute der Ausgangsrelation, wie z.B. die Güte eines Meßwerts oder die Bauart eines Meßbrunnens, die genau den Meßtupeln zugeordnet gewesen sind, sind dann i.a. nicht mehr sinnvoll. Nur in Ausnahmefällen, wenn z.B. die Werte eines zusätzlichen Attributs aller für die Interpolation benötigten Tupel gleich sind, wäre die Erhaltung dieses zusätzlichen Attributs sinnvoll. Da dies nur sehr aufwendig oder, in anderen Fällen, gar nicht überprüfbar ist, sollen zusätzliche Attribute bei der Interpolation grundsätzlich nicht in die Ergebnisrelation aufgenommen werden.

Wenn in einer Meßrelation mehrere kontinuierliche Attribute geführt werden, wird durch eine Interpolationsfunktion immer nur eins bzw. eine zusammengehörige Gruppe (z.B. Koordinaten) zur Zeit bearbeitet. Die übrigen kontinuierlichen Attribute werden dann wie zusätzliche Attribute behandelt. Hier muß so vorgegangen werden, weil i.a. jedes kontinuierliche Attribut (z.B. Raum oder Zeit) ein anderes Interpolationsverfahren benötigt, das z.B. auch die für die Interpolation benötigten Umgebungstupel nach anderen Kriterien auswählt. Zusätzlich könnten innerhalb eines Tupels Werte für einige kontinuierliche Attribute fehlen. Für diese müssen dann weitere Umgebungstupel gesucht werden.

Zu jedem kontinuierlichen Attribut müssen mehrere Interpolationsfunktionen definiert werden können. Zum einen kann bei einer mehrdimensionalen Meßbasis über das eine, das andere oder alle Basisattribute interpoliert werden, zum anderen können auch verschiedene Interpolationsverfahren existieren, die wahlweise z.B. schneller berechnet werden oder genauer sind. So muß also, wenn der Benutzer Interpolation wünscht, angegeben werden, welche Interpolationsverfahren auf welchen Attributen zu welchen Werten der variierenden Basisattribute berechnet werden sollen. (Das Attribut muß zumindest bei Mehrdeutigkeiten wie dem Verbund einer Relation mit sich selbst angegeben werden.)

Es gibt Probleme, die sind abhängig von individuellen Wünschen der Benutzer oder von der Realisierung des Interpolationsverfahrens. Solche Probleme sind:

- Was geschieht bei sog. “Treffern”, wenn der Benutzer für die variierenden Basisattribute (zufällig oder bewußt) Werte angegeben hat, die in der Datenbasis zur Zeit vorhanden sind?
- Wann ist eine Interpolation nicht mehr sinnvoll, weil die umliegenden Meßwerte zeitlich oder räumlich zu weit entfernt sind ? Wann unterbleibt die Interpolation aus diesen Gründen ?

Diese Fragen werden in die Interpolationsfunktionen selbst verlagert. Die Interpolationsfunktionen werden vom System in jedem Fall aufgerufen und ‘entscheiden’ dann, ob sie einen Wert zurückliefern und, wenn ja, wie dieser Wert gefunden wird.

5.2 Die Schnittstelle zum Benutzer

In vielen Fällen wird der Benutzer eine Interpolation mit bestimmten Voraussetzungen wie Einschränkungen, Schrittweiten nur einmal oder nur wenige Male aufrufen, wie etwa bei spontanen interaktiven Anfragen, bei der Aufbereitung der Daten für Modellrechnungen oder Statistiken für Projekt-Präsentationen. Die wichtigste Schnittstelle für den Endbenutzer, der die Interpolation nutzen will, ist also die interaktive. Bei den genannten Anlässen werden in den meisten Fällen nur ein ganz bestimmter Ausschnitt (Zeitraum, örtliche Begrenzung) der Daten interpoliert. Es ist also wünschenswert, daß Einschränkungen, genauer Selektionen, wie sie in Anfragen relationaler Sprachen definiert werden können, im Zusammenhang mit Interpolationen angegeben werden können.

Prinzipiell gibt es zwei Möglichkeiten, dem Endbenutzer einen interaktiven Zugriff mit Interpolation auf Meßdaten zu ermöglichen: Entweder muß eine neue Schnittstelle dafür entworfen und implementiert werden oder die Interpolation muß in eine vorhandene Datenbankanfragesprache eingebettet werden. Eine eigene Interpolationsschnittstelle könnte - in einer einfachen Version - schlichter gestaltet werden und wäre damit auch einfacher zu implementieren. Sie hätte aber einige schwerwiegende Nachteile: Der Benutzer müßte eine weitere Anfragesprache lernen und diese wäre auch weniger mächtig als die bekannten Anfragesprachen bezüglich Einschränkungen und Verknüpfungsmöglichkeiten mit anderen Daten. Je mächtiger und ähnlicher den relationalen Anfragesprachen so eine Schnittstelle gestaltet wird, desto höher wird auch der Implementierungsaufwand.

Das Einbetten der Interpolation in eine vorhandene (relationale) Anfragesprache hat zwar auch einige Nachteile, die Vorteile überwiegen aber deutlich. Nachteilig ist hier, daß das Zusammenwirken mit der vorhandenen Sprache genau definiert werden muß und daß evtl. einige Einschränkungen gemacht werden müssen. Es soll dem Benutzer jedoch möglichst wenig Aufwand bei der Einarbeitung in die Interpolationsschnittstelle entstehen. Jeder der einmal z.B. mit SQL gearbeitet hat, wird auch auf die vielfältigen Möglichkeiten, Ergebnismengen zu definieren, nicht mehr verzichten wollen. Es bietet sich deshalb an, eine bereits existierende, möglichst weit verbreitete Anfragesprache zu erweitern. Als Grundlage für die Interpolationsschnittstelle wurde Interaktives SQL gewählt (ISQL [ANSI86], siehe u.a. [INGR89]), und zwar aus den folgenden Gründen:

- SQL ist eine standardisierte Anfragesprache [ANSI86, ANSI89, ISO90] und auf den meisten relationalen DBS verfügbar. Damit wird die Abhängigkeit von einem bestimmten DBS verringert.

- Der Benutzer kommt weiterhin mit einer einzigen interaktiven Schnittstelle zum DBS und einer einzigen Anfragesprache aus.
- Zu SQL gibt es immer auch eine in höhere Programmiersprachen eingebettete Version (Embedded SQL [INGR89b]) und in vielen Fällen auch eine 4GL und andere Hilfsmittel wie Reportgeneratoren, Maskensysteme u.ä. Diese können dann auch analog um Interpolation erweitert werden.
- Da die Erweiterung einer bestehenden Anfragesprache weniger aufwendig ist, kann ein Prototyp schneller zur Verfügung gestellt werden als bei der Entwicklung einer ganz neuen Benutzerschnittstelle. Für SQL ist als freie Software eine fertige Grammatik als Eingabe für die Parser-/ Compiler-Generatoren LEX / YACC erhältlich, die einfach um weitere Typen von Anweisungen erweitert werden kann.

Die Anfragesprache, die aus SQL und der Erweiterung um Interpolation auf Meßdatenrelationen mit kontinuierlichen Attributen entsteht, wird im folgenden *kontinuierliches SQL* oder *CSQL* (Continuous SQL) genannt. CSQL wird in Kapitel 6 behandelt.

5.3 Die Schnittstelle zum DBS

Es stehen prinzipiell zwei Alternativen zur Verfügung, die Funktionalität eines DBS bzw. seiner interaktiven Benutzerschnittstelle zu erweitern. Entweder werden die Erweiterungen direkt in das DBS eingebunden, oder es wird ein Präprozessor erstellt, der die Erweiterungen erkennt und bearbeitet. Die direkte Einbettung der Interpolationsfunktionen in das DBS hätte folgende Vorteile:

- Jede Anfrage wird nur einmal bearbeitet. Jeder Präprozessor erfordert einen weiteren Verarbeitungslauf mit zusätzlichen Datenbankzugriffen, der das Antwortzeitverhalten verschlechtert.

- Die Anfragen können besser optimiert werden, abgestimmt auf die jeweilige Interpolationsfunktion. Bei einem Präprozessor muß zweimal unabhängig voneinander optimiert werden, einmal im Präprozessor die Interpolation und einmal die von Interpolation bereinigten Teile der Anfrage im DBS an sich.

Da jedoch aus den in Kapitel 4.6 beschriebenen Gründen ein kommerzielles DBS gewählt werden mußte, und zu solchen Systemen kein Quellcode erhältlich ist, mußte die Präprozessor-Alternative gewählt werden. Dies hat auch einige Vorteile:

- Der Präprozessor muß mit Hilfe von Embedded SQL und Dynamic SQL implementiert werden. Diese SQL-Varianten werden auch genormt, so daß der fertige Präprozessor einfach auf andere relationale DBS portiert werden kann.

- Das Ergänzen neuer Interpolationsfunktionen ist einfacher: Nur der Präprozessor, nicht das ganze DBS, muß beim Ergänzen neuer Interpolationsfunktionen heruntergefahren und neu geladen werden. Fehlerhafte Interpolationsfunktionen können nicht das gesamte DBS blockieren oder gar Datenbestände zerstören. Dies ist insbesondere dann ein Vorteil, wenn die Interpolationsfunktionen von DB-unerfahrenen Anwendern programmiert werden.

Interpolationsverfahren sind immer speziell auf eine Meß-Methode, auf eine Klasse von Meßwerten abgestimmt. Deshalb sollen sie im DBS auch genau einer Relation oder Klasse von Relationen zugeordnet werden. ‘Generische Interpolationsverfahren’, die sich auf beliebige (Meß-) Werte anwenden lassen, werfen eine Reihe von Problemen auf. Die Implementierung wäre aufwendiger, denn schon die unterschiedliche Anzahl von Basisdimensionen (z.B. im Raum oder in der Fläche gewonnene Meßwerte) ist ein Problem. Näheres zu diesem Aspekt findet sich auch in Kap. 9. Es besteht zwar immer die Möglichkeit, Interpolationsfunktionen ‘generisch’, also praktisch für alle Relationen zu implementieren, so wie eine Funktion auf alle Werte eines Datentyps anwendbar ist. Dann würde jedoch nicht überprüft, ob als zu interpolierende Relation wirklich eine Meßrelation angegeben wurde.

Bei einer direkten Einbettung der Interpolationsfunktionen in das DBS besteht die Möglichkeit des festen Bindens an einzelne Relationen in der Regel nicht, bzw. ist sehr aufwendig. Denn die Integration der Interpolation müßte hier vor der Definition der Meßrelationen geschehen. Eine andere Möglichkeit bietet hier das neuartige relationale DBS Illustra [Illu94], das eine hierarchische Typisierung von Relationen erlaubt. Hier könnte eine Interpolationsfunktion an einen bestimmten Relationentyp bzw. -untertyp gebunden werden.

Aber auch genau an einzelne Relationen angepaßte Interpolationsfunktionen haben einige Nachteile: Der Programmierer muß das Schema und die Struktur der Datenbank, wie Relationen- und Attributnamen, Beziehungen zwischen Relationen, und Sekundärschlüssel sehr genau kennen. Die Interpolationsfunktionen sind nicht transparent gegenüber - auch kleineren - Änderungen des DB-Schemas d.h., sie müssen an jede Änderung des Datenbankschemas angepaßt werden. Die Interpolationsfunktionen sind nicht einfach auf andere Relationen in derselben oder einer anderen DB übertragbar, die Werte mit derselben Semantik enthalten.

Teilweise können diese Nachteile, insbesondere die fehlende Übertragbarkeit, zumindest technisch dadurch behoben werden, daß bei der Programmierung Dynamisches SQL [INGR89b] verwendet wird und die benötigten Namen beim Funktionsaufruf übergeben werden. Dann müssen bei geringfügigen Änderungen der Relationenschemata und bei der Übertragung eines Verfahrens auf eine semantisch ähnliche Relation nur einige Beschreibungen und Definitionen in der DB, nicht aber die Implementierung der Funktion selbst geändert werden. Es können so auch Daten über Sichten interpoliert werden. In denen können Sekundärschlüsselbeziehungen schon denormalisiert sein, gewisse geeignete Zugriffspfade schon eingerichtet sein und die zu durchsuchende Tupelmenge schon eingeschränkt sein (siehe Kap. 9).

Fest an einzelne Relationen gebundene Interpolationsfunktionen haben auch den Vorteil gegenüber ‘generischen’, daß für den Benutzer nicht die Gefahr besteht, ein ungeeignetes Verfahren auszuwählen.

Unter den gegebenen Randbedingungen und unter Abwägung der Vor- und Nachteile der verschiedenen Optionen erschien es, zumindest für die erste Version, am günstigsten, einen Präprozessor zu implementieren, in den die Interpolationsfunktionen, fest an einzelne Relationen gebunden, eingebunden werden. In diesem Ansatz werden die Interpolationsfunktionen in einer höheren Programmiersprache implementiert und greifen über Embedded und Dynamic SQL direkt auf die DB zu. Dabei müssen typische, während einer Interpolation benötigte Suchverfahren jedesmal bei der Interpolation selbst mit implementiert werden.

Abhilfe schaffen könnten hier vordefinierte SQL-Anfragen für die üblichen Suchoperationen, in die innerhalb der Interpolationsfunktion die passenden Relationen- und Attributnamen eingesetzt werden müssen (siehe Kap. 7).

6. Die SQL-Erweiterung CSQL

Die Interpolation auf Meßwerten in der Datenbank wird dem Benutzer über eine SQL-Erweiterung, CSQL, zur Verfügung gestellt (siehe Kap. 5.2). Diese SQL-Erweiterung umfaßt sowohl Erweiterungen des Definitionsteils der Datensprache als auch des Anfrageteils. In diesem Kapitel werden zunächst die Meßwertrelationen mit den entsprechenden Attributen und die Interpolationsoperation formal eingeführt. Dann wird darauf die Syntax definiert und an kurzen Beispielen verdeutlicht. Anschließend werden die dafür im DBMS notwendigen Voraussetzungen beschrieben.

CSQL erwartet einen mit SQL zumindest oberflächlich vertrauten Anwender. Er sollte in der Lage sein, Relationen zu erzeugen und einfache Anfragen mit Einschränkungen und Verbundoperationen zu stellen.

6.1 Allgemeine Anforderungen an die Spracherweiterung CSQL

Die Spracherweiterungen müssen sowohl den Definitionsteil der Sprache (DDL) als auch den Anfrageteil (DML) betreffen. In beiden Teilen sollen die besonderen Eigenschaften der Meßwerte berücksichtigt werden. Da die besonderen Eigenschaften der Meßdaten während ihrer gesamten Lebenszeit in der Datenbank bestehen, erscheint es sinnvoll, sie auf konzeptioneller oder logischer Ebene während des Datenbankentwurfs festzuschreiben. Deshalb muß das Datenmodell zwei Arten von Relationen unterscheiden können: konventionelle Relationen und Meßwertrelationen. Denn auch eine Meßwertdatenbank muß neben den Meßwerten zusätzlich konventionelle Daten enthalten. Im konventionellen Teil der PWAB-Datenbank (Kap. 2) können z.B. Ernteerträge (kg Mais pro m^2) liegen, mit denen Berechnungen zur Wirtschaftlichkeit der verschiedenen Anbaumethoden durchgeführt werden. Im Definitionsteil der Anfragesprache müssen deshalb Meßwertrelationen und konventionelle Reltionen unterschieden werden können. Das soll dadurch geschehen, daß Meßwertrelationen mit ihren kontinuierlichen und Basisattributen gesondert definiert werden.

Weiterhin muß dem System bekanntgegeben werden, welche Interpolationsverfahren es gibt und auf welches kontinuierliche Attribut welcher Relation sie angewendet werden können. Dabei soll es möglich sein, mehrere Interpolationsverfahren für dasselbe kontinuierliche Attribut

zu definieren, denn es können verschiedene gleichwertige Methoden existieren, die von den Benutzern unterschiedlich bevorzugt werden. Oder es können für Interpolationen über verschiedene variierende Basisattribute unterschiedliche Interpolationsverfahren existieren.

Die Erweiterungen der DML müssen so gestaltet werden, daß die volle Mächtigkeit von SQL in allen Anfragen ohne Interpolation erhalten bleibt. In Anfragen mit Interpolation sollte die volle Mächtigkeit von SQL erhalten bleiben, soweit dies sinnvoll erscheint. Das heißt, auch in Anfragen mit Interpolation können Projektionen, Restriktionen, Verbunde, Vereinigung, Gruppierung und Sortierung durchgeführt werden. Innerhalb einer Anfrage kann Interpolation für ein oder mehrere kontinuierliche Attribute durchgeführt werden. Verbunde zwischen zwei Basisattributen oder einem Basisattribut (interpoliert) und einem ‘normalen’ Attribut sind möglich. Außerdem soll Interpolation auch innerhalb von geschachtelten Anfragen möglich sein.

Bezüglich der Interpolation sollte der CSQL-Anfrageteil folgendes leisten:

- Die Werte eines kontinuierlichen Attributs können für jede mögliche Kombination der Basisattributwerte innerhalb des gültigen Wertebereichs abgefragt werden.
- Äquidistante Folgen von Meßwerten in einer oder mehreren Dimensionen, höchstens gleich der Anzahl der Basisattribute, können abgefragt werden.
- Eine von mehreren Interpolationsmethoden kann spezifiziert werden.

Bisher ist es nicht vorgesehen, eine Schachtelung von Interpolationen zuzulassen. Soll z.B. bei einer Meßgröße über den Ort und über die Zeit interpoliert werden, und es existiert nur je eine Funktion zu Interpolation über Ort oder Zeit, ist es nicht möglich, diese beiden Verfahren in einer Anfrage geschachtelt nacheinander aufzurufen. Es muß eine neue Funktion implementiert werden, die die beiden vorhandenen integriert. Der Grund hierfür ist, daß jede Interpolationsfunktion für die Interpolation auf einer Relation mit einem ganz bestimmten Schema definiert ist. Das Ergebnis, das sie liefert, existiert nur als temporäre Relation in der DB, bis das Ergebnis ganz abgefragt ist. Außerdem hat die Ergebnisrelation, wie bei vielen ‘normalen’ SQL-Anfragen auch, ein etwas anderes Schema, als die Original-Meßwertrelation. Die zweite Interpolation müßte dann auf diesem Zwischenergebnis, der temporären Relation mit anderem Schema, durchgeführt werden. Dafür ist diese Funktion aber nicht definiert.

Diese eingeschränkte Lösung, die Schachtelungen verbietet, kann auch vom Antwortzeitverhalten günstiger realisiert werden, weil innerhalb der Interpolationsfunktion optimiert werden kann (siehe Kap.9.3).

6.2 Syntax und Semantik von CSQL

In diesem Abschnitt werden zunächst formal die Erweiterungen beschrieben, die notwendig sind, damit Interpolationen vom DBS unterstützt durchgeführt werden können. Anschließend werden die formalen Beschreibungen syntaktisch in SQL-Erweiterungen umgesetzt.

6.2.1 Differenzierte Relationentypen

Da die besonderen Eigenschaften der Meßdaten während ihrer gesamten Lebenszeit in der Datenbank bestehen, erscheint es sinnvoll, sie auf konzeptioneller oder logischer Ebene während des Datenbankentwurfs festzuschreiben. Deshalb muß das Datenmodell, ein erweitertes relationales Modell, zwei Arten von Relationen unterscheiden können:

1) Konventionelle Relationen R_i werden formal als Mengen von (geordneten) Wertekombinationen definiert, den Tupeln. Die Domänenfunktion D liefert den Wertebereich eines Attributs A:

   $R_i (A_{i1}, ..., A_{in}) \subseteq D (A_{i1}) \times ... \times D (A_{in}), i \in \mathbf{N}$

   wobei die A_{ij} die Attribute der Relation R_i sind. Jede Relation muß mindestens ein Attribut enthalten. Die Schlüsselattribute von R_i sind frei wählbar.

2) Meßwertrelationen sind eine Teilmenge aller Relationen, die einige zusätzliche Eigenschaften haben, welche nicht alle Relationen haben, und auf denen weitere Operationen definiert sind, die auf anderen Relationen nicht angewendet werden dürfen. Eine Meßwertrelation ist also ein Spezialfall einer Relation, die alle Eigenschaften einer ‘normalen’ Relation hat und noch einige mehr.

   Formal sieht eine Meßwertrelation wie folgt aus:

   $$MR_i (A_{ib1}, ..., A_{ibk}, A_{ic1}, ..., A_{icj}, A_{i1}, ..., A_{in}) \subseteq$$
   $$D (A_{ib1}) \times ... \times D (A_{ibk}) \times D (A_{ic1}) \times ... \times D (A_{icj}) \times D (A_{i1}) \times ... \times D (A_{in}),$$
   $$i \in \mathbf{N}, \quad k, j \geq 1;$$

   wobei die A_{ib} die Basisattribute bezeichnen, die A_{ic} die kontinuierlichen Attribute und die A_i sonstige Attribute. Jede Meßwertrelation muß mindestens zwei Attribute, ein Basisattribut und ein kontinuierliches Attribut aufweisen. Eine wesentliche Eigenschaft der Basisattribute ist, daß sie einen Schlüssel oder zumindest einen Teil eines Schlüssels bilden. Die Basisattribute enthalten den Kontext des Meßwerts oder der Meßwerte in jedem Tupel, der die Meßwerte eindeutig gegenüber anderen Werten identifiziert. Im Beispiel in Kap. 6.2.3 ist der Meßwert ‘gdw_stand’ (Grundwasserstand) ein kontinuierliches Attribut. Die zugehörigen Basisattribute sind die Koordinaten ‘x_koord’ und ‘y_koord’ und der Zeitpunkt ‘zeitpunkt’.

Da das für die Implementierung verwendete DBS INGRES [INGR91a] wie die meisten der heute weit verbreiteten relationalen DBS keinerlei Differenzierung von Relationen kennt, wird diese Kennzeichnung der Meßwertrelationen durch die in Abschnitt 6.2.3 beschriebenen DDL-Erweiterungen und die in Abschnitt 6.4 beschriebenen zusätzlichen Systemrelationen implementiert. Mit einem objekt-relationalen DBS wie Informix [Info96], das eine Typisierung von Relationenschemata ermöglicht, wäre eine einfachere Implementierung möglich (siehe auch Kap. 9).

In den Anwendungen des hier beschriebenen Systems bilden ein kontinuierliches Attribut und die zugehörigen Basisattribute eine stetige mathematische Funktion $f \in F$, die die Wertekombinationen der Basisattribute in den Wertebereich des kontinuierlichen Attributs abbildet:

$$f_i: A_{b1} \times A_{b2} \times ... \times A_{bk} \rightarrow A_{ci}, i = 1, ..., j.$$

Zu jedem Zeitpunkt und/oder an jedem Ort des Versuchs nimmt das kontinuierliche Attribut einen bestimmten Wert an. Diese Werte, also die Funktion f_i, können mit Hilfe umliegender Werte berechnet werden. Die Funktion f_i selbst ist nicht bekannt. Jedoch kann ein geeignetes Interpolationsverfahren mit Hilfe einiger vorhandener Werte Teilbereiche von f_i berechnen.

6.2.2 Neue Operationen

In diesem Abschnitt werden für die oben aufgeführten Erweiterungen zwei neue, auf allen Meßrelationen ausführbare Operationen eingeführt, beschrieben in der Notation der Abstrakten Datentypen (ADT), die u.a. in [Gutt77] und [LMWW79] als geeignetes Mittel dazu eingeführt werden. Die Definitionen werden nicht in eine vollständige Datenbankspezifikation eingebettet, wie etwa in [Maib85] angegeben, sondern es werden nur die Erweiterungen spezifiziert. Die Spezifikation dient als Grundlage für die Implementierung. Die Syntax ist ähnlich der in [WSSH88] verwendeten.
Im Gegensatz zu anderen Ansätzen [WSSH88], [StRG83] wird der ADT-Mechanismus hier nicht dem Benutzer angeboten, um selbst neue Datentypen und Operationen darauf zu definieren. Meßrelationen und Interpolationsfunktionen sind alle vom gleichen, hier vorgestellten Daten- bzw. Operationstyp. Der Benutzer erhält ein bereits fertig auf seine Bedürfnisse zugeschnittenes System, in das er nur noch neue Instanzen von Meßrelationen und Interpolationsfunktionen einfügen muß.

Sei R_i die Menge aller möglichen Relationen und A_i seien die Attribute abgeleitet aus den möglichen Wertebereichen. Dann wird die neue Menge Meßwertrelation MR wie folgt definiert:

type: MR **from** R, A, f

operations: $\text{new}: R \times (D(A_{b1}), ..., D(A_{bk})) \times (D(A_{c1}), ..., D(A_{cj})) \rightarrow MR$

...

$\text{interpolation}_1: MR \times f_1 \times (A_{b1}, ..., A_{bk}) \rightarrow A_{c1}$

...

$\text{interpolation}_j: MR \times f_j \times (A_{b1}, ..., A_{bl}) \rightarrow A_{cj}$

variables: $mr \in MR$

relations: $D(\prod_{Ab1 \ldots Abk}(mr)) \subseteq D(A_{b1}, ..., A_{bk})$

$D(\prod_{Aci}(mr)) \subseteq D(A_{ci})$

Diese einfache Interpolation eines Wertes reicht nicht aus, um die in den Abschnitten 3.4.2 und 3.4.3 beschriebenen Probleme zu lösen. Das erste Problem ist der Vergleich von zwei Meßreihen, formal ausgedrückt: der Verbund zwischen ihren Basisattributen. Das zweite ist die Erzeugung äquidistanter Eingabereihen für Modellrechnungen. Hierfür wird eine komplexere Operation benötigt. Dieser nächste Schritt ist die Berechnung äquidistanter Folgen von Werten auf dem kontinuierlichen Attribut. Dafür benötigen die Interpolationsfunktionen weitere Eingabeparameter, die die Distanzen oder Schrittweiten zwischen den Basisattributen der zu interpolierenden Werte angeben. Die Schrittweiten $A_{step\ i}$ müssen eine Intervallbreite aus dem Wertebereich des zugehörigen A_{bi} enthalten, $A_{step\ i} \in \{u\text{-}v \mid u, v \in D(A_{bi}), u > v\}$:

operations: interpol_series$_1$:

$$MR \times f_1 \times (A_{b1}, ..., A_{bk}) \times A_{step1} \times ... \times A_{step\ m} \rightarrow R (A_{b1}, ..., A_{bk}, A_{c1})$$

. . .

interpol_series$_j$:

$$MR \times f_j \times (A_{b1}, ..., A_{bl}) \times A_{step1} \times ... \times A_{step\ n} \rightarrow R (A_{b1}, ..., A_{bk}, A_{cj}) \qquad m \leq k,\ n \leq l$$

variables: $mr \in MR$

relations: $D (\prod_{Ab1, ..., Abk} (mr)) \subseteq D (A_{step1} \times ... \times A_{stepk})$

A_{step} kann hier ein- oder mehrdimensional sein, also eine Interpolation entlang einer Geraden (im Raum) oder auf einer n-dimensionalen Ebene. (Implementiert wurde zunächst die Interpolation entlang einer Geraden im Raum, weil bei der zweiten Variante leicht Performance-Probleme auftreten können. Denn die Anzahl der zu interpolierenden Werte wächst bei der zweiten Variante exponentiell mit der Anzahl variierender Basisattribute n, denn für jeden Wert des ersten Basisattributs werden alle möglichen Werte bzw. Kombinationen von Werte der weiteren Basisattribute generiert. Werden bei zwei Basisattributen je 100 Schritte angegeben, werden in der ersten Variante 100 Werte interpoliert, in der zweiten Variante jedoch 100 * 100, also 10000 Werte, siehe auch Bild 9.2, Kap. 9.2.)

6.2.3 DDL-Erweiterungen

Syntaktisch sind einige geringfügige Erweiterungen der DDL notwendig, um dem DBS mitzuteilen, welche Attribute kontinuierliche Attribute sind, und welches die zugehörigen Basisattribute sind. Eine Relation, die kontinuierliche Attribute und (Verweise auf) Basisattribute enthält, wird als Meßwertrelation erkannt. (Die DDL-Syntax folgt [INGR91a].)

```
CREATE  TABLE  <relation name >
                (<attr.name>    <data format>   BASE ATTRIBUTE
                {,<attr.name>   <data format>   BASE ATTRIBUTE}
                 ,<attr.name>   <data format>   CONTINUOUS ATTRIBUTE
                {,<attr.name>   <data format>   CONTINUOUS ATTRIBUTE}
                {,<attr.name>   <data format>} )
                [WITH [LOCATION = (<location> {, <location>})]
                                  [[NO] JOURNALING [[NO]DUPLICATES]]
```

mit

```
<data format> = <data type> [NOT NULL [WITH | NOT DEFAULT] | WITH NULL]
```

Die Grundwasserstände-Relation (siehe Kap. 10) wird mit dieser Syntax wie folgt definiert:

```
CREATE  TABLE  gwmessung
               (x_koord     float4     BASE ATTRIBUTE,
                y_koord     float4     BASE ATTRIBUTE,
                zeitpunkt   date       BASE ATTRIBUTE,
                gdw_stand   float4     CONTINUOUS ATTRIBUTE,
                gwm_id      vchar(8))
```

Ein Interpolationsverfahren wird im DBS wie folgt deklariert:

```
DECLARE METHOD <interpol.meth.>
                ON TABLE <rel.name>
                CONTINUOUS ATTRIBUTE <attr.name>
                VARYING BASE ATTRIBUTES <attr.name> {, <attr.name>}
```

Die Deklaration des Interpolationsverfahrens für Grundwasserstände sieht dann so aus:

```
DECLARE  METHOD gdw_staende
                ON TABLE gwmessung
                CONTINUOUS ATTRIBUTE gdw_stand
                VARYING BASE ATTRIBUTES x_koord, y_koord
```

6.2.4 Erweiterungen der Anfragesprache

Um Interpolation in eine DB-Anfrage einzubetten, müssen folgende Angaben innerhalb der Anfrage zusätzlich spezifiziert werden: Das Interpolationsverfahren und das kontinuierliche Attribut, auf das es angewendet werden soll, der Punkt, der interpoliert werden soll und bei Bedarf die Schrittweite für Folgen äquidistanter Interpolationswerte.

Die Syntax wurde so gewählt, daß sie an die natürlichsprachliche Formulierung entsprechender Anfragen angelehnt ist, ähnlich wie dies bei SQL ja auch gemacht wurde: Klauseln und Namen stehen an der Stelle in einer Anfrage, an der sie auch in einem frei formulierten, natürlich-

sprachlichen Satz auftreten würden. Somit hat CSQL die gleichen Vorteile wie SQL, daß Benutzer der natürlichen Sprache folgend damit umgehen können und CSQL relativ leicht erlernbar ist. CSQL hat aber auch die gleichen Nachteile wie SQL, nämlich, daß Namen u.ä. nicht in der logisch richtigen, für die Verarbeitung günstigen Reihenfolge angegeben werden.

Um alle notwendigen Angaben für eine Interpolation machen zu können, werden drei Klauseln benötigt, eine Klausel für den Namen der Interpolationsfunktion, eine Klausel für den zu interpolierenden Punkt und eine optionale Klausel für eventuelle Schrittweiten. Diese drei Klauseln sollen hier erklärt werden.

Die BY METHOD-Klausel

Die BY METHOD-Klausel gibt die Methode, also das Interpolationsverfahren an und das Attribut, das interpoliert werden soll. Die Angabe des zu interpolierenden Attributs ist zwar nur bei Verbund einer Meßwertrelation mit sich selbst zwingend erforderlich, wird aber aus Gründen der Einfachheit und Übersichtlichkeit immer gefordert. Die BY METHOD-Klausel folgt im Anschluß auf die FROM-Klausel, wenn die an der Anfrage beteiligten Relationen (evtl. Verbund) bekannt sind. Es dürfen nur Attribute aus diesen Relationen interpoliert werden. Die BY METHOD-Klausel hat folgendes Aussehen:

BY METHOD <method> (<cont. attr.>)

<method> ist hier der Name der Interpolationsfunktion und <cont. attr.> der Name des kontinuierlichen Attributs. In einer Anfrage kann für jedes kontinuierliche Attribut in der Ergebnisrelation höchstens eine BY METHOD-Klausel angegeben werden.

Die ENTRY-Klausel

Die ENTRY-Klausel gibt die Basis an, zu der ein Wert interpoliert werden soll. Oder sie gibt den Anfangswert, den ‘Eintrittswert’ für eine Reihe äquidistanter zu interpolierender Werte an. Die ENTRY-Klausel hat mindestens einen und höchstens so viele Einträge wie das Verfahren variierende Basisattribute hat. Fehlende variierende Basisattribute werden wie feste Basisattribute behandelt, d.h., sie werden zu den derzeit in der DB vorhandenen Werten (Wertekombinationen) ausgewertet, soweit nicht in der WHERE-Klausel weiter eingeschränkt wird. Derzeit sind in der ENTRY-Klausel nur numerische Werte, keine Referenzen, arithmetischen Ausdrücke u.ä. zulässig. Sie hat folgende Syntax:

ENTRY (<base_attr.> = <value> {, <base_attr.> = <value> })

Zu jeder BY METHOD-Klausel muß es genau eine ENTRY-Klausel geben.

Die STEP-Klausel

Die STEP-Klausel dient der Spezifikation von äquidistanten Folgen von Werten. Für alle in der ENTRY-Klausel angegebenen Werte werden die Abstände, die ‘Schrittweiten’ angegeben. Auch hier sind nur numerische Angaben zulässig. Die STEP-Klausel sieht wie folgt aus:

STEP (<distance> {, <distance> })

Zu jeder ENTRY-Klausel gibt es höchstens eine STEP-Klausel. Eine STEP-Klausel ohne ENTRY-Klausel ist nicht zulässig. Die STEP-Klausel muß genauso viele Werte beinhalten wie die zugehörige ENTRY-Klausel und die Attribute in der gleichen Reihenfolge ansprechen.

Die CSQL-Syntax, die die Spezifikation von Interpolationsmethoden, beliebigen ‘Meßpunkten’ (ENTRY-Punkten) und Schrittweiten (STEP) für äquidistante Folgen von Werten erlaubt, sieht folgendermaßen aus (ähnlich der Syntax in [INGR91a]). Die Basisattribute, die in der STEP-Spezifikation nicht auftreten, werden zu den in der Datenbank vorhandenen Werten ausgewertet:

```
SELECT [ALL | DISTINCT] “*” | [<result_column> =] <expression>
                        {, [<result_column> =] <expression>}
FROM    <table> [<corr_name>] | <table> [<corr_name>]
                        BY METHOD <method> (<cont.attr.>)
                          ENTRY (<base_attr.> = <value> {, <base_attr.> = <value> })
                         [ STEP (<distance> {, <distance>}) ]
                        { BY METHOD <method> (<cont.attr.>)
                              ENTRY (<base_attr.> = <value> {, <base_attr.> = <value> })
                            [ STEP (<distance> {, <distance>} ) ] }
          {, <table> [<corr_name>] | ...}
[WHERE <search_condition> ]
[GROUP BY <attr.> {, <attr.>}
[HAVING <search_condition>] ]
[ORDER BY <result-attr.>  [ASC | DESC] {,<result_attr.> [ASC | DESC] } ]
```

Hier entspricht die Erweiterung

```
<table> [<corr_name>]
BY METHOD <method> (<cont.attr.>)
  ENTRY (<base_attr.> = <value> {, <base_attr.> = <value> })
```

der Spezifikation einer Interpolationsfunktion für einen Wert

$$\text{interpolation}_j: MR \times f_j \times (A_{b1}, ..., A_{bk}) \rightarrow A_{cj}$$

und die Erweiterung

```
<table> [<corr_name>]
BY METHOD <method> (<cont.attr.>)
  ENTRY (<base_attr.> = <value> {, <base_attr.> = <value> })
  STEP (<distance> {, <distance>})
```

entspricht der Interpolationsfunktion für eine Folge von äquidistanten Werten

interpol_series$_j$:

$$MR \times f_j \times (A_{b1}, ..., A_{bk}) \times A_{step1} \times ... \times A_{step\ m} \rightarrow R\ (A_{b1}, ..., A_{bk}, A_{cj}), \quad m \leq k$$

In der Syntax entspricht <table> der Meßrelation MR, und <method> gibt den Namen der Interpolationsmethode f_j . In der ENTRY-Klausel werden die A_{b1}, ..., A_{bk} gegeben, und in der STEP-Klausel die $A_{step1} \times ... \times A_{step\ m}$.

Ähnliche Erweiterungen von QUEL wurden unter dem Namen POSTQUEL von [StRo86] eingeführt, um Aggegrationsfunktionen und komplexe Objekte mit geometrischen Operationen zu handhaben. (Komplexe) Objekte können Attribute erhalten, die vom Datentyp ‘Prozeduraufruf’ oder ‘DML-Operation’ sind und auch als Wert jeweils einen Prozeduraufruf bzw. eine DML-Operation enthalten. Der Aufruf dieser Funktionen und DML-Operationen erfolgt dann über die Auflistung dieser speziellen Attribute in der SELECT-Klausel und ist in der Anfrage ohne Kenntnis des Schemas nicht als Prozeduraufruf zu erkennen. Diese Erweiterungen sind damit Erweiterungen der Definitionssprache aber nicht der Anfragesprache. Der Aufruf von Prozeduren wird damit komplizierter, er ist nämlich nur noch über Attribute möglich, die diesen besonderen Datentyp haben. Damit können dann auch sehr allgemeine und für den Benutzer schwer nachvollziehbare Operationen durchgeführt werden. (Z.B. können weitere Relationen berührt werden, die ein bestimmter Benutzer gar nicht sehen darf. Er bekäme dann für ihn unerklärlich ein leeres Ergebnis.) Das ist für ein System, mit dem Anwender arbeiten sollen, ungeeignet. Deshalb wurde hier im CSQL-System eine möglichst einfache Syntax mit einleuchtender Semantik gewählt.

6.3 Verträglichkeit mit Standard-SQL

Zunächst soll hervorgehoben werden, daß Meßwertrelationen natürlich weiterhin auch als konventionelle Relationen betrachtet werden können und alle relationalen Operationen auf sie angewendet werden können[1]. Es ist nicht sinnvoll und wünschenswert, Interpolation innerhalb von Datenmanipulation einzusetzen, weil Änderungen auf temporär berechneten Ergebnissen nicht sinnvoll sind und berechnete und gemessene Werte keinesfalls vermischt werden dürfen. Die einzige Ausnahme bildet die Interpolation innerhalb eines SELECT-Statements zum Kopieren von Daten in neue bzw. andere Relationen.

1: Die SQL-Anweisungen HELP und CREATE PROCEDURE sind im CSQL-System nicht möglich, weil CSQL mit Embedded SQL/C (siehe Kap. 8) implementiert wurde. Dort sind diese Anweisungen auch nicht zulässig.

6.3.1 Anfrageteil von SQL

Interpolation kann praktisch in jede Datenbankanfrage eingebettet werden. Voraussetzungen sind lediglich, daß mindestens eine der beteiligten Relationen eine Meßwertrelation ist und daß dem DBS ein passendes Interpolationsverfahren bekannt ist. Wichtig für die Auswertung der Anfragen sind die Fragen, welchen Einschränkungen diese Anfragen unterliegen und an welcher Stelle der Auswertung die Interpolation ausgeführt werden kann und darf.

Die Einbeziehung der Interpolation besteht aus eigenen Klauseln, die sinngemäß zur SELECT-Klausel gehören. Deshalb können auch nach jeder SELECT- (und WHERE-) Klausel Interpolations-Klauseln, d.h. BY METHOD-Klauseln, evtl. mit STEP-Klauseln folgen.

a) Auswertung von Interpolationsanfragen

Die Auswertung der Anfragen soll hier nur soweit beschrieben werden, wie es notwendig ist, um festzustellen, ob und wie weit die Interpolation mit Standard-SQL zusammenarbeitet. Weitere Ausführungen hierzu sind in Kapitel 8 und auch in [Kaja90] zu finden. Eine zusammenfassende Übersicht gibt die Tabelle 6.1 am Ende dieses Abschnitts.

Restriktionen (WHERE-Klausel)

Für jede Einschränkung ist zu klären, ob die Interpolation vor oder nach deren Auswertung angewendet werden soll. Vier Fälle müssen hier unterschieden werden:

- Einschränkungen auf festen Basisattributen.
  Für das Ergebnis bedeutet es keinen Unterschied, ob diese Einschränkungen vor oder nach der Interpolation ausgewertet werden, weil für eine Interpolation stets nur Tupel mit gleichen Werten in festen Basisattributen herangezogen werden. Unter Berücksichtigung des Antwortzeitverhaltens sollte die Auswertung aber so früh wie möglich erfolgen, um die Anzahl der Interpolationsvorgänge zu verringern.

- Einschränkungen auf kontinuierlichen Attributen.
  Sie müssen auf den berechneten Werten durchgeführt werden, damit ein Benutzer auch ein Ergebnis erhält, das den von ihm spezifizierten Bedingungen genügt. Würde eine solche Einschränkung schon vor der Interpolation ausgewertet, könnten durch die Interpolation wieder Werte erzeugt werden, die diese Einschränkung verletzen. Außerdem könnten für die Interpolation relevante Tupel ausgeschlossen werden.

  Beispiel:
  Gegeben sei eine Relation

```
CREATE TABLE temperatur (meßwert float CONTINOUS ATTRIBUTE,
                         zeitpunkt date BASE ATTRIBUTE)
```

Dabei ist meßwert das kontinuierliche Attribut und zeitpunkt das variierende Basisattribut. Der Benutzer spezifiziert die Restriktion meßwert > 0. Wird diese Restriktion vor der Interpolation ausgewertet, wird die Interpolation für Tage, an denen die Temperatur tagsüber über 0 Grad lag und nachts darunter, wenn sie den sinusförmigen Verlauf einer Temperaturkurve nachbildet, für die nächtlichen Zeitpunkte wieder negative Werte berechnen. Diese möchte der Benutzer aber gar nicht sehen. Sie müssen also nach der Interpolation entfernt werden. Außerdem würde die Interpolation insgesamt ungenauer, weil ein Teil der Werte, die negativen, nicht mitberücksichtigt werden.

- Einschränkungen auf weiteren Attributen.
  Die Anwendung von Interpolation impliziert eine Projektion auf ein kontinuierliches Attribut und die Basisattribute. Nach einer Interpolation, die aus Werten aus verschiedenen Tupeln einen neuen Wert berechnet, läßt sich über die weiteren Attribute i.a. überhaupt keine Aussage mehr machen. Deshalb müssen Einschränkungen auf weiteren Attributen zwingend vor der Interpolation ausgewertet werden.

  Beispiel:
  Wird z.B. bei Messungen des Grundwasserstands in Meßpegeln jeweils vermerkt, welche Person diese Messung durchgeführt hat, kann dieses Attribut Person nach der Interpolation keinen sinnvollen Wert mehr enthalten, weil ja keine echte Messung durchgeführt wurde.

- Einschränkungen auf variierenden Basisattributen.
  Diese Einschränkungen dürfen eigentlich erst nach der Interpolation ausgewertet werden, weil sonst Tupel wegfallen könnten, die zur Interpolation herangezogen werden sollten. Andererseits können diese Restriktionen bei frühzeitiger Auswertung dazu dienen, die Anzahl der Interpolationsvorgänge entscheidend zu verringern. Ist bekannt, in welcher Distanz von den eigentlichen Meßpunkten die Interpolationsfunktion noch gültige Werte liefert, kann eine solche Restriktion nach Addition dieser Distanz auch vor der Interpolation schon ausgewertet werden. Die Restriktion muß dann aber nach der Interpolation noch einmal überprüft werden.
  Da dieser Fall kompliziert ist, soll er im folgenden ausführlich erläutert werden.

Einschränkungen auf variierenden Basisattributen müssen nach der Interpolation, bzw. aus Performancegründen vor und nach der Interpolation durchgeführt werden. Trotzdem kann es manchmal zu vom Benutzer unerwarteten Ergebnissen kommen, bzw. wünschenswerte Ergebnismengen lassen sich nicht spezifizieren. Betrachtet man die Relation in einer Wetterdatenbank

```
CREATE TABLE wetter_60 (zeitpunkt date BASE ATTRIBUTE,
                        strahlung float CONTINOUS ATTRIBUTE,
                        guete_str vchar(3))
```

mit der Basis zeitpunkt, dem kontinuierlichen Attribut strahlung und dem sonstigen Attribut guete_str, die Werte zur Intensität der Sonneneinstrahlung und die Qualität dieser Werte (guete_str) zu bestimmten Zeitpunkten an einer Wetterstation enthält. Der Strahlungswert ist

umso höher, je stärker die Sonne scheint; nachts ist er gleich null. Negative Strahlungswerte gibt es nicht, bzw. es sind fehlerhafte Messungen.

Durch einen Defekt des Meßgeräts wurden nachts z.T. negative Werte aufgezeichnet. Stellt nun der Benutzer die folgende Anfrage

```
SELECT zeitpunkt, strahlung
FROM   wetter_60   BY METHOD strahlung_intp (strahlung)
                   ENTRY (zeitpunkt = '22/04/90 06:00:00')
                   STEP ('3 hours 0 minutes 0 seconds')
WHERE  zeitpunkt   <= '06/05/90 18:00:00' AND
       strahlung   >=  0;
```

so benutzt er die Restriktion ‘strahlung >= 0‘, um damit fehlerhafte Aufzeichnungen auszublenden. Es kann nun zu folgenden Ergebnissen kommen:

Wird strahlung >= 0 **vor** der Interpolation ausgewertet, werden die fehlerhaften Werte ausgeblendet, und passende Werte (= 0) werden von der Interpolationsfunktion (siehe auch [Fabr91]) geliefert. Der Benutzer erhält das gewünschte Ergebnis.

Wird die Bedingung erst **nach** der Interpolation ausgewertet, werden auch alle aufgrund der fehlerhaften Messung interpolierten Werte, die kleiner 0 sind, entfernt: Es läßt sich keine äquidistante Folge von Werten erzeugen, obwohl dies von der Anzahl der vorhandenen gültigen Werte und der Qualität der Interpolationsfunktion her möglich wäre. Um das gewünschte Ergebnis zu erhalten, muß in diesem Fall also **vor** der Interpolation ausgewertet werden.

Stellt nun ein anderer Benutzer folgende Anfrage

```
SELECT zeitpunkt, strahlung
FROM   wetter_60   BY METHOD strahlung_intp (strahlung)
                   ENTRY (zeitpunkt = '22/04/90 06:00:00')
                   STEP ('3 hours 0 minutes 0 seconds')
WHERE  zeitpunkt   <= '06/05/90 18:00:00' AND
       strahlung   >=  1;
```

weil er ein Ergebnis ohne ganz niedrige Strahlungswerte wünscht, die er für seine weiteren Berechnungen nicht benötigt, so gibt es folgende Möglichkeiten: Wird die Bedingung ‘strahlung >= 1‘ **vor** der Interpolation ausgewertet, werden evtl. zu viele Tupel ausgeblendet; das Ergebnis wird ungenau. Eine Auswertung **nach** der Interpolation liefert hier das gewünschte Ergebnis.

Je nachdem, was der Benutzer wünscht, eine Ausblendung fehlerhafter Meßwerte oder eine willkürliche Einschränkung seiner Ergebnismenge, ist hier eine andere Auswertungsstrategie angezeigt. Da man i.a. von einer willkürlichen Einschränkung der Ergebnismenge ausgehen muß, ist eine Auswertung der Einschränkungen auf variierenden Basisattributen **nach** der Interpolation vorzunehmen.

In dem hier geschilderten Fall könnte der Güteparameter dem ersten Anwender helfen. Fehlerhafte Meßwerte, wie sie z.B. immer in den ersten Minuten nach dem Einschalten des Geräts auftreten, werden schon vom Meßgerät mit einer entsprechenden Kennzeichnung, guete_str = '-', versehen, gültige mit guete_str = '+'. Fehlerhafte Messungen könnte man mit guete_str = '+' vor der Interpolation abfangen, wenn Bedingungen auf variierenden Basisattributen nach der Interpolation ausgewertet werden:

```
SELECT zeitpunkt, strahlung
FROM   wetter_60   BY METHOD strahlung_intp (strahlung)
                   ENTRY (zeitpunkt = '22/04/90 06:00:00')
                   STEP ('3 hours 0 minutes 0 seconds')
WHERE  zeitpunkt   <= '06/05/90 06:00:00' AND
       guete_str   =  '+'                 AND
       strahlung   >= 0;
```

Diese Anfrage sollte im Normalfall auch ohne die Qualifikation 'strahlung >= 0' ein korrektes Ergebnis liefern, denn korrekte Strahlungswerte sind immer größer als Null (tagsüber) oder gleich Null (nachts). Für den zweiten Anwender, der nur Strahlungswerte größer als eins wünscht, könnte mit guete_str = '+' eine weitere Verbesserung des Ergebnisses erreicht werden, denn es werden alle fehlerhaften Werte, auch wenn sie größer als eins sind, bei der Interpolation nicht mit betrachtet.

Der Kern des hier geschilderten Problems liegt darin, daß man bei variablen Basisattributen und kontinuierlichen Attributen die Werte vor und nach der Interpolation nicht unterschiedlich referenzieren kann. Die in Kap. 9 geschilderte Alternative für Syntax und Auswertungsstrategie von CSQL löst dieses Problem.

Verbunde (Joins)

Es werden i.a. die Basisattribute über einen (natürlichen) Verbund verknüpft, und zwar nach der Interpolation, um die kontinuierlichen Attribute vergleichen zu können. Ein Verbund der kontinuierlichen Attribute ist in den meisten Fällen nicht sinnvoll. Wenn so ein Verbund jedoch gewünscht wird, muß er nach der Interpolation stattfinden, weil erst dann die Werte der Attribute vorliegen. Weitere Attribute wurden durch die Interpolation bereits herausprojiziert und dürfen daher nicht in den Verbundbedingungen auftreten.

Projektionen

Die Interpolation bedingt eine Projektion auf das kontinuierliche Attribut und die Basisattribute Diese Projektion muß (s.o.) vor der Interpolation durchgeführt werden. Weitere Projektionen nach der Interpolation - auch in einer Verbundrelation - sind möglich.

Gruppierung, eingebaute Funktionen (Aggregation) und Sortierung

Diese Operationen werden in SQL auf den Ergebnissen der in der FROM- und WHERE-Klausel spezifizierten Relationen (entspricht der Lese-Operation), Verbund-Operationen und Restriktionen durchgeführt. Sie müssen deshalb nach der Interpolation, die auch in der FROM-Klausel spezifiziert wird und zur Spezifikation der Relation gehört, erfolgen. In CSQL dürfen sie somit nicht auf weiteren Attributen spezifiziert werden, weil die weiteren Attribute zum Zeitpunkt der Auswertung dieser Anweisungen bereits hinausprojiziert wurden.

Schachtelung von Anfragen

Interpolation innerhalb geschachtelter (innerer) SELECT-Anfragen ist nur dann erlaubt, wenn sich diese entschachteln lassen oder zu einem konstanten Wert berechnen lassen. Näheres dazu, welche Typen von geschachtelten Anfragen sich prinzipiell entschachteln lassen, welche sich im Präprozessor-Ansatz auf dem DBS INGRES entschachteln lassen und welche sich in der derzeitigen CSQL-Version entschachteln lassen, findet sich in Kapitel 8.5 über Optimierung im CSQL-System und in [Jahk91].

Fehlerbehandlung

Eine grundsätzlich neue Art der Fehlerbehandlung bei Eingabefehlern des Benutzers ist nicht erforderlich. Durch die CSQL-Erweiterungen erhöht sich lediglich die Anzahl der möglichen verschiedenen Fehler und damit der Fehlermeldungen. Es können jetzt zusätzlich Fehler wie

“Relation ‘R’ ist keine Meßwertrelation”,

“Attribut ‘A’ ist kein Basisattribut” oder

“ Für Attribut ‘A’ ist Interpolationsverfahren ‘I’ nicht definiert”

auftreten. Wenn eine Interpolation aufgrund fehlender umgebender Werte nicht möglich ist, wird ein leeres Ergebnis geliefert, wie sonst auch, wenn in der Datenbank keine auf die gegebenen Qualifikationen passenden Werte gefunden werden können.

Anwendung von	Kontinuierliche Attribute	Feste Basisattribute	Variierende Basisattribute	Sonstige Attribute
Restriktionen	nachher	vorher / nachher	(vorher u.) nachher	vorher
Verbunde (Joins)	(nachher)	nachher	nachher	- -
Projektionen	- -	nachher	nachher	obligat. vorher
Aggregationen	nachher	(nachher)	(nachher)	- -
Gruppierung	nachher	nachher	nachher	- -
Sortierung	nachher	nachher	nachher	- -

Tabelle 6.1: Zusammenspiel von Interpolation und relationalen Operationen

Wenn ein Benutzer umständliche aber prinzipiell berechenbare Kombinationen von Klauseln und Bedingungen eingibt, z.B. eine Interpolation in der inneren Schleife und ein EXISTS-Prädikat auf das variable Basisattribut in der äußeren Schleife, wird die Anfrage nicht unbedingt abgewiesen oder automatisch vereinfacht, sondern langwierig berechnet.

b) Einschränkungen für Anfragen mit Interpolation

Es gibt einige Kombinationen von Standard-SQL-Klauseln und CSQL-Erweiterungen, die nicht sinnvoll sind und deshalb vom System als Fehler zurückgewiesen werden müssen. Das sind die folgenden Kombinationen:

- Arithmetische Ausdrücke, in denen sowohl kontinuierliche (interpolierte) Attribute als auch zusätzliche Attribute auftreten, sind nicht möglich, weil die ersten nur nach der Interpolation und die zweiten nur vor der Interpolation verfügbar sind (siehe auch Ausführungen zur Auswertung).

- Dadurch, daß manche Restriktionen bzw. Attribute in Restriktionen vor der Interpolation ausgewertet werden müssen, sind nicht alle denkbaren Restriktionen erlaubt. Es sind keine Restriktionen erlaubt, in denen sonstige Attribute direkt mit kontinuierlichen Attributen verglichen werden sollen. Es sind auch keine Verbundbedingungen erlaubt, die sonstige Attribute mit Attributen aus Verbundrelationen vergleichen, weil Verbunde erst nach der Interpolation durchgeführt werden.

Es gibt auch einige Kombinationen von Standard-SQL-Klauseln und CSQL-Erweiterungen, die zwar ein korrektes Ergebnis liefern, die aber auch ohne Interpolation das gleiche Ergebnis liefern würden bzw. deren Ergebnis von der (benutzergegebenen) Belegung der variablen Basisattribute abhängt. Die Funktionalität des CSQL-Systems bleibt also gewährleistet, auch wenn solche Anfragen nicht bearbeitet werden (können). Das sind folgende Kombinationen:

- EXISTS-Prädikate (und auch NOT EXISTS-Prädikate), wenn in der inneren Anfrage interpoliert wird, und der Bezug zur äußeren Anfrage über das kontinuierliche Attribut gebildet wird, wie

```
SELECT  *
FROM    a
WHERE  EXISTS ( SELECT *
                FROM   b  BY METHOD b1 (b_cont)
                          ENTRY (basis = 10)
                          STEP (5)
                WHERE basis   < 50      AND
                          b_cont >  a.attribut);
```

  Diese Anfrage liefert alle Tupel der Relation a, für die es in der Relation b Tupel gibt, in denen der Wert des kontinuierlichen Attributs b_cont größer ist als der Wert des Attributs a der Relation a. Bei Interpolation aus den umliegenden Werten entsteht i.a. kein Wert, der größer

ist als die in der DB gespeicherten Werte (s.u.). Diese Anfrage würde ohne Interpolation also zumindest annähernd das gleiche Ergebnis liefern, weil die interpolierten Werte aus den umliegenden Werten abgeleitet werden.

Besteht der Bezug zur äußeren Anfrage über die variierenden Basisattribute, ist Interpolation völlig überflüssig und das Ergenis wesentlich einfacher zu erlangen, denn die variierenden Basisattribute sind ja durch die STEP-Klausel festgelegt. Das Ergebnis der Anfrage

```
SELECT *
FROM   a
WHERE  EXISTS (SELECT *
               FROM   b  BY METHOD b1 (b_cont)
                         ENTRY (basis = 10)
                         STEP (5)
               WHERE basis  < 50       AND
                     basis  =  a .attribut);
```

ließe sich auch durch folgende Anfrage erzielen:

```
SELECT *
FROM   a
WHERE  a.attribut  IN (10, 15, 20, 25, 30, 35, 40, 45);
```

Besteht der Bezug zur äußeren Anfrage über ein festes Basisattribut, ist die Anfrage gleichwertig zu einer Anfrage ohne Interpolation. (Es müssen auch umständlich formulierte Anfragen korrekt bearbeitet werden, solange sie syntaktisch richtig sind.)

- Aggregationen im Zusammenhang mit Interpolation sind häufig wenig sinnvoll, wenn sie nicht in Verbindung mit Gruppierungen benötigt werden, weil ihre Ergebnisse auch ohne Interpolation erreichbar sind, wie bei Aggregationen auf festen Basisattributen und beim Zählen in Verbindung mit Interpolation. Bei Aggregation auf variierenden Basisattributen wird der Inhalt der Datenbank eigentlich gar nicht benötigt. Die Ergebnisse von Aggregationen auf interpolierten kontinuierlichen Attributen sind denen der nicht interpolierten i.a. sehr ähnlich. Das Maximum der interpolierten Werte eines Intervalls wird dicht beim Maximum der gemessenen Werte liegen. (Das Maximum der interpolierten Werte kann größer sein als das gemessene Maximum, wenn zur Interpolation z.B. eine Sinusfunktion untergelegt wird. Z.B., wenn am heißesten Tag des Sommers die Messung zu der Uhrzeit fehlt, zu der gewöhnlich das Tagesmaximum gemessen wird.) Da Interpolation jedoch auch angewendet werden kann, um Meßreihen zu ‘glätten’, sollten solche Anfragen dennoch bearbeitet werden. Insbesondere die Summierung von interpolierten Werten kann sinnvoll sein.

Fragen und Probleme bei der Einbettung

Es ist wichtig zu klären, was geschehen soll, wenn ein Benutzer die Interpolation für einen in der Datenbank vorhandenen Wert spezifiziert. Dieses Problem wird hier in die Interpolationsfunktionen verlagert. Es darf Funktionen geben, die diese Werte nur aus der Datenbank lesen,

und solche, die sie auch berechnen. Die Interpolationsfunktionen müssen entsprechend dokumentiert werden, so daß die Benutzer die Ergebnisse richtig interpretieren können. Wenn auch diese Werte berechnet werden, kann man die Interpolation gezielt einsetzen, um Unregelmäßigkeiten in den Messungen aufzuspüren.

Ein anderes Problem kann entstehen, wenn eine Interpolation weitere Informationen aus anderen (Meßwert-) Relationen in der Datenbank zu Hilfe nimmt. Z.B. könnte die Interpolationsfunktion zur Berechnung des Grundwasserstands an beliebigen Punkten die Beschaffenheit des Untergrunds (Sand, Lehm, Ton, Fels) heranziehen. Hier kann es zu leeren Ergebnissen kommen, die der Benutzer zunächst nicht versteht, wenn z.B. die Beschaffenheit des Untergrunds an einigen Interpolationsstellen fehlt. Eine gute Dokumentation und detaillierte Meldungen können jedoch Abhilfe schaffen. (Ähnliche Probleme können im “normalen” SQL bei der Arbeit mit Sichten auftreten.)

Probleme bereitet es auch, unsinnige bzw. sehr umständliche Anfragen zurückzuweisen oder entsprechend zu vereinfachen. Z.B. sind die Aggregationen MIN und MAX auf jeden Fall und die Aggregationen SUM und AVG, wenn nur in eine Dimension interpoliert wird, auf dem variierenden Basisattribut nicht sinnvoll bzw. ohne Interpolation gleichwertig und wesentlich einfacher zu berechnen. Hier muß der Benutzer, wenn er solche Anfragen stellt, eine entsprechend lange Bearbeitungszeit in Kauf nehmen, wie er das im “normalen” SQL evtl. auch muß, wenn er sehr umständliche Anfragen formuliert.

Interpolation in der HAVING-Klausel als geschachteltes SELECT könnte in manchen Fällen sinnvoll sein. Da aber das DBS INGRES in der zur Projektlaufzeit verfügbaren Version derartige Anfragen nicht korrekt bearbeitete, wurde hier auf eine Implementierung verzichtet.

6.3.2 Datenmanipulationsteil von SQL

Prinzipiell ist Interpolation überall dort erlaubt, wo SQL ein sog. Subselect, also eine (geschachtelte) SELECT-Anweisung erlaubt. Das bedeutet, daß Interpolation in den WHERE-Bedingungen von UPDATE- und DELETE-Anweisungen vorkommen darf. Außerdem können interpolierte Werte in neue bzw. eigene Relationen eingefügt werden, also permanent gespeichert werden, mit den Anweisungen:

```
CREATE TABLE <tablename> AS SELECT . . .
```

```
INSERT INTO <tablename> SELECT . . .
```

Interpolierte Werte oder ENTRY- / STEP-Werte können natürlich nicht geändert oder gelöscht werden, da sie nicht in der DB materialisiert sind. Deshalb dürfen sie nur in andere Relationen eingefügt werden. (Im derzeitigen Prototyp von CSQL ist Interpolation in Verbindung mit Änderungen der DB noch nicht möglich, weil entsprechende Überprüfungen noch nicht implementiert sind.)

6.4 Zusätzliche Systemrelationen

Damit die Interpolation mit dem beschriebenen Funktionsumfang in ein DBS eingebettet werden kann, muß das DBS selbst bzw. das CSQL-Modul einige Informationen über die Interpolation enthalten. Diese Angaben werden über zusätzliche Systemrelationen implementiert, denn diese Informationen müssen permanent gespeichert und zugreifbar sein. Die zusätzlichen Systemrelationen werden bei der Installation des CSQL-Systems angelegt und vom CSQL-System bei der Definition von Meßwertrelationen und Interpolationsverfahren automatisch gefüllt, genau wie “normale” Systemrelationen bei der Definition von Relationen gefüllt werden. Der einfache CSQL-Anwender kann sie nicht ändern.

Zunächst muß dem System bekanntgegeben werden, welche Interpolationsverfahren es gibt und auf welches kontinuierliche Attribut welcher Relation sie angewendet werden dürfen. Dabei muß es möglich sein, mehrere Interpolationsverfahren für dasselbe kontinuierliche Attribut zu definieren, denn es können verschiedene gleichwertige Methoden existieren, die von den Benutzern unterschiedlich bevorzugt werden. Für Interpolation über verschiedene variierende Basisattribute müssen verschiedene Interpolationsverfahren definiert werden. Dieses Verfahren erlaubt auch, unter anderen Namen verschiedene Versionen derselben Methode bereitzustellen.

Es soll nicht möglich sein, ein Interpolationsverfahren mit demselben Namen für unterschiedliche Relationen zu definieren. Damit soll erschwert werden, daß eine Interpolation auf eine Messung angewendet wird, auf die das Verfahren nicht paßt. Außerdem greifen die Interpolationsfunktionen direkt auf die Datenbank zu. Dabei lesen sie außer der Meßwertrelation auch u.U. noch Daten aus anderen Relationen, um das Ergebnis zu verbessern. Wie oben erwähnt, kann es bei der Interpolation von Grundwasserständen über die Fläche z.B. nützlich sein, außer den Grundwasserständen umliegender Meßpegel auch noch die Bodenbeschaffenheit der Umgebung des zu interpolierenden Punkts hinzuzuziehen. Solcher Abhängigkeiten wird sich der Benutzer eher bewußt, wenn er für jedes kontinuierliche Attribut eine andere Interpolationsfunktion definieren bzw. spezifizieren muß.

Der CSQL-Präprozessor muß später überprüfen können, ob Interpolationsverfahren auf kontinuierlichen Attributen definiert wurden und ob die Benutzer die Interpolationsfunktionen auf die richtigen Attribute anwenden. Zunächst müssen die Meßwertrelationen mit ihren kontinuierlichen und Basisattributen ausgezeichnet werden. Dazu werden zwei Systemrelationen benötigt, weil die Anzahl der kontinuierlichen Attribute pro Relation und die Anzahl der Basisattribute pro kontinuierlichem Attribut nicht begrenzt sind. Zu jedem Basisattribut wird außerdem festgehalten, ob es direkt in der Meßrelation steht oder über einen Fremdschlüssel aus einer anderen Relation geholt werden muß.

Weiterhin werden zwei Systemrelation für die Interpolationsverfahren selbst benötigt. Diese Relationen müssen enthalten, wie ein Verfahren heißt, für welches kontinuierliche Attribut welcher Meßwertrelation es anwendbar ist und welche Basisattribute variabel sind. Die Anzahl der variablen Basisattribute ist wiederum unbegrenzt. Im derzeitigen Prototypen wird das folgende

Schema (hier in der DDL-Syntax von INGRES-SQL) verwendet, das allen oben genannten Anforderungen genügt:

```
/* Erzeugen der Relation, die die Messwertrelationen aufnimmt:                */
CREATE TABLE messrelst (  messrel vchar(24) NOT NULL NOT DEFAULT,
                          messatt vchar(24) NOT NULL NOT DEFAULT)
        WITH JOURNALING;
CREATE UNIQUE INDEX messr_idx ON messrelst (messrel, messatt);

/* Erzeugen der Relation, die die Basisattribute zu den kontinuierlichen      */
/* Attributen aufnimmt:                                                       */
CREATE TABLE basis ( messrel vchar(24)  NOT NULL NOT DEFAULT,
                     basatt  vchar(24)  NOT NULL NOT DEFAULT,
                     basrel  vchar(24),
                     verweis vchar(1)   NOT NULL)
        WITH JOURNALING;
CREATE UNIQUE INDEX bas_idx ON basis (messrel, basatt);

/* Erzeugen der Relation, die die Namen der Interpolationsverfahren enthaelt  */
CREATE TABLE interpol (   verf     vchar(20) NOT NULL NOT DEFAULT,
                          messrel  vchar(24) NOT NULL NOT DEFAULT)
        WITH JOURNALING;
CREATE UNIQUE INDEX intp_idx ON interpol (verf);

/* Erzeugen, der Relation, die die variablen Basisattribute zu den            */
/* Interpolationsverfahren enthaelt:                                          */
CREATE TABLE interpolatt ( verf     vchar(20) NOT NULL NOT DEFAULT,
                           messatt  vchar(24) NOT NULL NOT DEFAULT,
                           varbasatt vchar(24) NOT NULL NOT DEFAULT,
                           basrel   vchar(24),
                           verweis  vchar(1)  NOT NULL)
        WITH JOURNALING;
CREATE UNIQUE INDEX intat_idx ON interpolatt (verf, messatt, varbasatt);
```

Das Attribut verweis enthält jeweils, ob die Basisattribute direkt in der Meßwertrelation gespeichert sind oder als Verweis (Fremdschlüssel) auf eine andere Relation. In späteren Versionen sollte es noch weitere Relationen geben, die Erklärungen und Kommentare zu den Interpolationsverfahren enthalten, die bei Bedarf dem Benutzer ausgegeben werden können.

Gibt der Benutzer eine neue Interpolationsfunktion dem System bekannt, werden die zusätzlichen Systemrelationen vom CSQL-System ergänzt. Es wird auch überprüft, ob alle Einschränkungen bezüglich der Namen der Funktionen eingehalten werden. Wie bei anderen Systemrelationen auch, können sie vom DBA auch direkt verändert werden. In diesem Fall trägt der Benutzer das Risiko falscher oder inkonsistenter Eintragungen.

7. Die Schnittstelle zwischen Interpolationsmodul und DBS

Das CSQL-System bietet einen geeigneten Rahmen für die Einbettung von Interpolation in ein DBS. Wie aber aus den Ausführungen im Kapitel 6, insbesondere 6.3.1 hervorgeht, sind auch bei der Implementierung der Interpolationsfunktionen noch einige anspruchsvolle Aufgaben zu lösen. Die in dieser Arbeit implementierten Beispiel-Interpolationsfunktionen wurden jeweils in Zusammenarbeit von Informatikern bzw. DB-Programmierern und Anwendern erstellt (siehe Kap. 10). Wenn es jedoch nicht möglich ist, dem Anwender einen DB-Spezialisten zur Seite zu stellen, müssen vom System Hilfen angeboten werden. In diesem Kapitel wird kurz erläutert, welche Hilfen möglich sind und wie sie aussehen könnten. Aus Zeitgründen wurden solche Hilfen jedoch bisher nicht implementiert.

Die Interpolationsroutinen greifen über Embedded SQL/C bzw. Dynamic SQL/C auf die Datenbank bzw. die in der Datenbank enthaltenen Werte zu (siehe Kap. 5), also über eine der Standardschnittstellen zur Datenbank. Da dies aber bei den meisten Interpolationsroutinen in ähnlicher Weise geschieht, bietet sich hier eine Möglichkeit, den Anwender geeignet zu unterstützen. Da geometrische Operationen u.ä. in SQL direkt nicht vorhanden sind und sie ohnehin z.B. mit Hilfe von C-Funktionen implementiert werden müssen, könnte man dem Anwendungsprogrammierer durch Bereitstellen passender parametrisierter Funktionen das Verwenden von SQL sogar ganz ersparen.

Auch bei einem weiteren Problem kann dieses Vorgehen hilfreich sein: CSQL mit eingebetteter Interpolation löst z.B. noch nicht das Problem, wie man die dichtesten (und weitere umgebende) Punkte zu einem geographischen Objekt findet. Aber es versteckt dieses Problem vor dem Endbenutzer und bietet ihm eine einfach zu handhabende, klare Syntax. Das Problem wird auf die Implementierung der Interpolationsfunktionen verlagert, wo es zentral effizienter gelöst werden kann. Hier könnten dem Anwendungsprogrammierer vorgefertigte Verfahren angeboten werden.

7.1 Standard-Zugriffe auf die DB

Es ist absehbar, daß man bei Interpolation und verwandten Operationen immer wieder die gleichen Suchfunktionen einsetzt, wie z.B. das Finden der dichtesten Punkte zu einem anderen oder das Finden aller Objekte in einem umschreibenden Rechteck. Die Funktionen unterscheiden sich nur in den Namen der Relationen und ihrer Attribute, auf denen sie arbeiten. Deshalb könnten diese Operationen parametrisiert implementiert werden und dem Anwender als Programmbibliothek zur Verfügung gestellt werden. Das würde dem Programmierer sowohl die Arbeit mit SQL als auch mit dem (z.T. etwas umständlich zu handhabenden) ESQL-Präprozessor ersparen. Außerdem könnte die Dokumentation dieser Programmbibliothek noch Hinweise enthalten der Art “Diese Operation wird geeignet durch einen ISAM-Index auf dem Attribut a1 der Relation R1 unterstützt”. Mit Dynamischem SQL [INGR91b] lassen sich solche parametrisierten Operationen als Prozeduren relativ einfach implementieren. Eine andere Methode zur wirkungsvollen Unterstützung einfacherer Operationen wäre eine Bibliothek vordefinierter SQL-Anfragen oder SQL-Prozeduren.

Wichtige Operationen, die eine Programm- oder Prozedurbibliothek unbedingt enthalten sollte, sind die folgenden (Auswahl):

- Finden des dichtesten Punkts zu einem Punkt x (1D / 2D / 3D)
- Finden der n dichtesten Punkte zu einem Punkt x (1D / 2D / 3D)
- Finden aller Punkte innerhalb eines bestimmten Radius um einen Punkt x (2D / 3D)
- Finden aller Punkte innerhalb eines gegebenen Rechtecks / Quaders
- Finden aller Werte innerhalb eines geschlossenen Polygonzugs

Zusätzlich sollten auch noch folgende Operationen angeboten werden, die zwar nicht direkt auf die DB zugreifen müssen, häufig aber sofort im Anschluß auf DB-Zugriffe ausgeführt werden (und in Anfragen eingebettet werden sollten):

- Abstand zwischen zwei Punkten (1D / 2D / 3D)
- Länge eines Polygonzugs
- Kleinstes umschreibendes Rechteck (Quader) um einen Polygonzug / ein Objekt
- Gemeinsamer Teil (Schnitt) zweier Polygonzüge / Flächen

Wenn man erreichen will, daß der Anwendungsprogrammierer ganz auf die Verwendung von SQL verzichten kann, müssen evtl. noch einige einfache Lese-Operationen angeboten werden, ähnlich der früher verwendeten CALL-Schnittstellen zu DBS. Dies ist zwar auf den ersten Blick ein Rückschritt, kann aber dem Anwender einigen Aufwand ersparen. Er muß z.B. nicht wegen einer einzigen Lese-Operation ein SELECT FROM (WHERE)-Statement in seinem Programm verwenden und so den ganzen mit der Verwendung von Embedded SQL/C verbundenen Aufwand treiben.

7.2 Auswirkungen auf die Leistung des DBS

In einem weiteren Schritt könnten die in Abschnitt 7.1 beschriebenen standardisierten (Prozedur-) Zugriffe auf die DB dann ausgewertet werden, um Hinweise zu finden, wie man das Antwortzeitverhalten bei Anfragen, die geometrische Operationen enthalten, verbessern kann. Denn ohne geeignete Speicherstrukturen und Zugriffspfade wird die Performance solcher Anfragen bei sehr großen Datenmengen eher schlecht sein und damit die Akzeptanz der eingebetteten Interpolation beim Endbenutzer insgesamt nur begrenzt.

Bei eindimensionalen Basen für Messungen wie etwa Zeit, Temperatur oder Druck allein, ist dies noch kein Problem. Hier reichen die in konventionellen DBS angebotenen Speicherstrukturen und Zugriffspfade aus. Sie müssen aber unbedingt eingerichtet und bei Bedarf auch gepflegt werden. Anders sieht es aus für mehrdimensionale Meßbasen, insbesondere räumliche Daten, aber auch andere mehrdimensionale Meßbasen (z.B. die Temperatur und seine Dicke bestimmen die Festigkeit eines Stahlblechs). Hier gibt es in vielen konventionellen DBS noch immer keine geeignete Unterstützung. Prinzipiell ist hier ein mehrdimensionaler Index immer noch viel besser als gar keine Unterstützung. Hilfreich kann auch ein Clustering über mehrere Relationen sein, z.B. das Abspeichern von Punkten auf einer Fläche und den daraus gebildeten Polygonzügen auf einer Speicherseite. Dies wird von einigen konventionellen DBS [ORAC86] schon seit längerer Zeit angeboten.

Für die Unterstützung räumlicher (mehrdimensionaler) Operationen gibt es in der Forschung schon vielversprechende Ansätze wie etwa räumliche Indexe (siehe [Oren86]) oder neuere baumartige Datenstrukturen für schnellen Zugriff auf geographische Daten wie sie u.a. in [Guen90], in [SeKr90] mit dem Buddy-Tree, in [FrBa90] mit dem Fieldtree und in [Same90] mit dem Quadtree vorgestellt werden. Verschiedene Strukturen und Zugriffspfade für geometrische, also auch räumliche Daten werden in [Guen88] verglichen und bewertet. In letzter Zeit befaßt sich die Forschung speziell mit der Auswertung räumlicher Operationen auf solchen Datenstrukturen. [RoSV95] etwa befaßt sich mit der Auswertung von Anfragen wie das Finden der k dichtesten Nachbar-Objekte zu einem Punkt im Raum in R-Tree-Datenstrukturen. In [PTSE95] werden räumliche Beziehungen auf kleinsten umschließenden Rechtecken in räumlichen Datenstrukturen wie R^+-Trees und R^*-Trees behandelt.

Für die Optimierung der Anfragen auf räumlichen Daten wäre es am besten, wenn es zu jeder räumlichen Operation (Prozedur im der in Kap. 7.1 erwähnten Bibliothek) eine passende Speicherstruktur und einen geeigneten Zugriffspfad in der DB geben würde, so daß für (häufig genutzte) Anwendungen diese mit der Einbindung der Interpolationsfunktion angelegt werden könnten.

7.3 Implementierung der Standardzugriffe für Interpolationsfunktionen

Im INGRES DBS (und einigen anderen kommerziellen DBS auch) gibt es zwei Möglichkeiten, parametrisierte Zugriffe auf die DB bereitzustellen, Datenbankprozeduren und Dynamisches SQL. Datenbankprozeduren (siehe auch Kap. 4.4.1) haben jedoch den Nachteil, daß sie immer einer DB fest zugeordnet werden müssen. D.h., möchte ein Programmierer sie in einer anderen DB verwenden, muß er sie zunächst installieren.

Datenbankprozeduren weisen bei der Programmierung von geometrischen Operationen jedoch noch weit größere Nachteile auf: Der Programmierer muß sich auf bestimmte Kontrollstrukturen beschränken und kann nicht beliebig Code in einer höheren Programmiersprache einfügen. Zudem konnten in den zur Projektlaufzeit zur Verfügung stehenden Versionen von Datenbankprozeduren Relationen- und Attributnamen noch nicht parametrisiert werden.

Die Erstellung einer Programmbibliothek mit Hilfe von Dynamischem SQL ist zwar wesentlich aufwendiger, weil der Speicherplatz für die Ergebnisse in jeder Funktion vom Programmierer selbst über die sogenannte *SQL Dynamic Area* (SQLDA) allokiert und verwaltet werden muß, dafür stehen ihm aber alle Möglichkeiten der höheren Programmiersprache und der Parametrisierung offen. Wenn jede Operation sich eigenen Speicherplatz reserviert, können alle Funktionen unabhängig voneinander aufgerufen und kombiniert werden.

Das beliebige Vermischen von SQL-Anfragen und mit Dynamic SQL implementierten geometrischen Operationen ist einfach für den Anwendungsprogrammierer möglich, wenn das Öffnen der Datenbank in einer eigenen Operation vorab geschieht. (Das Vermischen von SQL-Anfragen und Datenbank-Prozeduren ist sowieso immer möglich.)

Wenn Interpolationsfunktionen für das CSQL-System von Anwendungsprogrammierern (ohne Hilfe von DB-Programmierern) in größerem Umfang implementiert werden sollen, muß es eine Programmbibliothek mit Funktionen für geometrische Operationen bei DB-Zugriffen geben. Nur so kann erreicht werden, daß die Programmierung der Interpolationsfunktionen schnell geht, und daß die CSQL-Anfragen ein akzeptables Antwortzeitverhalten aufweisen.

8. Verarbeitung von CSQL

Im diesem Kapitel wird zuerst beschrieben, nach welchem Prinzip CSQL auf INGRES aufgesetzt ist, und es wird ein Überblick über das Gesamtsystem und die Komponenten, aus denen es besteht, gegeben. Dann werden die verschiedenen Komponenten einzeln erläutert.

8.1 CSQL im DBS INGRES

Das CSQL-System muß folgendes leisten: Es muß CSQL-Anfragen als solche erkennen, sie auf syntaktische und semantische Korrektheit überprüfen und bearbeiten. Die Bearbeitung erfolgt mit Hilfe der Interpolationsfunktionen und INGRES/SQL. Das CSQL-System muß die Namen der Interpolationsfunktionen, die der Benutzer verwendet, kennen und auch über deren Code verfügen. Es muß neue Funktionen kennenlernen und integrieren können, ihre Voraussetzungen erlernen und sie aufrufen können.

Von “außen”, für den Endbenutzer soll CSQL genauso aussehen wie INGRES/SQL, nur mit erweitertem Funktionsumfang. Dazu wurden zunächst die INGRES/ISQL-Benutzerschnittstelle und die INGRES/SQL-Schnittstelle nachgebildet [Bosc90]. Auf die Nachbildung der INGRES/ESQL/C-Schnittstelle wurde wegen des hohen Aufwands (s.u.) zunächst verzichtet. (Hinweise für eine Nachbildung der INGRES/ESQL/C-Schnittstelle sind in [Bosc90, Riet90] zu finden.)

In den CSQL-Benutzerschnittstellen werden die eingegebenen CSQL- und SQL-Anweisungen zuerst analysiert und - wenn sie syntaktisch korrekt sind - aufgeteilt in unterschiedlich zu bearbeitende Gruppen:

- CSQL-DDL-Anweisungen werden direkt verarbeitet. Sie werden z.B. aufgeteilt in Einfüge-Anweisungen in die zusätzlichen CSQL-Systemrelationen und “normale” SQL-DDL-Anweisungen.

- SQL-DDL-Anweisungen werden dahingehend überprüft, ob sie Auswirkungen auf die Interpolation haben könnten, und entsprechende Anweisungen werden zusätzlich gegeben. Wird

z.B. eine Meßrelation gelöscht, müssen alle auf ihr definierten Interpolationen verboten werden.

- Reine SQL-DML-Anweisungen und reine SQL-Anfragen werden direkt an das INGRES-System weitergereicht.

- CSQL-Anfragen werden vorverarbeitet, d.h. die Anfrage wird in einen sog. Anfragebaum (s.u.) umgeformt, Interpolation wird aufgerufen und die Ergebnisse werden in temporäre Relationen geschrieben. Dann werden entsprechend modifizierte Anfragen an das INGRES-System weitergegeben.

Die Vorverarbeitung der CSQL-Anweisungen ist der aufwendigste Teil im Gesamtsystem, weil hier auch eine *Anfrageoptimierung* stattfinden muß. Interpolation ist aufwendig, denn sie beinhaltet einen Prozeduraufruf mit mehreren DB-Zugriffen und Berechnungen. Insbesondere die Anzahl der Datenbank-Zugriffe ist fast immer höher als bei einer einfachen Selektion. Deshalb muß versucht werden, die Anzahl der Interpolationsaufrufe soweit wie möglich zu reduzieren (siehe Kap. 9).

Ein wesentlicher Teil der interpolationsbezogenen Anfrageoptimierung ist die Entschachtelung von geschachtelten Anfragen. Zum Teil ist diese Entschachtelung sogar unbedingt notwendig, nämlich dann, wenn die Interpolation in einer inneren SELECT-Klausel stattfindet. Wird hier nicht entschachtelt, muß die Kontrolle zwischen Interpolation und DBS wechseln, das ist aufgrund der Präprozessor-Implementierung (s.u.) aber nicht möglich.

Die Anfrage, die der SQL-Anfrageauswertung schließlich übergeben wird, unterscheidet sich nicht nur durch die temporäre(n) Relation(en) von der CSQL-Anfrage. Sie kann auch vereinfacht worden sein, weil z.B. zusätzliche Attribute hinausprojiziert werden mußten und auch Restriktionen auf diesen Attributen dann wegfallen. Auch kann eine Entschachtelung vorgenommen worden sein (siehe Kap. 8.3).

Am Schluß muß evtl. noch das Ergebnis aufbereitet werden. Der gesamte Ablauf ist in Bild 8.1 dargestellt.

Da vom verwendeten kommerziellen DBS INGRES kein Quellcode verfügbar war, wurde das gesamte CSQL-System als Präprozessor realisiert. Dadurch wird aber auch eine eventuelle Portierung auf andere relationale DBS vereinfacht.

Für den Präprozessor-Ansatz wurde zunächst die Benutzerschnittstelle (Ein- und Ausgabe) mit dem INGRES/Forms-System [INGR91] nachgebildet; das ist relativ einfach möglich, da INGRES die in den eigenen Schnittstellen verwendeten Bildschirmmasken auch überwiegend mit diesem System erzeugt hat. Der Parser für die lexikalische und syntaktische Analyse wurde mit den UNIX-Werkzeugen LEX und YACC [SUN88, SUN89] erstellt. Das gesamte CSQL-System besteht aus C-Programmen, in die sowohl die INGRES-Masken (Embedded Forms System) als

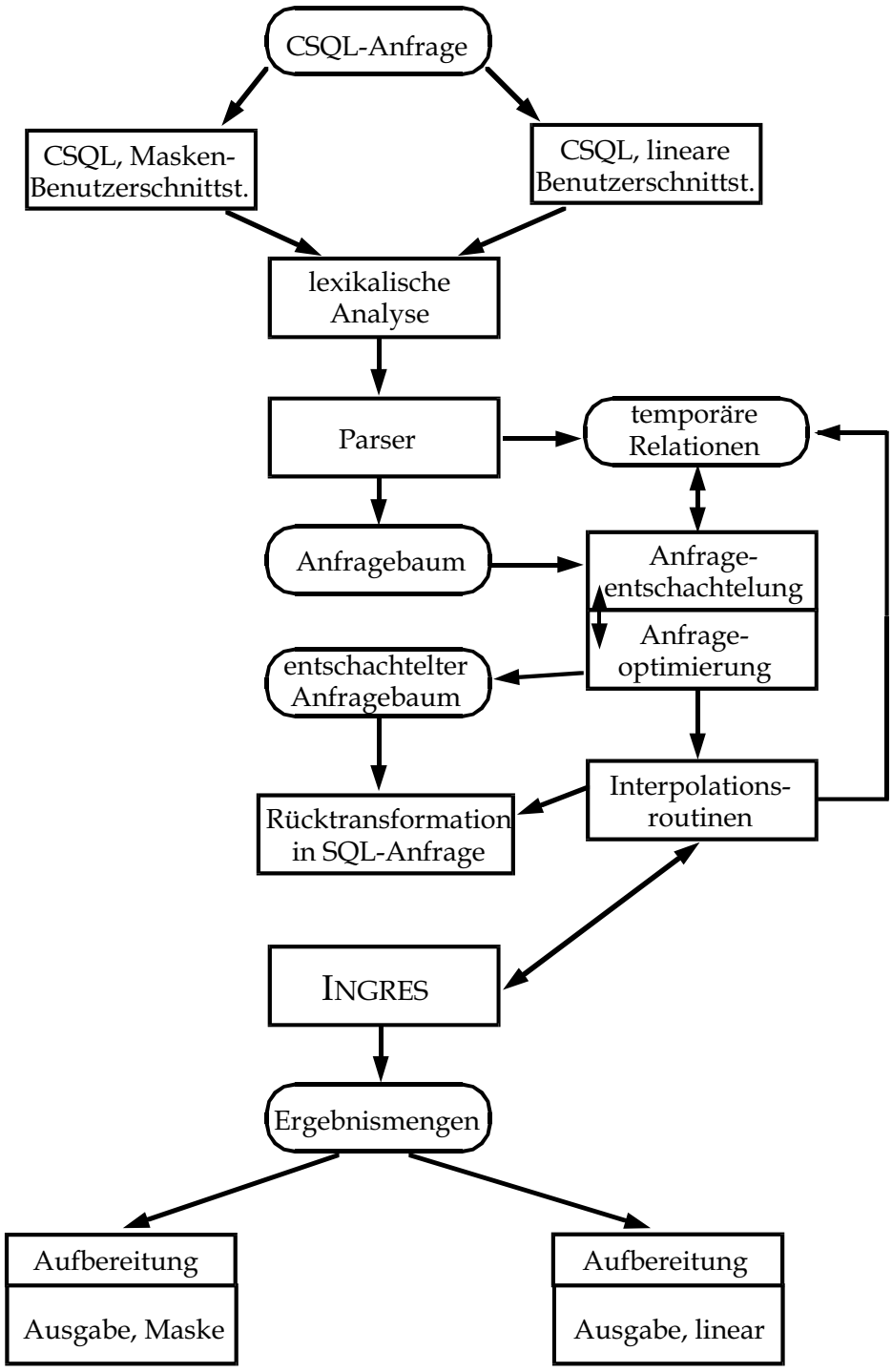


Bild 8.1: Übersicht über das CSQL-System

auch die gesamte Steuerung der Anfragebearbeitung, die Optimierung und Auswertung über Embedded SQL bzw. Dynamic SQL eingebunden sind. Der größte Teil des CSQL-Systems wurde im Rahmen von studentischen Arbeiten implementiert [Bosc90, Riet90, Kaja90, Jahk91]. Die Interpolationsroutinen selbst sind nicht fester Bestandteil des CSQL-Systems sondern auswechselbar. Die bisher implementierten Routinen werden innerhalb von Kapitel 10 beschrieben.

8.2 CSQL-Benutzerschnittstellen

Relationale DBS bieten dem Benutzer i.a. mehrere Schnittstellen zu den Datenbanken an. Für SQL bietet INGRES dem Benutzer

- eine zeilenorientierte Schnittstelle (INGRES/SQL), die sowohl interaktiv als auch aus Dateien heraus in Hintergrund-Programmen oder Batch-Jobs genutzt werden kann;
- eine maskenorientierte Schnittstelle (INGRES/ISQL), die nur interaktiv am Bildschirm genutzt werden kann;
- eine Schnittstelle zu höheren Programmiersprachen, die es erlaubt, SQL-Anweisungen direkt (Embedded SQL) oder parametrisiert bzw. als Zeichenkette (Dynamic SQL) aus Programmen heraus zu geben.

Zusätzlich gibt es in INGRES noch alle diese Schnittstellen auch für die Anfragesprache QUEL. Außerdem gibt es Masken als Schnittstellen zum DBS, und zwar sowohl vordefinierte als auch selbstdefinierte Masken. Solche oder ähnliche Schnittstellen bieten heute fast alle relationalen DBS an.

Im CSQL-System mußten nun die CSQL-Schnittstellen den SQL-Schnittstellen nachempfunden und reimplementiert werden. Zunächst wurden nur die SQL- [Riet90] und ISQL-Schnittstelle [Bosc90] implementiert, da Embedded / Dynamic SQL bereits selbst über einen SQL-Präprozessor realisiert wird und also ein “Prä-Präprozessor” hätte geschrieben werden müssen. Größere Probleme träten auch bei der Implementierung eines Dynamic CSQL auf: Hier müßte der Inhalt von Programmvariablen (Zeichenketten) analysiert, bearbeitet und evtl. verändert werden. Das ist kompliziert, weil diese Variablen erst zur Laufzeit belegt werden. Praktisch müßte zuerst der ESQL-Präprozessor nachimplementiert werden, der die “EXEC SQL ...”-Aufrufe in Funktionsaufrufe umwandelt. Dieser “ECSQL”-Präprozessor müßte zunächst einmal andere, neue Funktionen aufrufen. Diese müßten dann die Funktionen, mit denen INGRES das Dynamic SQL bearbeitet, nach der CSQL-Vorverarbeitung aufrufen. Das ist praktisch nicht möglich, weil INGRES die Schnittstellen dieser Funktionen nicht offenlegt und sich auch vorbehält, Funktionsnamen u.ä. mit jeder neuen Version wieder zu verändern. Hier ist der Präprozessor-Ansatz also nicht geeignet. Die Modifikationen würden besser auf einer tieferen Ebene des DBS ansetzen.

Beide CSQL-Schnittstellen lesen die CSQL- (und SQL-) Anweisungen zeilenweise ein und reichen sie, sobald ein Semikolon die Eingabe einer vollständigen Anweisung anzeigt, an den Parser weiter. Die interaktive CSQL-Schnittstelle [Bosc90] besteht aus einer INGRES-Maske, die eine Tabelle mit einem Attribut, einer Bildschirmzeile, enthält. Somit zeigt sie auch alle dem ISQL und allen INGRES-Masken eigenen Eigenschaften. Über in der Menü-Zeile verfügbare Funktionen konnten die im ISQL-Menü vorhandenen Funktionen identisch (bis auf geringfügige Verbesserungen) nachgebildet werden. Die wichtigsten Funktionen hier sind: eine Anfrage zur Verarbeitung abschicken, einen Editor aufrufen, vorwärts und rückwärts blättern, Dateien einlesen und ausschreiben, den Bildschirminhalt löschen und das System beenden. Fehlermeldungen werden auf die gleiche Art wie in ISQL angezeigt. Nachteilig ist bisher noch, daß die

verwendeten Masken bei der Installation des CSQL-Systems in jede einzelne DB geladen werden müssen.

Die zeilenorientierte CSQL-Schnittstelle entstand innerhalb der Arbeit von [Riet90] praktisch nebenbei, um die Arbeiten [Riet90, Kaja90] vor der Fertigstellung der interaktiven CSQL-Schnittstelle ausführlich testen zu können.

Im Ausgabeteil der CSQL-Schnittstellen gibt es zu den INGRES Schnittstellen einen Unterschied, den u.U. auch der Endbenutzer bemerkt: Im INGRES DBS wird die Anfrage nur soweit ausgewertet, daß die erste Bildschirmseite mit Ergebnistupeln beschrieben werden kann. Erst, wenn der Benutzer im Ergebnis ‘weiterblättern’ will, wird weiter ausgewertet. Das hat den Vorteil, daß der Benutzer die ersten Ergebnistupel schneller sieht, aber auch den Nachteil, daß nach dem Blättern evtl. wieder gewartet werden muß. Die CSQL-Schnittstellen mußten jedoch so implementiert werden, daß das Ergebnis erst nach vollständiger Auswertung der Anfrage gezeigt werden kann, auch bei reinen SQL-Anfragen. Andernfalls müßte die Kontrolle wieder zwischen CSQL und INGRES wechseln.

8.3 Analyse der CSQL-Anweisungen

Bei der Analyse (lexikalisch, syntaktisch, semantisch) der CSQL-Anweisungen, bzw. der SQL-Anweisungen mit und ohne CSQL-Erweiterungen sollte ein Parser implementiert werden, der die folgenden Anforderungen möglichst gut erfüllt:

- Als Ergebnis sollte eine einfache, gut weiter zu verarbeitende Datenstruktur geliefert werden.
- Der Programmcode mußte übersichtlich aufgebaut und einheitlich strukturiert sein, damit die Sprache und damit der Parser später einfach erweitert werden können.
- Der Programmieraufwand sollte so gering wie möglich sein.
- Das System sollte so portabel wie möglich werden.

Für die Sprache SQL war bereits eine übersichtliche, korrekte kontextfreie Grammatik allgemein verfügbar [Cain89]. Für die CSQL-Erweiterungen[1] ließ sich leicht eine übersichtliche, korrekte Erweiterung dieser Grammatik aufstellen [siehe Riet90, Anhang B]. Es bot sich also an, die Analyse mit Hilfe von Parsergeneratoren durchzuführen, um den Programmieraufwand gering zu halten. Ein weiterer positiver Effekt dabei ist, daß ein Parsergenerator die Konfliktfreiheit also Korrektheit der eingegebenen Grammatik überprüft. Die Vor- und Nachteile von Parsergeneratoren sind in [Riet90] ausführlich diskutiert: Um auch die Portabilität des Gesamtsystems sicherzustellen, mußten für die Erstellung des Scanners und Parsers allgemein verfügbare Parsergeneratoren wie die Werkzeuge LEX (lexikalische Analyse) und YACC (Yet

1: Dies wurde beim Entwurf von CSQL beachtet.

Another Compiler Compiler) [SUN88, SUN89] verwendet werden. Mit LEX und YACC wird ein LR(1) Bottom-Up-Parser erzeugt.

Ein besonderer Vorteil eines mit YACC erzeugten Parsers ist, daß neben der üblichen Parser-Kellerstruktur, auf der die Grammatik-Elemente gespeichert und reduziert werden, parallel eine weitere Kellerstruktur, der sogenannte Werte-Keller (value stack) gehalten wird. Mit dieser zweiten Kellerstruktur ist es auf einfache Weise möglich, die Anfrage in eine Baumstruktur umzuwandeln, im folgenden *Anfragebaum* genannt, wie in [Riet90] gezeigt wird.

Alle Anweisungen, die über die CSQL-Schnittstellen eingegeben werden, werden zunächst an den Parser weitergereicht. Sobald sie aber einer bestimmten Gruppe von Anweisungen zugeordnet werden können, werden sie auf unterschiedliche Weise weiter verarbeitet:

- SQL-Änderungsanweisungen.
  Sobald festgestellt wurde, daß eine Anweisung eine Änderung beinhaltet, wird die Analyse abgebrochen und die Anweisung direkt an SQL übergeben. (Wie oben erwähnt, ist Interpolation in Unteranfragen von Änderungsanweisungen noch nicht zulässig.) Da bereits nach dem ersten Symbol einer Anweisung feststeht, ob es sich um eine Änderungsanweisung handelt (INSERT ..., UPDATE ..., DELETE ...), ist die Verzögerung, die hier durch den CSQL-Präprozessor entsteht, relativ gering.

- SQL-Anfragen (SELECT-Anweisungen).
  SELECT-Anweisungen werden immer vollständig analysiert. Sollte sich dabei herausstellen, daß sie keine Interpolation enthalten, werden sie unverändert an SQL weitergereicht; es sei denn, der Parser hat bereits grobe Fehler entdeckt und dem Benutzer gemeldet.

- CSQL-Anfragen.
  CSQL-Anfragen werden auch vollständig analysiert und auf syntaktische und semantische Korrektheit überprüft. Bei der Analyse wird parallel der Anfragebaum aufgebaut, auf dem später die gesamte Bearbeitung und Auswertung stattfindet. Die Kapitel 8.4 bis 8.7 und 9 befassen sich ausschließlich mit dieser Gruppe von Anweisungen.

- CREATE-Anweisungen.
  Sie werden vollständig analysiert und auf syntaktische Korrektheit überprüft. Und es wird festgestellt, ob sie für CSQL relevante Teile enthalten. Wenn ja, werden die notwendigen Aktionen vom Präprozessor eingeleitet. Auf jeden Fall wird die CREATE-Anweisung, evtl. bereinigt, an SQL weitergereicht.

- DROP-Anweisungen.
  Es wird geprüft, ob sie die Interpolation irgendwie betreffen. Wenn ja, werden vom Präprozessor entsprechende Aktionen ausgeführt. Anschließend wird die Anweisung von SQL bearbeitet.

- DECLARE METHOD-Anweisungen.
  Sie werden analysiert und anschließend vollständig vom Präprozessor bearbeitet. Das Einfügen der neuen Interpolationsmethode in die zusätzlichen CSQL-Systenrelationen geschieht dabei auch mit Hilfe von SQL-Anweisungen.

Die Bearbeitung der (CSQL-) Anweisungen ist in Bild 8.2 dargestellt. Das Element “weitere Verarbeitung” wird in den folgenden Unterkapiteln aufgeschlüsselt und beschrieben.

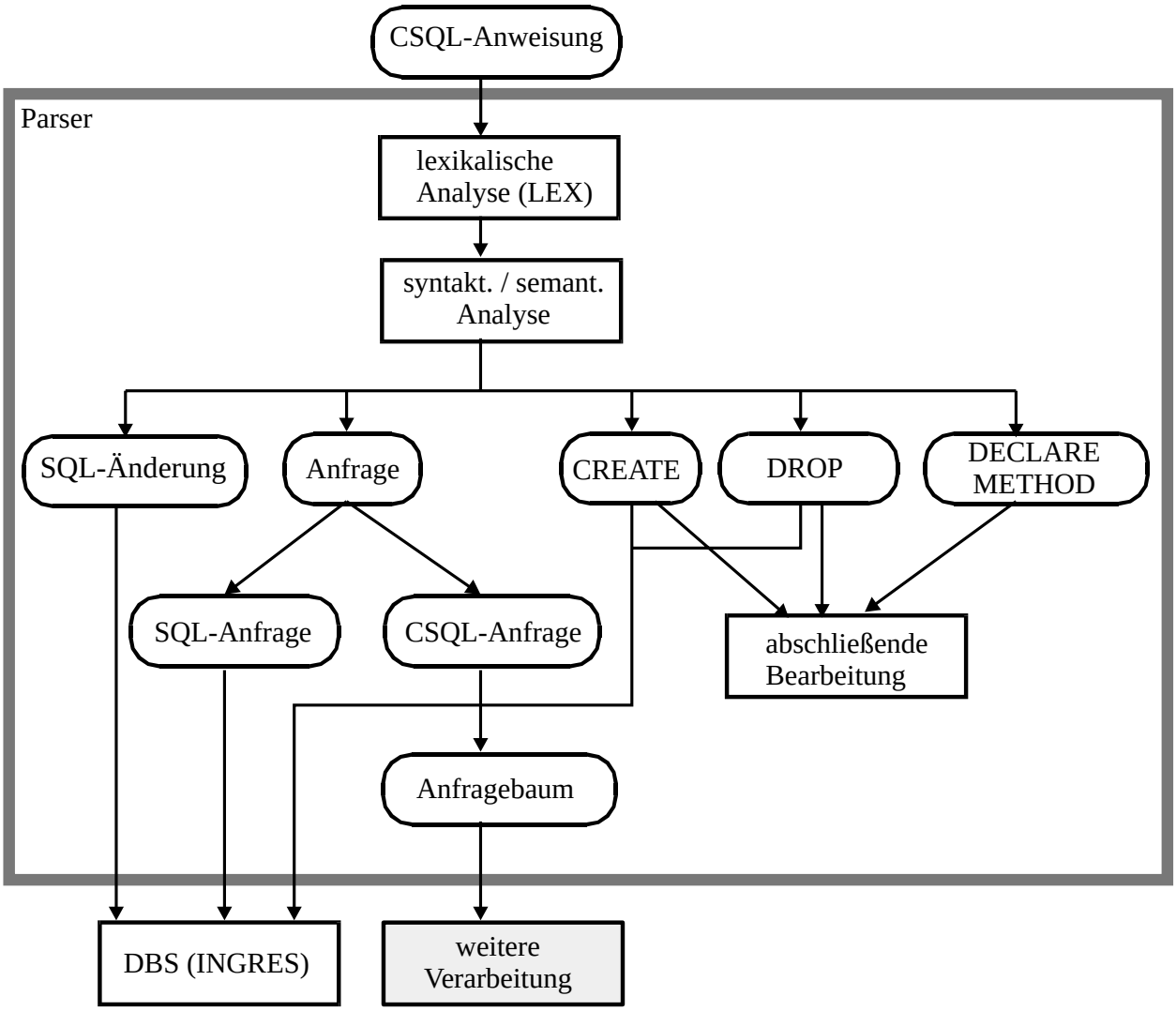


Bild 8.2: Bearbeitung von Anweisungen durch den CSQL-Parser

8.4 Ablaufsteuerung bei der Interpolation

Die gesamte weitere Bearbeitung der CSQL-Anweisungen, bevor die Anfrage dem SQL-System übergeben wird, erfolgt mit Hilfe des vom Parser aufgebauten Anfragebaums, der in diesem Kapitel zunächst kurz motiviert und erläutert werden soll.

Die Zwischendarstellung der CSQL-Anfragen in einer baumartigen, verzeigerten Datenstruktur wurde aus folgenden Gründen gewählt:

- In dieser Darstellung sind die verschiedenen Teile der Anfrage wie Interpolationsangaben oder WHERE-Bedingungsliste leicht zu finden und schnell zugreifbar.

- Bekannte Methoden zur Anfrageoptimierung arbeiten häufig mit einer baumartigen Darstellung der Anfrage [Haer78, Kim82, JaKo83, GaWo87, Hard87]. Sie können dann direkt auf CSQL übertragen werden.

- Die Bildung der SQL-Anfrage nach erfolgter Interpolation ist bei dieser Darstellung besonders einfach. Einige von CSQL eingeführte Optimierungen können erhalten bleiben (siehe Kap. 8.6).

Der Anfragebaum jeder SELECT-Anweisung - davon kann es mehrere, durch UNION verbundene geben - teilt sich in die drei Zweige Projektionsliste, Relationenliste (FROM) und die Liste der Einschränkungen (WHERE). Die Relationenliste enthält auch die Interpolationsanweisungen. Die Liste der Einschränkungen kann geschachtelte SELECT-Anweisungen enthalten (siehe Bild 8.3). Die in der Anfrage vorkommenden Namen und Konstanten bilden die Blätter des Baumes. Als konkretes Beispiel wird hier die folgende Anfrage in einen Anfragebaum umgewandelt, siehe Bild 8.4 (nach [Rieth90, S.29]):

```
SELECT x_koord, y_koord, gdw_stand
FROM   gwmessung BY METHOD gdw_staende
                         ENTRY (x_koord = 50.0, y_koord = 225.0)
WHERE  zeitpunkt = '29/03/88'
```

In der Anfrageauswertung werden als erstes geschachtelte Anfragen entschachtelt. Dieser Vorgang wird in Abschnitt 8.5 und [Jahk91] genau beschrieben. So können Interpolationen auf verschiedenen 'Ebenen' der Anfrage vermieden werden. Die Ablaufkontrolle braucht nicht zwischen Interpolation und SQL-System zu wechseln. Dies ist eine für einen Präprozessor notwendige Voraussetzung. Deshalb kann das CSQL-System auch Anfragen, die Interpolationsanweisungen in einer inneren SELECT-Anweisung enthalten und nicht entschachtelt werden können, bisher nicht bearbeiten [Jahk91].

Anschließend werden die Bedingungen in der WHERE-Klausel (nach der Entschachtelung sollte nur noch eine übrig sein) in die konjunktive Normalform gebracht (siehe [Kaja90]). Dieser

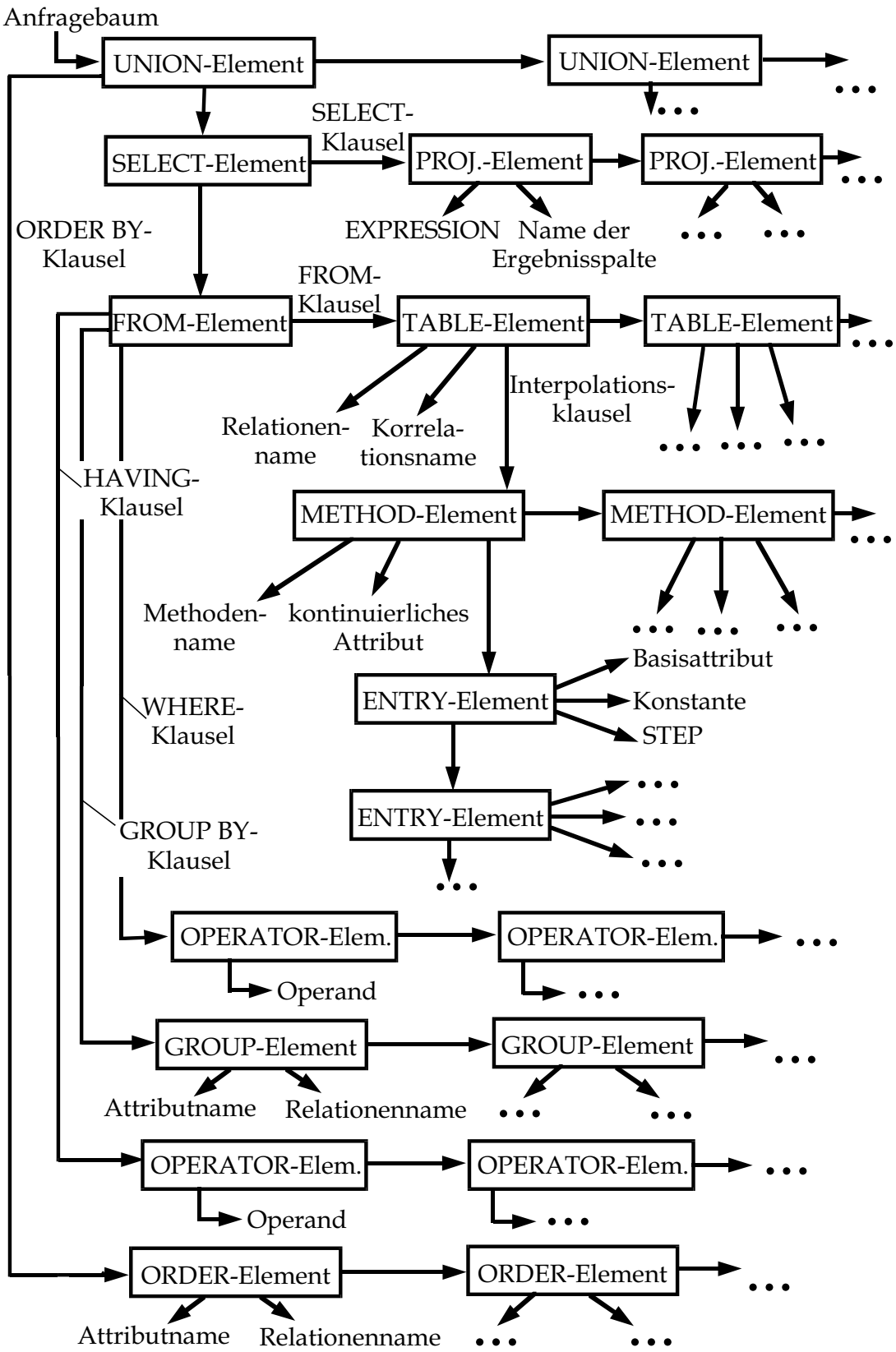


Bild 8.3: Allgemeine Struktur eines Anfragebaums

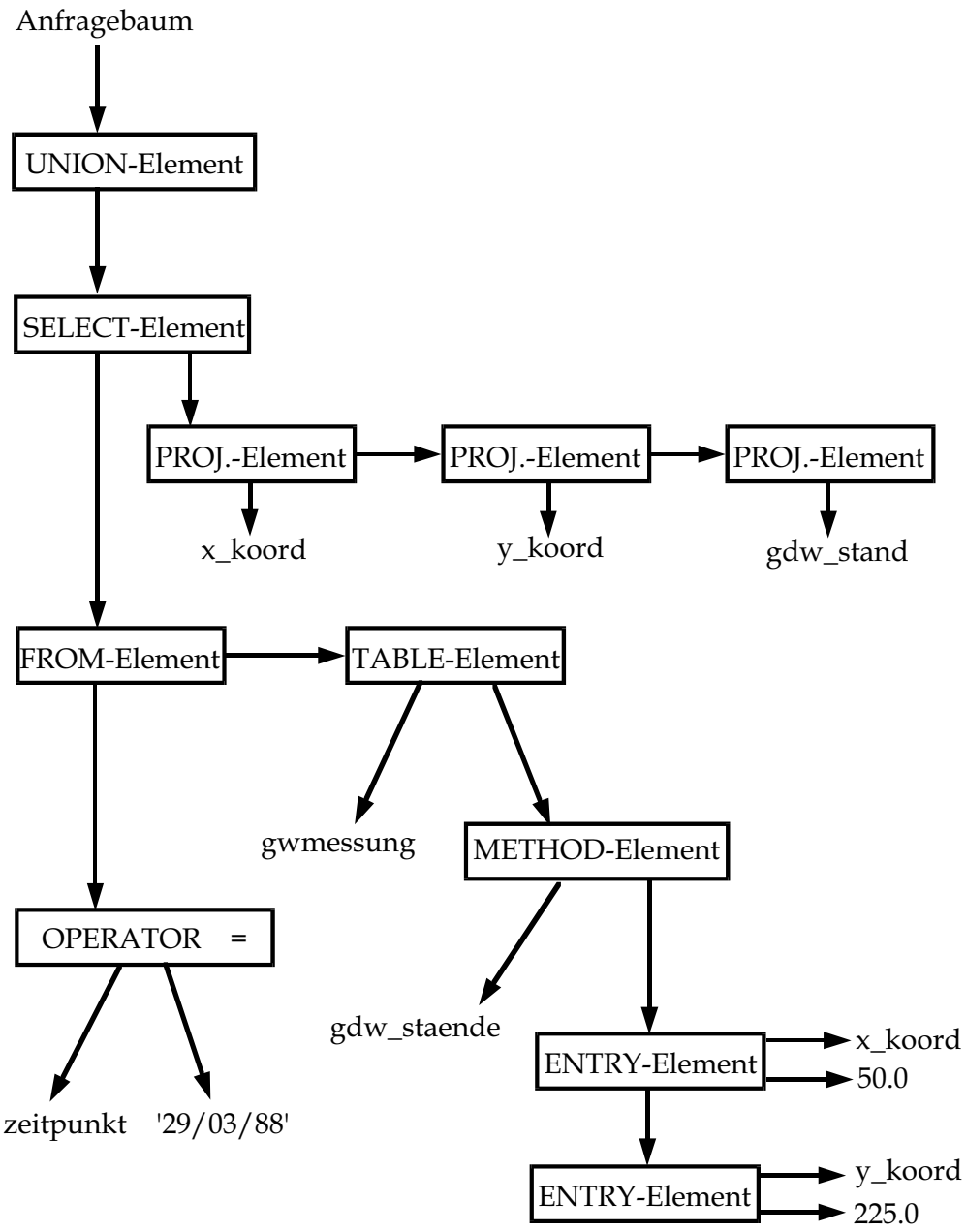


Bild 8.4: Beispiel eines Anfragebaums

Schritt ist notwendig, um die Restriktionen, die vor der Interpolation ausgewertet werden müssen, abgrenzen zu können und um später die bereits ausgewertete Qualifikationen einfach aus der WHERE-Klausel entfernen zu können.

Die vorgezogene Auswertung von Restriktionen geschieht dadurch, daß auf der Meßrelation eine Sicht definiert wird, die genau diesen Bedingungen genügt. Der Name dieser Sicht wird der Interpolationsroutine dann anstatt des Namens der Meßrelation übergeben.

Der Name dieser Sicht wird, wie auch der Name der Ergebnisrelation für die Interpolationsroutine, unter Zuhilfenahme der aktuellen Prozeßnummer gebildet, so daß er, wie DB-Objekte sonst auch, eindeutig für diesen DB-Knoten (Rechner) ist. Dadurch bleibt die Mehrbenutzerfähigkeit des DBS im CSQL-System erhalten, ohne daß sich verschiedene CSQL-Benutzer, die sonst nur Leseoperationen ausführen, gegenseitig behindern.

Die Interpolationsroutinen führen immer nur die Interpolation eines Wertes bei einem Aufruf durch. Das Hochzählen der Basisattribute, im folgenden auch als Generierung der *STEP-Werte* bezeichnet, erfolgt im CSQL-System. Die Interpolationsroutinen erhalten mit einem Übergabestring den Namen der Relation oder der Sicht, auf der sie interpolieren sollen, den Namen der Ergebnisrelation, die Namen der variierenden und festen Basisattribute und des kontinuierlichen Attributs, sowie die aktuellen Basiswerte, für die sie interpolieren sollen. In einem zweiten Übergabeparameter bekommt die Interpolationsfunktion mitgeteilt, ob weitere Funktionsaufrufe folgen, damit sie in gewisser Weise optimieren kann, indem z.B. Zwischenergebnisse gespeichert werden. Der Rückgabewert der Funktion zeigt an, ob sie mit den in der DB vorhandenen Werten überhaupt eine Interpolation durchführen konnte. Diesen Wert kann die Steuerung verwenden, um endlose Generierung von STEP-Werten und Aufrufe der Interpolationsfunktion zu verhindern, wenn sonst keine Bereichseinschränkungen als Abbruchbedingungen für die Generierung von STEP-Werten gegeben wurden. Das Ergebnis der Interpolation schreibt die Routine selbst mit den entsprechenden Basisattributen in die Ergebnisrelation.

Das CSQL-System erkennt, wenn alle STEP-Werte innerhalb der gültigen Bedingungen bearbeitet wurden, oder wegen zu großer Lücken bei den Meßwerten in der DB keine weitere Interpolation mehr möglich ist. Eine SQL-Anfrage wird gebildet und zur weiteren Auswertung dem DBS übergeben.

Der gesamte Ablauf im CSQL-System vom Anfragebaum zur Ergebnismenge ist in Bild 8.5 dargestellt.

8.5 Entschachtelung und Optimierung von Interpolationsanfragen

Die Optimierung der CSQL-Anfrage auf dem Anfragebaum gliedert sich in folgende Schritte [Kaja90]:

(1) Zuerst werden geschachtelte Anfragen, also Anfragen mit geschachtelten WHERE- oder HAVING-Klauseln entschachtelt. Wenn Interpolation in diesen Unteranfragen (Subselects) stattfindet, muß entschachtelt werden, sonst kann die Interpolation nicht bearbeitet werden.
(2) Dann werden die Prädikate in den WHERE- und HAVING-Klauseln in eine konjunktive Normalform (KNF) gebracht.
(3) Selektionen werden vorab ausgewertet, soweit dies möglich ist, ohne die Interpolation zu verfälschen, um die Anzahl der Interpolationsschritte gering zu halten.

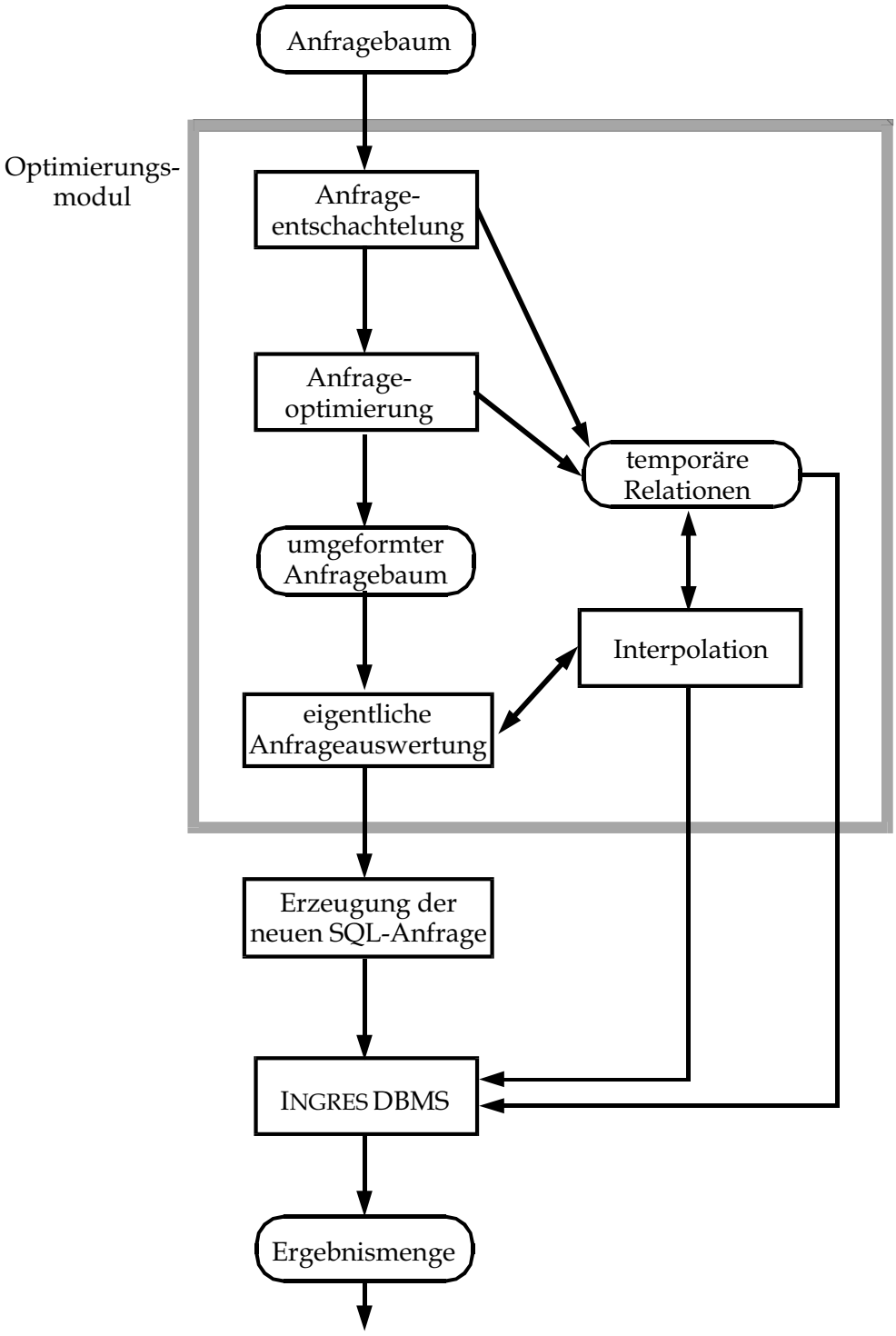


Bild 8.5: Ablaufsteuerung im CSQL-System

Die Entschachtelung wurde innerhalb einer Diplomarbeit [Fabr91] untersucht und implementiert. Dazu wurde auf die in [Kim82, GaWo87] beschriebene Klassifizierung von Schachtelungstypen und die dort ebenfalls skizzierten Algorithmen zur Entschachtelung zurückgegriffen.

Grob gesehen gibt es vier Typen von Schachtelungen bei SQL-Anfragen (die Benennung folgt hier [Kim82]):

1) Typ-A-Schachtelung.
   Der innere Anfrageblock kann ohne Berücksichtigung der übrigen Anfrage zu einem einzelnen festen Wert ausgewertet werden.

2) Typ-AG-Schachtelung.
   Der innere Anfrageblock kann ohne Berücksichtigung der übrigen Anfrage zu einer Menge von Werten ausgewertet werden. Für die Entschachtelung gibt es hier zwei Möglichkeiten. Entweder wird jedes Element der Ergebnismenge einzeln in einer Reihe von mit ODER verbundenen Qualifikationen in die Anfrage eingefügt. Oder es wird eine Sicht definiert, die der Bedingung der inneren Anfrage genügt, und diese Sicht wird über eine Verbundoperation in die Anfrage aufgenommen.

3) Typ-N- oder Typ-J-Schachtelung.
   Eine geschachtelte Abfrage mit dem Mengenoperator IN und einem Bezug zur äußeren Anfrage in der inneren Qualifikation (WHERE-Klausel) ist gleichwertig zu einer Anfrage mit Gleich-Verbund. Hierbei ist jedoch zu beachten, daß es Probleme mit Duplikaten geben kann. Um dies zu vermeiden, müssen an geeigneten Stellen Duplikateliminierungen eingeführt werden. Für den Mengenoperator NOT IN ist ebenfalls eine spezielle Behandlung erforderlich. Hier muß das sog. Anti-Join-Prädikat eingesetzt werden. Näheres dazu sowie Beispiele siehe [Kim82, Jahk91].

4) Typ-JA-Schachtelungen.
   Eine Typ-JA-Anfrage ist eine Anfrage, die in der inneren Anfrage eine Aggregation enthält und in der inneren Qualifikation einen oder mehrere Bezüge zur äußeren Relation. Die Entschachtelung ist hier aufwendig, weil die innere Anfrage zuerst über eine temporäre Relation, die eine Gruppierung enthält, aufgelöst werden muß. Dann wird ähnlich wie bei einer Typ-J-Schachtelung verfahren. In [GaWo87] wird gezeigt, daß bei dieser Typ-JA-Entschachtelung in einigen Fällen Probleme auftreten. In [GaWo87] und [Jahk91] wird ausführlich erklärt, wie diese Probleme vermieden werden können. In einigen Fällen kann nur mit Hilfe eines sog. “äußeren Verbundes” (outer join) eine äquivalente ungeschachtelte Abfrage gefunden werden. Da INGRES den äußeren Verbund nicht unterstützt, können diese Anfragen im CSQL-System noch nicht bearbeitet werden.

Die im CSQL-System implementierten Entschachtelungsalgorithmen können bisher nur skalare Vergleichsoperatoren und die Mengenoperatoren IN bzw. NOT IN bearbeiten. Für die Prädikate EXISTS bzw. NOT EXISTS, ANY und ALL gibt es auch Entschachtelungsverfahren. Diese werden in [Jahk91] skizziert, ihre Implementierung wurde aber zugunsten einer schnelleren Fertigstellung einer lauffähigen Entschachtelungskomponente zurückgestellt.

Die Entschachtelungskomponente bestimmt zuerst auf dem Anfragebaum den Schachtelungstyp. Dann wird der entsprechende Entschachtelungsalgorithmus aufgerufen. Falls eine Anfrage mehrere Schachtelungen enthält, wird die Anfrage rekursiv, beginnend mit dem innersten Anfrageblock, entschachtelt. Auf dem entschachtelten Anfragebaum können dann weitere Optimierungen stattfinden.

Für die weitere Optimierung [Kaja90] werden die Selektionen in WHERE- und HAVING-Klauseln zunächst in die KNF umgeformt. Dann werden die Selektionen für die zusätzlichen Attribute der Meßwertrelation zusammengefaßt und ausgewertet. Die Selektionen der Basisattribute werden zusammengefaßt und soweit wie möglich ausgewertet zur Bestimmung der Interpolationsgrenzen (variierende Basisattribute) und zur Einschränkung der Anzahl der zu bearbeitenden nichtvariierenden Basisattribute. Dabei werden für jedes Basisattribut Minimum, Maximum und die dazugehörenden Operatoren ermittelt. Dann werden alle gültigen Kombinationen der nichtvariierenden Basisattribute bestimmt. Dann werden die gültigen variierenden Basisattributwerte für jedes dieser Tupel bestimmt (Generierung der STEP-Werte) und für jede Kombination wird die Interpolationsroutine aufgerufen. Diese schreibt die Ergebnisse in eine temporäre Relation.

Im Anfragebaum wird dann die Originalrelation durch die temporäre Ergebnisrelation ersetzt. Selektionen auf zusätzlichen Attributen sowie andere, vollständig ausgewertete Selektionen werden entfernt. Aus dem Anfragebaum kann nun wieder eine SQL-Anfrage generiert werden, deren weitere Auswertung durch das DBS erfolgen kann.

8.6 Generierung der SQL-Anfragen

Die Generierung der SQL-Anfragen erfolgt aus dem Anfragebaum umgekehrt zu dessen Erstellung. Die Interpolationsklauseln werden zuvor vollständig aus dem Anfragebaum entfernt. Schon ausgewertete Bedingungen, die nicht auf den Interpolationsergebnissen nochmals ausgewertet werden müssen, wurden zuvor aus dem Anfragebaum entfernt (s.o.). Das sind Bedingungen,

- die zusätzliche Attribute und
- Vergleiche fester Basisattribute mit Konstanten enthalten.

Evtl. durchgeführte Entschachtelungen und Vereinfachungen der Bedingungen in den WHERE- und HAVING-Klauseln bleiben natürlich erhalten. Die neu eingeführten Ergebnisrelationen erhalten als Korrelationsnamen die alten Relationennamen (oder deren Korrelationsnamen). So braucht nicht die gesamte Anfrage geändert zu werden.

Es wird nur dann eine SQL-Anfrage generiert und an das DBS weitergeleitet, wenn zuvor keine Fehler festgestellt werden konnten, andernfalls bekommt der Benutzer nur eine entsprechende Fehlermeldung.

8.7 Nachbehandlung

Das Ergebnis, das das DBS zurückliefert, ist inhaltlich korrekt und vollständig. Nur die äußere Form muß noch nachgebessert werden: Anstatt der temporären Relation, muß wieder der Name der Meßwertrelation eingesetzt werden. Anschließend wird das Ergebnis in einer dem INGRES-System nachempfundenen Art dem Benutzer auf dem Bildschirm präsentiert. Dabei kommt es zu dem in Abschnitt 8.2 beschriebenen veränderten Verhalten, daß die Präsentation der ersten Ergebnisseite evtl. etwas länger braucht, das Weiterblättern dafür jedoch schneller geht.

Wenn Fehler aufgetreten sind, werden diese vom CSQL-System ebenfalls ähnlich wie bei INGRES dem Benutzer mitgeteilt, und zwar unabhängig davon, ob diese Fehler von INGRES bei der Auswertung der SQL-Anfrage oder schon vorher im CSQL-System aufgetreten sind.

Wird CSQL vom Benutzer beendet, werden alle temporären Relationen und Sichten wieder aus dem DBS gelöscht.

8.8 Stand der Prototyp-Implementierung

Das CSQL-System ist bis auf die im folgenden aufgeführten Einschränkungen funktionsfähig und steht auf UNIX-Systemen mit dem DBS INGRES zur Verfügung. Folgende Einschränkungen bestehen bisher noch:

- Interpolation ist nur auf reinen Anfragen möglich. In Verbindung mit Bedingungen in Änderungsoperationen ist es bisher noch nicht zugelassen. Dazu müßten zwei Dinge geändert werden:
  - Für Änderungsoperationen mit Interpolation müßten auch ‘Anfragebäume’ aufgebaut werden.
  - Die Transaktionsverwaltung müßte strenger gehandhabt werden. (In diesem Fall müßte die gesamte Interpolation in Transaktionsklammern gesetzt werden, weil es vorkommen kann, daß Relationen, die geändert werden sollen, bereits bei der Interpolation angesehen werden, etwa um die Generierung von STEP-Werten zu begrenzen. In der damaligen Version von INGRES war so aber ein Mehrbenutzerbetrieb während der Interpolation praktisch nicht mehr möglich; deshalb wurden so grobe Transaktionsklammern nicht gesetzt.)

  Beides wurde zugunsten einer schnellen Prototyperstellung zurückgestellt.

- Geschachtelte Anfragen, die sich nicht entschachteln lassen, wie etwa Ungleichheit in Verbindung mit Aggregationsfunktionen, können noch nicht bearbeitet werden. Hier müßte die Kontrolle zwischen dem CSQL-Präprozessor und der Anfrageauswertung des DBS wechseln; das ist auf einfache Weise bei einem Präprozessor aber nicht möglich.

- Interpolation in geschachtelten SELECT-Klauseln innerhalb der HAVING-Klausel ist bisher nicht möglich. Hierzu müßte, wenn diese Anfragen ohne Interpolation im INGRES-System korrekt funktionieren würden, ebenfalls der Anfragebaum erweitert werden. Außerdem müßte dann an zwei Stellen Interpolation ausgewertet werden, bevor die Anfragen an SQL weitergegeben werden.

- Eingebaute INGRES-Funktionen, z.B. Typ-Konvertierungen oder String-Operationen, können nicht bearbeitet werden, wenn sie für die vorgezogene Auswertung einer Bedingung benötigt werden. Die INGRES-Funktionen können aus dem CSQL-Präprozessor heraus nicht direkt eingesetzt werden; sie müßten explizit nachimplementiert werden.

- Die INGRES-Anweisungen HELP und CREATE PROCEDURE sind in CSQL nicht zulässig. Sie gehören auch nicht zum Funktionsumfang des INGRES/Embedded SQL. CREATE PROCEDURE ist direkt in einem Präprozessor überhaupt nicht möglich, weil die Kontrolle mehrmals zwischen INGRES und dem Präprozessor wechseln müßte. Möglich wäre hier höchstens der Umweg über ein Kommando in einer Datei, die mit SQL ausgeführt wird.
  HELP müßte vollständig neu implementiert werden mit Hilfe der SQLDA und mehreren Anfragen an die Verwaltungsrelationen der Datenbank. Da für CSQL im Rahmen einer anderen Arbeit eine eigene Verwaltungsschnittstelle implementiert worden ist [Lieb90], wurde auf diesen aufwendigen Teil innerhalb des CSQL-Systems verzichtet.

- An einigen Stellen sind nicht alle in INGRES möglichen alternativen Schreibweisen zulässig. Da hier keine Beeinträchtigung des Funktionsumfangs entsteht, wurde dies zugunsten einer einfacheren Implementierung zurückgestellt (siehe [Bosc90, Anhang B Benutzerhandbuch]).

- Es fehlt noch die Unterstützung des Benutzers beim Einfügen neuer und beim Entfernen veralteter Interpolationsroutinen aus dem CSQL-System. Deshalb gibt es auch noch keine DROP METHOD-Anweisung. Dies wäre in einer späteren Version sinnvoll.

- Basisattribute können noch nicht über Sekundärschlüssel den Meßwerten zugeordnet werden. Dies wäre bei mehrdimensionalen Basen sinnvoll, z.B., wenn bei Pegelständen nur noch die Identifikatoren der Meßbrunnen beigefügt wären, denen in einer anderen Relation die Koordinaten zugeordnet sind.

- Die Generierung der STEP-Werte erfolgt noch auf eine gewisse Art ‘eindimensional’, auch bei mehreren variierenden Basisattributen, entlang einer Geraden im Raum. Um dies zu ändern, also ein mehrdimensionales Raster von STEP-Werten zu erzeugen, müßten folgende Änderungen eingeführt werden (hier in einem C-ähnlichen Pseudo-Code dargestellt):

Folgende Deklarationen sind gegeben:

```
float entry [1 ... n]      /* ENTRY-Angaben für n Dimensionen                  */
float step [1 ... n];      /* STEP-Angaben für n Dimensionen                   */
float w_wert [1 ... n];    /* Variable für die STEP-Werte für n Dimensionen    */
bool möglich;              /* weitere Interpolation möglich?                   */

/* Initialisierungen:                                                          */
for i=1 to n do w_wert[i] = entry[i];
möglich = ja;
```

Bisher lief das Erzeugen der STEP-Werte so ab:

```
while (möglich) do
{  Eingabe-Parameterstring generieren;
   möglich = interpol (Eingabe-Parameterstring, weitere_folgen);
   for i=1 to n do w_wert[i] = w_wert[i] + step[i];
};
```

Später soll das Erzeugen der STEP-Werte dann wie folgt ablaufen:

```
while (möglich) do                        /* 2.- n-te Dim. heraufzählen       */
{  while (möglich) do                     /* 1. Dimension bearbeiten          */
   {    Eingabe-Parameterstring generieren;
        möglich = interpol (Eingabe-Parameterstring, weitere_folgen);
        if (möglich) then w_wert[1] = w_wert[1] + step[1];
   };
   /* möglich == nein                                                          */
   i = 1;
   while ((w_wert[i] == entry[i]) and (i < n) do
   {    i = i + 1;
        if (i < n) then w_wert[i] = entry[i];       /* Dimension zurücksetzen  */
        w_wert[i+1] = w_wert[i+1] + step[i+1];  /* nächste Dimension heraufzählen*/
        möglich = ja;
   };
};
```

Für den zweidimensionalen Fall würden die spezifizierten Punkte dann bei einer Bedingung mit Abstand von vorhandenen Werten in der in Bild 8.6 beschriebenen Art und Weise abgearbeitet:

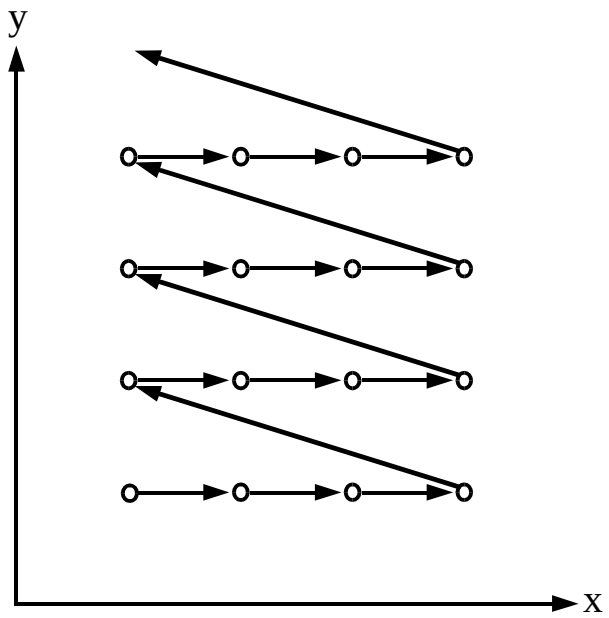


Bild 8.6: Zweidimensionale Erzeugung von STEP-Werten

8.9 Mögliche Verbesserungen bei direkter Einbettung

Da das CSQL-System auf einem kommerziellen DBS aufsetzt, für das keine Quellen zur Verfügung stehen, war der Präprozessor-Ansatz der einzig mögliche. Es entstehen jedoch einige Nachteile dadurch.

Zunächst muß die gesamte Anfrage zweimal geparst werden, einmal vom CSQL-System und einmal von DBS. Nach Bearbeitung der Interpolation muß zusätzlich noch aus dem Anfragebaum wieder eine SQL-Anfrage aufgebaut werden. Bei einer direkten Einbettung könnten das Aufbauen der SQL-Anfrage und das zweite Parsen entfallen und damit die Anfragebearbeitung beschleunigt werden. Außerdem entfällt das Nachbilden der DBS-Benutzerschnittstellen. Vor allem, wenn die Erweiterungen einmal in die DBS-Anfrageauswertung eingebettet sind, stehen sie in allen DBS-Schnittstellen zur Verfügung. Auch entfällt die unterschiedliche Bearbeitung derselben Klauseln an verschiedenen Stellen der Anfrage im CSQL-Aufsatz und im eigentlichen DBS.

Die Probleme mit geschachtelten Anfragen bestünden nicht mehr, denn

- das Wechseln der Kontrolle zwischen Interpolation und Anfrageauswertung würde kein Problem mehr darstellen;
- das DBS würde die meisten geschachtelten Anfragen ohnehin zur Optimierung entschachteln;
- wo dies nicht geschieht, wäre eine weitere Entschachtelung kein Problem, weil die in Zwischenschritten notwendigen Duplikat-Eliminierungen eingebaut werden könnten.

Die Interpolation könnte besser optimiert werden, weil sie mit anderen Auswertungsschritten vermischt und gekoppelt werden könnte (siehe auch Kap. 9).

Dennoch wurde mit den beschriebenen Methoden ein System erstellt, das für kleinere bis mittlere Anwendungen akzeptable Antwortzeiten bietet (siehe Kap. 10) und als Prototyp für erste Benutzer verwendet werden kann.

9. Alternative Syntax und Optimierung

In diesem Kapitel werden zunächst einige Schwächen und Nachteile der CSQL-Anfragesprache und des CSQL-Systems aufgezählt. Diese dienen dann als Motivation für einen zweiten Entwurf einer Syntax und die Einbettung in ein DBS. Dieser zweite Entwurf geht jedoch davon aus, daß das unterliegende DBS einige Eigenschaften der EDBS besitzt (siehe Kap. 4), die auf dem im Projekt verwendeten DBS INGRES noch nicht alle gegeben waren.

Anschließend werden die möglichen Ansatzpunkte für Optimierungen in der Anfrageauswertung für Anfragen mit eingebetteter Interpolation beschrieben. Es wird herausgestellt, welche Optimierungen für CSQL-Anfragen möglich sind und welche erst im zweiten Ansatz angewendet werden können. Im letzten Unterkapitel (Kap. 9.4) werden beide Ansätze gegeneinander abgewogen und es wird gezeigt, welcher Ansatz für welche Anwendungssituation eher geeignet ist.

9.1 Gründe für einen zweiten neuen Ansatz

Die in Kapitel 6 gegebene CSQL-Syntax ist für den Endbenutzer zwar leicht zu verstehen, hat aber einige Nachteile, insbesondere für den geübteren SQL-Benutzer:

- Bei CSQL-Anfragen ist keine direkte Umformung in ein Relationenkalkül möglich. Gerade dieser formale Hintergrund ist aber einer der wesentlichen Vorteile von SQL. Das Ergebnis einer SQL-Anfrage ist damit immer genau definiert. (Das Ergebnis einer CSQL-Anfrage ist natürlich auch immer genau definiert. Es ist nur keine so einfache direkte kurze Umformung in das Relationenkalkül möglich.)

- Wenn Einschränkungen für Basisattribute in der WHERE-Klausel gegeben werden, kann der Benutzer nicht spezifizieren, ob sie **vor** oder **nach** der Interpolation angewendet werden sollen. (Eine ausführliche Beschreibung dieses Problems befindet sich in Kap. 6.3.1.) Die Anwendung der Einschränkungen erfolgt nach festen Regeln. Manchmal wäre aber eine andere Anwendung wünschenswert.

Durch den CSQL-Ansatz sind auch einige Nachteile des Gesamtsystems bedingt, die bei einer anderen Syntax, die eine andere Vorgehensweise bei der Auswertung ermöglicht, nicht unbedingt auftreten müssen. Diese Nachteile des Systems sind vor allem folgende:

- Das CSQL-System läßt sich nicht einfach für andere Klassen eingebetteter Prozeduren oder Funktionen erweitern. Dazu ist es zu sehr auf die Bearbeitung von Interpolation ausgelegt.

- Obwohl in Kap. 7 eine Möglichkeit aufgezeigt wurde, wie man dem Programmierer der Interpolationsroutinen den DB-Zugriff vereinfachen kann bzw. ihn vom direkten DB-Zugriff via SQL befreien kann, bleibt diese Lösung teilweise unbefriedigend. Eigentlich sollte der DB-Zugriff der Anfrageformulierung überlassen bleiben, und die Interpolationsfunktionen sollten nur Eingabewerte (bzw. Objekte wie ganze Tupel oder Relationen) erhalten und Ausgaben produzieren. Diese Form erleichtert auch die Optimierung der gesamten Anfrage.

- Wenn in der WHERE-Klausel keine Bereichseinschränkungen für die variablen Basisattribute gegeben wurden, ist es für das CSQL-System sehr schwierig, z.T. sogar unmöglich zu bestimmen, wann es mit der Generierung von STEP-Werten aufhören soll. (Im folgenden wird immer dann von *Extrapolation* gesprochen, wenn nur auf einer Seite einer Interpolationsstelle Meßwerte vorhanden sind, die zur Interpolation herangezogen werden können, weil es entweder keine weiteren Meßwerte mehr gibt oder weil diese zu weit entfernt sind.) Um zumindest unbegrenzte und unendliche Extrapolation zu verhindern, müssen die derzeitigen Interpolationsfunktionen alle selbst überprüfen, ob sie mit den in der DB vorhandenen Werten überhaupt sinnvoll interpolieren können und dies dem CSQL-System über die Ausgabe mitteilen. Trotzdem ist es häufig kompliziert zu bestimmen, ob noch weitere Werte interpoliert werden können, wie das Beispiel in Bild 9.1 zeigt.

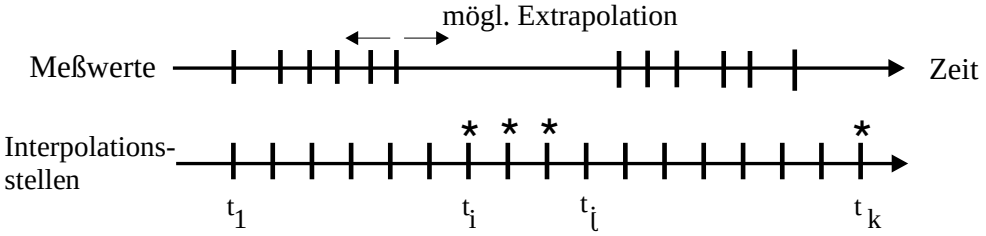


Bild 9.1: Mögliche und nicht mögliche (*) Extrapolation an verschiedenen Stellen auf einer Zeitachse

In Bild 9.1 zeigen die Pfeile, wie weit von irgendeinem Meßzeitpunkt aus, die Interpolationsfunktion noch Werte liefert. Zum Zeitpunkt t_i liefert sie bereits zurück, daß keine weitere Interpolation möglich ist. Zum Zeitpunkt $t_j > t_i$ wäre das aber doch wieder der Fall. Erst zum Zeitpunkt t_k ist endgültig keine Extrapolation mehr möglich.

In diesem eindimensionalen Fall kann noch mit min/max-Werten gearbeitet werden: Eine Interpolation/Extrapolation ist genau dann nicht mehr möglich, wenn die gewünschte Interpola-

tionszeit größer ist als der maximale Meßwert plus der möglichen Extrapolation. Zum Zeitpunkt t_k steht dann sicher fest, daß Interpolation für größere Werte nicht mehr möglich sein kann. Aber im mehrdimensionalen Fall wird dieses Verfahren sehr aufwendig und die Abbruchbedingungen sind kompliziert zu bestimmen, insbesondere dann, wenn das System auch 'mehrdimensional' die STEP-Werte erzeugt. Hinzu kommt, daß die Interpolationsfunktion nicht weiß, ob der Benutzer eine einfache, sinnvolle Bereichseinschränkung für die variablen Basisattribute gegeben hat. Die aufwendige Abbruchbedingung wird immer überprüft.

9.2 Der zweite Ansatz

Für den neuen Ansatz werden zunächst drei Erweiterungen von SQL eingeführt, die dann zu einer CSQL äquivalenten Form kombiniert werden können. Natürlich können alle drei Erweiterungen auch einzeln genutzt werden. In diesem Fall verlangen sie aber auch mehr Verständnis der Anfragesprache vom Benutzer.

Die folgenden Erweiterungen sollen auch an einem Beispiel gezeigt werden, und zwar an einer Erweiterung des CSQL-Beispiels 1) aus Kapitel 10, das im folgenden auch hier als Beispiel 1) bezeichnet wird. Die gegebene Relation lautet:

gwmessung (x_koord, y_koord, zeitpunkt, gdw_stand, gwm_id);

Es wird die Selektion von Grundwasserständen an einer Reihe von bestimmten, nicht gemessenen Punkten gewünscht. Die CSQL-Anfrage lautet:

```
SELECT x_koord, y_koord, gdw_stand
FROM   gwmessung BY METHOD gdw_staende
                         ENTRY (x_koord = 50.0, y_koord = 225.0)
                         STEP (0, 10)
WHERE  zeitpunkt = '29/03/88'      AND
       y_koord   < 350;
```

1) Zunächst muß eine Implementierung von SQL gewählt werden, in der geschachtelte SELECT-Anfragen, sog. Subselects auch in der FROM-Klausel möglich sind, oder das verwendete SQL muß so erweitert werden. Das würde dann wie folgt aussehen:

```
SELECT A.b1, A.b2, B.b1, B.B2
FROM   R1 A, ( SELECT b1, b2
                  FROM   R2
                  WHERE  ...) B
WHERE  A.b3 = B.b1        AND ...;
```

In der inneren WHERE-Klausel, die Relation B erzeugt, sollten keine Attribute der Relation A referenziert werden dürfen, weil es sich hier praktisch um 'parallele', unabhängige Rela-

tionen handelt, wie etwa zwei verschiedene Subselects in einer WHERE-Klausel. Innerhalb der Interpolation definiert diese neue Relation die Ergebnisrelation für die interpolierten Werte. Sie wird nicht dauerhaft gespeichert und entspricht der temporären Relation im CSQL-System.

Das Subselect in der FROM-Klausel enthält dann die Interpolation. Die Originalrelation vor der Interpolation ist dann nach außen nicht mehr sichtbar. In der 'Restanfrage' können nur die interpolierten Werte und die neuen Werte der variablen Basisattribute angesprochen werden. Innerhalb der Interpolation können die Attribute der Original-Relation mit ihrem alten Attributnamen qualifiziert werden, die neuen bekommen einen neuen Namen. Im folgenden sollen sie mit dem Präfix "new_" angesprochen werden. In der SELECT-Klausel des Subselects werden die Attribute der temporären Relation benannt. Der Übersichtlichkeit halber sollten sie den Attributnamen der Originalrelation entsprechen, soweit die Attribute übernommen werden.

2) Das Berechnen von interpolierten Werten aus umliegenden Werten ähnelt den Aggregationsfunktionen. Es muß also dem Benutzer ermöglicht werden, eigene, mächtigere Aggregationsfunktionen zu definieren und zu programmieren. Diese Aggregationsfunktionen sollten zusätzlich
   - konstante Eingabeparameter akzeptieren und
   - ganze Tupel oder Objekte übergeben bekommem.

   Die folgende Funktion z.B. summiert Werte des Attributs m auf, die nicht weiter als x Maßeinheiten von einem gegebenen Wert (meas_value) entfernt sind:

   ```
   SUM_DIST (m, meas_value, x); aufgerufen: SUM_DIST (gdw_stand, r.gdw_stand, 10)
   ```

   Die zweite Beispiel-Funktion berechnet das gewichtete Mittel der x dichtesten Meßstellen:

   ```
   WEIGHT_X ((m, x_alt, y_alt), x_neu, y_neu, x);
   ```

   aufgerufen für Beispiel 1):

   ```
   WEIGHT_X ((gdw_stand, x_koord, y_koord), new_x_koord, new_y_koord, 3)
   ```

   Vorn in der inneren Klammer stehen die betroffenen Attribute der Relation, dahinter die Parameter: der Punkt (x_neu, y_neu), zu dem die dichtesten Meßstellen gefunden werden sollen und die Anzahl x der zu berücksichtigen Meßstellen.

3) Um die ENTRY- und STEP-Angaben von CSQL darzustellen, wird eine bestimmte Art von relationenwertiger Funktion (table function) benötigt, die die folgenden Eingabeparameter erwartet:
   - die Original-Meßwertrelation,
   - die ENTRY- und STEP-Angaben für jedes Basisattribut und

- eine *Abbruchbedingung*, die angibt, in welchem Bereich die STEP-Werte generiert werden sollen.

Diese Funktion erzeugt praktisch ein kartesisches Produkt aus der Original-Meßwertrelation und den möglichen Interoplationspunkten. Die Anzahl der Interpolationspunkte ist durch die Abbruchbedingung auf jeden Fall begrenzt. Diese relationenwertige Funktion wird hier ext_table (extended table) genannt, weil sie die Original-Relation um einige Attribute erweitert. Die allgemeine Form, die Syntax dieser Funktion könnte so aussehen:

```
ext_table (<orig_relation>, <var_attr_name1>, entry1, step1, ..., <var_attr_namen>, entryn, stepn
        <stop_condition>)
```

Für das Beispiel 1) sähe der Aufruf dieser Funktion dann so aus:

```
ext_table (gwmessung, 'x_koord', 50.0, 0, 'y_koord', 225.0, 10.0,
        sqrt (exp (abs (x_koord - new_x_koord))
          + (exp (abs (y_koord - new_y_koord))) <= 500)
```

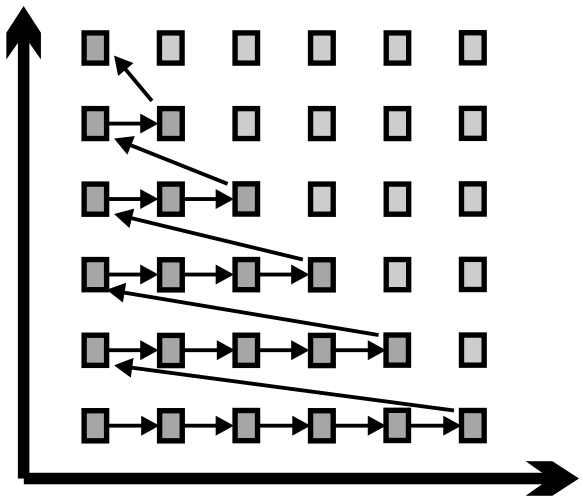
Die Abbruchbedingung ist ein Ausdruck, der die neu generierten variablen Basisattribute (hier: new_x_koord, new_y_koord) enthalten muß und einen booleschen Wert liefert. Solange dieser Wert zu "wahr" ausgewertet wird, werden Tupel um diese neuen Werte erweitert. Es sind auch feste Abbruchbedingungen möglich, die die alten Original-Basisattribute nicht enthalten, wie z.B.

```
new_x_koord < 150.0 AND new_y_koord < 350.0
```

Feste Bedingungen, die die alten variablen Basisattribute nicht enthalten, brauchen nur einmal pro generiertem Wertepaar ausgewertet zu werden, während die variablen Bedingungen für jedes einzelne Tupel ausgewertet werden müssen. Bei allen Bedingungen muß überprüft werden, ob sie überhaupt geeignet sind, die Generierung von STEP-Werten abzubrechen. Obergrenzen für negative Schrittweiten wären z.B. sinnlos. Diese Überprüfung ist innerhalb dieser ext_table-Funktion einfacher als während der Steuerung der Interpolation. Es könnte auch sinnvoll sein, mehrere derartige Funktionen für verschiedene Arten von Abbruchbedingungen bereit zu halten.

Prinzipiell treten bei den variablen Abbruchbedingungen die gleichen Probleme auf, wie sie in Kap. 8.7 beschrieben wurden (siehe Bild 9.2). Wenn jedoch die Formel zur Berechnung in der Abbruchbedingung schon gegeben wurde, können hier die Kombinationen von Extremwerten aller Dimensionen eingesetzt werden.

Wenn dann mit der in SQL-Anfragen üblichen Gruppierung nach den neuen Variablen und den festen Basisattributen gruppiert wird, läßt sich die jeweils richtige Eingabe-Untermenge für die Aggregationsfunktion bestimmen.



→ Reihenfolge, in der die STEP-Werte, die die Abstandsbedingung erfüllen, erzeugt werden (z.B. mit dem Algorithmus von Kap. 8.7)

Bild 9.2: Zweidimensionale Erzeugung von STEP-Werten

Insgesamt, sieht die Syntax des zweiten Ansatzes wie folgt aus:

```
SELECT  * | ...
FROM    R1, ... Rm, ( SELECT f_agg (m {, bi, ..., bk}), b1, ..., bn
                      FROM    ext_table (<orig_relation>, <att_namei>, entryi, stepi,
                              <new_att_namei>..., <att_namek>, entryk, stepk,
                              <new_att_namek>, <stop_condition>)
                      WHERE   user-defined restrictions
                      GROUP BY b1, ..., new_bi, ... new_bk, ..., bn)
WHERE   ...;
```

b1, ..., bn Basisattribute; bi, ..., bk variierende Basisattribute

Die Anfrage aus Beispiel 1 in Kap. 10 würde dann also insgesamt so aussehen:

```
SELECT  x_koord, y_koord, gdw_stand
FROM    (SELECT     gdw_stand = weight_x ((gdw_stand, x_koord, y_koord),
                                          new_x_koord, new_y_koord, 3),
                    x_koord = new_x_koord, y_koord = new_y_koord, zeitpunkt
         FROM       ext_table (gwmessung, 'x_koord', 50.0, 0, 'y_koord', 225.0, 10.0,
                               sqrt (exp (abs (x_koord - new_x_koord))
                                + exp (abs (y_koord - new_y_koord))) <= 500)
         WHERE      guete     = 'g'          AND
                    zeitpunkt = '29/03/88'
        GROUP BY new_x_koord, new_y_koord, zeitpunkt)
WHERE   y_koord < 350.0;
```

Diese Syntax ist SQL ähnlicher, aber sie ist schwieriger zu lesen und erwartet vom Endbenutzer mehr Wissen über Struktur und Inhalte der Datenbank sowie über den Ablauf der Interpolation.

Statt nur zu wissen, welche Interpolationsfunktionen es gibt, muß der Endbenutzer wissen, welche Funktionen der Art von ext_table es gibt, welche neuen Aggregationsfunktionen vorhanden

sind und wie er ext_table-Funktionen und Aggregationsfunktionen miteinander kombinieren darf. Er muß in der Lage sein, sinnvolle Abbruchbedingungen zu formulieren.

Der Programmierer der Interpolationsfunktionen hat es leichter, er programmiert nur noch die neuen Aggregationsfunktionen. Ein Datenbank-kundiger Programmierer sollte die Funktionen zur Relationenerweiterung erstellen.

9.3 Auswertung und Optimierungsmöglichkeiten bei Interpolationsanfragen

In diesem Abschnitt soll nur die Auswertung und Optimierung des Interpolationsanteils der Anfragen betrachtet werden. Sobald die Interpolation geschehen ist, kann die ‘Restanfrage’ wie andere Datenbank-Anfragen auch ausgewertet werden. Die Auswertung der Restanfrage wird nur in soweit betrachtet, wie die Kombination von Schritten in der Interpolation und der weiteren Auswertung Vorteile bringen könnte. Für das als Präprozessor arbeitende CSQL-System muß die Optimierung und Auswertung der Interpolation auf jeden Fall vor der Optimierung und Auswertung der restlichen Anfrage stattfinden und kann auch nicht damit vermischt werden.

Der Interpolationsanteil der Anfragen soll hier zunächst in kleinere Schritte zerlegt werden, die dann einzeln für die Optimierung und Auswertung betrachtet werden; eine bekannte Technik in der Anfrageoptimierung. Grob betrachtet zerfällt die Interpolationsauswertung in folgende Schritte, (die bis zu einem gewissen Grad vertauscht werden können):

(1) Lesen der Meßwertrelation aus der Datenbank
(2) Generierung der STEP-Werte
(3) Kombination der Tupel aus der Meßwertrelation mit den STEP-Werten, bzw. Erweiterung der Meßwerttupel um die STEP-Werte
(4) Zwischenspeicherung der neuen erweiterten Relation
(5) Lesen (von Teilen) der erweiterten Relation
(6) Auswertung der Aggregationsfunktion
(7) Speicherung der Zwischenrelation mit den Interpolationsergebnissen
[(8) Auswertung der restlichen Anfrage wie gewohnt]

Das Bild 9.3 zeigt diesen Vorgang allgemein und das Bild 9.4 für das Beispiel 1.

Mit der Optimierung kann man nun auf zwei verschiedene Arten ansetzen: Zunächst kann versucht werden, jede der oben aufgezählten Teiloperationen zu optimieren. Weil diese Optimierung innerhalb der in CSQL definierten Interpolationsroutinen stattfindet, wird sie im folgenden als *innere Optimierung* bezeichnet. Außerdem kann die Verarbeitung der ganzen Anfrage auf die Tatsache ausgerichtet werden, daß die Anfrage Interpolation enthält. Dies wird von nun an *äußere Optimierung* genannt.

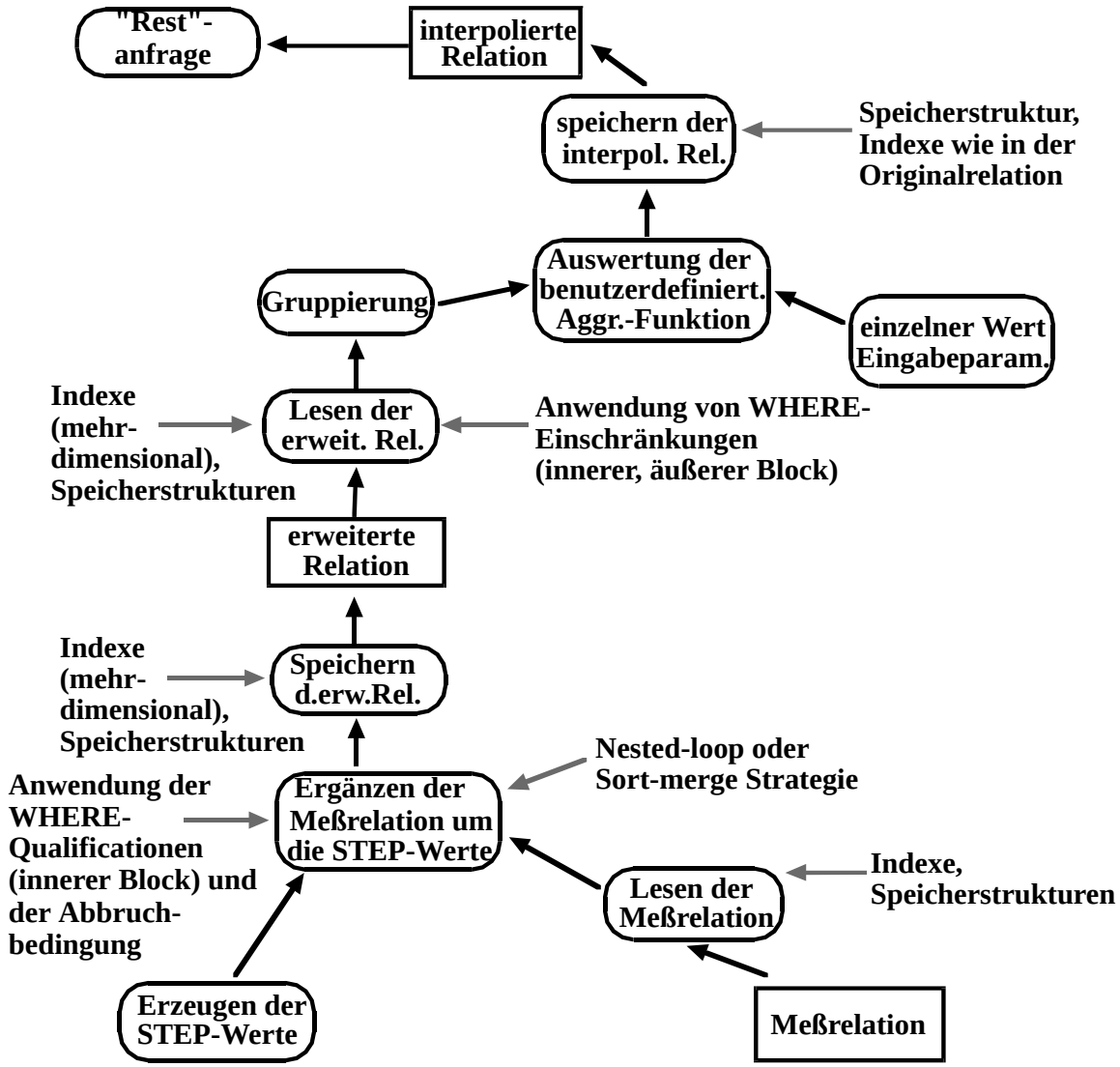


Bild 9.3:Anfragebaum mit Ansatzpunkten für die Optimierung

9.3.1 Innere Optimierung

Die Auswertung der Interpolation beginnt damit, daß die Meßwertrelation gelesen wird und die STEP-Werte erzeugt werden. Diese beiden Teile werden dann ähnlich einem kartesischen Produkt zu einer erweiterten Relation kombiniert. Wenn die Abbruchbedingung keine Verweise auf die ‘alten’ Basisattribute enthält (oder die WHERE-Klausel im CSQL keine entsprechenden Einschränkungen), wird jedes Tupel der Meßwertrelation mit jedem möglichen STEP-Wert (-Tupel) kombiniert. Eine Möglichkeit, dies zu tun, ist es, alle möglichen STEP-Werte nacheinander zu generieren und dann die entsprechenden Tupel der Meßwertrelation damit zu ergänzen. Oder es wird eine STEP-Werte-Kombination erzeugt und damit werden alle in Frage kommenden Tupel ergänzt. Wenn die Meßwertrelation nicht in den Hauptspeicher paßt, das ist gewöhnlich immer so, muß sie hier im ungünstigsten Fall so oft gelesen werden, wie STEP-Wert (-Tupel) erzeugt worden sind. Das wäre im Beispiel eins 13 mal, wenn die gegebene feste Abbruchbedingung

y_koord < 350.0 genutzt werden kann, aber 75 mal, wenn die gegebene variable Abbruchbedingung genutzt wird und der Meßpegel mit der höchsten y-Koordinate (10, 470) ist. Wenn die Abbruchbedingung Verweise auf die alten Basisattribute enthält, können Indexe auf diesen Attributen dazu beitragen, die Anzahl der zu lesenden Tupel beträchtlich zu verringern.

Es gibt nun Möglichkeiten, diesen Aufwand beträchtlich zu reduzieren: In den meisten Fällen wird es günstiger sein, die Originalrelation nur einmal zu lesen und für jedes Tupel die entsprechenden STEP-Werte zu erzeugen. Denn das Erzeugen der Werte ist viel schneller als das Lesen der ganzen Relation. Hier ist es geschickt, die Tupel in der geordneten Reihenfolge der festen Basisattribute zu lesen, wenn ein entsprechender Index vorhanden ist. Diese Reihenfolge wird nämlich im nächsten Schritt der Anfrageauswertung benötigt, dem Gruppieren der ergänzten Tupel nach allen Basisattributen.

Wenn die Basis mehrdimensional ist, wäre ein mehrdimensionaler Zugriffspfad am günstigsten. Index-Strukturen wie ISAM oder B-Tree, wie sie in konventionellen DBS vorhanden sind (s. [GrRe93]), sind nicht ausreichend. Strukturen, wie sie auch für räumliche Daten vorgeschlagen werden, erscheinen geeigneter. Beispiele hierfür sind der Buddy-Tree [SeKr90], der Quadtree [Same90] oder Cell-Trees mit zusätzlichen Ablageplätzen für übergroße Elemente [Guen90].

Während der Gruppierung können weitere Einschränkungen aus der WHERE-Klausel angewendet werden, um die Anzahl der zu verarbeitenden Tupel zu verringern. Sogar Bedingungen in der - bezogen auf den zweiten Ansatz äußeren WHERE-Klausel - die die neuen oder festen Attribute betreffen, können manchmal zu diesem Zeitpunkt schon ausgewertet und ausgenutzt werden, wenn sie keine weiteren Beziehungen zur äußeren Anfrage aufweisen. Im gegebenen Beispiel könnte die Qualifikation “meas_date = 29/03/88” während des Lesens der Relation ausgewertet werden. Das verringert die Größe der erweiterten Zwischenrelation auf einen Bruchteil der Originalrelation. Sind Messungen mit zehn verschiedenen Datumsangaben vollständig gespeichert, wird die Zwischenrelation nur noch ein Zehntel so groß wie ohne diese Einschränkung. Eine andere Einschränkung der äußeren WHERE-Klausel, “y_koord < 350.0”, kann ebenfalls ausgwertet und ausgenutzt werden. Wenn eine variable Abbruchbedingung gegeben wurde, wird so die Häufigkeit des Lesens der Originalrelation, von 75 auf 13 Läufe reduziert.

Wie zu Beginn dieses Abschnitts erwähnt, ist das Speichern der Zwischenrelation der nächste Schritt, an dem nach Optimierungsmöglichkeiten gesucht werden kann. Da bekannt ist, daß im nächsten Schritt der Interpolation nach allen Basisattributen gruppiert werden soll, ist die Speicherung der Tupel in der Reihenfolge, in der sie später benutzt werden sollen, ein vielversprechender Ansatz. Hier muß sorgfältig abgewogen werden, ob das Speichern der Tupel in der gewünschten Reihenfolge oder das Lesen und Sortieren der unsortierten Relation hinterher teurer ist. Ein temporärer Index über alle Basisattribute wäre eine andere Lösung für dieses Problem. Letzlich bleibt die Wahl zwischen den beiden folgenden sinnvoll erscheinenden Möglichkeiten:

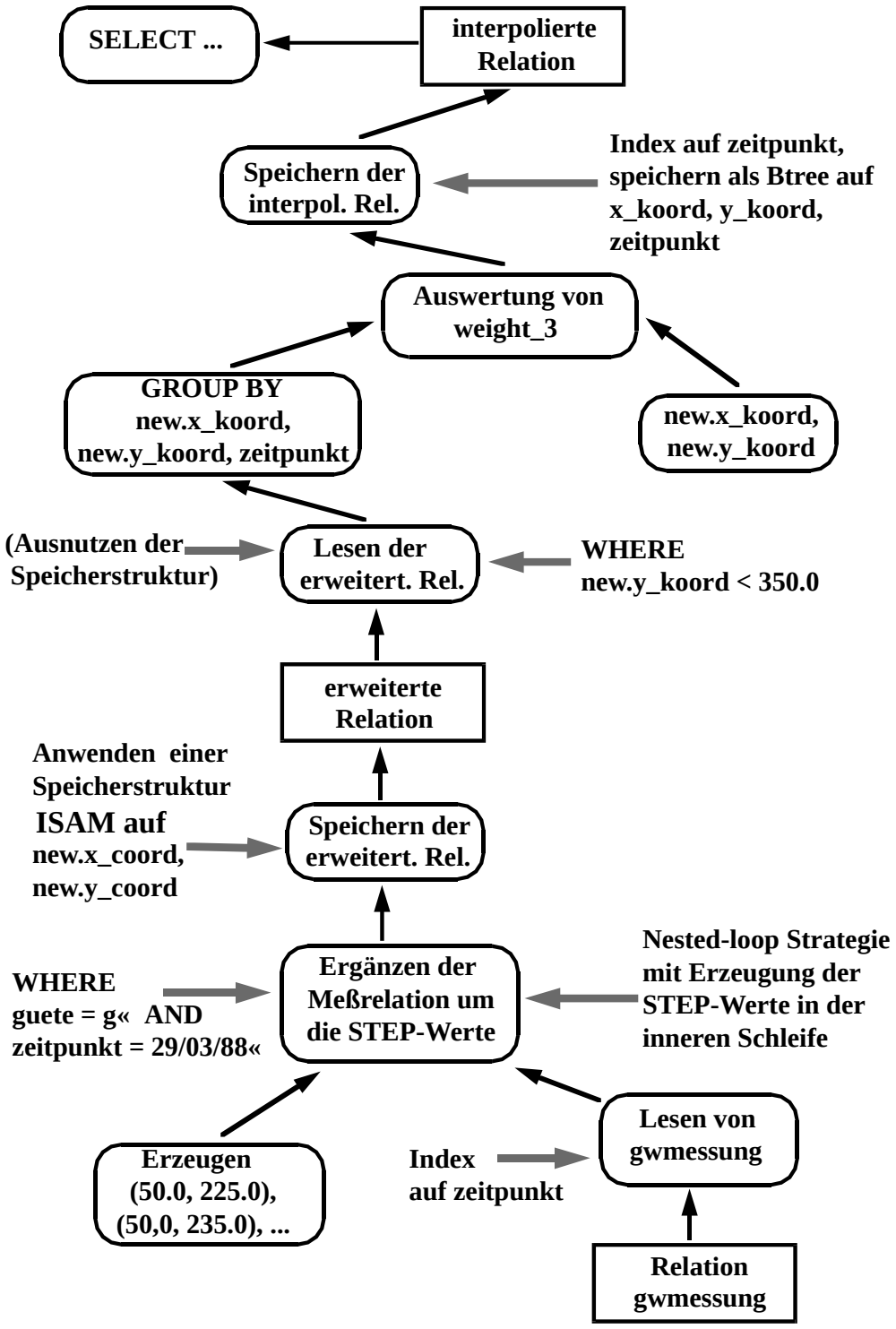


Bild 9.4: Anfragebaum mit Optimierung für das Beispiel 1)

- Speicherung der Tupel innerhalb einer Speicherstruktur, die für den späteren Zugriff die richtige Reihenfolge vorhält, z.B. B-Tree oder ISAM,
- Speicherung der Tupel in der Reihenfolge, in der sie auftreten und Anlegen eines Indexes über alle Basisattribute.

Im gegebenen Beispiel wären die folgenden Optimierungen sinnvoll: Nachdem die Bedingung in der WHERE-Klausel das feste Basisattribut auf einen einzigen Wert reduziert hat, sollte ein Index über die neuen x- und y-Koordinaten angelegt werden. Die Nützlichkeit dieses Schritts ist natürlich abhängig von der Implementierung des Zugriffspfads und des Indexes im DBS, von der Anzahl und Breite der Tupel in der Zwischenrelation, (die annähernd berechnet werden kann,) von der Anzahl der STEP-Werte und von der Selektivität der Einschränkungen (die berechnet werden können, wenn Statistiken geführt werden). Die Zeit, die das DBS zum Aufbau von Zugriffspfaden benötigt, abhängig von der Anzahl und Breite der Tupel, kann durch Leistungsmessungen annähernd im voraus bestimmt werden und so dem Optimierer bekannt sein. Nach diesen Vorarbeiten kann die Gruppierung zusammengehörender Tupel und die Auswertung der Aggregationsfunktionen sehr effizient durchgeführt werden.

Die Auswertung der benutzerdefinierten Aggregationsfunktion kann recht kompliziert werden. Hier kann es günstig sein, Zwischenergebnisse aufzubewahren, so daß sie nicht für jede Gruppe neu berechnet werden müssen. (s.a. Kap. 10.2.3.: Interpolation von Niederschlagswerten). Etwas Ähnliches wird auch in [HeNa96] für die Berechnung teurer Prädikate in DB-Anfragen vorgeschlagen.

Die Speicherung der Ergebnisrelation, einer ebenfalls temporären Relation, sollte im Hinblick darauf geschehen, daß diese Relation während der weiteren Anfrageauswertung noch gelesen wird, evtl. sogar mehrmals. In vielen Fällen kann es nützlich sein, die Speicherstruktur und die Zugriffspfade der Originalrelation zu verwenden, weil diese Relation die Originalrelation in der ‘Restanfrage’ ersetzt, also auf sie genauso zugegriffen wird wie auf die Originalrelation.

Folgender Punkt dabei ist noch wichtig: Wenn der Anfrageoptimierer ‘weiß’, daß eine Anfrage Interpolation enthält, weiß er auch, daß diese Teiloperationen in der gegebenen Reihenfolge ablaufen müssen, und kann dahingehend optimieren. Zumindest für diesen Teil der Anfrage kann damit auch das Erzeugen von alternativen Auswertungsplänen entfallen.

9.3.2 Äußere Optimierung

Das Prinzip ist hier, die Anzahl der Interpolationen zu verringern, wann immer dies mit vernünftigem Aufwand möglich ist. Denn Interpolation bedeutet, daß anstatt des Lesens eines Tupels mehrere Tupel (bis zur gesamten Relation) gelesen werden und eine Funktion ausgewertet wird. Hier ähnelt die Optimierung für Interpolationsanfragen der Optimierung von Anfragen mit aufwendig zu berechnenden Prädikaten oder geschachtelten Anfragen, die nicht entschachtelt werden können, wie sie von Hellerstein und seiner Arbeitgruppe mehrfach beschrieben wird [HeSt93, HeNa96, Hell98]. Die dort vorgeschlagenen Verfahren können aber nicht direkt übernommen werden, denn Interpolation kann nicht einfach als teures Prädikat betrachtet werden. Interpolation sollte bei der Anfrageauswertung so früh wie möglich durchgeführt werden, weil der Anfrageoptimierer für die Optimierung der ‘Restanfrage’ Informationen über die Ergebnisrelation aus der Interpolation benötigt. Außerdem müssen die in Kap. 6.2.4, Tabelle 6.1 gegebenen Einschränkungen für die Auswertungsreihenfolge unbedingt beachtet werden. Es muß des-

halb eine Art zweistufiger Optimierung durchgeführt werden: Zunächst wird die Interpolation optimiert, und dann - unter Einbeziehung der Interpolationsergebnisse - die Restanfrage. (Wenn ein als Präprozessor konzipiertes System verwendet wird, kann auch gar nicht anders gearbeitet werden.)

Zunächst einmal wird die Anzahl der Interpolationen schon durch das möglichst frühzeitige Auswerten geeigneter Restriktionen (Prädikate) deutlich verringert, und zwar genau in der Größenordnung, wie auch die Anzahl der Leseläufe der Originalrelation verringert wird.

Um die Anzahl der Interpolationen insgesamt zu reduzieren, ist das Entschachteln von Anfragen, die Interpolation in den inneren Unteranfragen enthalten, unerläßlich. Dies entspricht in etwa den Vorteilen, die ein Sort-Merge-Verbund gegenüber einem Nested-Loop-Verbund hat. Es können hier die bekannten, in Kap. 8.5 beschriebenen Verfahren zur Entschachtelung angewendet werden [Kim82, BaWo87].

9.3.3 Optimierung in CSQL

Im derzeitigen Prototyp-CSQL-System wird die in Kapitel 6.2 gegebene Syntax direkt ausgewertet (s.a. Kapitel 8). Zur Zeit führt das System folgernde Optimierungen durch:

- Einschränkungen, die in WHERE-Klauseln gegeben werden, werden verwertet.
- Es werden Index-Strukturen über das DBS verwendet.
- Geschachtelte Anfragen, die Interpolation in der inneren Unteranfrage enthalten, werden entschachtelt.
- Ob Zwischenergebnisse während der Interpolation von Reihen aufbewahrt werden, ist abhängig von der Interpolationsroutine.

Aber diese Lösungen haben einige Nachteile:

- Einige mögliche Optimierungen können nicht angewendet werden. Z.B. muß die Originalrelation mehrmals gelesen werden, weil für jeden gegebenen STEP-Wert einzeln interpoliert wird.
- Direkter Datenbankzugriff innerhalb der Interpolationsroutinen ist mühsam und verwirrend für viele Anwender, die ihre eigenen Interpolationsverfahren implementieren möchten. Man kann von ihnen nicht erwarten, daß sie bei der Implementierung stets die Verarbeitungsgeschwindigkeit bedenken und daß sie anschließend Benchmarks auf dem DBS laufen lassen, um die am besten geeignete Daten- bzw. Indexstruktur für ihr Interpolationsverfahren herauszufinden.

CSQL-Interpolationen können also bei gleicher Anzahl und Breite der Tupel ein recht unterschiedliches Antwortzeitverhalten aufweisen, abhängig von der Art und Qualität der Interpolationsroutinen und von den Datenstrukturen und Zugriffspfaden im DBS.

9.3.4 Optimierung im zweiten Ansatz

Das Lesen der Meßwertrelation und ihre Ergänzung mit den STEP-Werten kann effizient optimiert werden. Wenn jedes Tupel mit allen möglichen STEP-Werten ergänzt werden muß, dann ist die Zeit, die benötigt wird, um die Originalrelation zu lesen, nahezu unabhängig von der Anzahl der Werte, die interpoliert werden sollen. Die Zeit zum Lesen der Originalrelation kann jedoch noch erheblich reduziert werden, wenn WHERE-Restriktionen gegeben sind, die hoch selektiv sind und die von Indexen oder Speicherstrukturen unterstützt werden. Die Zeit, die benötigt wird, die Originalrelation zu lesen, ist dann nicht länger abhängig von der Anzahl der zu generierenden STEP-Werte sondern nur noch von der Größe der Relation.

Wenn die Restriktionen in der WHERE-Klausel keine Unteranfragen enthalten, sollten sie immer so früh wie möglich ausgewertet werden, weil sie häufig die Anzahl der weiter zu verarbeitenden Tupel erheblich reduzieren; und ausgewertet werden müssen sie sowieso. Restriktionen auf zusätzlichen Attributen müssen auf jeden Fall vor der Ergänzung mit STEP-Werten ausgewertet werden aus drei Gründen:

(1) Die kürzeren (schmaleren) Tupel können schneller verarbeitet werden.

(2) Die temporäre Relation wird kleiner. Wenn die Restriktionen hoch selektiv sind, kann die temporäre Relation sehr viel kleiner werden.

(3) Nach der Aggregation sind die Werte der sonstigen Attribute nicht mehr gültig und werden hinausprojiziert. Sobald die Restriktionen auf diesen Attributen ausgewertet sind, werden sie nicht mehr benötigt und die Projektionen können sofort durchgeführt werden. Wenn eine temporäre Relation benötigt wird, reduziert das die Länge der Tupel und erhöht so die Verarbeitungsgeschwindigkeit.

(1) und (2) stimmen auch für feste Basisattribute. (Sie stimmen jedoch nicht für variable Basisattribute. Hier betreffen die Restriktionen die STEP-Werte, nicht die ursprünglichen Attributwerte.)

Während der nächsten beiden Schritte, Speicherung und erneutem Lesen der temporären Relation, kann die Verarbeitungsgeschwindigkeit nicht so beträchtlich verbessert werden. Ob die Tupel in der richtigen Reihenfolge gespeichert sind, ob ein passender Index existiert oder ob die Tupel sortiert wurden, nachdem sie in beliebiger Reihenfolge gelesen wurden: die Größenordnung ist immer O(n log n), wobei n die Anzahl der Tupel der temporären Relation ist. Die Zeit, die benötigt wird, um ein Tupel zu lesen oder zu speichern ist O(log n), und es gibt n Tupel. Dies entspricht der durchschnittlichen Zeit, die es dauert, n Tupel zu sortieren, denn Sortieralgorithmen sind bekannterweise von der Größenordnung O(n log n). Welche Methode man hier am besten verwendet, hängt von der Implementierung der Speicher- und Indexstrukturen im aktuellen DBS ab, von der Anzahl der Tupel und ihrer Breite. Ähnliche Betrachtungen gelten für das Speichern und erneute Lesen der temporären Relation, die die interpolierten Werte enthält.

9.4 Vergleich und Bewertung beider Ansätze

In diesem Abschnitt sollen der CSQL-Ansatz und der zweite Ansatz verglichen und bewertet werden. Die Bewertung soll nicht ausschließlich aber schwerpunktmäßig bezüglich der Optimierung erfolgen, denn eine gute Optimierung und damit ein gutes Antwortzeitverhalten sind entscheidend für die Akzeptanz beim Benutzer. Es soll auch beschrieben werden, inwieweit die in Abschnitt 9.1 erwähnten Nachteile behoben werden könnten und ob neue Probleme entstehen würden. Auch wenn so ein System mit den gegebenen Voraussetzungen nicht zu implementieren war, sollten jedoch konkretere Überlegungen bezüglich dieser Punkte angestellt werden können.

Der zweite Ansatz ist dem ursprünglichen SQL näher. Es gibt keine neuen Klauseln; die Interpolation wird über Funktionen durchgeführt. Die erweiterten Aggregationsfunktionen entsprechen der Funktion f_j aus Kap. 6.2.2, die ext_table-Funktion liefert den Funktionen $interpolation_j$ und $interpol\_series_j$ die Argumente.

Der Benutzer kann für Einschränkungen auf den Basisattributen genau bestimmen, ob sie vor oder nach der Interpolation durchgeführt werden sollen. Einschränkungen auf zusätzlichen Attributen, die später hinausprojiziert werden, kann er nur zu einem Zeitpunkt geben, an dem sie noch ausgewertet werden können.

Erweiterte Aggregationsfunktionen, tabellenwertige Funktionen und Schachtelungen in der FROM-Klausel lassen sich in vielen anderen Anwendungen verwerten. Allerdings muß für jede Erweiterung dann wieder durchdacht werden, wie diese Bausteine zu kombinieren sind und welche Besonderheiten beachtet werden müssen. Die ext_table-Funktion jedoch ist wieder so speziell, daß sie nur für Interpolationszwecke zu verwenden ist.

Sobald die ext_table-Funktion einmal fertiggestellt ist, müssen für die verschiedenen Interpolationen nur noch die Aggregationsfunktionen geschrieben werden. Das ist auch für einen Anwender relativ einfach möglich, weil nicht auf die Datenbank zugegriffen werden muß, und eine Abbruchbedingung für die Erzeugung der STEP-Werte muß vom Endbenutzer mitgegeben werden. Damit wird das Problem, unendliche Extrapolation zu vermeiden, auf den Endbenutzer geschoben. An dieser Stelle ist es aber auch einfacher zu lösen. Oft wird dem Endbenutzer eine feste Abbruchbedingung genügen.

Im zweiten Ansatz können alle beschriebenen möglichen Optimierungen eingesetzt werden, ohne den Anwendungsprogrammierer zu belasten. Insbesondere sind auch alle verschiedenen Varianten von Entschachtelungen möglich. Ein Nachteil jedoch ist, daß einige temporäre Relationen, die durch die ext_table-Funktion entstehen, viel größer werden. Hier muß genügend Plattenspeicherkapazität vorgehalten werden. Das Lesen dieser großen Zwischenrelation erfolgt jedoch nur einmal und wird so insgesamt nicht länger dauern, als wenn die Originalrelation für jeden STEP-Wert neu gelesen werden muß. Ein entscheidender Vorteil ist, daß hier die Optimierung durch das DBS erfolgen kann und nicht vom Anwendungsprogrammierer abhängig ist.

So sind auch bei einem unerfahrenen Anwendungsprogrammierer akzeptable Antwortzeiten zu erwarten.

Die Syntax des zweiten Ansatzes ist zwar deutlich mehr an SQL angepaßt und damit insgesamt orthogonaler als die CSQL-Syntax aber auch nicht so einfach anzuwenden. So muß sich der Endbenutzer etwa über die Abbruchbedingungen und deren Formulierung selbst im Klaren sein. Die Vorteile des zweiten Ansatzes werden also auch mit einigen Nachteilen erkauft. Der zweite Ansatz eignet sich damit für eine andere Zielgruppe als CSQL. CSQL eignet sich für mit SQL und DBS allgemein relativ ungeübte Benutzer, wenn erfahrene Anwendungsprogrammierer die Interpolationsfunktionen schreiben (bzw. die Anwendungsprogrammierer entsprechend unterstützt werden). Dann ist auch eine gewisse Optimierung und damit ein günstiges Antwortzeitverhalten gewährleistet. Der zweite Ansatz eignet sich besser für den SQL-kundigen Anwender, der seine Interpolationsfunktionen schnell selbst schreiben möchte.

10. Fallstudie

In diesem Kapitel wird an einem Beispiel - einer Datenbank, einigen Interpolationsfunktionen und einigen Beispiel-Anfragen - gezeigt, wie das CSQL-System arbeitet. Die konkrete INGRES-Installation wird kurz beschrieben, insbesondere die Möglichkeiten, die für Leistungs- und Zeitmessungen vorhanden sind. Abschließend wird das CSQL-System anhand der Vereinfachung der Anfragen und der Ergebnisse der Messungen bewertet.

10.1 Die Beispiel-Datenbank

Das gegebene Datenbankschema bildet einen kleinen, aber repräsentativen Ausschnitt aus der Projektdatenbank. Alle Beispiele innerhalb dieses Kapitels beziehen sich auf dieses Schema. Die Beispiel-Relationen stammen mit Ausnahme der Relation bodenprobe aus zwei Teilprojekten, dem Geohydrologie-Teilprojekt (siehe Kap. 2.2) und dem Pflanzenbau-Teilprojekt (siehe Kap. 2.5).

grdwamess (zeitpunkt, gwm_id, tiefe, messpkthoehe, bauart);

gwmessung (x_koord, y_koord, zeitpunkt, gdw_stand, gwm_id);

bohrprobe (gwm_id, probe_id, tiefe_oben, tiefe_unten, zusatz, bodenart);

siebanalyse (sieb_id, tiefe_oben, tiefe_unten, gesamt_gewicht, bru_gewicht, volumen, korndichte, probe_id)

permvers (probe_id, leiter, kf_10, spez_r_ent, spez_r_wass, leitf_wass, entw_poros, restfeuchte, ntot_1, ntot_2, gwm_id)

wetter_info (messwert, kanalnr, einheit, sensortyp, relation, attribut, messbeginn, messende);

wetter_10 (zeitpunkt, regen, aregen, windweg, awindweg, tsl_k, atsl_k, tslf_k, atslf_k, tsl_a, atsl_a, tslf_a, atslf_a, windritng, awindritng, feuchte, afeuchte);

wetter_60 (zeitpunkt, global, aglobal, netto, anetto, par, apar, lufttempr, alufttempr, btemp5, abtemp5, btemp20, abtemp20, btemp35, abtemp35, btemp50, abtemp50, btemp80, abtemp80, btemp110, abtemp110 btemp140, abtemp 140, btemp5a, abtemp5a, btemp20a, abtemp20a, btemp35a, abtemp35a, btemp50a, abtemp50a, btemp80a, abtemp80a, btemp110a, abtemp110a, btemp140a, abtemp140a);

bodenproben (zeitpunkt, probe_id, x_koord, y_koord, tiefe, beschr)

Bild 10.1: Das Schema der Beispiel-Datenbank

Die Relationen grdwamess bis permvers (siehe Bild 10.1) stammen aus dem Geohydrologie-Projekt, die Wetter-Relationen aus dem Pflanzenbau-Projekt. (Das sind nicht alle Daten, die in diesen Projekten angefallen sind, für die Beispiele sind sie jedoch ausreichend.)

Diese Relationen wurden ausgewählt, weil die genannten Teilprojekte von Beginn des Gesamtprojekts an liefen und weil die Daten, vor allem die Schemata, frühzeitig vorlagen. Die Daten aus dem Geohydrologie-Projekt sind insbesondere wegen ihrer Dreidimensionalität interessant, die Wetterdaten wegen ihres Umfangs, einige 10 000 bis einige 100 000 Tupel.

Im Geohydrologie-Teilprojekt wurden Meßbrunnen (auch Meßpegel genannt) gebohrt, in denen anschließend regelmäßig die Grundwasserstände gemessen wurden, sowie zusätzliche Versuche durchgeführt wurden. Die Bohrkerne der Meßbrunnen wurden auf vielfältige Weise untersucht.

Die Relation grdwamess (Grundwassermeßstelle) beschreibt einen Meßbrunnen, und zwar den Zeitpunkt, an dem er gebohrt wurde, seine Tiefe von der Erdoberfläche aus gemessen, die Höhe ü.N.N. des Geländes an dieser Stelle (messpkthoehe), die Bauart (alt - neu gebohrt, Bohrverfahren) und gibt jedem Meßpegel einen eindeutigen Identifikator (gwm_id).

Die Relation gwmessung (Grundwasserstandsmessung) nennt die Meßstelle (gwm_id), ihre Koordinaten (x_koord, y_koord), den Zeitpunkt der Messung und den Grundwasserstand (gdw_stand) [in m, Höhe ü.N.N.], also den eigentlichen Meßwert.

Die Relation bohrprobe beschreibt die einzelnen Proben, das sind die Abschnitte, in die jeder Bohrkern zwecks weiterer Untersuchungen unterteilt wurde. Jedes Tupel enthält die Meßstelle (gwm_id), aus der der Bohrkern stammt, eine eindeutige Identifikation der einzelnen Probe (probe_id), den Abschnitt des Bohrkerns (tiefe_oben, tiefe_unten), einen Zusatz (z.B. ‘großer Stein’) und grob die Bodenart (z.B. Sand, Kies, Lehm).

An jede einzelne Probe wurden im Labor elektrische Spannungen angelegt, um verschiedene Bodenparameter zu bestimmen. Diese sind in der Relation permvers (Permeameterversuch) aufgezeichnet. Jedes Tupel enthält die eindeutige Identifikation der Probe (probe_id), den Namen des Versuchsleiters (leiter), die eindeutige Identifikation der Grundwassermeßstelle, aus der sie stammt (gwm_id) und die folgenden Meßparameter: den Mittelwert der Durchlässigkeit k_{f10} (kf_10), den spezifischen elektrischen Widerstand am entwässerten Kern (spez_r_ent), den spezifischen elektrischen Widerstand am wassergesättigten Kern (spez_r_wass), die Leitfähigkeit des Wassers (leitf_wass), die entwässerbare Porosität in % (entw_poros), die Restfeuchte, die Gesamtporosität nach der Formel $n_{tot} = n_e + n$ (ntot_1) und die Gesamtporosität nach einer weiteren Formel $n_{tot} = L^e - (PT*1000)/(2.7*PL*A)$ (ntot_2). Die Forscher im Wasserbau-Teilprojekt mußten häufig mehrere sehr ähnliche, nach nur leicht unterschiedlichen Methoden zu bestimmende Parameter in ihre Messungen und Berechnungen aufnehmen, um sich nach allen Seiten gegen den Vorwurf unvollständiger Arbeit abzusichern.

Jede Probe, sofern sie nicht aus Felsgestein besteht, wurde abschließend mehrfach durch Siebe mit verschiedenen Gittergrößen gesiebt. Die Ergebnisse sind in der Relation siebanalyse festgehalten. Für jede Siebgröße gibt es für jede Probe ein Tupel. Jedes Tupel enthält hier einen eigenen Identifikator (sieb_id), die Zuordnung zur Bohrprobe (probe_id), den Abschnitt der Probe in m unter Gelände (tiefe_oben, tiefe_unten), das Gewicht der Gesamtprobe (gesamt_gewicht), das Gewicht der Bruchstücke (bru_gewicht), das Volumen der Probe und die Korndichte.

Im Geohydrologie-Projekt sind noch viele weitere Daten angefallen, etwa bei durchgeführten Pump- und Tracerversuchen. Da diese Daten in den Beispielen nicht gebraucht werden, werden sie hier der Einfachheit halber weggelassen. Das vollständige Schema der Teildatenbank des Wasserbau-Teilprojekts sowie eine kurze Erklärung aller Meßparameter sind in [Kaja89, Kueb89] zu finden.

Im Pflanzenbau-Teilprojekt wurde einerseits das Wetter aufgezeichnet, weil es beim Wachstum der Pflanzen eine entscheidende Rolle spielt, sowie einige Parameter zum Pflanzenwachstum. Für die Beispiele hier sowie für Leistungsmessungen sind nur die Wetterdaten interessant, deshalb werden nur sie hier beschrieben. Die Relation wetter_info enthält Zusatzinformationen zu den eigentlichen Wetterdaten. Die Parameter der Relationen wetter_10 und wetter_60 werden hier beschrieben. Zu jedem Parameter wird gesagt, was gemessen wurde (messwert), auf welchem Kanal der Meßstation gemessen wurde (kanalnr), die Einheit, in der gemessen wurde (z.B. Grad C oder Liter/m^2), der Typ des Meßsensors wird angegeben (sensortyp), die Relation (wetter_10 oder wetter_60) und der Attributname werden genannt, sowie Beginn und Ende der Messungen. (Nicht alle Parameter wurden ganzjährig und während der gesamten Projektlaufzeit erhoben.)

Die Relation wetter_10 enthält die Parameter, die i.a. im Abstand von 10 min erhoben wurden, die Relation wetter_60 die Parameter, die im Abstand von 60 min erhoben wurden. Zu jedem Parameter gibt es jeweils einen Indikator (gespeichert in einem Attribut beginnend mit a..., z.B. aglobal zu global), der die Qualität des Meßwerts anzeigt (Werte, die während der Anlaufzeit eines Meßgeräts aufgezeichnet wurden, können beispielsweise noch fehlerhaft sein), bzw. besagt, daß dieser Meßwert zur Zeit gar nicht erhoben wurde.

In der Relation wetter_10 beschreibt der Parameter regen den Niederschlag pro m^2 innerhalb von 10 min, windweg die Windgeschwindigkeit, tsl_k die TSL-Strahlung auf einer konventionell bewirtschafteten Parzelle, tslf_k die TSLF-Strahlung auf derselben Parzelle, tsl_a die TSL-Strahlung auf einer alternativ bewirtschafteten Parzelle, tsfl_a die TSLF-Strahlung auf derselben Parzelle, windritng die Windrichtung und feuchte die Luftfeuchtigkeit. In der Relation wetter_60 beschreibt global die Globalstrahlung, netto die Nettostrahlung (der Sonne) und par die photosynthetisch aktive Strahlung. lufttempr nennt die Lufttemperatur und btempx jeweils die Bodentemperatur auf einer bestimmten konventionell bewirtschafteten Parzelle in der Tiefe x (x = 5, 20, 35, 50, 80, 110, 140cm). btempxa beschreibt die Bodentemperaturen auf einer anderen alternativ bewirtschafteten Parzelle.

Die Relation bodenproben stammt aus dem Hydrochemie-Teilprojekt. Jede Probe erhält einen eindeutigen Identifikator (probe_id). Zu jeder Probe wird der Zeitpunkt der Probenahme festgehalten, sowie die Koordinaten (x_koord, y_koord) und die mittlere Tiefe des Höhenabschnitts der Gesamtprobe (z.B. 30cm, 50cm). Welche Probenstücke zu einer insgesamt genommenen Probe gehören, erkennt man an den Koordinaten und an den Hunderterstellen des Identifikators. Außerdem wird eine Beschreibung (beschr) der Stelle des Versuchsfelds, an der die Probe gezogen wurde, gegeben (z.B. umweltschonend, Wiese). Mit diesen Proben wurden später verschiedene Analysen durchgeführt. Die Beschreibung, welche Tiefen oben und unten zu einer mittleren Tiefe gehören, findet sich in einer anderen Relation, ebenso die Ergebnisse der verschiedenen Analysen.

10.2 Die Beispiel-Interpolationsverfahren

Als Beispiel-Interpolationsfunktion wurde zunächst ein einfaches Verfahren zur räumlichen Interpolation von Grundwasserständen implementiert. Dies erschien u.a. wegen der Dreidimensionalität (zwei Dimensionen Basis und die Zeit als zusätzliches Attribut) interessant. Anschließend werden Interpolationsverfahren auf einigen Parametern der Wetterdaten beschrieben. Hier ist zu unterscheiden zwischen Meßwerten, die aufsummiert werden (z.B. Niederschlag) und solchen, wo stets Absolutwerte gemessen werden (z.B. Lufttemperatur).

10.2.1 Die Interpolation von Grundwasserständen

Von Beginn des Projekts an war klar, daß eine Funktion zur Interpolation von Grundwasserständen an beliebigen Punkten benötigt würde, weil, nachdem die Meßpegel einmal gebohrt wurden, an diesen Stellen keine Bodenproben mehr genommen werden können (siehe auch Kap. 3.4.2). Auch aus problemorientierter Sicht ist diese Interpolation besonders interessant, weil die Grundwasserstände drei Basiswerte haben: die Koordinaten (x, y) und die Zeit. Es bestehen die Möglichkeiten, über die Zeit an einem Ort zu interpolieren, über den Ort (Koordinaten) zu einem Zeitpunkt zu interpolieren und beides zu kombinieren. Im letzten Fall ist dann noch die Reihenfolge zu beachten. Für den Anfang wurde die Interpolation über den Ort gewählt, weil die Grundwasserstände ohnehin meistens gemessen wurden, wenn jemand auf dem Meßfeld andere Messungen durchführte oder Bodenproben nahm. Außerdem sieht man an diesem Beispiel sehr gut, daß geometrische Built-in-Funktionen im DBS sehr hilfreich wären.

Die Interpolation von Grundwasserständen erfolgt nach folgendem Algorithmus:

Die Funktion, gw_level, erhält als Eingabe einen String, der folgendes enthält: den Namen der Meßwertrelation (oder des Views), den Namen der Ergebnisrelation, in die der interpolierte Wert eingetragen werden soll, den Namen des festen Basisattributs (Datum/Uhrzeit), seinen Wert, die Namen der variablen Basisattribute (X- und Y-Koordinate) und ihre Werte sowie den Namen des Meßwerts (kontinuierlichen Attributs).

1. Auspacken des Eingabestrings und Überprüfung, ob die entsprechende Relation in der DB vorhanden ist.

2. Erzeugen einer temporären Sicht auf die Relation, die nur die Tupel zum angegebenen Datum sieht (also Auswertung des festen Basisattributs) und Zählen der an diesem Datum gemessenen Pegelstände.

3. Falls mehr als 3 (n) Pegelstände vorhanden sind, werden die 3 (n) räumlich dichtesten gemessenen Pegel bestimmt.
   [Falls sich dabei herausstellt, daß genau an der gefragten Stelle ein Meßpegel liegt und dort auch gemessen wurde, wird dieser Wert geliefert.]

4. Gewichtete Interpolation des Grundwasserstands; der dichteste Pegelstand wiegt am stärksten.

5. Einfügen des Ergebnisses in die übergebene Ergebnisrelation, Löschen der Sicht und Zurückliefern von “erfolgreich interpoliert”.

Sollten zu dem gegebenen Datum keine Pegelstände vorhanden sein, wird ein “nicht erfolgreich interpoliert” zurückgeliefert, aber auch einen Hinweis darauf, daß weitere Interpolationen möglich sind. Tritt ein DB-Fehler auf, wenn z.B. das Schreiben in die Ergebnisrelation ist nicht möglich, weil kein Plattenplatz mehr vorhanden ist, wird zurückgeliefert, daß die Interpolation nicht erfolgreich war und keine weiteren Interpolationen möglich sind.

10.2.2 Die Interpolation von Niederschlag

Die Interpolation von Niederschlag wird hier stellvertretend für alle sehr unregelmäßigen Meßparameter beschrieben, wie etwa auch Windgeschwindigkeit, Windrichtung oder Luftfeuchtigkeit. Es wird davon ausgegangen, daß der Niederschlag am Tage unterschiedlich verteilt ist, daß verschiedene Tage sehr unterschiedliche Mengen von Niederschlag haben können (von Tagen ohne Niederschlag bis zu Tagen mit starkem oder ununterbrochenem Niederschlag) und daß die verschiedenen Monate unterschiedliche Niederschlagssummen aufweisen. Das hier beschriebene Verfahren eignet sich zur Interpolation des Niederschlags in Nordeuropa oder ähnlichen Klimazonen. Für Klimagebiete mit sehr regelmäßigem Niederschlag (Mittagsregen, jährlicher Monsunregen) ist es weniger geeignet. Für die Interpolation von Niederschlag und ähnlichen Meßparametern haben wir in der gängigen Literatur kein passendes Verfahren gefunden. Das hier beschriebene Verfahren wurde in Diskussionen der Autorin mit dem Diplomanden Thilo Jahke [Jahk91] und dem Mitarbeiter im Pflanzenbau-Teilprojekt, Thomas Entenmann (siehe Kap. 2.5) gefunden. Eine detaillierte Beschreibung des Verfahrens findet sich in [Jahk91].

Das hier beschriebene Verfahren eignet sich zur Berechnung von Niederschlagswerten an einzelnen Zeitpunkten oder Reihen von äquidistanten Meßwerten mit demselben Meßabstand wie die Original-Meßwerte. Das ist darin begründet, daß die Niederschlagswerte über den Meßabstand (hier 10 min) aufsummiert werden. Einzelne Werte stellen auch immer eine Aufsummie-

rung über diesen Meßabstand dar. (Deshalb ist es auch nicht möglich, Werte mit unregelmäßigem Meßabstand in einem Arbeitsgang zu berechnen. Dies ist aber in CSQL auch nicht vorgesehen.) Eine Umrechnung der Werte auf einen anderen Meßabstand (etwa 5 min, 15 min oder 1 h) erforderte ein ganz anderes Verfahren.

Grob beschrieben verläuft die Interpolation von Niederschlag in folgenden Schritten ab:

1. Auspacken des Eingabestrings und Überprüfung, ob die entsprechende Relation in der DB vorhanden ist.

2. Zunächst wird überprüft, ob der gesuchte Wert in der Meßrelation (oder dem Meßview) vorhanden ist. In diesem Fall wird nicht interpoliert, der gesuchte Wert in die Ergebnisrelation eingetragen und ein “erfolgreich interpoliert” zurückgeliefert. [Es wird auch geprüft, ob der gesuchte Wert bereits in der Ergebnisrelation steht. - Das ist beim derzeitigen CSQL-System nicht möglich. In diesem Fall würde nichts getan und ein Erfolg zurückgeliefert.]

3. Berechnung einzelner fehlender Werte oder von Werten zu anderen Meßzeitpunkten.
   Es wird überprüft, ob Meßwerte in der ‘Umgebung’ des gesuchten Wertes vorhanden sind. Die Umgebung wird hier durch ein Vielfaches des Meßabstands definiert. (Sinnvoll erscheint hier bei stündlichen Messungen etwa ein Vielfaches von zwei, bei zehnminütigen Messungen ein Vielfaches von bis zu sechs.) Der nächst größere und der nächst kleinere Wert werden selektiert und aus diesen Werten wird linear, gewichtet interpoliert. Der gesuchte Wert wird in die Ergebnisrelation eingetragen und ein “erfolgreich interpoliert” wird zurückgeliefert.

4. Berechnung einer kleineren Reihe fehlender Werte, bzw. einzelner Werte aus dieser Reihe.
   Als nächstes wird überprüft, ob am vorhergehenden und am nächsten Tag Werte am gesuchten Zeitpunkt vorhanden sind. Damit auch Werte gefunden werden, die nicht exakt 24 Stunden vor oder nach dem gesuchten Wert liegen, wird in einer δ-Umgebung des gesuchten Zeitpunkts gesucht. Sinnvoll ist hier ein δ, das dem halben Meßabstand entspricht, bzw., dem halben Meßabstand minus einer Sekunde. Wenn diese beiden Werte vorhanden sind, wird aus diesen Werten linear, gewichtet interpoliert. Der gesuchte Wert wird in die Ergebnisrelation eingetragen und ein “erfolgreich interpoliert” wird zurückgeliefert.

5. Berechnung einer größeren Reihe fehlender Werte, bzw. einzelner Werte aus dieser Reihe.
   Fehlen die Werte über mehr als 24 Stunden hinweg, muß aufwendiger interpoliert werden. Betrachtet wird ein Intervall von Tagen vor und nach dem gesuchten Meßzeitpunkt. (Die Intervallbreite ist einstellbar und muß auch nicht vor und nach dem Meßzeitpunkt gleich sein.) Es werden nur Tage betrachtet, an denen die Meßwerte oder auch interpolierten Werte vollständig vorhanden sind. Aus diesen Werten wird ein ‘durchschnittlicher Regentag’ berechnet, d.h., es wird aufgeschlüsselt, wieviel Prozent des Regens in welchen Zeiträumen fallen. Der für den gesuchten Zeitraum ermittelte Wert wird als Interpolationsergebnis eingesetzt. Ein “erfolgreich interpoliert” wird zurückgeliefert.

In der derzeitigen Form ist das Verfahren nicht geeignet, leicht verschobene Meßreihen bei längeren Reihen fehlender Werte zu berechnen. Das liegt daran, daß bei der Ermittlung des ‘durchschnittlichen Regentags’ auch bereits berechnete Werte mit betrachtet werden. Dies ließe sich bei Bedarf leicht abändern. Außerdem darf man nicht Original-Meßwerte und interpolierte Werte zu anderen Zeitpunkten gemeinsam betrachten. Hierbei würde eine zu große Regenmenge vorgetäuscht. Diese Einschränkungen gelten nur für Meßwerte, bei denen aufsummiert wird, also bei den in diesem Projekt aufgezeichneten Meßparametern nur beim Niederschlag. Der Benutzer ist hier z.Zt. noch selbst für ein korrektes Einsetzen der Interpolationsverfahren verantwortlich.

10.2.3 Die Interpolation von Lufttemperatur

Die Interpolation von Lufttemperatur wird hier stellvertretend für die Interpolation von Wetterparametern beschrieben, die als Kurve über die Zeit aufgetragen, ein annähernd sinusförmiges Verhalten zeigen, und zwar übers Jahr und an jedem einzelnen Tag. Diese Wetterparameter sind neben der Lufttemperatur alle Bodentemperaturen und alle Strahlungsparameter, mit der Einschränkung, daß Strahlungswerte nicht negativ werden können. Auch hier gilt, daß das Verfahren für nordeuropäisches Klima implementiert wurde, und andere Klimazonen, wie etwa tropische Gebiete mit sehr gleichmäßiger Temperatur, evtl. Modifikationen erfordern würden. Das hier beschriebene Verfahren wurde aus dem in [FrTh84] beschriebenen entwickelt und ist in [Jahk91] ausführlich dargestellt.

In den ersten vier Punkten gleicht dieser Algorithmus dem aus Kap. 10.2.2. Der Vollständigkeit halber sind hier aber alle Punkte noch einmal aufgeführt.

1. Auspacken des Eingabestrings und Überprüfung, ob die entsprechende Relation in der DB vorhanden ist.

2. Zunächst wird überprüft, ob der gesuchte Wert in der Meßrelation (oder dem Meßview) vorhanden ist. In diesem Fall wird nicht interpoliert, der gesuchte Wert in die Ergebnisrelation eingetragen und ein “erfolgreich interpoliert” zurückgeliefert. [Es wird auch geprüft, ob der gesuchte Wert bereits in der Ergebnisrelation steht. - Das ist beim derzeitigen CSQL-System nicht möglich. In diesem Fall würde nichts getan und ein Erfolg zurückgeliefert.]

3. Berechnung einzelner fehlender Werte oder von Werten zu anderen Meßzeitpunkten.
   Es wird überprüft, ob Meßwerte in der ‘Umgebung’ des gesuchten Wertes vorhanden sind. Die Umgebung wird hier durch ein Vielfaches des Meßabstands definiert. (Sinnvoll erscheint hier bei stündlichen Messungen etwa ein Vielfaches von zwei, bei zehnminütigen Messungen ein Vielfaches von bis zu sechs.) Der nächst größere und der nächst kleinere Wert werden selektiert und aus diesen Werten wird linear, gewichtet interpoliert. Der gesuchte Wert wird in die Ergebnisrelation eingetragen und ein “erfolgreich interpoliert” wird zurückgeliefert.

4. Berechnung einer kleineren Reihe fehlender Werte, bzw. einzelner Werte aus dieser Reihe.
   Als nächstes wird überprüft, ob am vorhergehenden und am nächsten Tag Werte am gesuchten Zeitpunkt vorhanden sind. Damit auch Werte gefunden werden, die nicht exakt 24 Stunden vor oder nach dem gesuchten Wert liegen, wird in einer δ-Umgebung des gesuchten Zeitpunkts gesucht. Sinnvoll ist hier ein δ, das dem halben Meßabstand entspricht, bzw., dem halben Meßabstand minus einer Sekunde. Wenn diese beiden Werte vorhanden sind, wird aus diesen Werten linear, gewichtet interpoliert. Der gesuchte Wert wird in die Ergebnisrelation eingetragen und ein “erfolgreich interpoliert” wird zurückgeliefert.

5. Berechnung einer größeren Reihe fehlender Werte, bzw. einzelner Werte aus dieser Reihe.
   Fehlen die Werte über mehr als 24 Stunden hinweg, muß aufwendiger interpoliert werden. Zunächst werden dazu sämtliche Tagesminmal- und Tagesmaximalwerte berechnet. Berechnet werden dabei nur die Werte an den Tagen an denen mindestens P * 100 % der Meßwerte vorhanden sind (0<P<=1). Fehlende Werte werden aus dem vorhergehenden und nachfolgendem Minimal- bzw. Maximalwert gemittelt. Ist nur ein vorhergehender oder nachfolgender Wert vorhanden, wird der fehlende Wert diesem gleichgesetzt. Es müssen auch hier nicht die Werte des gesamten Jahres oder sogar mehrerer Jahre betrachtet werden. Betrachtet wird hier ein Intervall von ca. 100 Tagen vor und nach dem gesuchten Meßzeitpunkt. Die Intervallbreite ist einstellbar und muß auch nicht vor und nach dem Meßzeitpunkt gleich sein. Eine zu geringe Anzahl von Tagen, etwa zehn, führt jedoch zu sehr ungenauen Ergebnissen.
   Aus diesen äquidistanten Werten können mit Hilfe der sog. *Diskreten Fourierinterpolation* (auch *Trigonometrischen Interpolation*) durch Linearkombinationen von Sinus- und Kosinusfunktionen die Funktionen *f_min* und *f_max* berechnet werden. [FrTh84, Jahk91]. Diese Funktionen wären jeweils exakt, wenn die Anzahl der Sinus- und Kosinusglieder, also der Fourierkoeffizientenpaare, halb so groß wäre wie die Anzahl der Minimal- bzw. Maximalwerte. Für hinreichend genaue Ergebnisse sorgen aber auch schon zwei bis zehn Koeffizientenpaare. So werden der Minimal- und Maximalwert des gesuchten Tages berechnet.

6. In einem weiteren Schritt wird jetzt der Wert am gesuchten Zeitpunkt des Tages berechnet. Zwar ist der Temperaturverlauf auch innerhalb eines Tages näherungsweise sinusförmig, eine lineare Interpolation ist aber hinreichend genau und wird aus Leistungsgründen vorgezogen. Ein “erfolgreich interpoliert” wird zurückgeliefert.

7. Falls vom aufrufenden Programm signalisiert wurde, daß keine weiteren Interpolationen folgen, werden berechnete Minimal- und Maximalwerte sowie Fourierkoeffizienten gelöscht, andernfalls werden sie für weitere Interpolationsschritte aufbewahrt.

Das Verfahren zur Interpolation der Lufttemperatur ist in Bild 10.2 noch einmal kurz dargestellt.

Für ein sehr schnelles Interpolationsverfahren könnte man auf die Berechnung der Fourierkoeffizienten verzichten und direkt aus den geschätzten Tagesminimal- und Tagesmaximalwerten mitteln. Die Ergebnisse wären entsprechend ungenauer.

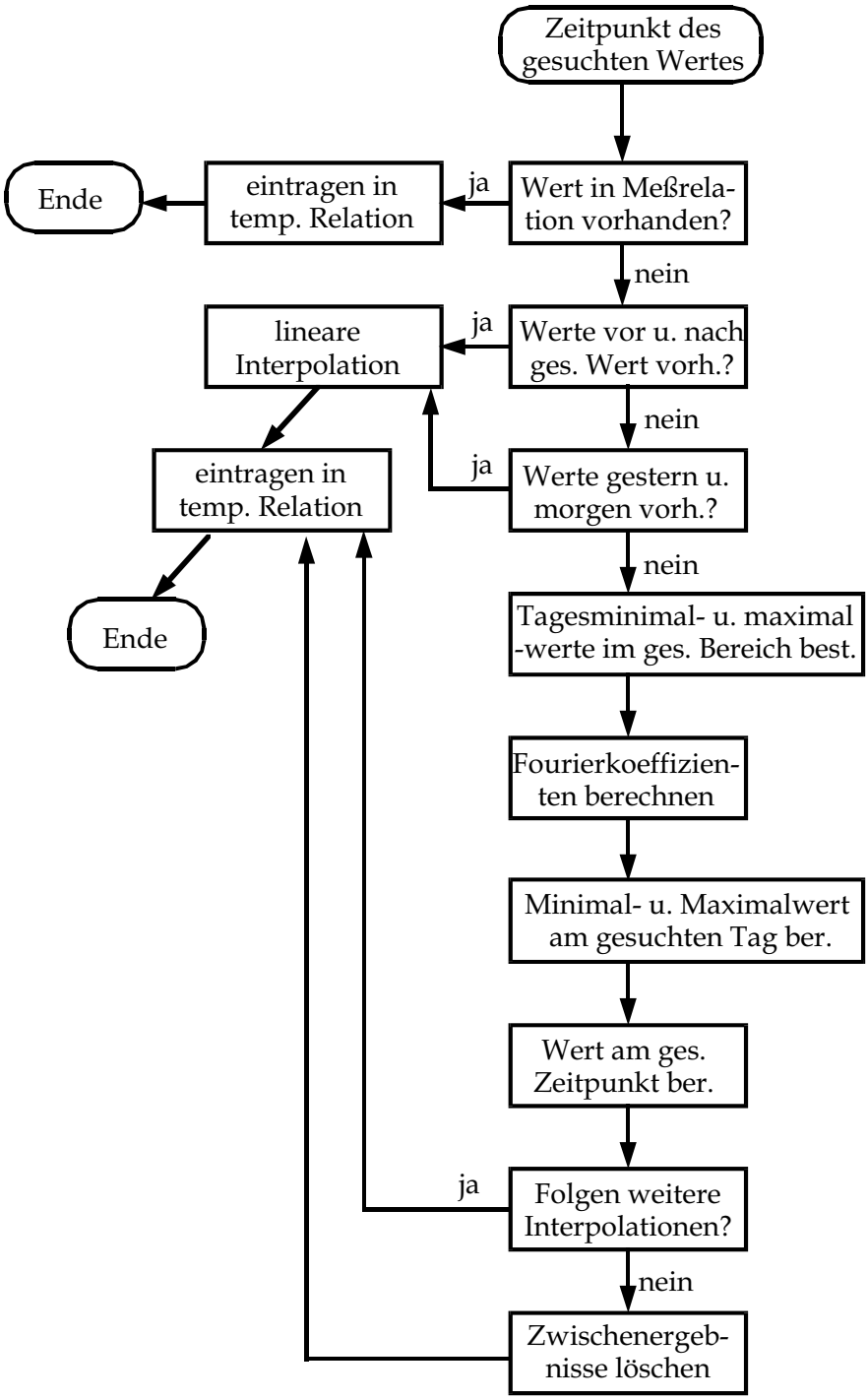


Bild 10.2: Interpolation der Lufttemperatur

10.3 Anwendungen auf der Beispiel-Datenbank

Im folgenden werden nun einige Beispiele gegeben, die zeigen, wie das CSQL-System mit den Interpolationsfunktionen angewendet werden kann. Es wird jeweils auch versucht, eine (oder

mehrere aufeinanderfolgende) SQL-Anweisungen zu geben, die zumindest annäherungsweise das Gleiche leisten. Das Antwortzeitverhalten dieser Beispielanfragen wurde auch gemessen. Die Ergebnisse sind in Kap. 10.5 zu finden.

1. Beispiel: Interpolation eines einzelnen Wertes

Hier wurde die Selektion von Grundwasserständen an beliebigen Punkten gewählt. Diese Anfrage wird häufig benötigt, z.B., wenn man abschätzen will, ob Pflanzen auch während Trokkenperioden genügend Wasser erhalten. Die CSQL-Anfrage lautet dann:

```
SELECT x_koord, y_koord, gdw_stand
FROM   gwmessung BY METHOD gdw_staende
                    ENTRY (x_koord = 50.0, y_koord = 225.0)
WHERE DATE_TRUNC ('DAYS', zeitpunkt) = '29/03/88'
```

(In der derzeitigen Version kann das CSQL-System die Funktion DATE_TRUNC noch nicht verarbeiten. Hier muß die DATE_TRUNC-Funktion entweder vorher durch einen SQL-View implementiert werden oder die Uhrzeit muß mit angegeben werden. Da die Grundwasserstände fast immer mit 12:00:00 Uhr gespeichert sind, bot sich die zweite Lösung für die Messungen an.)

Mit den folgenden Beispieldaten:

x_koord	y_koord	zeitpunkt	gdw_stand	gwm_id
35.0	260.0	26/03/88	158.689	p10
50.0	212.5	26/03/88	158.694	p11
62.5	215.0	26/03/88	158.687	p12
75.0	217.5	26/03/88	158.674	p13
42.5	235.0	26/03/88	158.687	p14
25.0	257.5	26/03/88	158.680	p9
35.0	260.0	29/03/88	157.119	p10
50.0	212.5	29/03/88	157.144	p11
62.5	215.0	29/03/88	157.137	p12
75.0	217.5	29/03/88	157.124	p13
42.5	235.0	29/03/88	157.127	p14
25.0	257.5	29/03/88	157.110	p9

liefert diese Anfrage das folgende Ergebnis:

x_koord	y_koord	gdw_stand
50.0	225.0	157.136

Mit den realen Daten, 6603 Tupel bei 67 verschiedenen Meßpegeln, liefert diese Anfrage das Ergebnis 157.139.

Um mit “normalem” SQL ein ähnliches Ergebnis zu erhalten, könnte der Benutzer die folgende Anfrage formulieren:

```
SELECT  50.0, 225.0, AVG (gdw_stand)
FROM    gwmessung
WHERE   x_koord >= 40.0    AND    x_koord <= 65.0        AND
        y_koord >=215.0    AND    y_koord <= 235.0       AND
        DATE_TRUNC (‘DAYS’, zeitpunkt) = ‘29/03/88’
```

Das Ergebnis wäre dann bei den oben gegebenen Daten das Mittel aus den Werten 157.137 und 157.127 (p12 und p14), also 157.132.

Hier muß der Benutzer genau wissen, wie er die Grenzen für die x- und y-Koordinaten zu setzen hat, damit er ein akzeptables Ergebnis erhält, oder aber ausprobieren. Bei den Originaldaten führte dies nach mehreren Versuchen zu der folgenden Einschränkung:

```
...
WHERE   x_koord >= 40.0    AND    x_koord <= 65.0        AND
        y_koord >=215.0    AND    y_koord <= 235.0       AND
...
```

Dann werden sieben Meßwerte berücksichtigt und das Ergebnis lautet 157.150. Dabei wird für den Benutzer sicher die Abweichung vom interpolierten Ergebnis von geringerer Bedeutung sein als die Tatsache, mehrfach probieren zu müssen, bis in den angegebenen Grenzen eine akzeptable Anzahl Tupel zu finden ist und dies auch durch Voranfragen überprüfen zu müssen.

2. Beispiel: Ein Verbund zwischen berechneten Werten und Meßwerten

Das zweite Beispiel zeigt einen Vergleich von Bodenproben und Grundwasserständen, implementiert durch einen Verbund. Die Grundwasserstände werden an den Stellen berechnet, an denen Bodenproben genommen wurden. Die Bodenproben stammen aus einem dem Benutzer bekannten äquidistanten Raster. (Für die Funktion DATE_TRUNC gilt das in Beispiel 1 gesagte.)

Die Einschränkung

```
b.tiefe      < 20
```

dient nur dazu, die Menge der Ausgabetupel einzuschränken. Schließlich ist der Grundwasserstand für alle Einzelproben aus einer Probenahme gleich. Gemessen wurde die Anfrage mit und ohne diese Einschränkung.

```
SELECT  b.probe_id,  gw.gdw_stand, b.tiefe, b.beschr
FROM    bodenproben b,
        gwmessung gw BY METHOD gw_level
                     ENTRY (x_koord = -100.0, y_koord = -200.0)
                     STEP (18.0, 27.0)
WHERE   DATE_TRUNC ('DAYS', gw.zeitpunkt)  = '18/12/90'   AND
        DATE_TRUNC ('DAYS', b.zeitpunkt)   = '18/12/90'   AND
        gw.x_koord      <= 100.0                          AND
        gw.y_koord      <= 100.0                          AND
[       b.tiefe      < 20                                 AND  ]
        b.x_koord  = gw.x_koord                           AND
        b.y_koord  = gw.y_koord
```

Die Einschränkungen

```
        gw.x_koord      <= 100.0            AND
        gw.y_koord      <= 100.0
```

genauer, mindestens eine von beiden, müssen hier als Abbruchbedingungen eingefügt werden. Die eigentlichen Abbruchbedingungen

```
        b.x_koord  = gw.x_koord             AND
        b.y_koord  = gw.y_koord
```

sind durch die Variabilität von b.x_koord und b.y_koord für die Interpolationssteuerung zu kompliziert auszuwerten. Das Interpolationsverfahren gw_level hört zwar auf zu interpolieren, wenn die gesuchten Koordinaten einen bestimmten Abstand zu den vorhandenen Meßstellen überschritten haben. Jedoch ist es für die Interpolationssteuerung unmöglich zu unterscheiden, ob dies für den folgenden Schritt auch zutrifft oder nicht (siehe Kap. 9.1).

Da das Hochzählen der STEP-Werte noch entlang einer Geraden im Raum, also praktisch eindimensional abläuft (siehe Kap. 8.7), müßten bei einem zweidimensionalen Raster noch mehrere sonst identische Anfragen mit unterschiedlichen ENTRY-Punkten gestellt werden.

Um ohne CSQL mit einer "normalen" SQL-Anfrage an ein halbwegs brauchbares Ergebnis zu kommen, könnte es der Benutzer mit folgender Anfrage versuchen (wieder mit und ohne die Einschränkung b.tiefe <= 20.):

```
SELECT  b.probe_id, AVG (gw.gdw_stand), b.tiefe, b.beschr
FROM    bodenproben b, gwmessung gw
WHERE   DATE_TRUNC ('DAYS', gw.zeitpunkt)  = '18/12/90'   AND
        DATE_TRUNC ('DAYS', b.zeitpunkt)   = '18/12/90'   AND
        ABS (b.x_koord - gw.x_koord) <= 60                AND
        ABS (b.y_koord - gw.y_koord) <= 80                AND
[       b.tiefe      <= 20                                ]
GROUP BY b.probe_id, b.tiefe, b.beschr
```

Auch hier muß der Benutzer wieder einige Male (hier 11 mal) probieren, bis er einen geeigneten Wert für den Abstand zwischen Bodenproben-Entnahmestellen und Grundwassermeßstellen findet. Im ungünstigsten Fall bekommt er für einige Bodenproben keine Werte, ohne daß für andere Proben zu viele Pegel herangezogen werden. Im konkreten Beispiel oben schwankt die Zahl der zur Durchschnittsbildung herangezogenen Pegel zwischen einem und 39. Auf jeden Fall müssen vorab Testanfragen laufen, in denen die Gruppierung und Durchschnittsberechnung wegfällt, dafür aber die Anzahl der in die Durchschnittsberechnung einzubeziehenden Pegel geliefert wird. Die Tabelle 10.1 zeigt, wie die CSQL- und SQL-Ergebnisse voneinander abweichen und wieviele Pegel die SQL-Anfrage jeweils in die Mittelung einbezieht.

Die Ergebnisse der CSQL-Anfrage und der SQL-Anfrage mit Gruppierung und Durchschnittsbildung weichen z.T. sehr voneinander ab: einmal weniger als 10 cm (Bohrproben 4xx), einmal mehr als ein Meter (Bohrproben 11xx). Natürlich muß der CSQL-Wert nicht unbedingt immer der bessere sein, jedoch spricht für seine Qualität, daß nur eine begrenzte Anzahl der dichtesten Meßpegel gemittelt wird und dabei auch noch gewichtet wird. Im konkreten Beispiel wurden von CSQL drei Meßwerte gewichtet gemittelt, während für die Bohrproben 11xx 13 Meßwerte ungewichtet gemittelt wurden, wie sich aus den Voranfragen ergab. Bei den Grundwasserständen, die sich am Datum des gegebenen Beispiels insgesamt nur um ca. drei Meter unterschieden ist eine Abweichung von einem Meter schon erheblich.

probe_id	CSQL-Wert	SQL-Wert	# Pegel bei SQL
1xx	154.413	154.379	1
2xx	154.424	154.379	1
3xx	154.597	154.379	1
4xx	154.407	154.500	8
5xx	154.387	154.427	15
6xx	154.445	154.376	39
7xx	154.430	154.342	33
8xx	154.311	154.338	31
9xx	154.321	154.509	34
10xx	154.960	154.541	27
11xx	155.748	154.632	13
12xx	155.731	156.263	2

Tabelle 10.1: Vergleich von CSQL- und SQL-Ergebnissen

Wenn Indices auf x- und y-Koordinaten angelegt sind, liefern sowohl CSQL als auch SQL die beschriebenen Ergebnisse sehr schnell (näheres hierzu in Kap. 10.5.3).

3. Beispiel: Vergleich von zwei Meßreihen

Um die Abhängigkeiten zwischen Luftfeuchtigkeit, Luft- und Bodentemperaturen zu erforschen, werden die Meßwerte, die im April 1990 erhoben wurden, über den Verbund verglichen. Beide Meßreihen müssen interpoliert werden, um zu vergleichbaren Werten zu kommen. Diese Anfrage ist typisch für eine Situation, in der man sich einen ersten Überblick verschaffen will.

```
SELECT  w10.zeitpunkt, w10.feuchte, w60.lufttemp, w60.btemp5, w60.btemp80,
        w60.btemp140, w60.btemp5a
FROM    wetter_10 w10 BY METHOD feuchte_zeit_int (feuchte)
                              ENTRY (zeitpunkt = '01/04/90 04:00:00')
                              STEP ('12 hours 0 minutes 0 seconds')
        wetter_60 w60 BY METHOD lufttemp_zeit_int (lufttemp)
                              ENTRY (zeitpunkt = '01/04/90 04:00:00')
                              STEP ('12 hours 0 minutes 0 seconds')
                      BY METHOD btemp5_zeit_int (btemp5)
                              ENTRY (zeitpunkt = '01/01/90 04:00:00')
                              STEP ('12 hours 0 minutes 0 seconds')
                      BY METHOD btemp80_zeit_int (btemp80)
                              ENTRY (zeitpunkt = '01/01/90 04:00:00')
                              STEP ('12 hours 0 minutes 0 seconds')
                      BY METHOD btemp140_zeit_int (btemp140)
                              ENTRY (zeitpunkt = '01/01/90 04:00:00')
                              STEP ('12 hours 0 minutes 0 seconds')
                      BY METHOD btemp5a_zeit_int (btemp5a)
                              ENTRY (zeitpunkt = '01/01/90 04:00:00')
                              STEP ('12 hours 0 minutes 0 seconds')
WHERE   w60.zeitpunkt  > '01/04/90 00:00:00' AND
        w60.zeitpunkt  < '01/05/90 00:00:00' AND
        w60.zeitpunkt  = w10.zeitpunkt
ORDER BY w60.zeitpunkt;
```

Diese Anfrage liefert dann nach akzeptabler Wartezeit (siehe Abschnitt 10.5.3) die gewünschten Ergebnisse. Mit einer einzigen Anfrage sind für dieses Beispiel in Standard-SQL keine brauchbaren Ergebnisse zu erhalten. Akzeptable Ergebnisse müssen hier über den Umweg von View-Definitionen erreicht werden. Der Benutzer muß dazu wissen, daß die Werte in 10- bzw. 60-minütigen Abständen gemessen wurden. Er kann versuchen, Werte zu finden, die in einer δ-Umgebung von 5 bzw. 30 Minuten um den gesuchten Zeitpunkt liegen. (Bei Zeitpunkten, deren δ-Umgebung über Mitternacht hinausginge, wäre das noch etwas komplizierter als hier gezeigt.) Die View-Definitionen und die Anfrage könnte dann so aussehen:

```
CREATE VIEW w10 AS
SELECT *
FROM   wetter_10
WHERE  DATE_PART ('HOURS', zeitpunkt)     = 3     AND
       DATE_PART ('MINUTES', zeitpunkt)   > 55    OR
       DATE_PART ('HOURS', zeitpunkt)     = 4     AND
       DATE_PART ('MINUTES', zeitpunkt)   <= 5    OR
       DATE_PART ('HOURS', zeitpunkt)     = 15    AND
       DATE_PART ('MINUTES', zeitpunkt)   > 55    OR
       DATE_PART (HOURS', zeitpunkt)      = 16    AND
       DATE_PART ('MINUTES', zeitpunkt)   <=5;


CREATE VIEW w60 AS
SELECT *
FROM   wetter_60
WHERE  DATE_PART ('HOURS', zeitpunkt)     = 3     AND
       DATE_PART ('MINUTES', zeitpunkt)   > 30    OR
       DATE_PART ('HOURS', zeitpunkt)     = 4     AND
       DATE_PART ('MINUTES', zeitpunkt)   <=30    OR
       DATE_PART ('HOURS', zeitpunkt)     = 15    AND
       DATE_PART ('MINUTES', zeitpunkt)   > 30    OR
       DATE_PART (HOURS', zeitpunkt)      = 16    AND
       DATE_PART ('MINUTES', zeitpunkt)   <=30;


SELECT w10.zeitpunkt, w10.feuchte, w60.lufttempr, w60.btemp5, w60.btemp80,
       w60.btemp140, w60.btemp5a
FROM   w10, w60
WHERE  w10.zeitpunkt > '01/04/90 00:00:00'              AND
       w10.zeitpunkt < '03/04/90 00:00:00'              AND
       DATE_TRUNC ('DAYS', w10.zeitpunkt) =
       DATE_TRUNC ('DAYS', w60.zeitpunkt)               AND
       (DATE_PART ('HOURS', w10.zeitpunkt) > 12         AND
       DATE_PART ('HOURS', w60.zeitpunkt) > 12          OR
       DATE_PART ('HOURS', w10.zeitpunkt) < 12          AND
       DATE_PART ('HOURS', w60.zeitpunkt) < 12);
```

Das Antwortzeitverhalten läßt sich u.U. verbessern, wenn man folgenden Teil der Qualifikation in die View-Definitionen aufnimmt:

```
zeitpunkt > '01/04/90 00:00:00'          AND
zeitpunkt < '03/04/90 00:00:00'
```

Wenn die Tupel nach dem Attribut zeitpunkt geclustert sind, bzw. ein passender Index auf zeitpunkt definiert ist, wird so sichergestellt, daß nur die Tupel angefaßt werden, die diese Qualifikation erfüllen. Sonst ist das abhängig von der Güte des Optimierers. Es wird dann jedoch aufwendiger, das Zeitintervall zu ändern.

10.4 Das INGRES-Datenbanksystem

In diesem Abschnitt sollen die für das CSQL-System verwendeten Installationsparameter des INGRES DBS beschrieben werden. Außerdem werden einige Parameter der für die Messungen verwendeten Datenbank beschrieben. Die Werte der INGRES-Parameter setzten sich aus den Empfehlungen der Handbücher sowie den Erfahrungen verschiedener Mitarbeiter für ihre DBS-Anwendungen zusammen. Sie enthielten sicher nicht überall für das CSQL-System optimale Werte, insgesamt waren sie jedoch akzeptabel.

Die Installation umfaßte das INGRES-Grundsystem (Anfragebearbeitung und -auswertung), SQL, ISQL, Embedded SQL/C, Dynamic SQL/C, alle vorhandenen Möglichkeiten, mit Masken zu arbeiten (Forms, Embedded Forms, ABF, VISION), und die INGRES Object Management Extension. VISION und die Object Management Extension kamen jedoch erst später hinzu (anfangs gehörten sie nicht zum Lieferumfang von INGRES), so daß sie bei der Implementierung des CSQL-Systems nicht berücksichtigt werden konnten. Die hier verwendete Installation lag auf einem Rechner SUN Sparc 2 und zwei Fujitsu-Platten von ein bzw. zwei GByte Größe mit 11 msec mittlerer Zugriffszeit für Daten und Protokolle.

Für das DBS stand Platz auf zwei Platten zur Verfügung, auf einer für die Daten, auf einer für Protokolle und gelegentliche DB-Gesamtabzüge. Für die Transaktions-Protokolldatei stand ein sog. *raw device* zur Verfügung. Das ist ein zusammenhängender Plattenabschnitt, auf den das DBS unter Umgehung des Betriebssystems in dem von ihm gewünschten Format schreiben kann. Die Protokolldatei war zum Ende des Projekts 104 576 Kbytes groß. Wenn man die Relation wetter_10 betrachtet, die damals über 100 000 Tupel enthielt, war das eher klein und konnte sogar bei Änderungsoperationen, die in CSQL jedoch nicht vorgenommen wurden, zu Problemen führen. Alle anderen Parameter, wie die maximale Anzahl der Transaktionen und Sperrtabellen im System (82), die maximale Anzahl Sperren pro Transaktion (80), die maximale Anzahl Sperren im System (4656) waren für das CSQL-System immer ausreichend. Erwähnt sei noch, daß die sog. *fast commit*- und *write behind*-Optionen gesetzt waren. Das erlaubt dem DBS, die commit-Records mehrerer kleiner Transaktionen zusammenzufassen und gemeinsam hinauszuschreiben, ohne daß dadurch die Sicherheit geringer wird. Damit kann das Antwortzeitverhalten deutlich verbessert werden.

Alle Relationen der Datenbank, auf der die Messungen durchgeführt wurden, der *messdb* lagen physisch auf einer Platte. Es wäre sinnvoll gewesen zu testen, ob eine Aufteilung sehr großer Relationen in Partitionen auf mehreren Platten einen Leistungsvorteil gebracht hätte. Das war jedoch nicht möglich, da nicht auf mehreren gleichwertigen Platten Platz zur Verfügung stand. Alle Relationen wurden mit der sog. *journaling*-Option angelegt, d.h., erfolgte Änderungen wurden über das Ende einer Transaktion hinaus protokolliert, so daß die geforderte Wiederherstellbarkeit der Daten auch nach Systemzusammenbrüchen und Plattenfehlern gesichert war.

Für alle Meßwertrelationen wurde die ISAM-Datenstruktur (index sequential access method) auf dem Attribut zeitpunkt gewählt, weil laut Handbuch sich diese Datenstruktur am besten für

Relationen eignet, bei denen oft Bereichsanfragen auf dem Index-Attribut gestellt werden und auf denen nicht zu häufig Änderungen stattfinden. Näheres hierzu siehe Abschnitt 10.5. Bei der Definition der Datenstruktur konnte angegeben werden, wie weit die Blattknoten beim Aufbau der Struktur gefüllt sein sollten. Für die Messungen wurde immer angegebgen, daß die Blattknoten möglichst gut (100%) gefüllt sein sollten. Es wurde darauf geachtet, daß aktuelle Statistiken vorhanden waren und daß die Meßwertrelationen für die Messungen neu reorganisiert waren.

10.5 Leistungsmessungen

In diesem Abschnitt werden zunächst die Möglichkeiten erläutert, die INGRES bietet, um Leistungsmessungen vorzunehmen. Dann werden einige Meßergebnisse vorgestellt, die anhand einer Reihe von SQL- und CSQL-Testanfragen gewonnen wurden. Abschließend werden dann die Beispiele aus Abschnitt 10.2 gemessen.

10.5.1 Leistungsmessung im INGRES-System

Die einzige Möglichkeit, direkt im INGRES-System Leistungsmessungen vornehmen zu können, besteht darin, Variablen der INGRES-Funktion dbmsinfo() abzufragen. Diese Funktion liefert verschiedene Statusinformationen, von denen die folgenden für Leistungsvergleiche verwendet werden können [INGR89]:

- _bio_cnt: liefert die Anzahl der gepufferten Ein-/ Ausgabeanforderungen vom Benutzerprogramm für diese Sitzung. Für eine bestimmte Anfrage oder Transaktion ist dieser Wert immer gleich, unabhängig von der derzeitigen Last des Rechners oder von Speicherstrukturen.

- _dio_cnt: liefert die Anzahl der Ein-/ Ausgabeanforderungen für Daten auf Platten für diese Sitzung.

- _cpu_ms: liefert die verbrauchte CPU-Zeit dieser Sitzung (in msec).

- _et_sec: liefert die verstrichene Zeit dieser Sitzung (in sec). Diese Variable enthält auch die Pausen, in denen ein Benutzerprogramm anderes bearbeitet.

- _pfault_cnt: liefert die Anzahl der sog. “page faults”, d.h., daß keine freie Hauptspeicherseite für den INGRES-Server vorhanden ist.

Es wurden immer alle fünf Statusvariablen abgefragt, obwohl für den CSQL- (und SQL-) Endbenutzer eigentlich nur _cpu_ms und _et_sec interessant sind. Die anderen Variablen können nur Hinweise darauf geben, warum einzelne Anfragen besonders viel Zeit benötigen. Alle diese

Variablen zählen immer nach oben, definierte Anfangszustände gibt es nicht. Deshalb ist es immer notwendig, die Variablen vor und nach einer Anfrage bzw. Transaktion abzufragen und die Differenz zu bilden. Wie in [INGR89] vorgeschlagen, wurden die zu messenden Anfragen immer von folgenden Anfragen umrahmt:

```
INSERT  INTO mess_prot (v_bio_cnt, v_cpu_ms, v_dio_cnt, v_et_sec, v_pfault_cnt,
                        anfrage, lauf, ben_system, ingres_system, programm)
SELECT int4 (dbms_info ('_bio_cnt')), int4 (dbms_info ('_cpu_ms')),
        int4 (dbms_info ('_dio_cnt')), int4 (dbms_info ('_et_sec')),
        int4 (dbms_info ('_pfault_cnt')),
        '<Anfrage-Bezeichnung>', <Anfrage-Nr>, 'phoenix', 'phoenix', '<Programmname>'
FROM    one_row_rel;
<zu messende Anfrage / Transaktion>
UPDATE mess_prot
SET     n_bio_cnt     = int4 (dbmsinfo ('_bio_cnt')),
        n_cpu_ms      = int4 (dbmsinfo ('_cpu_ms')),
        n_dio_cnt     = int4 (dbmsinfo ('_dio_cnt')),
        n_et_sec      = int4 (dbmsinfo ('_et_sec')),
        n_pfault_cnt  = int4 (dbmsinfo ('_pfault_cnt'))
WHERE   anfrage       = '<Anfrage-Bezeichnung>'     AND
        lauf          = <Anfrage-Nr>                AND
        programm      = '<Programmname>';
```

Die Relation mess_prot (Meßprotokoll) ist vom Benutzer selbst anzulegen und kann neben den eigentlichen Meßwerten und den Angaben, die notwendig sind, um eine Anfrage beim UPDATE und später wieder eindeutig zu identifizieren, noch beliebige weitere Angaben enthalten. Hier sind jeweils die fünf vorher und nachher gemessenen Werte enthalten (z.B. v_cpu_ms und n_cpu_ms), die Angaben zur eindeutigen Identifizierung einer Anfrage, also die Bezeichnung der Anfrage, die Nummer des Laufs, wenn eine Anfrage mehrmals gemessen wird und das Programmsystem, mit dem gemessen wird (SQL oder CSQL). Außerdem können noch der Rechner angegeben werden, auf dem das Benutzersystem (SQL, CSQL) läuft und der Rechner, auf dem die Datenbank liegt. Für diese Arbeit wurden jedoch nur Messungen auf einem Rechner durchgeführt. Zur Betrachtung der Meßergebnisse wird dann am besten eine Sicht definiert, die die Differenzen aus den Vorher-/ Nachher-Werten bildet und evtl. uninteressante Informationen ausblendet.

Diese Meßanfragen können sowohl von SQL und ISQL als auch von CSQL in der interaktiven und in der zeilenverarbeitenden Form bearbeitet werden. Die Anlaufzeit der Systeme wird dabei nie mitgemessen. Es empfiehlt sich, die Anfragen aus vorbereiteten Dateien einlesen zu lassen (zeilenorientierte Varianten), da sonst die Zeit zum Eintippen oder Einkopieren in _et_sec mitgemessen wird. Die Leistung, die diese Meßanfragen selbst benötigen, ist mit ca. 120 CPU msec und drei bis sechs Plattenzugriffen vernachlässigbar gering. Die insgesamt verstrichene Zeit (_et_sec) ist in der Einheit sec überhaupt nicht meßbar.

Eine andere Möglichkeit für Leistungsmessungen hätte darin bestanden, entsprechende Statusinformation des Betriebssystems aus dem CSQL-Programm heraus abzufragen und gesondert auszugeben. Das hätte jedoch eine - wenn auch geringe - Änderung der CSQL-Programme gefordert. Dann wäre auch ein direkter Vergleich mit SQL-Anfragen, die über die Original-SQL-Schnittstelle gestellt werden, nicht möglich gewesen. Deshalb wurden die Messungen allein mit den INGRES-Statusvariablen durchgeführt.

10.5.2 Antwortzeitverhalten im CSQL-System

Um das Antwortzeitverhalten des CSQL-Systems zu untersuchen, wurden zunächst einmal verschiedene - nicht unbedingt immer sehr sinnvolle - SQL- und CSQL-Anfragen gemessen, um herauszufinden, welche Faktoren die Meßergebnisse wie stark beeinflussen. Wo möglich, wurden die Anfragen auch mit SQL gemessen, um vergleichen zu können, wie stark CSQL die Anfragebearbeitung verzögert. Die Anfragen und die Meßergebnisse im einzelnen sind im Anhang aufgeführt, hier werden die Ergebnisse nur tendenziell beschrieben. Im einzelnen wurden folgende Beobachtungen gemacht:

- In früheren INGRES-Versionen gab es bei Messungen der Autorin und eines Diplomanden [Diet89] z.T. sehr starke Schwankungen, wenn dieselbe Anfrage unter denselben Randbedingungen mehrfach ausgeführt wurde. Es gibt noch immer Schwankungen in den Meßergebnissen (außer bei '_bio_cnt'); die sind aber relativ gering. Keinesfalls mehr braucht dieselbe Anfrage bei einem Lauf ein Vielfaches der Zeit wie bei einem anderen Lauf. Auch die sonstige Last des Rechners wirkt sich - selbst bei '_et_sec' - kaum auf das Meßergebnis aus.

- In [Diet89] wurde auch festgestellt, daß dieselbe Anfrage im zweiten (und jedem weiteren) Lauf hintereinander deutlich schneller bearbeitet wurde als beim ersten Lauf. Als Ursache dafür wurde herausgefunden, daß INGRES auf noch vorhandene Daten im UNIX-Dateipuffer und im INGRES local cache zurückgegriffen hat und so die Zugriffszeit verkürzt wurde. Bei den Messungen für diese Arbeit wurden keine deutlichen Unterschiede zwischen dem ersten und weiteren, direkt folgenden Läufen einer Anfrage festgestellt. Bezüglich des INGRES local cache läßt sich dazu sagen, daß dieser Puffer bei jedem erneuten Aufruf von SQL oder CONNECT bei ESQL neu initialisiert wird, was bei den jetzigen Messungen immer geschehen ist. Bezüglich des UNIX-Dateipuffers läßt sich nur vermuten, daß entweder INGRES seine Zugriffsstrategie so verändert hat, daß der alte Inhalt dieses Puffers keine Rolle mehr spielt. Oder die Zeit, die benötigt wird, um eine Seite in den Dateipuffer zu lesen, wird gegenüber der gesamten Antwortzeit vernachlässigbar gering. Denn obwohl bei Anfragen mit vorheriger Dateipufferleerung tendenziell mehr page faults auftreten, hat das keine Auswirkung auf die Antwortzeit. (Für Details siehe Tabelle A-1 im Anhang, Lauf 1 bis 4 mit Dateipufferleerung im Vergleich zu Lauf 5 und 6 ohne Dateipufferleerung.)

- Wie oben beschrieben, hatte zwar die allgemeine Last des Rechners kaum einen Einfluß auf die Meßergebnisse, wenn jedoch auf derselben Datenbank parallel gearbeitet wird, wirkt sich das leicht negativ auf das Antwortzeitverhalten aus. Die reine CPU-Zeit bleibt zwar gleich,

jedoch steigt die Anzahl der page faults leicht, so daß die Antwortzeit insgesamt etwas steigt. (Siehe Tabelle A-2 im Anhang, Lauf 10 bis 12 im Vergleich zu den vorhergehenden Läufen 1 bis 6.) Bei parallel laufenden reinen Anfragen auf derselben Relation läßt sich bei SQL ebenfalls eine leichte Verschlechterung des Antwortzeitverhaltens feststellen (siehe Lauf 14). Im CSQL-System kann es bei parallelen Anfragen auf derselben Relation zu Verklemmungen kommen, weil hier, bedingt durch das Einrichten von temporären Zwischenrelationen z.Zt. die Sperren noch etwas restriktiver gesetzt werden.

- In den INGRES-Handbüchern wird häufig darauf hingewiesen, daß die Befehle ‘optimizedb’ (Aktualisierung der Statistiken für die Anfrageoptimierung) und ‘sysmod’ (Optimierung der Systemtabellen) Laufzeitverbesserungen bewirken können. Statistiken waren für die verwendete Meßdatenbank schon lange eingerichtet. Die Anwendung der oben genannten Befehle hat auch nach einigen Veränderungen in der Datenbank zu Meßzwecken keine deutlichen Verbesserungen gebracht. Sie sind also nur nach wirklich großen Veränderungen erforderlich. Andererseits laufen diese Befehle sehr schnell, so daß man sie ruhig einmal mehr laufen lassen kann. (Für Details siehe Lauf 30 und 31 im Vergleich zu Lauf 1 bis 6, Tabelle A-3 im Anhang.)

- Einen deutlichen Einfluß auf die Antwortzeit hat jedoch die richtige Datenstruktur. Zu Testzwecken wurden einige der Anfragen (anfr1, anfr1c, anfr2d, anfr3, anfr3a, anfr3b) auf Relationen laufen gelassen, die zuvor in eine für Bereichsanfragen wenig geeignete Struktur, die Hash-Struktur (bezüglich Seitenfüllgrad etc. mit den INGRES defaults) transformiert. Im günstigsten Fall (anfr1) erfährt der Benutzer hier eine durchschnittlich 50prozentige Verlängerung seiner Antwortzeit; im schlechtesten Fall (anfr3) muß er 15 (SQL) bis 20 (CSQL) mal so lange auf seine Antwort warten. (Für Details siehe Lauf 20 und 21 im Vergleich zu Lauf 1 bis 6, Tabelle A-4 im Anhang.)
  Interessanterweise läßt sich hier eine deutliche Abnahme der page faults im zweiten Lauf gegenüber dem ersten feststellen. Dies hat jedoch nur einen geringen Einfluß auf die benötigte CPU-Zeit und einen kaum erkennbaren Einfluß auf die gesamte Antwortzeit.

- Als letztes wurde noch der Einfluß der Größe der Relationen auf das Antwortzeitverhalten untersucht. Alle übrigen Versuche liefen auf den Relationen wetter_60 (anfr1, anfr2) mit 30 262 Tupeln und wetter_10 (anfr3) mit 181 727 Tupeln. Aus der Relation wetter_60 wurden drei Abschnitte ausgewählt: 120 Tupel, das entspricht fünf Tagen für Lauf 1; 1487 Tupel, das entspricht zwei Monaten für Lauf 2; 4367 Tupel, das entspricht sechs Monaten für Lauf 3. Die gefragten Tupel lagen jeweils in der Mitte dieser Abschnitte. Aus der Relation wetter_10 wurden zwei Abschnitte ausgewählt: 22 025 Tupel, das entspricht fünf Monaten für Lauf 1; 52 385 Tupel, das entspricht einem Jahr für Lauf 2. (Kleinere Bereiche konnten hier nicht gewählt werden, weil die Anfragen einen Zeitraum von fünf Monaten umspannen.) Bei den meisten getesteten Anfragen (anfr1a, anfr2a, anfr3, anfr3a) hat die Relationengröße keine erkennbaren Auswirkungen auf die Antwortzeit. Denn die Speicherstrukturen wurden so gewählt, daß mit wenigen Seitenzugriffen auf die Tupel mit den gesuchten Werten zugegriffen werden konnte. Bei den komplizierteren CSQL-Anfragen (anfr2c, anfr3b) kann man jedoch

eine direkte Abhängigkeit zwischen der Relationengröße und der Antwortzeit erkennen. Hier muß also trotz geeigneter Speicherstrukturen mehr gesucht und bearbeitet werden. (Für Details siehe Vergleich normaler mit Mini-Relationen, Läufe 1m, 2m, 3m, in Tabelle A-5 im Anhang.)

Natürlich sollte auch herausgefunden werden, wie sehr das CSQL-System die Bearbeitung reiner SQL-Anfragen verzögert. Deshalb wurden die SQL-Anfragen (anfr1, anfr1a) auch mit dem CSQL-System gemessen. Das CSQL-System benötigt hier jeweils etwa doppelt so lange wie SQL, siehe Tabelle A-6 im Anhang. (Die Gründe hierfür sind in Kap. 8.9 und 9 beschrieben.)

Bei der Messung der Beispielanfragen wurde aufgrund der oben beschriebenen Ergebnisse vor allem auf geeignete Datenstrukturen geachtet. Alle anderen Randbedingungen haben keinen größeren Einfluß und wurden während der Messungen auch nicht verändert.

10.5.3 Messung der Beispielanfragen

Die Beispielanfrage 1 lief sowohl im SQL- als auch im CSQL-System in der jeweiligen Form schnell. Ein anderes Ergebnis war auch nicht zu erwarten, da die Anfragen neben einigen einfachen Bereicheinschränkungen nur jeweils eine Interpolation (CSQL) bzw. eine Aggregation (reines SQL) enthalten. Die Ergebnisse im einzelnen lauten:

bio_cnt	cpu_ms	dio_cnt	et_sec	pfault_cnt	Programm
6	1020	32	2	0	SQL
165	3800	262	9	0	CSQL

Damit benötigt die CSQL-Anfrage ca. viermal so viel Zeit wie die SQL-Anfrage. Wie in Kap. 10.5.2 herausgefunden, benötigen auch SQL-Anfragen im CSQL-System die doppelte Zeit. Die weitere zeitliche Verzögerung ist dadurch bedingt, daß für die Interpolation die Anfrage in mehrere Anfragen aufgeteilt wird (siehe Kap. 10.2).

Die Beispielanfrage 2 wurde jeweils zweimal gemessen, einmal ohne (Lauf 1) und einmal mit der in Kap. 10.3 beschriebenen Einschränkung der Tiefe, die vor allem die Menge der Ausgabetupel einschränkt. Zusätzlich wurde im CSQL-System noch eine Variante der Anfrage mit nur einer Interpolation (ohne `STEP`-Angabe) auch ohne Tiefeneinschränkung gemessen.

bio_cnt	cpu_ms	dio_cnt	et_sec	pfault_cnt	Lauf	Programm
17	5170	438	11	0	1: —	SQL
6	490	39	1	0	2: tiefe < 20	SQL
600	5230	688	18	0	1: —	CSQL
585	5310	658	16	0	2: tiefe < 20	CSQL
167	1970	234	6	0	3: ohne STEP	CSQL

Es fällt auf, daß die SQL-Anfrage durch die Tiefeneinschränkung deutlich gewinnt. Daraus kann man schließen, daß der Optimierer erkennt, daß diese Einschränkung vorgezogen ausgewertet werden kann und so den Umfang der notwendigen, durch die Berechnungen sehr teure Verbundoperation deutlich einschränkt. Im CSQL-System fällt dieser Vorteil weniger deutlich aus: Für die Interpolation ist die Tiefe in der anderen Relation ohnehin irrelevant. Und die Zwischenrelation mit den interpolierten Werten ist klein und der Verbund mit ihr ohne komplizierte Berechnungen durchzuführen. Ohne Tiefenbeschränkung kostet damit die CSQL-Anfrage nicht ganz doppelt so viel wie die entsprechende SQL-Anfrage, mit Tiefenbeschränkung fast das 16fache. Um zu sehen, wie teuer eine Interpolation ist, wurde Lauf 3 ohne STEP-Werte (und ohne die Einschränkung der Tiefe) gemessen. Hier sieht man, daß bereits die erste Interpolation relativ teuer ist. Die weiteren Interpolationen dagegen sind relativ günstig zu haben.

Die Beispeilanfrage 3 wurde jeweils mit drei verschiedenen Zeitintervallen gemessen; und zwar über zwei Tage (Lauf 1), über eine Woche (7 Tage, Lauf 2) und über einen Monat (30 Tage, Lauf 3). Die SQL-Anfrage wurde jeweils zweimal gemessen, einmal mit der zeitlichen Einschränkung in den Sichten (a) und einmal mit der zeitlichen Einschränkung in der Anfrage (b).

bio_cnt	cpu_ms	dio_cnt	et_sec	pfault_cnt	Lauf	Programm
10	930	96	4	15	1a: 2 Tage	SQL
10	1540	105	3	0	2a: 7 Tage	SQL
11	4630	158	8	15	3a: 30 Tage	SQL
10	31820	880	40	25	1b: 2 Tage	SQL
10	33520	937	46	0	2b: 7 Tage	SQL
11	35250	908	50	14	3b: 30 Tage	SQL
3719	517360	22233	750	75	1: 2 Tage	CSQL
11919	1779920	76420	2628	0	2: 7 Tage	CSQL
49641	7307530	324300	12936	457	3: 30 Tage	CSQL

Die zweite, für den Benutzer komfortablere SQL-Version (b) benötigt etwa zehnmal so viel Zeit wie die erste (a). Die CSQL-Version braucht dann noch einmal 10 bis 250 mal so lange wie die SQL-Version b. Die Gründe dafür sind, daß in CSQL sechs voneinander unabhängige Interpolationen ablaufen und die Zahl der Plattenzugriffe dadurch erheblich steigt. Dafür hat der Benutzer auch die Sicherheit, daß er in jedem Intervall auch genau einen Wert bekommt.

Man sieht hier, daß in der CSQL-Anfrage die Laufzeit liniear abhängig ist von der Anzahl der bearbeiteten Tupel bzw. der durchgeführten Interpolationen. In Lauf 3 werden 15 mal so viele Tupel angefaßt und Interpolationen durchgeführt und die Laufzeit ist auch fast 15 mal so lang. Bei den SQL-Anfragen läßt sich eine so klare Formel nicht sofort angeben. Die Laufzeit steigt zwar mit der Anzahl der angefaßten Tupel und durchgeführten Vergleichsoperationen, aber nicht so stark wie diese. Zieht man jedoch in der Anfrage (a) einen Sockelbetrag von ca. 600 CPU-msec ab, steigt die Laufzeit wieder linear mit der Anzahl der angefaßten Tupel. In der Anfrage (b) läßt sich auch so eine Abhängigkeit nicht nachvollziehen. Hier wird vermutlich die meiste Zeit benötigt, um alle Tupel zu lesen und zu sehen, ob sie im gefragten Intervall liegen oder nicht. Die Weiterverarbeitung der gefundenen Tupel schlägt nur noch mit relativ geringem Aufwand zu Buche.

10.6 Bewertung der Ergebnisse

Die Auswertung und Messung der Beispielanfragen hat gezeigt, daß sich der Einsatz von CSQL bei entsprechenden Problemstellungen in den meisten Fällen für den Benutzer lohnt. In der Beispielanfrage 1 kostet schon ein weiterer Probelauf den Benutzer weit mehr Zeit als die ca. sieben Sekunden, die CSQL länger braucht. In der Beispielanfrage 2 spricht allein schon die Qualität der Ergebnisse (siehe Vergleich in Tab. 10.1) für den Einsatz von CSQL. Wenn man bedenkt, daß auch ein geübter Benutzer hier in der SQL-Version mindestens elf Probeläufe braucht, um geeignete Einschränkungen zu finden, ist eine zwei bis 16 mal so lange Antwortzeit (im sec-Bereich) durchaus noch ein Vorteil.

In der Beispielanfrage 3 sieht man, daß hier die bessere Qualität der Ergebnisse mit einer deutlich längeren Antwortzeit bezahlt werden muß. Bei den vorliegenden, überwiegend vollständigen Daten ist auch die Zeit, die der Benutzer zum Formulieren und Testen der komplizierteren SQL-Transaktion benötigt, kürzer als die CSQL-Antwortzeit. Bei sehr unvollständigen Daten kann das anders aussehen. Allerdings kann die Antwortzeit von dreieinhalb Stunden die Geduld des Benutzers schon auf eine harte Probe stellen. Eine so lange Antwortzeit ist nur vertretbar, wenn wirklich qualitativ sehr gute Werte für die Weiterverarbeitung benötigt werden oder wenn die SQL-Anfrage aufgrund fehlender Werte (oder zu unregelmäßiger Messungen) keine brauchbaren Ergebnisse liefert.

Insgesamt läßt sich feststellen, daß CSQL gerade für den in SQL unerfahrenem Benutzer, der im Formulieren von komplizierten Anfragen wenig Übung hat, oder für einen Benutzer, der mit den vorliegenden Meßdaten nicht vertraut ist, also nichts oder wenig über deren Meßabstände, Vollständigkeit und dergleichen weiß, eine große Hilfe ist. Die CSQL-Antwortzeit mag im Einzelfall lang erscheinen, ein mehrfaches Probieren und Umformulieren von SQL-Anfragen dauert aber noch länger. Somit bietet CSQL gerade für die Benutzer aus Anwenderprojekten, die auch mit Daten arbeiten müssen, die sie nicht selbst erhoben haben, eine geeignete Hilfe.

11. Zusammenfassung und Ausblick

Diese Arbeit begann mit dem interdisziplinären Forschungsprojekt “Naturmeßfeld ‘Horkheimer Insel’”. Doch gegen Ende des Projekts war bereits abzusehen, daß einige der Punkte, auf die sich das DB-Teilprojekt zunächst konzentriert hatte, schon bald durch kommerzielle Lösungen abgedeckt sein würden. Z.B. ist heute der Einsatz kommerzieller verteilter DBS für ähnliche Aufgaben Stand der Technik und nicht mehr der Forschung vorbehalten. Auch einige der damals technischen und organisatorischen Probleme, wie die physische Vernetzung aller Projektstandorte, würden heute im Zeitalter des schnell expandierenden Internet so nicht mehr auftreten.

Für andere Schwerpunkte des Projekts, nämlich die Aufbereitung und Verarbeitung der Meßwerte, war die Entwicklung von kommerziellen Produkten nicht so günstig. Nach Gesprächen mit den Anwendern aus den unterschiedlichen Fachgebieten stellte sich bald heraus, daß folgende Punkte die wichtigsten Anliegen aller Beteiligten waren: das Schließen von (kleinen und größeren) Meßlücken, das Vergleichen von Meßwerten auf verschobener Meßbasis und das Erzeugen von Meßreihen mit äquidistanten Werten aus vorhandenen Meßreihen mit unterschiedlichen oder anderen Abständen. Die Lösung dieser Probleme wurde Schwerpunkt des DB-Teilprojekts und damit Gegenstand dieser Arbeit.

Prinzipiell lassen sich diese Probleme alle durch Interpolation aus den vorhandenen Meßwerten lösen. Geeignete Interpolationsverfahren sind i.a. bekannt, siehe z.B. [FrTh84]. Die Aufgabe des Datenbank-Teilprojekts war es nun, diese Interpolation in ein DBS zu integrieren, und zwar mit den folgenden Randbedingungen:

(1) Es wird ein kommerzielles relationales DBS verwendet, das zu Beginn des Projekts Stand der Technik gewesen ist und von dem weitere Verbesserungen zu erwarten sind.

(2) Der Endbenutzer soll die Interpolation direkt in seine Datenbank-Anfrage integrieren können. Das System soll ihm dabei soviel Hilfe wie möglich geben.

Um den Punkt (2) zu erfüllen, wurde zunächst festgelegt, was Interpolation innerhalb einer Datenbank-Anfrage bedeuten kann und dann wurde die SQL-Spracherweiterung CSQL entwickelt. Hierbei wurde davon ausgegangen, daß ein SQL-gewohnter Benutzer nicht zuviele neue Klauseln hinzulernen möchte und daß er weiterhin - wie bei reinem SQL - seine Anfragen in Anleh-

nung an die natürliche Sprache formulieren möchte. Wegen der Randbedingung (1) mußte die Implementierung als Präprozessor erfolgen. Das hatte aber auch den Vorteil, daß die anderen Forderungen der Anwender, insbesondere bezüglich Datenkonsistenz und -sicherheit durch den vollen Funktionsumfang eines relationalen DBS automatisch erfüllt waren.

Die Implementierung des CSQL-Systems erfolgte dann im wesentlichen im Rahmen von studentischen Arbeiten [Bosc90, Jahk91, Riet90], wobei auch die ersten Interpolationsfunktionen selbst mit implementiert wurden [Fabr91]. Mit diesem ersten Prototypen sollte gezeigt werden, daß dieser Ansatz prinzipiell für Anwender nützlich und hilfreich ist, daß ein akzeptables Antwortzeitverhalten möglich ist; und es sollten Schwächen und Verbesserungsmöglichkeiten aufgezeigt werden.

Nachdem durch die Fertigstellung eines funktionsfähigen ersten Prototypen gezeigt worden war, daß dieser Ansatz prinzipiell funktioniert, fiel auch gleich der erste Nachteil auf: Die Implementierung der Interpolationsfunktionen ist für einen mit den Interna eines DBS nicht vertrauten Anwender zu kompliziert. Das führte einerseits zu den in Kap. 7 gemachten Vorschlägen für standardisierte, vordefinierte Schnittstellen zum DBS und andererseits zu dem im Kap. 9 vorgestellten zweiten Ansatz. Dieser konnte unter den gegebenen Voraussetzungen, die Verwendung eines kommerziellen DBS ohne Quellcode, nicht implementiert werden, da eine Präprozessorlösung nicht möglich war.

So konnten Leistungsmessungen auch nur für das CSQL-System durchgeführt werden. Die Ergebnisse hier (siehe Kap. 10 und Anhang) waren für einen Forschungsprototypen durchaus zufriedenstellend, außer für komplizierte Anfragen mit mehreren Interpolationen. Solange jedoch mit einem Präprozessorsystem gearbeitet werden muß, lassen sich einige wichtige Optimierungen in der Anfragebearbeitung nicht durchführen.

Interessant wäre jetzt herauszufinden, auf welche Art man die erwähnten Nachteile am einfachsten abstellen kann. Ob man CSQL oder den zweiten Ansatz wählt, hängt sicher im wesentlichen davon ab, ob man für den Endbenutzer eine einfache Schnittstelle wünscht (CSQL) oder ob man den Programmierer der Interpolationsfunktionen entlasten will (zweiter Ansatz). Wichtig ist dabei als erstes, daß für einen neuen Prototypen gewisse Software-Voraussetzungen geschaffen werden. Entweder sollte der Quellcode eines relationalen DBS (eher kommerziell, jedoch Stand der Technik) vorliegen oder es sollte ein objekt-relationales erweiterbares DBS [StBM99] verwendet werden wie etwa Informix Universal Server [Info96], die kommerzielle Weiterentwicklung von POSTGRES.

Die Verwendung eines für Erweiterungen vom Anwender vorgesehenen DBS wie der Informix Universal Server hätte außerdem weitere Vorteile: Die Schnittstellen sind genau definiert; damit werden Änderungen und das Anpassen an andere DBS einfacher. Der Aufwand wird geringer, weil einige Teile der Präprozessor-Lösung, etwa die aufgesetzte Benutzerschnittstelle entfallen. Insgesamt entsteht also mit weniger Aufwand ein besseres Produkt.

Läge der Quellcode eines DBS vor, ließen sich natürlich alle möglichen Erweiterungen ergänzen. Jedoch je mehr ergänzt wird, desto größer wird auch der Aufwand. Bei der Integration von CSQL in das DBS könnte man insbesondere bei der Anfrageoptimierung einige Verbesserungen erzielen. Zunächst einmal sollte man überprüfen, ob die Annahme, daß frühes Interpolieren immer günstig ist, richtig ist (ähnlich den Betrachtungen, die in [HeSt93, Hell98] gemacht werden). Außerdem kann dann auch in der inneren Anfrage einer geschachtelten Anfrage interpoliert werden, so daß auch Anfragen mit Interpolation in der inneren Unteranfrage, die nicht entschachtelt werden können (siehe Kap. 8.5) bearbeitet werden können. Hier können dann Optimierungsverfahren, die mit dem Aufbewahren von Zwischenergebnissen arbeiten [HeNa96], eingesetzt werden. Wenn der zweite Ansatz dann auf demselben DBS implementiert wird, kann das Antwortzeitverhalten beider Varianten direkt verglichen werden. So läßt sich dann entscheiden, welcher Ansatz bezüglich der Leistung der erfolgversprechendere ist.

Mit einem erweiterbaren DBS ließe sich auch der zweite Ansatz ohne Veränderung des Quellcodes implementieren. Hier wäre die Realisierung des zweiten Ansatzes sogar einfacher, weil keine neuen Klauseln hinzukommen und so die Erweiterung des Parsers entfällt. Mit Hilfe der Relationentypen lassen sich die zusätzlichen Systemrelationen vermeiden. (Das gilt für beide Varianten.) Auch hier ließe sich dann ein direkter Vergleich der Leistung und des Implementierungsaufwandes durchführen.

In den letzten Jahren wurde noch eine weiteres Werkzweug entwickelt, das geeignet sein könnte, die Akzeptanz von (entsprechend angepaßten) DBS auch bei skeptischen Anwendern zu erhöhen: Innerhalb von *COM* (Component Object Model) [COM97, KiVS97] wurde die DB-Schnittstelle *OLE DB* (Object Linking and Embedding for Databases) [Blak97] entwickelt. OLE DB erlaubt den Zugriff mit SQL sowohl auf Daten innerhalb von DBS als auch auf andere Daten in tabellarischen Strukturen etwa in lesbaren Dateien, Spreadsheets, ISAM-Dateien, E-Mails und nicht-relationalen Datenbanken. Wenn nun Anwender gelegentlich auf alte Daten zugreifen müssen, (von denen genügend Sicherungskopien vorhanden sind und die nicht zu umfangreich sind,) die sich nicht bzw. nicht mehr im DBS befinden, kann OLE DB eine große Hilfe sein, denn alle DB-Operationen sind auf diesen Daten möglich aber das aufwendige Laden (und wieder Löschen) der Daten entfällt. Es müßte aber noch geklärt werden, inwieweit sich Interpolation und andere Erweiterungen etwa für räumliche Daten in OLE DB integrieren lassen bzw. damit zusammen arbeiten.

Zur Zeit sind die Aktivitäten auf den Gebieten Umweltinformatik und insbesondere Umwelt-DBS nicht mehr ganz so stark, und die Entwicklung entsprechender Lösungen und Produkte läuft hier nicht mehr so schnell wie zu Beginn des interdisziplinären Umweltforschungsprojekts. Die allermeisten Umweltprobleme sind aber weiterhin ungelöst, so daß der Bedarf für ein System wie CSQL bestehen bleibt. Die mit dem ersten Prototypen von CSQL erzielten Ergebnisse (siehe Kap. 10 und Anhang) zusammen mit dem aufgelisteten Potential an Verbesserungsmöglichkeiten und einem neuartigen objekt-relationalen DBS lassen erwarten, daß ein System zur Integration von Interpolation (auf Meßwerten) auf einem DBS realisiert werden kann, das auch den Anforderungen von Anwendern genügt, die an kommerzielle Produkte gewöhnt sind.

12. Literatur

[ABDD90] Malcolm Atkinson, Francois Bancilhon, David DeWitt, Klaus Dittrich, David Maier, Stanley Zdonik: **The Object-Oriented Database System Manifesto**; in: Datenbank-Rundbrief, Mitteilungsblatt der GI-FG Datenbanken, Ausgabe 5 - Mai 1990, pp. 28-36

[ACM91] Communications of the ACM: **Next Generation Database Systems**; Vol.34, No. 10, October 1991

[ACM96] Communications of the ACM: **Data Mining**; Vol.39, No. 11, November 1996

[AMKP85] Hamideh Afsarmanesh, Dennis McLeod, David Knapp, Alice Parker: **An Extensible Object-Oriented Approach to Databases for VLSI/CAD**; in: Proc. of the 11th Conf. on Very Large Data Bases (VLDB), Stockholm, Sweden 1985, pp. 13-24

[ANSI86] American National Standards Institute: **Database Language SQL**; Document X3.135-1986, auch verfügbar als International Standards Organization Document ISO-9075-1987(E)

[ANSI89] American National Standards Institute: **Database Language SQL with integrity enhancement**; Document X3.135-1989, auch verfügbar als International Standards Organization Document ISO-9075-1989(E)

[ANSI89a] American National Standards Institute: **Database Language: Language Embeddings**; Document X3.168-1989

[Baye85] Rudolf Bayer: **Database Technology for Expert Systems**; in: Internationaler GI-Kongress 85, Wissensbasierte Systeme, Informatik Fachberichte 112, Springer-Verlag, Berlin, Februar 1985, pp. 1-16

[BaZi95] Annegret Baumewerd-Ahlmann, Leonore Zink: **Umweltdatenbanken**; in: Umweltinformatik, Informatikmethoden für Umweltschutz und Umweltforschung, 2. Auflage, B.Page / L.M. Hilty (Hrsg.), Handbuch der Informatik Bd, 13.3, R. Oldenbourg Verlag, München 1995, pp. 101-124

[BBGS88] D.S. Batory, J.R. Barnett, J.F. Garza, K.P. Smith, K. Tsukuda, B.C. Twichell, T.E. Wise: **GENESIS: An Extensible Database Management System**; in: IEEE Transactions on Software Engineering, Vol. 14, No. 11, November 1988, pp. 1711-1730

[BDHS96] Peter Buneman, Susan Davidson, Gerd Hillebrand, Dan Suciu: **A Query Language and Optimization Techniques for Unstructured Data**; in: Proc. of the 1996 ACM SIGMOD Int. Conf. on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996, pp.505-516

[BEHL90] C. Bartilla, Th. Entenmann, K. Helfersrieder, Th. Lang, B.E. Allison, G. Kahnt, R.R. van der Ploeg: **Das PWAB-Testfeld Wasser und Boden: Transportvorgänge und Sickerwasserqualität in der ungesättigten Zone unter konventionell und umweltschonend bewirtschafteten Ackerflächen**; in: Projekt Wasser - Abfall - Boden (PWAB), Bericht über das 2. Statuskolloquium am 13. Februar 1990 in Karlsruhe, Kernforschungszentrum (KfK-PWAB 5, Juni 1990), pp. 29-48

[BiFr91] R.Bill, D. Fritsch: **Grundlagen der Geo-Informationssysteme. Band 1: Hardware, Software und Daten**; Wichmann-Verlag, Karlsruhe, 1991

[BiFr95] R.Bill, D. Fritsch: **Grundlagen der Geo-Informationssysteme. Band 2: Analysen, Anwendungen und neue Entwicklungen**; Wichmann-Verlag, Karlsruhe, 1995

[Bill90] Ralf Bill: **Zur Eignung moderner Geo-Informationssysteme für Belange der Umweltinformatik**; in: Konzeption und Einsatz von Umweltinformationssystemen, O.Günther, H. Kuhn, R. Mayer-Föll, F.J. Radermacher (Hrsg.), Proc., Informatik-Fachberichte 301, Springer-Verlag, Berlin, 1990, pp. 331-339

[Bill95] Ralf Bill: **Raumbezogene Datenverarbeitung in Umweltinformationssystemen**; in: Umweltinformatik, Informatikmethoden für Umweltschutz und Umweltforschung, 2. Auflage, B.Page / L.M. Hilty (Hrsg.), Handbuch der Informatik Bd, 13.3, R. Oldenbourg Verlag, München 1995, pp. 125-148

[BKLM90] K.-H. van Bernem, H.L. Krasemann, A. Lisken, A. Müller, S. Patzig, R. Riethmüller: **Das Wattenmeerinformationssystems WATiS**; in: [Neug90a]

[Blak97] José A. Blakeley: **Data Access for the Masses Through OLE DB**; http://www.microsoft.com/data/oledb/dataaccessmassesthrougholedb.html, 1997

[Bosc90] Monika Bosch: **Einbindung von Erweiterungen in die relationale Anfragesprache SQL zur Selektion von Meßwerten über Interpolationsverfahren (Teil 1)**; Studienarbeit Nr. 872, IPVR, Universität Stuttgart, Juli 1990

[Buis89] Laurent Buisson: **Reasoning on Space with Object-Centered Knowledge Representations**; in: Proc. Symp. on the Design and Implementation of Large Spatial Databases, Santa Barbara, Cal., July 17-18, 1989, LNCS Bd. 409, Springer-Verlag, Berlin 1990, pp. 325-344

[CaDe85] Michael J. Carey, David J. DeWitt: **Extensible Database Systems**; in: Proc. of the Islamorada Workshop on Large Scale Knowledge Bases and Reasoning Systems, Feb., 1985, Islamorada, Florida, pp. 335-352

[CaDV88] Michael J. Carey, David J. DeWitt, Scott L. Vandenberg: **A Data Model and Query Language for EXODUS**; in: Proc. of SIGMOD Int. Conf. on Management of Data, Chicago, Illinois, June 1-3, 1988

[Cain89] Leroy Cain: **SQL Grammar for LEX, YACC**; from: Internet, comp databases, Columbia Union College, Mathematical Sciences Dept., 7600 Flower Ave., WH 406, Takoma Park, Md 20912

[CDFG86] Michael J. Carey, David J. DeWitt, Daniel Frank, Goetz Graefe, Joel E. Richardson, Eugenie J. Shekita, M. Muralikrishna: **The Architecture of the EXODUS Extensible DBMS: A Preliminary Report**; Computer Science Technical Report #644, May 1986, University of Wisconsin-Madison

[CDRS86] Michael J. Carey, David J. DeWitt, Joel E. Richardson, Eugene J. Shekita: **Object and File Management in the EXODUS Extensible Database System**; in: Proc of the 12th Int. Conf. on Very Large Data Bases (VLDB), Kyoto, Japan, August 25-28, 1986, pp. 91-100

[COM97] **Introduction to COM**; http://msdn.microsoft.com/library/default.asp?URL=/library/specs/S1CF80.HTM, 1997

[Date84] C.J. Date: **A Critique of the SQL Database Language**; in: ACM SIGMOD Record, Vol. 14, No. 3, November 1984

[Date86] C.J. Date: **An Introduction to Database Systems**; Vol. 1, Fourth Edition, The System Programming Series; Addison-Wesley Publishing Company, Reading, Mass. 1986

[Davi98] Judith R. Davis: **IBM's DB2 Spatial Extender: Managing Geo-Spatial Information Within the DBMS**; White Paper, prepared for IBM Corporation, May 1998

[Diet89] Jochen R. Dieter: **Leistungsmessungen im INGRES-Datenbanksystem**; Diplomarbeit Nr. 574, IPVR, Universität Stuttgart, Mai 1989

[EgFr89] Max J. Egenhofer, Andrew U. Frank: **PANDA: An Extensible DBMS Supporting Object-Oriented Software Techniques**; in: Datenbanksysteme in Büro, Technik und Wissenschaft, Proc. BTW'89; GI/SI-Fachtagung, Zürich, March 1989, pp. 74-79

[Fabr91] Joachim Fabricius: **Interpolationsverfahren für eine Wetterdatenbank auf dem relationalen DBMS INGRES**; Diplomarbeit Nr. 738, IPVR, Universität Stuttgart, Februar 1991

[FrBa90] Andrew U. Frank, Renato Barrera: **The Fieldtree: A Data Structure for Geographic Information Systems**; in: Proc. Symp. on the Design and Implementation of Large Spatial Databases, Santa Barbara, Cal., July 17-18, 1989, LNCS Bd. 409, Springer-Verlag, Berlin 1990, pp. 29-44

[FrTh84] J. France, J.H.M. Thornley: **Mathematical Models in Agriculture - A Quantitative Approach to Problems in Agriculture and Related Sciences**; Butterworth & Co (Publichers) Ltd, 1984, Chapter 6, pp. 95-113

[Gadi88] Shashi K. Gadia: **A Homogeneous Relational Model and Query Language for Temporal Databases**; in: ACM ToDS, Vol. 13, No. 4, Dec. 1988, pp. 418-448

[GaWo87] Richard A. Ganski, Harry K.T. Wong: **Optimization of Nested SQL Queries Revisited**; in: Proc. of the 1987 Annual Conf. on Management of Data (ACM SIGMOD), San Francisco, May 27-29, pp. 23-33

[GrDe87] Götz Graefe, David J. DeWitt: **The EXODUS Optimizer Generator**; in: Proc. of the 1987 Annual Conf. on Management of Data (ACM SIGMOD), San Francisco, May 27-29, pp. 160-172

[GrRe93] Jim Gray, Andreas Reuter: **Transaction Processing: Concepts and Techniques**; Morgan Kaufmann Publishers, San Mateo, California, 1993

[Guen88] Oliver Günther: **Efficient Structures for Geometric Data Management**; Lecture Notes in Computer Science, No. 337, Springer Verlag, Berlin 1988

[Guen90] Oliver Günther: **Data Management in Environmental Information Systems**; in: Proc. 5. Symposium Informatik für den Umweltschutz, Wien, Österreich, September 1990, Informatik-Fachberichte 256, Springer-Verlag, Berlin 1990, pp. 57-66

[Gutt77] John Guttag: **Abstract Data Types and the Development of Data Structures**; in: CACM, Vol. 20, No. 6, June 1977, pp. 396-404

[Haer78] Theo Härder: **Implementierung von Datenbanksystemen**; Carl Hanser Verlag, München 1978

[Hard87] Martin Hardwich: **Why ROSE is Fast: Five Optimizations in the Design of an Experimental Database Systems for CAD/CAM Applications**; in: Proc. of ACM SIGMOD'87, San Francisco, May 27-29, 1987, pp. 292-298

[HCLM90] Laura M. Haas, Walter Chang, Guy M. Lohman, John McPherson, Paul F. Wilms, Georges Lapis, Bruce Lindsay, Hamid, Pirahesh, Michael J. Carey, Eugene Shekita: **Starburst Mid-Flight: As the Dust Clears**; in: IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1, March 1990, pp. 143-160

[Hell98] Joseph M. Hellerstein: **Optimization Techniques for Queries with Expensive Methods**; in: ACM Transactions on Database Systems, Vol. 23, No. 2, June 1998, pp. 113-157

[HeNa96] Joseph M. Hellerstein, Jeffrey F. Naughton: **Query Execution Techniques for Caching Expensive Methods**; in: Proc. of the 1996 ACM SIGMOD Int. Conf. on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996, pp. 423-434

[HeSt93] Joseph M. Hellerstein, Michael Stonebraker: **Predicate Migration: Optimizing Queries with Expensive Predicates**; in: Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of Data, Washington, DC, May 26-28, 1993, pp. 267-276

[Illu94] Illustra Information Technologies, Inc.^{TM}: **Illustra User's Guide**; Illustra Server Release 2.1, June 1994

[Info96] Informix Corp.:**Developing DataBlade Modules for INFORMIX Universal Server**; Informix White Papers, 1996

[INGR89] INGRES Technical Note #6: **Measuring Query Performance**; Release 6.1 onwards, May 1989

[INGR91] INGRES Corp.: **Using INGRES Through Forms and Menus for the UNIX and VMS Operating System**; Release 6.4, December 1991

[INGR91a] INGRES Corp.: **INGRES/SQL Reference Manual for the UNIX and VMS Operating System**; Release 6.4, December 1991

[INGR91b] INGRES Corp.: **INGRES/Embedded SQL Companion Guide for C for the UNIX Operating System**; Release 6.4, December 1991

[INGR91c] INGRES Corp.: **INGRES Object Management Extension for the UNIX and VMS Operating Systems**; Release 6.3, 6.3/03 and 6.4, December 1991

[ISO90] International Standards Organization: **(ISO working draft) Database Language SQL2 and SQL3**; internal committee document ISO/IEC JTC1/SC21 WG3 SEL-3b, dated April 1990

[JaMa86] Donald A. Jardine, Aviram Matzov: **Ontology and Properties of Time in Information Systems**; in: Proc. IFIP Working Conf. Knowledge & Data (DS-2), Algarve, Portugal 1986, pp. F1-F16

[Jahk91] Thilo Jahke: **Erweiterung des Optimierers im CSQL-System, einem System zur Bearbeitung eingebetteter Prozeduren (Interpolationsroutinen) in SQL-Anfragen**; Studienarbeit Nr. 992, IPVR, Universität Stuttgart, Mai 1991

[JaKo84] M. Jarke, J. Koch: **Query Optimization in Database Systems**; in: Computing Surveys, Vol. 16, No. 2, June 1984, pp. 111-152

[Kaja89] Manfred Kaja: **Entwurf und Implementierung einer Anwenderdatenbank (Geohydrologie) auf dem relationalen DBS Ingres (Teil 2)**; Studienarbeit Nr. 791, IPVR, Universität Stuttgart, November 1989

[Kaja90] Manfred Kaja: **Auswahl und Implementierung von Optimierungsverfahren für eingebettete Prozeduren (Interpolationsroutinen) in SQL-Anfragen**; Diplomarbeit Nr. 679, IPVR, Universität Stuttgart, Juli 1990

[KeCh90] John D. Keenan, Pao-Hsing Chang: **A Statistical Method to Determine Pollutant Sources**; in: Proc. 5. Symposium 'Informatik für den Umweltschutz', Wien, Österreich, September 1990, Informatik-Fachberichte 256, Springer-Verlag, Berlin 1990, pp. 557-565

[Kim82] Won Kim: **On Optimizing an SQL-like Nested Query**; in: ACM ToDS, Vol. 7, No. 3, September 1828, pp. 443-469

[KiVS97] Eugene Kim, Andreas Vogel, Roger Sessions: **COM vs. CORBA**; http://www.clbooks.com/interviews/COMvsCORBA.html

[KLNR90] Horst Kremers, Mark P. Line, Leonore Neugebauer, Rolf Riethmüller, Wilhelm Windhorst: **Arbeitskreis "Umweltdatenbanken" - Ziele und erste Ergebnisse**; in: Proc. 5. Symposium 'Informatik für den Umweltschutz', Wien, Österreich, September 1990, Informatik-Fachberichte 256, Springer-Verlag, Berlin 1990, pp. 1-16

[Knut73] Donald E. Knuth: **The Art of Computer Programming**, Vol. 1, **Fundamental Algorithms**, 2nd Edition Addison Wesley, Reading, Massachusetts, 1973

[Kobu88] H. Kobus: **Das PWAB-Testfeld Wasser und Boden**; in: Bericht über das 1. Statuskolloquium, February 23, Karlsruhe, Projekt Wasser-Abfall-Boden, KfK-PWAB 1, 1988, pp. 96-116

[KTGP89] H. Kobus, G. Teutsch, C. Gillbricht, T. Ptak: **Schadstofftransport im Untergrund — Erkundungs- und Überwachungsmethoden**; Zwischenbericht PW 87 044, Universität Stuttgart, Institut für Wasserbau, Wissenschaftlicher Bericht Nr. 89/18 (HG 108), März 1989

[Kueb89] Dietmar Kübler: **Entwurf und Implementierung einer Anwenderdatenbank (Geohydrologie) auf dem relationalen DBS Ingres (Teil 1)**; Studienarbeit Nr. 790, IPVR, Universität Stuttgart, November 1989

[LeLi89] Tobin J. Lehman, Bruce G. Lindsay: **The Starburst Long Field Manager**; in: Proc. of the 15th Int. Conf. on Very Large Data Bases (VLDB), Amsterdam, The Netherlands, August 22-25, 1989, pp. 375-383

[Lieb90] Sabine Liebelt: **Implementierung eines Datenbankauskunftsystems**; Studienarbeit Nr. 830, IPVR, Universität Stuttgart, Februar 1990

[LiHa90] Bruce Lindsay, Laura Haas: **Extensibility in the Starburst Experimental Database System**; in: Proc. Database Systems of the 90s, Int. Symposium Müggelsee, Berlin, Nov. 5-7, 1990, Springer-Verlag LNCS 466

[LiMP87] Bruce Lindsay, John McPherson, Hamid Pirahesh: **A Data Management Extension Architecture**; in: ACM SIGMOD Record, Vol. 16, No. 3, Dec 1987, pp. 220-226

[LiNe86] Udo W. Lipeck, Karl Neumann: **Modelling and Manipulating Objects in Geoscientific Databases**; in: Proc. of 5th Int. Conf. on ER-Approach, Dijon, France, 1986, pp. 105-124

[LSBM83] Guy M. Lohmann, Joseph C. Stoltzfus, Anita N. Benson, Michael D. Martin, Alfonso F. Cardenas: **Remotely-Sensed Geophysical Databases: Experience and Implications for Generalized DBMS**; in: ACM Sigmod'83 Proceedings of Annual Meeting, San Jose, 1983 (SIGMOD Record Vol. 13, No. 4), pp. 146-160

[LMWW79] Peter C. Lockemann, Heinrich C. Mayr, Wolfgang H. Weil, Wolfgang H. Wohllieber: **Data Abstractions for Database Systems**; in: ACM ToDS, Vol. 4, No. 1, March 1979, pp. 60-75

[Maib85] T. S. E. Maibaum: **Database Instances, Abstract Data Types, and Database Specification**; in: The Computer Journal, Vol. 28, No. 2, 1985, pp. 154-161

[Matt96] Nelson M. Mattos: **An Overview of the SQL3 Standard**; Database Technology Institute, IBM, Santa Teresa Lab, San Jose, California, July 1996

[MuPS86] A.J. Musgrave, B. Page, M. Stopp: **INFUCHS - Ein Informationssystem für Umweltchemikalien und Störfälle**; in: Informatik im Umweltschutz - Anwendungen und Perspektiven (B. Page, ed.), Oldenbourg-Verlag, München 1986, pp. 144-177

[Neug89] Leonore Neugebauer: **Extending a Database to Support the Handling of Environmental Measurement Data**; in: Proc. Symp. on the Design and Implementation of Large Spatial Databases, Santa Barbara, Cal., July 17-18, 1989, LNCS Bd. 409, Springer-Verlag, Berlin 1990, pp. 147-165

[Neug89a] Leonore Neugebauer: **Datenbankunterstützung für ein langfristiges Umweltforschungsprojekt**; in: Proc. 4. Symposium Informatik im Umweltschutz, Karlsruhe, 6.-8. November 1989, Informatik-Fachberichte 228, Springer-Verlag, Berlin 1989, pp. 231-240

[Neug90] Leonore Neugebauer: **Einbettung von Interpolationsverfahren in die Anfragesprache SQL zur Bearbeitung von Umweltmeßwerten**; in: Proc. Workshop Umweltinformatik 1990, FAW-B-91002/II, FAW Ulm, 13. Februar 1991, pp. 71-91

[Neug91] Leonore Neugebauer: **Optimization and Evaluation of Database Queries Including Embedded Interpolation Procedures**; in: Proc. acm SIGMOD'91 Int. Conf. on Management of Data, May 29-31, Denver, Colorado 1991, pp. 118-127

[Neum88] Neumann, Karl-H.: **Eine Geowissenschaftliche Datenbanksprache mit Benutzerdefinierbaren Geometrischen Datentypen**; Diss., TU Braunschweig 1988

[Orac86] Oracle Corp.: **The ORACLE Database Administrator's Guide, Version 5.1**, Oracle Part Number: 3601-V5.1, 1986

[Oren86] Jack A. Orenstein: **Spatial Query Processing in an Object-Oriented Database System**; in: ACM SIGMOD Record, Vol. 15, No. 2, June 1986, pp. 326-336

[PeSp88] Lars-Ole Pedersen, Richard Spooner: **Datenorganisation in System 9**, Bericht der Prime Wild GIS, Heerbrugg, Schweiz 1988

[PiAn86] P. Pistor, F. Anderson: **Designing a Generalized NF^2 Model With an SQL-Type Language Interface**; in : Proc 12th Int. Conf. on Very Large Data Bases (VLDB), Kyoto, Japan, August 25-28, 1986, pp. 278-285

[PiJa90] W. Pillmann, A. Jaeschke (Hrsg.): **Informatik für den Umweltschutz**; Proc. 5. Symposium, Wien, Österreich, September 1990, Informatik-Fachberichte 256, Springer-Verlag, Berlin 1990

[PlKa89] R.R. van der Ploeg, G. Kahnt: **Transportvorgänge und Sickerwasserqualität in der ungesättigten Zone unter konventionell und alternativ bewirtschafteten Akkerflächen**; 2. Zwischenbericht PW 87.043, Universität Hohenheim, Institut für Bodenkunde und Standortslehre, Mai 1989

[POST90] POSTGRES Implementation Team: **POSTGRES / POSTQUEL Manual**; Version 2, Berkeley, CA, 1990

[PTSE95] Dimitris Papadias, Yannis Theodoridis, Timos Sellis, Max J. Egenhofer: **Topological Relations in the World of Minimum Bounding Rectangles: A Study with R-trees**; in: Proc. of the 1995 ACM SIGMOD Int. Conf. on Management of Data, an Jose, California, May 22-25, 1996, pp.92-103

[Reut85] Andreas Reuter: **Datenbanksysteme für KI-Anwendungen**; Vorlesungsskript, Institut für Parallele und Verteilte Höchstleistungsrechner, Abteilung Anwendersoftware, SS 1990

[Reut88] Andreas Reuter: **Datenbanken als Grundlage für große verteilte Meß-, Kontroll-, Analyse- und Simulationssysteme**; in: Proc. GI - 18. Jahrestagung I, Vernetzte und komplexe Informatk-Systeme, Hamburg, Oktober 1988, R. Valk (Hrsg.); Informatik-Fachberichte 187, Springer-Verlag, Berlin 1988, pp. 203-215

[ReWa90] Ernst Rudolf Reichl, Winfried H. Walter: **ZOODAT - die tiergeographische Datenbank Österreichs**; in [Neug90a]

[RLMN93] Thomas C. Rakow, Michael Löhr, Frank Moser, Erich J. Neuhold, Klaus Süllow: **Einsatz von objektorientierten Datenbanksystemen für Multimedia-Anwendungen**; in: it+ti 3/93 Informationstechnik und Technische Informatik 35 (1993) 3, pp. 4-17

[Riet90] Peter Rieth: **Einbindung von Erweiterungen in die relationale Anfragesprache SQL zur Selektion von Meßwerten über Interpolationsverfahren (Teil 2)**; Studienarbeit Nr. 871, IPVR, Universität Stuttgart, Juli 1990

[RoSV95] Nick Roussopoulos, Steven Kelley, Frédéric Vincent: **Nearest Neighbor Queries**; in: Proc. of the 1995 ACM SIGMOD Int. Conf. on Management of Data, an Jose, California, May 22-25, 1995, pp.71-79

[Same90] Hanan Samet: **Hierarchical Spatial Data Structures**; in: Proc. Symp. on the Design and Implementation of Large Spatial Databases, Santa Barbara, Cal., July 17-18, 1989, LNCS Bd. 409, Springer-Verlag, Berlin 1990, pp. 193-212

[ScSc83] H.-J. Schek, M. Scholl: **Die NF^2-Relationenalgebra zur einheitlichen Manipulation externer konzeptueller und interner Datenstrukturen**; in: Sprachen für Datenbanken, Fachgespräch auf der 13. GI-Jahrestagung, Hamburg, Oktober 1983, pp. 113-133

[SCFL86] P. Schwarz, W. Chang, J. C. Freytag, G. Lohman, J. McPherson, C. Mohan, H. Pirahesh: **Extensibility in the Starburst Database System**; in: Proc. of 1986 Int. Workshop on Object-Oriented Database Systems, Asilomar, Pacific Glore, Cal., pp. 85-92

[ScWa86] H.-J. Schek, W. Waterfeld: **A Database Kernel System for Geoscientific Applications**; in: Proc. of the 2nd Int. Symp. on Spatial Data Handling, Seattle, Washington, July 1986, pp. 273-288

[SeKr90] Bernhard Seeger, Hans-Peter Kriegel: **The Buddy-Tree: An Efficient and Robust Access Method for Spatial Data Base Systems**; in: Proc. of 16th Int. Conf. on Very Large Data Bases (VLDB), August 13-16, 1990, Brisbane, Australia, pp. 590-601

[Sell87] Timos K. Sellis: **Efficiently Supporting Procedures in Relational Database Systems**; in: SIGMOD Record, Vol. 16, No. 3, Dec. 1987, pp. 220-226

[SeSW96] Peter G. Selfridge, Divesh Srivastava, Lynn O. Wilson: IDEA: **Interactive Data Exploration and Analysis**; in: Proc. of the 1996 ACM SIGMOD Int. Conf. on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996, pp.24-34

[SeVK98] Roger Sessions, Andreas Vogel, Eugene Kim: **COM vs. CORBA**; computerliteracy, http://www.clbooks.com/interviews/COMvsCORBA.html,1998

[Shaw90] Phil Shaw: **Database Language Standards: Past, Present, and Future**; in: Proc. Database Systems of the 90s, Int. Symp. Müggelsee, Berlin, Nov. 5-7, 1990, Springer-Verlag LNCS 466

[SnAh85] Richard Snodgrass, Ilsoo Ahn: **A Taxonomy of Time in Databases**; in: Proc. of ACM-SIGMOD'85 Int. Conf. on Management of Data, Austin, Texas, 1985, pp. 236-246

[SPSW90] Hans-Joerg Schek, Heinz-Bernhard Paul, Marc H. Scholl, Gerhard Weikum: **The DASDBS Project: Objectives, Experiences, and Future Prospects**; in: IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1, March 1990, pp. 25-43

[StAH87] Michael Stonebraker, Jeff Anton, Eric Hanson: **Extending a Database System with Procedures**; in: ACM ToDS, Vol. 12, No. 3, September 1987, pp. 350-376

[StBM99] Michael Stonebraker, Paul Brown, Dorothy Moore: **Object-Relational DBMSs: Tracking the Next Great Wave**; 2nd Edition, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1999, ISBN 1-55860-452-9

[StOl93] Michael Stonebraker, Michael Olson: **Large Object Support in POSTGRES**; in: Proc. of the 9th Int. Conf. on Data Engineering, April 19-23, 1993, Vienna, Austria, pp. 355-362

[Ston97] Michael Stonebraker: **Architectural Options for Object -Relational DBMSs**; Informix White Paper, http://www.informix.com, California, 1997

[StRG83] Michael Stonebraker, Brad Rubenstein, Antonin Guttman; **Application of Abstract Data Types and Abstract Indices to CAD Data Bases**; in: ACM SIGMOD/SIGDBP Proc. of Annual Meeting, Engineering Design Applications, San Jose, May 1983, pp. 107-113

[StRH90] Michael Stonebraker, Lawrence A. Rowe, Michael Hirohama: **The Implementation of POSTGRES**; in: IEEE Transactions on Knowledge and Data Engineering, Vol. 2, No. 1, March 1990, pp. 125-142

[StRo86] Michael Stonebraker, Lawrence A. Rowe: **The Design of POSTGRES**; in: Proc. of ACM SIGMOD'86 Int. Conf. on Management of Data, Washington D.C., 1986, pp. 340-355

[SUN88] SUN Microsystems User Manuals: **LEX**; SUN Release 4.1, December 1, 1988

[SUN89] SUN Microsystems User Manuals: **YACC**; SUN Release 4.1, February 1, 1989

[Teut89] G. Teutsch et.al.: **The Experimental Field Site 'Horkheimer Insel': Research Program, Instrumentation and First Results**; 1989, eingereicht zur Veröffentlichung

[Teut90] G. Teutsch: **Das PWAB-Testfeld Wasser und Boden: Stofftransport im Untergrund, Erkundungs- und Überwachungsmethoden**; in: Projekt Wasser - Abfall - Boden (PWAB), Bericht über das 2. Statuskolloquium am 13. Februar 1990 in Karlsruhe, Kernforschungszentrum (KfK-PWAB 5, Juni 1990), pp. 17-28

[Ullm91] Jeffry D. Ullman: **A Comparison Between Deductive and Object-Oriented Database Systems**; in: Deductive and Object-Oriented Databases, Proc. of 2nd Int. Conf. DOOD'91, Munich, Germany, December 1991, C. Delobel, M. Kifer, Y. Masunaga (eds.), Springer-Verlag, Berlin, LNCS 566, pp. 263-277

[Wege91] K. Meyer-Wegener: **Multimedia-Datenbanken**; Leitfäden der angewandten Informatik, B.G. Teubner, Stuttgart 1991

[Wolf90] Andreas Wolf: **The DASDBS GEO-Kernel, Concepts, Experiences, and the Second Step**; in: Proc. Symp. on the Design and Implementation of Large Spatial Databases, Santa Barbara, Cal., July 17-18, 1989, LNCS Bd. 409, Springer-Verlag, Berlin 1990, pp. 67-88

[WSSH88] P. F. Wilms, P. M. Schwarz, H.-J. Schek, L. M. Haas: **Incorporating Data Types in an Extensible Database Architecture**; in: Proc. of 3rd Int. Conf. on Data and Knowledge Bases, Jerusalem, Israel, June 1988, pp. 180-192

[WSSM89] W. Windhorst, W. Schaefer, A. Salski, M. Meyer: **Erfassung, Verwaltung und Auswertung von Daten im Projektzentrum Ökosystemforschung der Christian-Albrechts-Universität zu Kiel**; in: Proc. 4. Symposium Informatik im Umweltschutz, Karlsruhe, 6.-8. November 1989, Informatik-Fachberichte 228, Springer-Verlag, Berlin 1989, pp. 251-272

Anhang:
Testanfragen und Meßergebnisse

Hier werden alle Testanfragen und Meßergebnisse aufgeführt, aus denen die allgemeinen Aussagen zur Laufzeit von CSQL und SQL im Kap. 10.5.2 abgeleitet wurden. Alle diese Testanfragen laufen auf den Wetterdaten, den Relationen wetter_60 und wetter_10, weil diese mit 30 262 bzw. 181 727 Tupeln eine realitätsnahe Größe haben. Zunächst werden hier alle Anfragen aufgeführt:

Eine Anfrage, die die Werte eines Jahres selektiert:

```
anfr1:
SELECT zeitpunkt, lufttempr
FROM   wetter_60
WHERE  zeitpunkt > '31/12/1989 23:59:59'   AND
       zeitpunkt < '01/01/1991 00:00:00';

nur CSQL, anfr1c:
SELECT zeitpunkt, lufttempr
FROM   wetter_60
WHERE  zeitpunkt > '31/12/1989 23:59:59'   AND
       zeitpunkt < '04/01/1991 00:00:00';
```

Eine Anfrage, die die Werte von drei Tagen selektiert:

```
anfr1a:
SELECT zeitpunkt, lufttempr
FROM   wetter_60
WHERE  zeitpunkt > '31/12/1989 23:59:59'   AND
       zeitpunkt < '04/01/1990 00:00:00';
```

Die verschiedenen Varianten der Anfrage 2 sind alle CSQL-Anfragen. Die Anfragen anfr2 und anfr2a behandeln überwiegend Treffer, d.h. Werte, die so in der Datenbank existieren, während die Anfragen anfr2c und anfr2d jeden Wert wirklich interpolieren müssen.

```
anfr2:
SELECT zeitpunkt, lufttempr
FROM   wetter_60   BY METHOD lufttempr (lufttempr)
                   ENTRY (zeitpunkt = '01/01/1990 00:26:15')
                   STEP ('2 hours 0 minutes 0 seconds')
WHERE  zeitpunkt > '31/12/1989 23:59:59'        AND
       zeitpunkt < '01/01/1991 00:00:00';

anfr2a:
SELECT zeitpunkt, lufttempr
FROM   wetter_60   BY METHOD lufttempr (lufttempr)
                   ENTRY (zeitpunkt = '01/01/1990 00:26:15')
                   STEP ('2 hours 0 minutes 0 seconds')
WHERE  zeitpunkt > '31/12/1989 23:59:59'        AND
       zeitpunkt < '01/02/1990 00:00:00';

anfr2c:
SELECT zeitpunkt, lufttempr
FROM   wetter_60   BY METHOD lufttempr (lufttempr)
                   ENTRY (zeitpunkt = '01/01/1990 00:30:00')
                   STEP ('2 hours 0 minutes 0 seconds')
WHERE  zeitpunkt > '31/12/1989 23:59:59'        AND
       zeitpunkt < '04/01/1990 00:00:00';

anfr2d:
SELECT zeitpunkt, lufttempr
FROM   wetter_60   BY METHOD lufttempr (lufttempr)
                   ENTRY (zeitpunkt = '01/01/1990 00:30:00')
                   STEP ('2 hours 0 minutes 0 seconds')
WHERE  zeitpunkt > '31/12/1989 23:59:59'        AND
       zeitpunkt < '01/01/1990 12:31:00';
```

Die Anfragen 3 laufen auf der großen Relation wetter_10. Die anfr3 ist die SQL-Anfrage, anfr3a eine CSQL-Anfrage mit nur Treffern und anfr3b eine CSQL-Anfrage mit echter Interpolation. Alle drei Anfragen sind geschachtelt mit Aggregation in der inneren SELECT-Klausel. Die Interpolation findet jeweils in der inneren Klausel statt.

```
anfr3:
SELECT  zeitpunkt, regen
FROM    wetter_10
WHERE   regen = ( SELECT  MAX (regen)
                  FROM    wetter_10
                  WHERE   zeitpunkt > '15/02/1990 06:00:00'     AND
                          zeitpunkt < '15/02/1990 10:00:00')
          AND
          zeitpunkt > '31/12/1989 23:59:59'     AND
          zeitpunkt < '01/06/1990 00:00:00';
```

```
anfr3a:
SELECT  zeitpunkt, regen
FROM    wetter_10
WHERE   regen = ( SELECT  MAX (regen)
                  FROM    wetter_10  BY METHOD regen (regen)
                                     ENTRY (zeitpunkt = '15/02/90 06:06:15')
                                     STEP ('1 hours 10 minutes 0 seconds')
                  WHERE   zeitpunkt < '15/02/1990 10:00:00')
          AND
          zeitpunkt > '31/12/1989 23:59:59'     AND
          zeitpunkt < '01/06/1990 00:00:00';

anfr3b:
SELECT  zeitpunkt, regen
FROM    wetter_10
WHERE   regen = ( SELECT  MAX (regen)
                  FROM    wetter_10  BY METHOD regen (regen)
                                     ENTRY (zeitpunkt = '15/02/90 06:00:00')
                                     STEP ('1 hours 10 minutes 0 seconds')
                  WHERE   zeitpunkt < '15/02/1990 10:00:00')
          AND
          zeitpunkt > '31/12/1989 23:59:59'     AND
          zeitpunkt < '01/06/1990 00:00:00';
```

Es folgen nun die Meßergebnisse in Tabellenform. Einige Meßwerte sind in unterschiedlichen Zusammenhängen mehrfach aufgeführt. Jeder Lauf hat jedoch eine (innerhalb einer Anfrage) eindeutige Nummer erhalten.

Die folgende Tabelle A-1 enthält jeweils sechs Läufe, von denen die ersten vier mit Dateipufferleerung und die Läufe fünf und sechs ohne Dateipufferleerung abgelaufen sind. (Die Anfrage 2 wurde nur einmal gemessen, weil sie eine deutlich längere Antwortzeit hat als alle anderen. Die allgemeinen Schlußfolgerungen können genauso gut aus den anderen Ergebnissen abgeleitet werden.)

bio_cnt	cpu_ms	dio_cnt	et_sec	pfault_cnt	Anfrage	Lauf	Programm
80	3720	208	39	14	anfr1	1	SQL
80	4050	207	42	0	anfr1	2	SQL
80	4160	208	53	0	anfr1	3	SQL
80	3760	204	40	0	anfr1	4	SQL
80	3860	204	39	0	anfr1	5	SQL
80	3940	202	39	0	anfr1	6	SQL
26232	22340	205	110	8	anfr1	1	CSQL
26232	23730	205	102	0	anfr1	2	CSQL
26232	23570	206	99	0	anfr1	3	CSQL

bio_cnt	cpu_ms	dio_cnt	et_sec	pfault_cnt	Anfrage	Lauf	Programm
26232	22800	204	94	0	anfr1	4	CSQL
26232	22720	204	94	0	anfr1	5	CSQL
26232	23470	202	97	0	anfr1	6	CSQL
7	90	6	0	0	anfr1a	1	SQL
7	140	7	0	0	anfr1a	2	SQL
7	160	8	1	0	anfr1a	3	SQL
7	110	8	0	0	anfr1a	4	SQL
7	100	4	0	0	anfr1a	5	SQL
7	90	4	0	0	anfr1a	6	SQL
26448	22400	204	117	0	anfr1c	1	CSQL
26448	23000	205	94	0	anfr1c	2	CSQL
26448	24590	206	101	0	anfr1c	3	CSQL
26448	23750	206	102	0	anfr1c	4	CSQL
26448	24550	204	115	0	anfr1c	5	CSQL
26448	23000	204	87	0	anfr1c	6	CSQL
591021	52193940	3582928	69612	222	anfr2	1	CSQL
39921	170310	518	295	10	anfr2a	1	CSQL
39921	171140	431	276	0	anfr2a	2	CSQL
39921	173990	440	288	0	anfr2a	3	CSQL
39921	173000	495	271	0	anfr2a	4	CSQL
39921	176130	435	284	0	anfr2a	5	CSQL
39921	175540	431	283	0	anfr2a	6	CSQL
5207	469140	24961	661	1	anfr2c	1	CSQL
5207	466630	24963	617	0	anfr2c	2	CSQL
5207	475760	24952	644	0	anfr2c	3	CSQL
5207	480430	24944	649	3	anfr2c	4	CSQL
5207	473710	25000	621	0	anfr2c	5	CSQL
5207	484180	24961	772	0	anfr2c	6	CSQL
1118	94440	4973	130	0	anfr2d	1	CSQL
1118	91090	5022	122	0	anfr2d	2	CSQL
1118	93360	4960	126	0	anfr2d	3	CSQL
1118	92940	4956	130	1	anfr2d	4	CSQL
1118	93510	4961	124	0	anfr2d	5	CSQL
1118	94280	5026	203	1	anfr2d	6	CSQL
6	4310	250	7	2	anfr3	1	SQL
6	4350	249	8	43	anfr3	2	SQL
6	4230	246	7	11	anfr3	3	SQL

bio_cnt	cpu_ms	dio_cnt	et_sec	pfault_cnt	Anfrage	Lauf	Programm
6	4300	246	8	0	anfr3	4	SQL
6	4200	246	6	0	anfr3	5	SQL
6	4200	246	7	0	anfr3	6	SQL
526	6810	375	18	75	anfr3a	1	CSQL
526	7040	370	16	68	anfr3a	2	CSQL
526	6670	371	13	33	anfr3a	3	CSQL
526	6620	367	14	0	anfr3a	4	CSQL
526	6390	433	13	0	anfr3a	5	CSQL
526	6230	375	11	0	anfr3a	6	CSQL
640	148580	4563	192	2	anfr3b	1	CSQL
640	149950	4564	214	6	anfr3b	2	CSQL
640	146500	4564	205	1	anfr3b	3	CSQL
640	147620	4560	209	0	anfr3b	4	CSQL
640	146370	4558	191	0	anfr3b	5	CSQL
640	145010	4560	183	0	anfr3b	6	CSQL

Tabelle A-1: Verschiedene Läufe mit und ohne Dateipufferleerung

Die nächste Tabelle A-2 zeigt, wie sich das parallele Arbeiten auf derselben Datenbank auf die Antwortzeiten auswirkt. In den Läufen eins bis sechs wurde nicht parallel gearbeitet; in den Läufen zehn bis zwölf wurde parallel auf derselben Datenbank auf einer anderen Relation gearbeitet. Im Lauf 14 wurde parallel auf dieselbe Relation lesend zugegriffen. (Paralleles Lesen auf derselben Relation wurde in CSQL nicht gemessen, weil - bedingt durch das Anlegen temporärer Relationen - die Sperren hier restriktiver gesetzt werden und es zu Verklemmungen kommen kann.)

bio_cnt	cpu_ms	dio_cnt	et_sec	pfault_cnt	Anfrage	Lauf	Programm
80	3720	208	39	14	anfr1	1	SQL
80	4050	207	42	0	anfr1	2	SQL
80	4160	208	53	0	anfr1	3	SQL
80	3760	204	40	0	anfr1	4	SQL
80	3860	204	39	0	anfr1	5	SQL
80	3940	202	39	0	anfr1	6	SQL
80	4050	208	53	43	anfr1	10	SQL
80	4000	206	61	0	anfr1	11	SQL
80	3870	206	52	0	anfr1	12	SQL
80	4160	204	70	0	anfr1	14	SQL

bio_cnt	cpu_ms	dio_cnt	et_sec	pfault_cnt	Anfrage	Lauf	Programm
26232	22340	205	110	8	anfr1	1	CSQL
26232	23730	205	102	0	anfr1	2	CSQL
26232	23570	206	99	0	anfr1	3	CSQL
26232	22800	204	94	0	anfr1	4	CSQL
26232	22720	204	94	0	anfr1	5	CSQL
26232	23470	202	97	0	anfr1	6	CSQL
26232	21390	200	179	1	anfr1	10	CSQL
26232	21360	200	152	0	anfr1	11	CSQL
26232	21750	200	164	0	anfr1	12	CSQL
5207	469140	24961	661	1	anfr2c	1	CSQL
5207	466630	24963	617	0	anfr2c	2	CSQL
5207	475760	24952	644	0	anfr2c	3	CSQL
5207	480430	24944	649	3	anfr2c	4	CSQL
5207	473710	25000	621	0	anfr2c	5	CSQL
5207	484180	24961	772	0	anfr2c	6	CSQL
5207	459240	24926	1775	90	anfr2c	10	CSQL
5207	464320	24914	1855	10	anfr2c	11	CSQL
5207	489390	24912	2090	136	anfr2c	12	CSQL
6	4310	250	7	2	anfr3	1	SQL
6	4350	249	8	43	anfr3	2	SQL
6	4230	246	7	11	anfr3	3	SQL
6	4300	246	8	0	anfr3	4	SQL
6	4200	246	6	0	anfr3	5	SQL
6	4200	246	7	0	anfr3	6	SQL
6	4470	245	16	0	anfr3	10	SQL
526	6810	375	18	75	anfr3a	1	CSQL
526	7040	370	16	68	anfr3a	2	CSQL
526	6670	371	13	33	anfr3a	3	CSQL
526	6620	367	14	0	anfr3a	4	CSQL
526	6390	433	13	0	anfr3a	5	CSQL
526	6230	375	11	0	anfr3a	6	CSQL
526	6990	350	1786	84	anfr3a	10	CSQL

Tabelle A-2: Messungen mit und ohne Parallelzugriff auf dieselbe DB

Die nächste Tabelle A-3 enthält die Antwortzeiten nach den INGRES-Kommandos “optimizedb” und “sysmod” zur Reorganisation und Optimierung der Statistiken der Systemtabellen in den Läufen 30 und 31. Zum Vergleich können hier alle Läufe aus Tabelle A-1 dienen. Der Übersichtlichkeit halber sind nur die Läufe eins und zwei aufgeführt.

bio_cnt	cpu_ms	dio_cnt	et_sec	pfault_cnt	Anfrage	Lauf	Programm
80	3720	208	39	14	anfr1	1	SQL
80	4050	207	42	0	anfr1	2	SQL
80	4010	215	34	2	anfr1	30	SQL
80	3920	208	40	0	anfr1	31	SQL
26232	22340	205	110	8	anfr1	1	CSQL
26232	23730	205	102	0	anfr1	2	CSQL
26232	22630	206	82	1	anfr1	30	CSQL
26232	23310	206	85	0	anfr1	31	CSQL
7	90	6	0	0	anfr1a	1	SQL
7	140	7	0	0	anfr1a	2	SQL
7	150	8	0	0	anfr1a	30	SQL
7	110	8	0	0	anfr1a	31	SQL
26448	22400	204	117	0	anfr1c	1	CSQL
26448	23000	205	94	0	anfr1c	2	CSQL
26448	23490	204	88	0	anfr1c	30	CSQL
26448	22980	204	91	0	anfr1c	31	CSQL
39921	170310	518	295	10	anfr2a	1	CSQL
39921	171140	431	276	0	anfr2a	2	CSQL
39921	169410	542	276	21	anfr2a	30	CSQL
39921	172840	477	301	2	anfr2a	31	CSQL
5207	469140	24961	661	1	anfr2c	1	CSQL
5207	466630	24963	617	0	anfr2c	2	CSQL
5207	450540	24962	603	11	anfr2c	30	CSQL
5207	457800	24946	659	0	anfr2c	31	CSQL
1118	94440	4973	130	0	anfr2d	1	CSQL
1118	91090	5022	122	0	anfr2d	2	CSQL
1118	91960	4963	136	1	anfr2d	30	CSQL
1118	89070	4965	121	0	anfr2d	31	CSQL
6	4310	250	7	2	anfr3	1	SQL
6	4350	249	8	43	anfr3	2	SQL
6	4170	251	7	0	anfr3	30	SQL
6	4470	247	7	0	anfr3	31	SQL

bio_cnt	cpu_ms	dio_cnt	et_sec	pfault_cnt	Anfrage	Lauf	Programm
526	6810	375	18	75	anfr3a	1	CSQL
526	7040	370	16	68	anfr3a	2	CSQL
526	6890	378	12	0	anfr3a	30	CSQL
526	6880	371	13	0	anfr3a	31	CSQL
640	148580	4563	192	2	anfr3b	1	CSQL
640	149950	4564	214	6	anfr3b	2	CSQL
640	150440	4587	212	9	anfr3b	30	CSQL
640	152050	4566	206	0	anfr3b	31	CSQL

Tabelle A-3: Messungen vor und nach “optimizedb” und “sysmod”

Die Tabelle A-4 zeigt den Einfluß der Datenstruktur auf die Antwortzeit. Die Läufe 20 und 21 wurden mit Hash-Struktur durchgeführt (INGRES defaults). Als Vergleich dienen hier wieder die Läufe eins bis sechs aus Tabelle A-1. Der Übersichtlichkeit wegen sind wieder nur die Läufe eins und zwei aufgeführt.

bio_cnt	cpu_ms	dio_cnt	et_sec	pfault_cnt	Anfrage	Lauf	Programm
80	3720	208	39	14	anfr1	1	SQL
80	4050	207	42	0	anfr1	2	SQL
80	8150	946	45	7	anfr1	20	SQL
80	8090	944	85	1	anfr1	21	SQL
26232	22340	205	110	8	anfr1	1	CSQL
26232	23730	205	102	0	anfr1	2	CSQL
26232	28430	942	126	2	anfr1	20	CSQL
26232	28360	942	200	1	anfr1	21	CSQL
26448	22400	204	117	0	anfr1c	1	CSQL
26448	23000	205	94	0	anfr1c	2	CSQL
26448	28070	942	130	0	anfr1c	20	CSQL
1118	94440	4973	130	0	anfr2d	1	CSQL
1118	91090	5022	122	0	anfr2d	2	CSQL
1118	223110	26449	533	80	anfr2d	20	CSQL
1118	222520	26386	680	26	anfr2d	21	CSQL
6	4310	250	7	2	anfr3	1	SQL
6	4350	249	8	43	anfr3	2	SQL
6	56000	8203	125	22	anfr3	20	SQL
6	53980	8201	102	0	anfr3	21	SQL
526	6810	375	18	75	anfr3a	1	CSQL
526	7040	370	16	68	anfr3a	2	CSQL
526	223730	37026	625	86	anfr3a	20	CSQL
526	225710	37012	461	46	anfr3a	21	CSQL
640	148580	4563	192	2	anfr3b	1	CSQL
640	149950	4564	214	6	anfr3b	2	CSQL
640	633360	86193	1417	152	anfr3b	20	CSQL
640	584580	86195	976	1	anfr3b	21	CSQL

Tabelle A-4: Vergleich von ISAM- und Hash-Struktur

Die Tabelle A-5 enthält Antwortzeitvergleiche von Anfragen auf Relationen verschiedener Größe. Der Lauf null wurde auf den Relationen der Größe 30 262 Tupel (anfr1x, anfr2x) bzw. 181 727 Tupel (anfr3x) gemessen. Zum Vergleich können hier zusätzlich die Läufe eins bis sechs der Tabelle A-1 auf den entsprechenden Anfragen hinzugezogen werden. Für die Anfragen anfr1x und anfr2x wurden der Lauf 1m auf 120 Tupeln (fünf Tage), der Lauf 2m auf 1487 Tupeln (zwei Monate) und der Lauf 3m auf 4367 Tupeln (sechs Monaten) ausgeführt. Für die Anfragen anfr3x wurden der Lauf 1m auf 22 025 Tupeln (fünf Monate) und der Lauf 2m auf

52 385 Tupeln (ein Jahr) gemessen. (Die Werte des Laufs 1m der Anfrage 2a sind vermutlich fehlerhaft, was sich am abweichenden Wert von ‘bio_cnt’ erkennen läßt. Vielleicht wurde hier zuviel gemessen, also mehr als die eine Anfrage; oder es wurde unbemerkt parallel auf dieser Relation gearbeitet.)

bio_cnt	cpu_ms	dio_cnt	et_sec	pfault_cnt	Anfrage	Lauf	Programm
7	140	8	0	2	anfr1a	0	SQL
7	110	9	0	1	anfr1a	1m	SQL
7	130	8	1	0	anfr1a	2m	SQL
7	100	10	0	0	anfr1a	3m	SQL
240	300	4	2	0	anfr1a	0	CSQL
240	280	8	1	0	anfr1a	1m	CSQL
240	310	8	3	0	anfr1a	2m	CSQL
240	340	8	1	0	anfr1a	3m	CSQL
39921	175030	563	349	121	anfr2a	0	CSQL
51401	281710	864	410	6	anfr2a	1m	CSQL
39921	175120	416	280	2	anfr2a	2m	CSQL
39921	173210	430	265	3	anfr2a	3m	CSQL
5270	417900	27481	540	37	anfr2c	0	CSQL
5270	23660	169	37	0	anfr2c	1m	CSQL
5270	42590	1399	64	0	anfr2c	2m	CSQL
5270	82700	3750	106	0	anfr2c	3m	CSQL
6	5100	294	9	0	anfr3	0	SQL
6	4760	255	12	6	anfr3	1m	SQL
6	4750	263	12	0	anfr3	2m	SQL
526	5100	435	15	88	anfr3a	0	CSQL
526	4760	433	17	73	anfr3a	1m	CSQL
526	4750	426	12	0	anfr3a	2m	CSQL
640	148650	5576	206	7	anfr3b	0	CSQL
640	17600	686	23	3	anfr3b	1m	CSQL
640	31190	1133	55	2	anfr3b	2m	CSQL

Tabelle A-5: Vergleich von Anfragen auf Relationen unterschiedlicher Größe

Zum Abschluß sind jetzt noch einmal alle Meßergebnisse von SQL-Anfragen zusammengestellt (anfr1, anfr1a), die sowohl mit SQL als auch mit CSQL gemessen wurden.

Interessanterweise trifft das Verhältnis auch auf anfr3 und anfr3a zu, obwohl anfr3a einige wenige Aufrufe der Interpolationsfunktion enthält. (Es wird dann allerdings nicht interpoliert, sondern es handelt sich um sog. Treffer, also zufällig in der DB vorhandenen Werte.) Deshalb wurden die Meßergebnisse dieser Anfragen mit in die Tabelle A-6 aufgenommen.

bio_cnt	cpu_ms	dio_cnt	et_sec	pfault_cnt	Anfrage	Lauf	Programm
80	4050	230	28	20	anfr1	0	SQL
80	3710	223	12	0	anfr1	1a	SQL
80	3720	208	39	14	anfr1	1	SQL
80	4050	207	42	0	anfr1	2	SQL
80	4160	208	53	0	anfr1	3	SQL
80	3760	204	40	0	anfr1	4	SQL
80	3860	204	39	0	anfr1	5	SQL
80	3940	202	39	0	anfr1	6	SQL
26232	24940	228	96	7	anfr1	0	CSQL
26232	23980	229	86	0	anfr1	1a	CSQL
26232	22340	205	110	8	anfr1	1	CSQL
26232	23730	205	102	0	anfr1	2	CSQL
26232	23570	206	99	0	anfr1	3	CSQL
26232	22800	204	94	0	anfr1	4	CSQL
26232	22720	204	94	0	anfr1	5	CSQL
26232	23470	202	97	0	anfr1	6	CSQL
7	140	8	0	2	anfr1a	0	SQL
7	90	6	0	0	anfr1a	1	SQL
7	140	7	0	0	anfr1a	2	SQL
7	160	8	1	0	anfr1a	3	SQL
7	110	8	0	0	anfr1a	4	SQL
7	100	4	0	0	anfr1a	5	SQL
7	90	4	0	0	anfr1a	6	SQL
240	300	4	2	0	anfr1a	0	CSQL

bio_cnt	cpu_ms	dio_cnt	et_sec	pfault_cnt	Anfrage	Lauf	Programm
6	5100	294	9	0	anfr3	0	SQL
6	4310	250	7	2	anfr3	1	SQL
6	4350	249	8	43	anfr3	2	SQL
6	4230	246	7	11	anfr3	3	SQL
6	4300	246	8	0	anfr3	4	SQL
6	4200	246	6	0	anfr3	5	SQL
6	4200	246	7	0	anfr3	6	SQL
526	6970	435	15	88	anfr3a	0	CSQL
526	6810	375	18	75	anfr3a	1	CSQL
526	7040	370	16	68	anfr3a	2	CSQL
526	6670	371	13	33	anfr3a	3	CSQL
526	6620	367	14	0	anfr3a	4	CSQL
526	6390	433	13	0	anfr3a	5	CSQL
526	6230	375	11	0	anfr3a	6	CSQL

Tabelle A-6: Verzögerung von SQL-Anfragen durch das CSQL-System

Lebenslauf

Name:	Leonore Zink, geb. Neugebauer
Geburtstag/-ort:	06.10.1959 in Hameln/Weser, Nds.
Familienstand:	verheiratet, 2 Kinder: Tochter Sarina, geb. 26.08.1993 Tochter Evelyn, geb. 30.01.1996
Schulabschluß:	1978 Abitur an der Viktoria-Luise-Schule in Hameln
Studium:	1978-1984 Informatik mit Anwendungsfach Verkehrslenkung und -sicherung an der TU Braunschweig Abschluß mit Diplom am 12.12.1984

Wissenschaftlicher Werdegang:

1982-1984 Wissenschaftliche Hilfskraft am Institut für Datenverarbeitung / Institut für Betriebssysteme und Rechnerverbund der TU Braunschweig

1985-1988 Wissenschaftliche Mitarbeiterin am Institut für Compilerbau und Informationssysteme, Abteilung Datenbanken der TU Braunschweig

1988-09/1991 Wissenschaftliche Mitarbeiterin am Institut für Parallele und Verteilte Höchstleistungsrechner (IPVR) der Universität Stuttgart

10/'91-09/'93 Zunächst freie dann wieder fest angestellte wissenschaftliche Mitarbeiterin ebenfalls am IPVR im Datenbankteilprojekt des interdisziplinären ingenieurwissenschaftlichen Projekts "Prozeßsimulation Umformtechnik (PSU)"

1995 Wiedereinstiegsstipendium für Frauen des Hochschulsonderprogramms II der Universität Stuttgart