Rückblick auf IRIS

Abschlußbericht eines Datenbank-Forschungsprojektes

Stefan Manegold, Carsten Hörner, Andreas Heuer

Juli 1993

TU Clausthal
Institut für Informatik
Projektgruppe Relationale Objektbanksysteme

Zusammenfassung

Dieser Bericht beschreibt abschließend die Ergebnisse des IRIS-Projekts der Projektgruppe Relationale Datenbanksysteme am Institut für Informatik der TU Clausthal. Das wichtigste Ergebnis ist die Implementierung von IRIS als Prototyp eines vollständig relationalen Datenbanksystems, der sich durch seine Fähigkeiten wesentlich von vergleichbaren kommerziellen Produkten abhebt:

IRIS bietet mehrere verschiedenartige interaktive Anfragesprachen, algebraische und regelbasierte für das Relationenmodell und dessen Erweiterung zu geschachtelten Relationen sowie eine Universalrelationen-Schnittstelle. Updates können sowohl interaktiv (menüorientiert) als auch mittels einer deklarativen Update-Sprache durchgeführt werden, wobei zur Integritätserhaltung Schlüssel- und Fremdschlüsselbedingungen herangezogen werden. Relationenalgebraische Anfragen und Update-Operationen stehen außerdem auch eingebettet in Pascal für die Anwendungsprogrammierung zur Verfügung.

Die Definition des Datenbankschemas wird in IRIS durch spezielle Algorithmen unterstützt, die funktionale und mehrwertige Abhängigkeiten zur Erreichung bestimmter positiver Schemaeigenschaften ausnutzen und entsprechend normalisierte Datenbankschemata erzeugen. Darüberhinaus bestehen Möglichkeiten zur Schemaevolution. Die Beschreibung des Datenbankschemas wird, zusammen mit statistischen Angaben, in einem relationalen Data Dictionary verwaltet. Das zugrundeliegende Typsystem von IRIS kann um benutzerdefinierte Typen (ADTs) erweitert werden.

Intern kann der Zugriff auf die Datenbank, neben den obligatorischen Pufferungsmechanismen, durch verschiedene Formen von Zugriffspfaden unterstützt werden, die auch zur Laufzeit dynamisch gewechselt werden können.

Die im IRIS-Projekt gesammelten Erfahrungen bilden die Basis für die bereits weit fortgeschrittene Implementierung des prototypischen objektorientierten Datenbanksystems OSCAR und einer parallel dazu laufenden Implementierung seines Datenmodells EXTREM auf Basis eines kommerziellen Objektbanksystems.

Vorwort

Entwicklungsziel des IRIS-Projektes war ein universitärer Prototyp eines relationalen Datenbanksystems, wobei insbesondere im Bereich des Datenbankentwurfs, der Anfragesprachen und der Speicherstrukturen moderne Konzepte ausgenutzt werden sollten.

11 Jahre sind nun seit Projektbeginn vergangen. Die IRIS-Gruppe startete damals als vierköpfige studentische Interessengemeinschaft mit der Konzeption eines relationalen Datenbanksystems. Zwei aus dieser Gruppe betreuten ab 1985 studentische Software-Praktika, in denen die ersten Module des IRIS-Systems entwickelt wurden. Die Hauptarbeit an IRIS wurde bis 1988 abgeschlossen. Seit diesem Termin wurden etliche Feinheiten geändert und letzte Komponenten entwickelt, wie etwa der algebraische Optimierer.

Seit 1993 kann das IRIS-Projekt jetzt als vollständig abgeschlossen gelten. Einige Dinge, wie etwa Mehrbenutzerbetrieb und Datenschutz- sowie Datensicherheits-Mechanismen waren in diesem Prototyp nicht geplant und sind deshalb auch nicht realisiert worden. Etwa fehlt ein vollständiges Transaktionskonzept, ein systemfehlerresistentes Recovery-Konzept und jede Art von Concurrency Control. Weiterhin gibt es Einschränkungen bei der Datenmenge, etwa in der Anzahl der Attribute einer Relation und der maximalen Länge von Strings.

Erfolgreich erprobt wurden dagegen diverse Speicherstrukturen, auch mehrdimensionale und dynamische, sowie mehrere Arten von Anfragesprachen (algebraisch, regelbasiert, Universalrelationen-Schnittstelle). Das Sichtkonzept wurde auf geschachtelte Relationen erweitert, die Fremdschlüsselverfolgung ernst genommen und der algebraische Optimierer von außen erweiterbar gestaltet.

Dieser Abschlußbericht basiert zunächst auf zwei Zwischenberichten über das IRIS-Projekt aus den Jahren 1985 und 1987, die ich selbst verfaßt habe. Weiterhin ist die für Implementierer im IRIS-Projekt gedachte "Meta-Dokumentation" von Carsten Hörner schwerpunktmäßig in diesen Bericht eingeflossen. Die Aktualisierung dieser Meta-Dokumentation und die Aufbereitung der verschiedenen Quellen zu einem hoffentlich gut lesbaren Abschlußbericht gelang schließlich Stefan Manegold. Diesen beiden Mitstreitern sei noch einmal herzlich gedankt, wie auch allen anderen Projektgruppen-Mitgliedern, die am Ende dieses Berichts noch einmal aufgeführt sind.

Im Juli 1993

Andreas Heuer

Inhaltsverzeichnis

Kapitel 1

Einführung

Seit Anfang der 80er Jahre wird am Institut für Informatik auf dem Gebiet der Datenbanktheorie und -technologie geforscht. Neben der Arbeit an Theorie und Konzepten bildete immer auch die Umsetzung in prototypische Systeme einen Schwerpunkt der Arbeit. Alle diese Aktivitäten wurden gebündelt in der Projektgruppe Relationale Datenbanksysteme, zunächst geleitet von Prof. Klaus Ecker, seit 1988 von Dr. Andreas Heuer. Die Organisationsform der Projektgruppe ermöglicht eine intensive und längerfristige Beteiligung von Studenten an den Forschungsarbeiten in Form von Software-Praktika, bezahlten Hilfskraft-Tätigkeiten und Diplomarbeiten. Begleitet werden die Aktivitäten der Projektgruppe durch Vorlesungen, Pro- und Hauptseminare, so daß durch die Wechselwirkungen zwischen Lehre, Forschung und praktischer Anwendung ein angesichts der geringen Größe des Institutes beachtliches Qualitätsniveau erreicht werden konnte.

Das IRIS-Projekt, dessen Ergebnisse wir in diesem Bericht vorstellen möchten, war das erste große Projekt dieser Gruppe. Es wurde Ende 1982 begonnen und kann nunmehr als weitgehend abgeschlossen angesehen werden. Ziel dieses Projekts war die Konzeption und prototypische Implementierung des „Interaktiven Relationalen Informations-Systems“ IRIS, eines relationalen Datenbank-Management-Systems, das — im Gegensatz zu den bekannten kommerziellen Produkten dieser Art — konsequent die theoretischen Konzepte des relationalen Datenbankmodells umsetzen und besonders für die interaktive Nutzung geeignet sein sollte.

Im Zuge der Fortentwicklung der Datenbanktheorie hin zu „mächtigeren“, ausdrucksstärkeren Datenbankmodellen — zunächst semantischen, dann objektorientierten — wurden auch in unserer Projektgruppe Pläne für ein objektorientiertes Datenbank-System geschmiedet, auf der Grundlage des EXTREM-Modells [Heu88, HH91]. An ihrer Umsetzung wird seit 1988 im Rahmen des OSCAR-Projekts [Kla92] gearbeitet; die Projektgruppe nennt sich daher inzwischen auch „Relationale Objektbanksysteme“. Auf dem Weg dorthin wurde jedoch zunächst der Übergang vom relationalen zum geschachtelt-relationalen (NF^2-) Datenmodell vollzogen und IRIS um Möglichkeiten zur Behandlung geschachtelter Relationen, die als spezielle Sichten angesehen werden, erweitert, insbesondere hinsichtlich entsprechender Anfragesprachen.

Schließlich wurde der nicht-interaktive Kern von IRIS dazu verwendet, einem ersten Prototypen von OSCAR als Subsystem für die interne Speicherung der Objekte als Tupel in Relationen zu dienen. Dazu war zunächst die Abbildung der Klassen des EXTREM-Modells

auf eine geeignete relationale Repräsentation zu bewerkstelligen und die IRIS-Metaebene, das Data Dictionary, entsprechend zu erweitern.

Zu Beginn der Implementierung von IRIS (Anfang 1985) diente die DEC PDP-11 des Instituts als Hardware-Plattform, als Implementierungssprache wurde Pascal gewählt. Es zeigte sich jedoch recht schnell, daß sich daraus zu enge Restriktionen ergaben[1], so daß 1986 auf Personal Computer unter MS-DOS umgestiegen wurde, was zu diesem frühen Zeitpunkt noch relativ leicht war. Die Programmiersprache Pascal wurde beibehalten. Das rasche Wachstum des Systems ließ uns jedoch auch hier an die Grenzen des Betriebssystems stoßen, weshalb OSCAR nun unter (PC-) UNIX implementiert wird, allerdings noch mit einem IRIS-Kernsystem unter DOS.

Die Wahl der Hardware-Plattform und die damals noch wenig ausgeprägten Möglichkeiten zur Vernetzung mehrerer PCs bedingten es, daß IRIS als Ein-Benutzer-System konzipiert und implementiert wurde. Dies stellt natürlich eine wesentliche Einschränkung dar, die aber unserer Meinung nach den Erfolg des Projekts, nämlich den Beweis der Realisierbarkeit bestimmter theoretisch motivierter Konzepte, nicht schmälert, uns jedoch die Durchführung des Projekts angesichts permanent knapper finanzieller und personeller Ressourcen später erleichterte.

Wir werden im folgenden die Konzepte von IRIS näher vorstellen und dazu mit einer Erläuterung der theoretischen Grundlagen und der Architektur des Systems beginnen. Nachfolgend werden dann die drei architektonischen Ebenen (externe, konzeptuelle und interne Ebene) mit ihren spezifischen Leistungen vorgestellt. Die oben erwähnten Erweiterungen von IRIS für den Einsatz als Kernsystem für OSCAR bilden bereits einen Teil des OSCAR-Projekts und werden daher hier nicht behandelt, zumal dazu auch recht weit auszuholen wäre.

Im Anhang werden weitere Detail-Informationen zu IRIS dargestellt (Menüstruktur, Relationen des Data Dictionary) sowie ein Überblick über die Dokumentationen zu IRIS („interne Literatur“) und die zugrundeliegende „externe“ Literatur gegeben. „Interne“ Literatur[2] bezeichnet hierbei die nicht veröffentlichten Dokumentationen der Implementierung und Bedienung der einzelnen IRIS-Komponenten, die wir Interessierten jedoch gern zur Verfügung stellen.

Auf eine vollständige Darstellung der Syntax der Anfrage- und Update-Sprachen sowie der Spezifika von IRIS-Pascal wird hier verzichtet, da dies den Rahmen dieses Berichtes sprengen würde. Hier sei auf die jeweilige interne Literatur verwiesen.

[1] Je Benutzer stand lediglich eine 64kB „kleine“ Hauptspeicher-Partition zur Verfügung.

[2] Bei Verweisen auf „interne“ Literatur besteht die Autorenkennung — im Gegensatz zu Verweisen auf „externe“ Literatur — nur aus Kleinbuchstaben.

Kapitel 2

Grundlagen

2.1 Grundbegriffe des Relationenmodells

Das relationale Datenmodell geht zurück auf Codd [Cod70]. Gegenüber seinen „Vorgängern", dem hierarchischen und dem Netzwerk-Modell, zeichnet es sich durch seine exakte mathematische Fundierung, aber auch eine geradezu elegante Schlichtheit aus. Der Erfolg des Relationenmodells als Beschreibungswerkzeug dürfte aber auch darauf zurückzuführen sein, daß seine Konzepte eine intuitive, informale Interpretation ermöglichen, durch die es auch weniger mathematisch geprägten Menschen verständlich wird. Wir wollen hier, dem Zweck des Berichts entsprechend, keine vertiefte Einführung in das Relationenmodell als theoretisches Fundament von IRIS geben. Der daran interessierte Leser sei auf die einschlägigen Lehrbücher verwiesen, etwa [Ull88, Dat90, Dat83, Vos87]. Stattdessen wollen wir die wesentlichen Grundbegriffe im folgenden informal, an der intuitiven Sichtweise orientiert, einführen und die Vertiefung bei der Vorstellung der jeweiligen Komponenten von IRIS vornehmen.

Intuitiv gesehen, beruht das Relationenmodell auf der Darstellung der in der Datenbank zu speichernden Daten in Form von Tabellen, die als **Relationen** bezeichnet werden. Die Spalten der Tabellen werden **Attribute** genannt, die einzelnen Zeilen **Tupel**. Im Tabellenkopf, dem **Relationenschema**, werden den Attributen eindeutige Namen und Wertebereiche (Typen bzw. **Domänen**) zugeordnet. Beziehungen zwischen verschiedenen Tabellen werden u.a. durch die Verwendung gleich benannter Attribute/Spalten hergestellt.

Für die Theorie des Relationenmodells entscheidend, in der Praxis leider oft vernachlässigt, ist die konsequente Ausrichtung am mathematischen Begriff der **Menge**, worunter ganz allgemein eine ungeordnete Sammlung gleichartiger Dinge ohne Duplikate verstanden wird. Ein Relationenschema ist also eine Menge von Attributen, eine Relation über einem Relationenschema eine Menge von Tupeln, die zu diesem Schema passen, und eine **Datenbank** schließlich eine Menge von Relationen und ihren Schemata. Die Menge aller im Datenbankschema definierten Attribute wird als **Universum** bezeichnet.

Schlüssel einer Relation sind (bzgl. Inklusion) minimale Teilmengen des Relationenschemas, also Mengen von Attributen, von denen die Werte der restlichen Attribute gewissermaßen abhängen. Konkret heißt das, daß es in einer Relation keine zwei Tupel geben darf, die auf allen Attributen eines Schlüssels übereinstimmen (und sich wegen des Mengen-Charakters der Relation dann auf den restlichen Attributen unterscheiden müßten).

Schlüssel stellen die wichtigste Art von **Integritätsbedingungen** dar: Das sind zusätzliche, im (erweiterten) Relationen- bzw. Datenbankschema angegebene Bedingungen, denen die Tupel der Relationen genügen müssen, damit sie als erlaubter, konsistenter Datenbestand gelten können. IRIS kennt außerdem noch Fremdschlüssel-Bedingungen, die wir aber erst im Abschnitt 3.3 erläutern wollen.

Was IRIS nun von anderen, insbesondere von den meisten kommerziellen Datenbanksystemen unterscheidet, ist die Tatsache, daß IRIS nach der Klassifikation von Codd als „vollständig relational“ bezeichnet werden kann, da es die folgenden 4 Kriterien erfüllt:

R1: Jegliche Datenbankinformation ist durch Attributwerte in Relationen repräsentiert.

R2: Es gibt keine für den Benutzer sichtbaren Verkettungen zwischen den Relationen.

R3: Es gibt „die“ vollständige Relationenalgebra in irgendeiner Syntax, aber ohne Verwendung von Iteration oder Rekursion und ohne Beschränkung bzgl. interner Strukturen.

R4: Das System unterstützt automatisch „Entity-Integrität“ und „referentielle Integrität“, d.h. jede Relation besitzt einen Schlüssel, ihren sog. Primärschlüssel, und Fremdschlüsselbedingungen können definiert werden.

Um Mißverständnissen vorzubeugen, weisen wir darauf hin, daß im „einfachen“ (oder „flachen“) Codd'schen Relationenmodell jedes Tupel einer Relation für jedes Attribut ihres Schemas *genau einen* Wert aus seiner Domäne enthalten muß: Dieser Wert darf weder fehlen, noch dürfen mehrere Werte gleichzeitig angegeben werden. Diese Forderung wird als **erste Normalform** (1NF) bezeichnet. (Zu weiteren Normalformen siehe Abschnitt 4.1.)

Eine auch in IRIS vorgenommene Erweiterung des Relationenmodells, das sog. **NF^2-Modell** (Non First Normal Form) oder Modell geschachtelter Relationen, hebt gerade diese Forderung auf und erlaubt als Attributwerte vollständige Relationen, also Mengen von Tupeln über einem festen Relationenschema, die ihrerseits wiederum Relationen als Attributwerte enthalten dürfen, usw. Dies erlaubt eine natürlichere Darstellung komplexer Daten, wobei viele Konzepte des „flachen“ Relationenmodells direkt übertragen werden können, aber auch einige Ergänzungen erforderlich werden, wie wir bei der Vorstellung der Anfragesprachen NERO und NaLogo! zeigen werden.

Das NF^2-Modell ist die entscheidende Zwischenstufe beim Übergang vom (wertorientierten) Relationenmodell zu unserem objektorientierten Datenbankmodell EXTREM mit seinen fast beliebig komplexen Attributstrukturen. Die Verwendung von IRIS als relationales Subsystem des auf EXTREM aufbauenden OSCAR-Systems ist nur deswegen möglich, weil es eine formal abgesicherte Übersetzung von EXTREM-Datenbankschemata in geschachtelte Relationen und damit auch in flache Relationen gibt — natürlich auf jeder Ebene mit gewissem Zusatzaufwand. Aber dazu an anderer Stelle mehr (z.B. in [Heu87]).

2.2 Die Architektur von IRIS

Zunächst wollen wir einen kurzen Überblick über den Aufbau des Systems geben. Auf die einzelnen Komponenten wird in den folgenden Kapiteln ausführlicher eingegangen.

IRIS folgt im wesentlichen den ANSI/X3/SPARC–Richtlinien zur Architektur von Datenbanksystemen [ANS75]. Genauere Ausführungen, was die Komponenten und Schnittstellen (ihre Numerierung in Abbildung 2.1 entspricht der in [ANS75]) dieses Modellentwurfs leisten sollen, finden sich in [Heu87].

Stark vergröbert, sieht dieser Entwurf eine Trennung in drei Ebenen vor (siehe auch Abbildung 2.1):

1. Eine **„interne Ebene“**, deren Aufgabenbereich die Speicherung der Daten als Bits und Bytes auf den vorhandenen Speichermedien und den effizienten Zugriff darauf umfaßt.

2. Eine **„konzeptuelle Ebene“**, die die theoretischen Konzepte des Datenmodells implementiert, bei einem relationalen System wie IRIS also die Konzepte „Relationenschema“, „Relation“, „Schlüssel“, „Relationenalgebra“ usw.

3. Schließlich eine **„externe Ebene“**, die diese Konzepte für den Benutzer aufbereitet und die Kommunikation zwischen Mensch und System steuert.

Zu jeder der drei Ebenen gibt es eine Transformations- und eine Definitions-Komponente, auf der externen Ebene finden sich daneben die IQL- und DML-Komponenten. (IQL steht für Interactive Query Language, also interaktive An- oder Abfragesprache; DML bedeutet Data Manipulation Language und kann sowohl als Sprache für die Veränderung der Daten im Sinne von Updates wie auch als Obermenge von IQL und Update-Sprache aufgefaßt werden.) Dreh- und Angelpunkt für alle drei Ebenen ist das **Data-Dictionary/Directory** (DD/D), sozusagen das Inhaltsverzeichnis der Datenbank, das alle systemweit benötigten **Metadaten** enthält. Metadaten sind „Daten über Daten“, im Falle einer relationalen Datenbank also die Beschreibung des Datenbankschemas mit Attributen, Domänen, Relationen, Schlüsseln usw.

Wie stellt sich diese Architektur bei IRIS dar?

Zunächst steht auch hier das DD/D im Mittelpunkt des Systems, da alle IRIS-Komponenten zur Erfüllung ihrer Aufgaben auf die Metadaten zugreifen müssen. Die Metadaten werden weitgehend wie Benutzerdaten behandelt, vor allem werden sie ebenfalls *in Form von Relationen* verwaltet und mit den Benutzerrelationen zusammen im `DATAF` (Data File) auf Platte gespeichert.

Hinsichtlich der Änderbarkeit wird natürlich sehr wohl zwischen Benutzer- und Dictionary-Daten unterschieden. Außerdem werden zur Beschleunigung der Arbeit Kopien der Directory-Daten im Hauptspeicher angelegt, um bei Anfragen und Updates auf den Benutzerdaten die erforderlichen Strukturinformationen nicht jedesmal von der Platte holen zu müssen. Und schließlich existiert auf der Platte noch ein „leeres“ Default-DD/D (genannt `MINDIC`), das gebraucht wird, wenn eine neue Datenbank eingerichtet werden soll.

Die direkte Verbindung zwischen DD/D und `DATAF` bildet die **Transformationskomponente der Internen Ebene**, bei IRIS meistens als „der **IRIS-Kern**“ bezeichnet. Ein wesentlicher Teil davon ist **UFO**, die „Universelle File-Organisation“, deren Aufgabe die Bereitstellung des `DATAF` in Form eines Puffers für mehrere Plattenseiten im Hauptspeicher und der tupelweise Zugriff darauf ist. Außerdem gehören die verschiedenen Zugriffsunterstützungen (indexsequentiell, indiziert- nichtsequentiell, kdB-Baum, Hashing) zum IRIS-Kern.

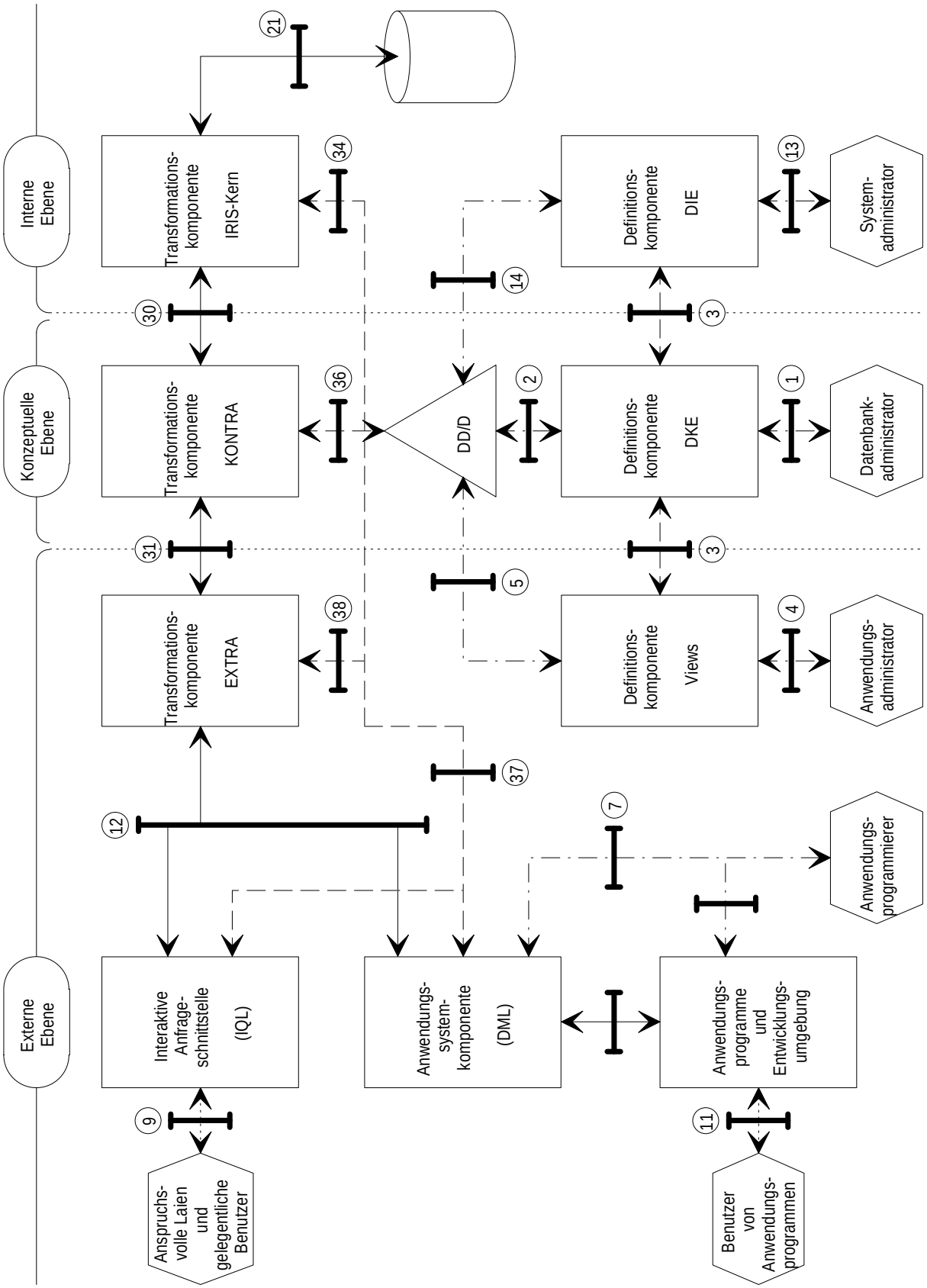

Abbildung 2.1: Architektur von IRIS nach der ANSI/X3/SPARC-Form

Die **Definitionskomponente der Internen Ebene (DIE)** ist jener Programmteil, der es dem Datenbank- oder Systemadministrator erlaubt, die für die einzelnen Relationen zu verwendenden Zugriffsunterstützungen festzulegen. Da sie speziell hierfür entwickelt wurde, gehört auch die Statistikkomponente von IRIS zur DIE. Sie besteht aus mehreren, vom System unterhaltenen und organisatorisch dem DD/D angegliederten Relationen.

Den ersten großen Abstraktionsschritt von Bytefolgen auf physikalischen Seiten hin zu Tupeln in Relationen stellt dann der Übergang von der internen zur konzeptuellen Ebene dar, die ebenfalls in eine Transformations- und eine Definitionskomponente zerfällt.

Die **Definitionskomponente der Konzeptuellen Ebene (DKE)** ist jene, in der die Definition und Änderung der Relationenschemata nebst Schlüssel- und (ggf.) anderen Integritätsbedingungen erfolgt – erneut also eine Komponente, die in erster Linie den DBA betrifft. Hierzu gehört auch die algorithmische Schemadefinition mit Hilfe des Synthese-Verfahrens (IRIS-SYNTHI).

Die **Transformationskomponente der Konzeptuellen Ebene** (KONTRA) implementiert eine erweiterte Relationenalgebra, indem eine (bereits optimierte) algebraische Anfrage übersetzt wird in eine Folge von einzelnen Tupelanforderungen, die dann vom IRIS-Kern zu erfüllen sind.

Über KONTRA angeordnet ist die **Transformationskomponente der Externen Ebene** (EXTRA). Diese dient allgemein dazu, IQL- und DML-Anweisungen, also Anfragen und Update-Anweisungen, in Operationen auf Relationen zu übersetzen. Da der Update-Bereich in IRIS relativ eigenständig ist, besteht die Hauptaufgabe in der Übersetzung von RELAX-, NERO- oder URQUEL-Anfragen in einen relationenalgebraischen Ausdruck und dessen anschließender algebraischer Optimierung.

Die **Definitionskomponente der Externen Ebene** beinhaltet bei IRIS die Definition und Änderung von (Benutzer-) Sichten (engl. Views).

Den Abschluß nach außen bilden neben den Update-Schnittstellen die interaktiven Anfragesprachen RELAX, URQUEL, NERO, LDML und NaLogo!. Die drei letztgenannten bauen ihrerseits auf RELAX auf. URQUEL-Anfragen werden mittels Fensterfunktionen und Tableau-Optimierung interpretiert und zu relationenalgebraischen Anfragen vervollständigt. Schließlich stellt IRIS noch eine Einbettung von RELAX und DML in Pascal („IRIS-PASCAL“) für die Entwicklung von Anwendungsprogrammen zur Verfügung.

Kapitel 3

Externe Ebene

Die Externe Ebene ist diejenige Schicht des Datenbank-Systems, die dessen Leistungen dem Endbenutzer in möglichst geeigneter Form zur Verfügung stellt und mit diesem kommuniziert. Dies geschieht entweder direkt, indem der Benutzer interaktiv Anfragen an die Datenbank stellt oder Änderungen des Datenbestandes veranlaßt, oder indirekt durch Anwendungsprogramme. Im Gegensatz zum anwendungsneutralen DBMS stellen letztere einen ganz spezifischen Anwendungskontext her. Die Externe Ebene des DBMS muß daher auch Möglichkeiten zur Entwicklung und Ausführung von Anwendungsprogrammen umfassen.

Hinweis: Die meisten Beispiele in diesem Abschnitt beziehen sich auf das in Anhang E angegebene Datenbankschema.

3.1 Interaktive Anfragesprachen

IRIS enthält fünf von der Konzeption her unterschiedliche Anfragesprachen:

- **RELAX:** Eine Sprache, die direkt auf der Relationenalgebra aufbaut und diese um zusätzliche Fähigkeiten erweitert.
- **URQUEL:** Eine konzeptuell eher am Relationenkalkül orientierte Universal-Relationen-Anfragesprache, deren Implementierung aber ebenfalls die Relationenalgebra verwendet.
- **LDML:** Eine regelbasierte Anfragesprache auf der Grundlage von RELAX.
- **NERO:** Eine algebraische Anfragesprache für geschachtelte Relationen.
- **NaLogo!:** Eine regelbasierte Anfragesprache für geschachtelte Relationen.

3.1.1 RELAX

RELAX (relational algebra extended) stellt Operationen der Relationenalgebra (siehe Abschnitt 4.3) in syntaktisch minimal aufbereiteter Form bereit, wodurch sich unmittelbar eine

wohldefinierte Semantik von RELAX-Anfragen ergibt. Gleichzeitig erweitert RELAX die „klassische“ Relationenalgebra [Ull88, Vos87] um Fähigkeiten zur Berechnung abgeleiteter Attribute, zum Umgang mit abstrakten Datentypen und zur Aufbereitung der Anfrageergebnisse.

RELAX ([dw88]) umfaßt folgende Operationen:

- **Mengenoperationen:** Für je zwei Relationen mit *gleichem* Relationenschema sind die folgenden Operationen definiert:
  - `VEREIN`: Mengenvereinigung
  - `DURCH`: Mengendurchschnitt
  - `DIFF`: Mengendifferenz
- **Relationenspezifische Operationen:** Für beliebige Relationen definiert sind die Operationen:
  - `PI`: Projektion auf die angegebenen Attribute
  - `SIGMA`: Selektion aller Tupel, die eine Bedingung erfüllen
  - `BETA`: Umbenennung von Attributen
  - `JOIN`: Natürlicher Verbund zweier Relationen über ihre gemeinsamen Attribute
- **Erweiterungen**:
  - `SPE`: Spaltenerweiterung, d.h. Einführung eines neuen Attributs, dessen Werte gemäß einem als Parameter übergebenen Ausdruck berechnet werden (s.u.)
  - `ORDNE`: Sortierung nach den Werten der angegebenen Attribute
  - `NEST`: „Nestung“, d.h. (informal) Zusammenfassung eines oder mehrerer „flacher“ Attribute zu einem mengenwertigen, aggregierten („relationenwertigen“) Attribut
  - **Zuweisung**: Um komplexe Anfragen zu entzerren, können Teilausdrücke als temporäre Relation benannt und im weiteren Anfragetext durch ihren Namen referenziert werden. Die Lebenszeit solcher temporärer Relationen ist auf die jeweilige Anfrage begrenzt.

Die Instantiierung einer Relation ist syntaktisch nicht als explizite Operation realisiert; statt $r(R)$ wie in der Relationenalgebra reicht der Name R.

Zulässige **Selektionsbedingungen** beinhalten Vergleiche von Attributwerten mit anderen Attributwerten, Konstanten (auch Mengen!) sowie mit Ausdrücken, die einfache arithmetische Operationen enthalten dürfen. Neben den üblichen Vergleichsoperatoren (`=, <, <=`, ...) stehen dafür die Operatoren `AUS` und `NAUS` für den Test auf Enthaltensein in einer Menge von Konstanten zur Verfügung, außerdem der Operator `WIE` für den Vergleich von Zeichenketten mit speziellen Stringkonstanten, die auch „Wildcards“ enthalten dürfen. Hierfür wurde ein Teilstring-Suchalgorithmus aus [BM77] implementiert. Diese einfachen Bedingungen können mit den logischen Operatoren `UND`, `ODER` und `NICHT` zu komplexen Bedingungen zusammengesetzt werden.

Auch bei RELAX können immer nur Attributwerte eines einzelnen Tupels in die Selektionsbedingungen eingehen, so daß ein spezieller Mechanismus benötigt wird, um diese Einschränkung zu durchbrechen. Dieser Mechanismus wird durch die **Spaltenerweiterung** `SPE` bereitgestellt, die einer Relation ein neues Attribut hinzufügt, dessen Werte nach einer angegebenen Vorschrift berechnet werden. Dazu stehen zur Verfügung:

- Die arithmetischen Standardfunktionen (Aggregatfunktionen) `MIN`, `MAX`, `SUM` und `ARIMI` (Minimum, Maximum, Summe und arithmetisches Mittel), die entweder auf die gesamte Relation oder gruppenweise (s.u.!) angewendet werden können,
- die Zählfunktion `ANZAHL`, ebenfalls wahlweise global oder gruppiert,
- Werte anderer Attribute desselben Tupels,
- Konstante,
- die arithmetischen Operatoren `+, -, *, /`
- sowie spezielle Funktionen der Abstrakten Datentypen (ADT's) `Datum` und `Zeit`, die in Abschnitt 4.4 näher vorgestellt werden.

Die Berechnung der Aggregatfunktionen kann wahlweise für die gesamte Relation oder gruppiert nach den Werten eines Attributs erfolgen. Eine „Gruppe" ist dabei die Menge aller Tupel, die auf diesem Attribut den gleichen Wert tragen. Die Anzahl der Gruppen ist also gleich der Anzahl der verschiedenen Werte, die bei diesem Attribut mindestens einmal vorkommen. Sinnvoll ist eine Gruppierung natürlich nur auf Attributen, die keinen Schlüssel für die Relation darstellen, ansonsten wären alle Gruppen einelementig.

Da das neue Attribut auch einen Namen erhält und somit „zugreifbar" wird, wurde mit der Spaltenerweiterung ein leistungsfähiges Werkzeug geschaffen, das eine „Ableitung" von Daten in sehr eleganter Weise ermöglicht und RELAX zu einer in dieser Hinsicht (mindestens) genauso mächtigen Sprache wie SQL, QUEL usw. werden läßt.

Der `ORDNE`-Operator bewirkt eine **Sortierung** der Tupel nach den angegebenen Attributen, betrifft also de facto nur die Ausgabe der Anfrageergebnisse.

Die **Nestung** betrifft in RELAX ebenfalls nur die Form der Ausgabe. Bei Tupeln, die sich nur durch die Werte der Nestungs-Attribute (auch als „Nester" bezeichnet) nicht aber durch die der übrigen Attribute unterscheiden, werden die Werte der Nicht-Nest-Attribute nur einmal ausgegeben wird. Näheres zur Ausgabe der Nester findet man im Abschnitt 3.4.

Beispiel: Tabelle 3.1 zeigt in Teil (a) eine nicht genestete Relation und in Teil (b) dieselbe Relation nach Anwendung von

```
NEST [Land, @Orte <- (Ort)] (...)
```

IRIS würde bei der Ausgabe die genestete Relation durch „Kollabieren" der Nester zunächst auf drei Zeilen verkürzen und die vollen Nester erst auf expliziten Wunsch und dann einzeln ausgeben. Dies wird im Abschnitt 3.4 genauer beschrieben. □

Land	**Ort**
Spanien	Alicante
Deutschland	Clausthal
Deutschland	Fischen
Deutschland	Garmisch
Frankreich	Paris
Frankreich	LaRochelle

(a) Einfache 1NF-Relation

Land	**Orte** **Ort**
Spanien	Alicante
Deutschland	Clausthal Fischen Garmisch
Frankreich	Paris LaRochelle

(b) Dieselbe Relation in genesteter Form

Tabelle 3.1: Beispiel zur Nestung

Die Auswertung von RELAX-Anfragen beginnt mit der Prüfung auf syntaktische und semantische Korrektheit. Dabei wird ein Operatorbaum aufgebaut, der anschließend algebraisch optimiert und schließlich in pre-order-Reihenfolge abgearbeitet wird. Mehr dazu im Abschnitt 4.3.

3.1.2 URQUEL

URQUEL ist die Abkürzung für „universal relation query language“, zu deutsch Universalrelationen- Anfragesprache.

Ein Nachteil üblicher relationaler Anfragesprachen besteht darin, daß der Benutzer zur Anfrageformulierung Wissen über das zugrundeliegende Datenbankschema benötigt, um angeben zu können, welche Daten ausgegeben werden sollen und woher sie stammen. Insbesondere muß er wissen, welche Attribute in welchen Relationen definiert sind und wie über diese Verbindungen die Daten miteinander zu kombinieren sind.

Zur Umgehung dieses Problems wurden Universalrelationen-Schnittstellen wie URQUEL entwickelt. Unter bestimmten Bedingungen, etwa bestimmten Formen der Azyklizität, lassen sich nämlich die Relationen in der Datenbank *eindeutig* zu einer einzigen großen Relation verbinden, ohne daß dabei Verluste hinsichtlich des Informationsgehaltes entstehen. Diese Relation wird dann als **Universalrelation** bezeichnet.

Da es wegen der unvermeidbaren Redundanzen allerdings nicht sinnvoll ist, die Daten auch in dieser Form zu speichern, braucht man Hilfsmittel, um dem Benutzer wenigstens die Illusion zu vermitteln, es gäbe diese Universalrelation wirklich. Die Mechanismen, die eine Anfrage an die Universalrelation auf eine gleichwertige Anfrage an die tatsächliche Datenbank abbilden, werden als **Fensterfunktionen** bezeichnet.

Fensterfunktionen lassen sich auf grundsätzlich verschiedene Weisen definieren, man unterscheidet operationale und semantische Fensterfunktionen. In [Heu85a] ist die in IRIS-URQUEL verwendete operationale Fensterfunktion näher beschrieben. Ein Beispiel einer semantischen Fensterfunktion beschreibt [BV85].

Noch ausführlichere Informationen über das theoretische Fundament von URQUEL enthält [Heu85a]. In [LH85] wird die Fensterfunktion von URQUEL der des Dortmunder „Konkurrenzprodukts“ DURST gegenübergestellt und schließlich gezeigt, daß unter bestimmten Voraussetzungen beide Funktionen trotz ihrer völlig verschiedenen Ansätze gleichwertig sind.

Diese „bestimmten Voraussetzungen“, die ganz allgemein für ein vernünftiges Funktionieren einer Fensterfunktion gelten, sind der Grund für den engen Zusammenhang der algorithmischen Schemaentwurfskomponente IRIS-SYNTHI mit URQUEL: URQUEL kann nur verwendet werden, wenn das Datenbankschema mit dem Synthese-Algorithmus erzeugt wurde. Ob dies der Fall ist, kann aus dem DD/D entnommen werden. Außerdem testet URQUEL zu Beginn das Datenbankschema auf Azyklizität, eine notwendige Voraussetzung für die Anwendbarkeit der Fensterfunktionen.

Wie bei RELAX gibt es auch bei URQUEL eine *konzeptuelle* Anfrageoptimierung. Hier kommt ein speziell auf Universalrelationen-Schnittstellen zugeschnittenes **Tableau-Optimierungsverfahren** zum Einsatz. Grundlage dafür ist der in [Sag83] vorgestellte Algorithmus. Die Optimierung ist hier jedoch nicht optional, sondern ist Voraussetzung für die Anfrageinterpretation mittels Fensterfunktionen.

Zur URQUEL-**Syntax** sei hier nur angemerkt, daß die Grundform der Anfrage – ähnlich etwa QUEL – lautet:

`SUCHE` *attribute* `MIT` *bedingung*

wobei hinter `SUCHE` zunächst die Attribute anzugeben sind, auf die die (gedachte) Universalrelation projiziert werden soll. Außerdem können Tupelvariable verwendet werden, wodurch quasi mehrere Kopien der Universalrelation zur Verfügung stehen.

Hinter `MIT` steht die Selektionsbedingung, die *keine* Join-Bedingungen enthält, wie man sie von SQL kennt. Hier sind neben den Standard-Vergleichsoperatoren die Operatoren `AUS`, `NICHT AUS` (zur Prüfung von Elementbeziehungen mit konstanten Mengen), `IST LEER`, `IST NICHT LEER` (zur Prüfung, ob ein Tupel in einer bestimmten Menge existiert oder nicht; dient der Berücksichtigung charakteristischer Attribute [LH85]) und `WIE` (zum Vergleich einer Zeichenkette mit einer „Spezialkonstanten“ mit Wildcards) erlaubt.

Weiterführende Informationen zu URQUEL sind in [lst?] zu finden.

3.1.3 LDML

LDML, die „logical data manipulation language“, ist eine **regelbasierte Anfragesprache** für IRIS, beruhend auf dem Konzept der Zugriffs-Konstruktions-Regeln [Bro89]. Sinn und Zweck solcher Sprachen ist allgemein, neben der rein deskriptiven Formulierung die Grenzen von Relationenalgebra bzw. Relationenkalkül hinsichtlich ihrer „Mächtigkeit“ zu überwinden, also Anfragen zu ermöglichen, die mit letzteren nicht ausgedrückt werden können.

Eine solche Grenze liegt dort, wo Anfragen ihrer Natur nach *rekursiv* sind. Die Konstruktion derart abgeleiteter Informationen läßt sich oft ziemlich elegant unter Benutzung von **Ableitungsregeln** beschreiben, wie man sie etwa von PROLOG kennt.

Als **Beispiel** denke man sich eine Relation, in der Eltern und ihre Kinder eingetragen sind. Mittels einer relationenalgebraischen Anfrage kann man daraus zwar, sofern diese Informationen überhaupt in den Ausgangsdaten enthalten sind, zu jedem Menschen die Großeltern, die Geschwister, Cousins oder Urenkel ermitteln. Möchte man aber etwa *alle* Vor- oder Nachfahren einer Person wissen, so ist zwar klar, daß aufgrund der endlichen Zahl vorhandener

Daten auch in endlich vielen Schritten die Frage komplett zu beantworten wäre, aber man weiß nicht von vornherein, in *wieviel* Schritten. Daher ist diese Anfrage nicht mehr mit Relationenalgebra oder -kalkül auszuwerten.

Hier hilft eine Formulierung mittels Regeln weiter, die in diesem Beispiel sogar der „natürlichen“ Definition des Vorfahrenbegriffs viel näher ist als eine Folge von Selektionen und Verbunden:

> Ist A Kind von B, so ist B Vorfahre von A.
> Ist C Vorfahre von B und B Vorfahre von A, so ist C Vorfahre von A.

LDML [rr88] ermöglicht die Formulierung und Auswertung solcher Datenbankanfragen in IRIS. Als Hilfsmittel wird dabei wiederum auf die (erweiterte) Relationenalgebra in Gestalt von RELAX zurückgegriffen, d.h.: LDML-Anfragen werden zwecks Auswertung zunächst in relationenalgebraische Ausdrücke umgeformt, deren wiederholte Auswertung jedoch durch einen speziellen Mechanismus kontrolliert werden muß.

Ganz grob betrachtet, zerfällt die Bearbeitung einer LDML-Anfrage in zwei Teile: Zunächst wird eine Menge von Regeln durch Eintragung in die für LDML vorgesehenen Systemrelationen definiert. Danach wird diese Regelmenge nach Angabe der gewünschten Zielrelation („intensionale Relation“) und eventueller Vorgabe von Attributwerten („gebundene Attribute“) von IRIS ausgewertet.

Beispiel: Sei die Relation „Flug“ (Tabelle 3.2) in der Datenbank abgespeichert. (Hier weichen wir vom Schema in Anhang E ab.) Weiterhin seien folgende Regeln vorhanden:

1:
```
Flug : Start Ziel
-----------------
Reise : Start Ziel
```

2:
```
Reise : Start zw(Ziel)
Flug : zw(Start) Ziel
---------------------
Reise : Start Ziel
```

Dann ist folgende Anfrage möglich:

> **Intensionale Relation:** `Reise`
> **Gebundenes Attribut:** `Start`
> **Start:** `Hamburg`
> **Start:** `Moskau`

Sie liefert als Ergebnis die Relation in Tabelle 3.3. Das heißt, daß alle Tupel der abgeleiteten, durch obige Regeln definierten Relation „Reise“ auszugeben sind, deren Attribut „Start“ entweder den Wert „Hamburg“ oder den Wert „Moskau“ enthält. Gesucht sind also alle Flugreisen von Hamburg nach Moskau mit beliebig vielen (auch null!) Zwischenlandungen.

Seien überdies die Relation „Schiffahrt“ (Tabelle 3.4) und folgende Regeln gegeben:

Start	Ziel
Hannover	Frankfurt
Hamburg	Frankfurt
Köln	Paris
Köln	Amsterdam
Frankfurt	Moskau
Frankfurt	New York
Frankfurt	Köln

Tabelle 3.2: Die Relation „Flug“.

Start	Ziel
Hamburg	Frankfurt
Hamburg	Moskau
Hamburg	New York
Hamburg	Köln
Hamburg	Paris
Hamburg	Amsterdam

Tabelle 3.3: Das Ergebnis der ersten LDML-Anfrage.

3:
```
Schiffahrt : Start Ziel
-----------------------
Reise : Start Ziel
```

4:
```
Reise : Start zw(Ziel)
Schiffahrt : zw(Start) Ziel
---------------------------
Reise : Start Ziel
```

Dann ist folgende Anfrage möglich:

Intensionale Relation: `Reise`
Gebundenes Attribut: `Start`
Gebundenes Attribut: `Ziel`
Start: `Hamburg`
Ziel: `Hamburg`
Start: `Hamburg`
Ziel: `Hannover`

Sie liefert als Ergebnis die Relation in Tabelle 3.5.

3.1.4 NERO

NERO bedeutet „nested relational algebra for objects“, womit bereits zum Ausdruck kommt, daß es sich hierbei um eine algebraische Anfragesprache für **genestete Relationen** han-

Start	**Ziel**
New York	Hamburg
Hamburg	Bombay
New York	Bombay

Tabelle 3.4: Die Relation „Schiffahrt“.

Start	**Ziel**
Hamburg	Hamburg

Tabelle 3.5: Das Ergebnis der zweiten LDML-Anfrage.

delt, die als Grundlage für die Implementierung einer algebraischen Anfragesprache für das Objektbanksystem OSCAR und sein Datenbankmodell EXTREM konzipiert wurde.

Unter genesteten (geschachtelten oder NF^2-) Relationen versteht man Strukturen, die zunächst wie die „flachen“ Relationen des Codd'schen Relationenmodells Mengen von Tupeln sind. Nur sind die Komponenten dieser Tupel nicht mehr nur elementare, nicht weiter zerlegbare Werte wie Zahlen oder Zeichenketten, sondern können wiederum ganze Relationen sein, die ihrerseits wieder Mengen von Tupeln mit beliebig komplexen Komponenten sind. Da diese Struktur der sog. „ersten Normalform“ (1NF) nicht mehr genügt, kam die Bezeichnung „Non First Normal Form“, abgekürzt NF^2, zustande. Für NERO sind allerdings nicht alle NF^2-Relationen zugelassen, sondern lediglich solche, bei denen auf jeder Ebene (d.h. in jeder Teilrelation) atomare Attribute existieren, die für diese Ebene einen Schlüssel darstellen. Solche Relationen werden als in Partitioned Normal Form (PNF) [VGF86] befindlich bezeichnet. Die Menge der „erlaubten“ Relationen ist hierbei größer als die in [AB86] zugelassene. Sie können immer durch vollständige Entnestung als flache Relationen dargestellt und ohne Verlust von Information zur ursprünglichen PNF-Relation zurückgenestet werden.

Die auf solchen Strukturen definierten algebraischen Operationen sind ähnlich denen des Relationenmodells. Bedingt durch die komplexere Struktur der bearbeiteten Objekte, ist jedoch die Semantik der einzelnen Operationen z.T. recht kompliziert. Projeziert werden darf nicht nur auf flache Attribute, sondern auch auf Nester und rekursiv in Nester hinein. Hierbei ist zu beachten, daß die Ergebnisrelation auch wieder in PNF ist, d.h. neben den explizit angegebenen Projektions Attributen müssen sich dort auch die flachen Attribute jedes erhaltenen Nestes (Schlüssel!) wiederfinden. In Selektionsbedingungen können neben den aus RELAX bekannten Beziehungen flacher Attribute auch Elementbeziehungen zwischen Attributen und Nester sowie Mengenvergleiche und Teilmengenbeziehungen zwischen Nestern getestet werden. Bei den Mengenoperationen Vereinigung, Differenz und Durchschnitt ist darauf zu achten, daß beide Relationen jeweils das gleiche Schema haben, inklusive der Nestungsstruktur. Joins über Nester sind nur möglich, wenn die Nester in beiden Relationen gleiche Namen und gleiche Struktur haben.

Im wesentlichen orientieren sich Sprachumfang und Syntax von NERO [gk90] an RELAX; die meisten RELAX-Anfragen sind daher auch gültige NERO-Anfragen. Dies ist insbesondere deshalb wichtig, weil IRIS ja eigentlich nur flache Relationen kennt und die Schachtelung

erst explizit bei der Anfrageformulierung herbeigeführt werden muß. Der dafür erforderliche `NEST`-Operator steht sowohl in RELAX als auch in NERO zur Verfügung, wenn auch mit unterschiedlicher Syntax. Während er in RELAX aber nur die Ausgabe des Resultats betrifft und daher nur ganz „außen“ angewandt werden kann, gibt es in NERO keine derartigen Einschränkungen.

Besonders interessant wird NERO durch die Fähigkeit, auch Mengen gleicher Bauart, etwa parallele „Nester“, in Selektionsbedingungen miteinander zu vergleichen, also zu überprüfen, ob Teilmengenbeziehungen bestehen oder sogar Gleichheit vorliegt. Auf diese Weise lassen sich Anfragen sehr elegant formulieren, die in RELAX (wenn überhaupt) nur mit großem Aufwand möglich wären.

Dazu ein **Beispiel:** Gegeben seien zwei Relationen, deren erste[1] Urlauber und ihre Hobbies (beliebig viele!) und deren zweite Urlaubsorte und ihre Freizeitangebote (ebenfalls beliebig viele!) enthält. Man versuche einmal, in RELAX die Frage zu beantworten, welcher Urlaubsort für welchen Urlauber geeignet ist, wenn wir unter „geeignet“ verstehen wollen, daß der Urlauber dort allen seinen Hobbies nachgehen kann.

In NERO sich die Anfrage etwa wie folgt formulieren:

```
Url := NEST [Hobby : @Hobbies] (Urlauber JOIN Hobbies);
Ang := NEST [Angebot : @Angebote] (Urlaubsorte);
PI [Name, Ort] (SIGMA [@Hobbies <= @Angebote] (Url JOIN Ang ))
```

Man beachte, daß der Vergleichsoperator `<=` hier die Bedeutung „ist Teilmenge von“ trägt, wenn auf beiden Seiten Nestnamen (erkennbar am Präfix `@`) stehen.

Im Prinzip läuft die Auswertung einer NERO-Anfrage genauso ab wie die einer RELAX-Anfrage: Zunächst werden Syntax und Semantik auf eventuelle Fehler überprüft. Dabei wird ein Operatorbaum aufgebaut, der allerdings nicht direkt abgearbeitet werden kann, sondern – sofern nicht vorher schon Fehler entdeckt wurden – zunächst in eine relationenalgebraische Anfrage in Gestalt einer Liste relationenalgebraischer Operatorbäume übersetzt werden muß.

Eine komplette Überarbeitung der ursprünglichen Implementierung von NERO lieferte NERO++ [gt92]. Korrigiert wurden hierbei die Selektionen mit `NICHT`, `AUS` und `NAUS` (`NICHT AUS`), Mengenoperationen (`VEREIN`, `DURCH`, `DIFF`), der Operator `UNNEST` sowie die Spaltenerweiterung mit komplexen Ausdrücken. Neu hinzugekommen sind Spaltenerweiterungen mit Stringdomänen und den zugehörigen Operatoren `STR` (Umwandlung von Integer/Real nach String), `VAL` (Umwandlung von String nach Integer/Real), `SUB` (Teilstring) und `&` (Konkatenation) sowie die Umbenennung von Nestern. Als weiterer Erfolg ergab sich eine drastische Performance-Steigerung.

3.1.5 NaLogo!/NaLogo!^{opt}

NaLogo! (Nested approach to a Logic for objects!) ist eine regelbasierte Anfragesprache für geschachtelte Relationen. Genau wie NERO handelt es sich auch hier um einen Zwischen-

[1] Um redundante Daten zu vermeiden, ist sie in unserem Beispiel (Anhang E) auf die Relationen „Urlauber“ und „Hobbies“ aufgeteilt, aus denen sie durch einen Join hergeleitet werden kann.

schritt beim Übergang vom wertorientierten Relationenmodell bei IRIS zum objektorientierten EXTREM-Modell des OSCAR-Systems.

Auch NaLogo! beruht auf dem Prinzip der Zugriffs-Konstruktions-Regeln [Bro89]. Das theoretische Fundament ist also weitgehend das gleiche wie das von LDML.

NaLogo!^{opt} ist eine Weiterentwicklung von NaLogo!, bei der die Performance der Regelauswertung durch einige Optimierungsverfahren (Differenzen-Verfahren, Stratifikation, Magic-Set-Verfahren: s.u.!) gesteigert werden sollte. „Repräsentative" Tests haben allerdings gezeigt, daß lediglich die Stratifikation — wenigstens in einigen Fällen — hierzu in der Lage ist. Die anderen Verfahren benötigten in den getesteten Fällen für die Anfrageauswertung (inklusive des Zusatzaufwands für die Optimierung) mehr Zeit als die Standard-Auswertung, zum Teil bis zu 270 % !

NaLogo!^{opt} stellt die folgenden fünf Auswertungs-Verfahren zur Verfügung:

Standard-Auswertung Dies ist die „alte" NaLogo!-Auswertung. In jeder Iteration werden *alle* anwendbaren Regeln verwendet, um Tupel zu berechnen, und dies solange, bis sich die Tupelanzahl aller Zwischenrelationen nicht mehr erhöht. Auf jegliche Optimierung wird also verzichtet.

Differenzen-Verfahren Hierbei wird im Gegensatz zum Standard-Verfahren in jedem Iterationsschritt nur der echte Tupelzuwachs (die Differenz) berechnet, der sich aus dem Tupelzuwachs des vorhergehenden Schrittes ergibt [Ban86]. Es werden also weniger Tupel in die Regel-Anwendungen eingebracht. Dazu wird der ursprüngliche Anfragetext algebraisch umgeformt.

Stratifikation Bei diesem Verfahren werden nicht *alle* Regeln in *jeder* Iteration angewandt, sondern es werden Schichten (Strata) von Regeln gebildet, in denen die wechselseitig rekursiven Regeln liegen. Diese Schichten sind so angeordnet, daß zu ihrer Berechnung nur Ergebnisse zuvor ausgewerteter Schichten benötigt werden. Die Schichten werden nacheinander mit dem Standard-Verfahren berechnet. Somit werden evtl. einige Regeln seltener verwendet.

Magic-Set-Verfahren Das Magic-Set-Verfahren beruht auf „Sideway Information Passing" (SIP), d.h. es muß mindestens ein Attribut der Regel durch eine Konstante gebunden sein. Dies entspricht einer Selektion und soll geschickt bei der Berechnung berücksichtigt werden.

Anmerkung: Dieses Verfahren kann nur auf Regeln in 1. Normalform (d.h. mit atomaren Attributen in Kopf und Rumpf) angewandt werden. Außerdem muß mindestens ein Attribut bei der Anfrage gebunden werden (s.o.!) und mindestens eins ungebunden bleiben, da nur die ungebundenen Attribute in der Ergebnisrelation auftauchen.

LDML-Auswertung Wie das Magic-Set-Verfahren verwendet auch die LDML-Auswertung SIP, um die Anfrageberechnung zu optimieren.

Anmerkung: Auch dieses Verfahren kann nur auf Regeln in 1. Normalform angewandt werden. Wiederum muß mindestens ein Attribut in der Anfrage gebunden werden.

Voreingestelltes Auswertungsverfahren ist die Stratifikation.

Folgendes Beispiel soll einen Eindruck von der **NaLogo!**–Syntax geben. `Verbindung` sei eine Basisrelation mit den zwei Attributen `Start` und `Ziel`.

1. Regel mit Bedingung:

```
Verbindung : Start Ziel,
Start = 'Clausthal'/
Clausthal_Verbindung : Start Ziel.
```

2. Selektion im Zugriff anstelle einer Bedingung (äquivalent zu 1.):

```
Verbindung : Start = 'Clausthal' Ziel/
Clausthal2_Verbindung : Start Ziel.
```

3. Nestung:

```
Verbindung : Start Ziel/
NestVerbindung : Start @Ziele({Ziel}).
```

4. Spaltenerweiterung:

```
Verbindung : Start Ziel/
neueVerbindung : Start Ziel Neu = anzahl(*) ; (Start).
```

Die Bedienung erfolgt so, daß zunächst unter dem Menüpunkt „Regelverwaltung“ Regeln eingegeben, geändert, gelöscht, abgespeichert und wieder eingelesen werden können.

Regeln werden mit einem Namen und einer Nummer versehen und in speziellen Relationen des erweiterten Data Dictionary gespeichert. Eine NaLogo!-Anfrage besteht dann nur noch in der Angabe *einer* durch die vorhandenen Regeln definierten Zielrelation. Zusätzlich kann der Benutzer (dank NaLogo!^{opt}) ein Verfahren für die Auswertung seiner Anfrage wählen.

Die Auswertung von Regel-Anfragen ist in [bus91] beschrieben. Eine Benutzer-Dokumentation zu NaLogo!^{opt} liegt in [kk92] vor. Auch Syntax-Diagramme sind dort zu finden.

3.2 IRIS-PASCAL

Im Gegensatz zu den bisher besprochenen rein interaktiven Anfragesprachen von IRIS stellt IRIS-PASCAL (im folgenden mit IP abgekürzt) eine komplette **Entwicklungsumgebung** dar. Mit IP wurde also eine Möglichkeit geschaffen, **Anwendungsprogramme** zu erstellen, die direkt auf IRIS zugreifen — übrigens ohne daß man als Programmierer erst das ganze Datenbankschema mühevoll definieren muß.

3.2.1 Grundlagen

Es besteht ein entscheidender Unterschied zwischen reinen Anfragesprachen und einer universellen Programmiersprache wie Pascal hinsichtlich ihrer **Mächtigkeit**: Pascal erreicht die Mächtigkeit von Turing-Maschinen, d.h.:

> *Alles, was intuitiv als Algorithmus angesehen wird, kann mit einer Turing-Maschine berechnet werden.*
>
> (Turing'sche These)

Diese These ist zwar nicht beweisbar, aber bei Turing-Maschinen ist noch nicht einmal entscheidbar, ob sie überhaupt einmal ihre Arbeit beenden — und genau das gilt damit auch für Pascal.

Für RELAX gilt das nicht, auch nicht für SQL oder irgendeine ähnliche Sprache. Diese Mächtigkeit will man auch gar nicht erreichen, da sie u.a. die Möglichkeit beinhaltet, Endlosschleifen zu konstruieren, die für ein Datenbank-System fatal wären. Da aber für viele Zwecke gerade solche Fähigkeiten gebraucht werden, ist es üblich, sie so zu verpacken, daß wenigstens die Datenbank von den denkbaren Folgen entlastet wird, also etwa in Form einer Programmiersprache, die inclusive Datenbankzugriffen „alles kann", letztere aber in einer „sicheren" Art und Weise durchführt.

Prinzipiell kann die Zusammenführung von Programmiersprache und Datenmanipulationssprache (DML[2]) auf drei verschiedene Weisen erfolgen:

Call DML: Hierbei wird der Text einer Anfrage in Form eines Strings an die DML übergeben. Vorteile sind dabei die Flexibilität und der geringe Aufwand bei der Realisierung, der entscheidende Nachteil ist allerdings, daß Syntaxfehler erst zur Laufzeit bei der Analyse des übergebenen Strings erkannt werden.

Embedded DML: Hier wird die Syntax der Gastsprache um Datenbankkonstrukte erweitert, welche im Quelltext durch Schlüsselworte markiert werden. Ein Precompiler erkennt die datenbankrelevanten Teile, überprüft deren Syntax und übersetzt sie in die Syntax der Gastsprache.

Integrated DML: Hierbei verschmelzen DML und Gastsprache zu einer Sprache mit einheitlicher Syntax, die die Möglichkeiten beider Sprachen vereint. Dies stellt die eleganteste Lösung für den Benutzer dar, erfordert aber die Entwicklung eines neuen Compilers.

Bei IP wurde die Variante „*embedded DML*" realisiert.

3.2.2 Anwendung und Bedienung

Aufgrund der gewählten Realisationsvariante als „embedded DML" besteht die Entwicklung von IP-Anwendungen aus mehreren Schritten. Zunächst wird das „gemischte" Anwendungsprogramm – in der IP-Nomenklatur als „Workfile" bezeichnet – geschrieben, danach übersetzt der Precompiler die Datenbank-Anteile, die in IP durch die Schlüsselworte `IRIS` und `IRISEND` zu klammern sind, in ein „reines" Pascal-Programm, das die IP-Library benutzt. Die weitere Bearbeitung erfolgt wie bei normalen Programmen mit (MS-Pascal-) Compiler und Linker. Der gesamte Bearbeitungsvorgang wird von IRIS aus kontrolliert.

[2]Hier im Sinne von „Anfragesprache" verwendet, nicht mit der Update-Schnittstelle zu verwechseln!

```
(* IRISPASCAL-Quellprogramm *)

PROGRAM  Beispiel ( input, output );

IRIS  USES Beispiel;  IRISEND;

PROCEDURE  print_RelAtt;
(* Ausgabe aller 'manuell' angelegten Relationen mit den zugeh\"origen *)
(* Attributen                                                          *)
BEGIN
  IRIS  QUERY ( Erg );
    pi [Relname, Rel_Id, Att_Id, Att_Name]
     ( ( ( sigma [Bemerkung='manuell'] (Relation) )
         join  Rel_Att )
       join  Attribut );
  IRISEND;
  writeln ( 'Relation Relation / Attribut' );
  writeln;
  writeln ( '            Relname |Rel_Id|Att_Id|            Att_Name' );
  writeln ( '--------------------+------+------+--------------------' );
  ErgCursor:=ErgWurzel;
  IF  IrisResult = 0
    THEN  WHILE  ErgCursor.r <> 0  DO
          BEGIN
            writeln ( ErgCursor^.Relname:20, '|',
                      ErgCursor^.Rel_Id:6,   '|',
                      ErgCursor^.Att_Id:6,   '|',
                      ErgCursor^.Att_Name:20 );
            ErgCursor := ErgCursor^.next;
          END
    ELSE  writeln ( 'Irisfehlernummer : ', IrisResult );
  ErgLoeschen;
END;

BEGIN
  DATAF_Datei := 'Beispiel.dat';
  FREI_Datei  := 'Beispiel.fre';
  IF  NOT einschleusen( DATAF_Datei, FREI_Datei )
    THEN  writeln ( 'Fehler !!!' )
    ELSE  print_RelAtt;
END.
```

Abbildung 3.1: Ein IRIS-PASCAL-Beispielprogramm

Es ist in IP *nicht* erforderlich, in jedem einzelnen Anwendungsprogramm erst einmal das komplette Datenbankschema explizit zu definieren. Stattdessen ist lediglich im ersten `IRIS ... IRISEND` - Block der Name der Datenbank anzugeben:

`USES` *dataf-name*

IP bzw. das Anwendungsprogramm entnimmt dieser Zeile, nach welchem Datenfile es suchen soll. Konkret werden statt der Default-Files `DATAF` und `FREI` die Dateien *dataf-name*`.DAT` und *dataf-name*`.FRE` benutzt. Aus dieser Datenbank liest IP während der Precompilation dann selbständig das Data Dictionary ein. Anhand dieser Informationen kann dann auch schon zur Precompile-Zeit (und nicht erst zur Laufzeit!) festgestellt werden, ob alle folgenden Datenbank-Zugriffe syntaktisch und semantisch korrekt sind. Hier wird u.a. geprüft, ob alle angesprochenen Attribute und Relationen existieren. Einziger „Haken“ dabei: Wenn am Schema einer Datenbank Änderungen vorgenommen werden (Schema-Evolution), müssen alle Anwendungsprogramme, die diese Datenbank benutzen, neu übersetzt werden. Ansonsten könnte nicht gewährleistet werden, daß sie trotz der Änderungen noch korrekt ablaufen.

Die Interaktion des Anwendungsprogramms mit IRIS beschränkt sich auf die Formulierung von RELAX-Anfragen und DML-Statements, also Anfragen und Update-Kommandos. Auf die Einbindung der anderen Anfragesprachen mußte mangels „Manpower“ verzichtet werden.

Interessant wird die Verwendung von IP auch dadurch, daß – im Gegensatz zur interaktiven Arbeit – die RELAX-Anfragen **parametrisiert** werden können, d.h. daß Selektionen nicht nur nach Konstanten, sondern auch nach Variablen des IP-Programms möglich sind. Das Anfrageergebnis kann dann zur weiteren Bearbeitung tupelweise durchlaufen werden. Dazu stehen spezielle Cursor-Variable zur Verfügung, die wie normale Pascal-Records verwendet werden können und jeweils ein Tupel der Relation enthalten. Die Attributnamen werden als Feld-Bezeichner übernommen. Dem Benutzer wird dabei von IP der Deklarationsaufwand für diese Variablen weitestgehend abgenommen.

Ähnlich ist die Vorgehensweise bei Updates. Als Ergebnis stellt IP die von IRIS erzeugten Statusmeldungen in besonderen System-Variablen bereit. Anhand dieser Werte kann festgestellt werden, ob die Änderung durchgeführt wurde, oder ob eine Schlüssel- oder Fremdschlüsselverletzung vorlag.

3.2.3 Implementierung

Die für das Gesamtprojekt bedeutendste Konsequenz aus der IP-Implementierung war die Einführung des „**IRIS-Treibers**“, dessen Aufgabe darin besteht, die Umschaltung zwischen IRIS und IP zu bewerkstelligen, die nötig wurde, weil beide Teile zusammen nicht mehr in den beschränkten DOS-Speicher passen. (Später wurde der Treiber auch für andere Komponenten genutzt; namentlich für die DML und SYNTHESE.) Daher kann IP auch nur aufgerufen werden, wenn IRIS als `IPEXEC` gestartet wurde.

IRIS und IP sind also eigentlich als getrennte Programme realisiert, die allerdings ausgiebig miteinander kommunizieren müssen, wozu auch die Benutzung von Hauptspeicher-Inhalten (Anfragebaum, globale Variable, Hauptspeicher-DD/D, internes `MINDIC`) sowie die gemeinsame Benutzung von Files gehört. Dies alles koordiniert der „Treiber“.

3.3 Die Update-Komponente

Bisher wurden die verschiedenen Möglichkeiten betrachtet, Daten aus der Datenbank zu extrahieren. Zuvor müssen sie aber erst einmal eingegeben werden. Außerdem muß es möglich sein, sie zu löschen oder zu ändern. Genau diesem Zweck dient die Update-Komponente.

Die eben geforderten Möglichkeiten der Datenmanipulation bietet IRIS-Update in zwei verschiedenen Formen: Erstens interaktiv mit Fenstertechnik und „Vorselektion“, zweitens in Form einer als DML (Data Manipulation Language) bezeichneten leistungsfähigen formalen Sprache.

3.3.1 Datenintegrität

Die interaktive Update-Komponente bietet – neben der komfortablen Bedienung – die bereits in der Einleitung angesprochene Fremdschlüssel-Verfolgung. Das bedeutet, daß zunächst alle in der Datenbank bestehenden Fremdschlüssel-Beziehungen ermittelt und in Form eines sog. „Fremdschlüssel-Graphen“ verwaltet werden.

Ein **Fremdschlüssel** ist dabei jede Teilmenge X der Attribute einer Relation r_1, die in dieser Relation *kein* Schlüssel ist, aber in einer anderen Relation r_2 einen Schlüssel bildet. Dabei wird davon ausgegangen, daß Attribute gleichen Namens (und in IRIS dann zwangsweise gleichen Typs) in verschiedenen Relationen dieselbe Bedeutung haben. Das ist sinnvoll, wenn man (wie bei IRIS) Attributdefinitionen als global ansieht und nicht auf den Kontext einer einzigen Relation beschränkt. Eine Fremdschlüssel-Bedingung bedeutet dann, daß alle Werte der betreffenden Attribute X in r_1 auch als Werte dieser Attribute in r_2 vorkommen müssen. Es handelt sich also um eine globale Integritätsbedingung, speziell um eine Form der Inklusionsbedingung.

Das heißt, daß ein Tupel nicht in r_1 eingefügt werden kann, wenn es in r_2 kein Tupel mit denselben X-Werten gibt. Entweder unterbleibt das Einfügen in r_1, oder in r_2 muß ein passendes Tupel eingefügt werden – ggf. auch mehrstufig. Umgekehrt darf ein Tupel nicht aus r_2 gelöscht werden, wenn es in r_1 noch Tupel mit denselben X-Werten gibt. Hier wäre dann entweder das Tupel in r_2 zu belassen, oder man müßte zuvor die entsprechenden Tupel aus r_1 entfernen. Das Ändern eines Tupels in r_1 oder r_2 kann ebenfalls zu Fremdschlüssel-Verletzungen führen, die dann in analoger Weise zu behandeln wären.

Treten im Fremdschlüsselgraphen Zyklen auf, was bei manueller Schemadefinition nicht auszuschließen ist, kann für Updates nur auf die DML-Schnittstelle ausgewichen werden.

Die DML-Schnittstelle nimmt nämlich keine Fremdschlüssel-Verfolgung vor, was auch schwierig wäre, da über die aus eventuellen Fremdschlüssel-Verletzungen resultierenden Aktionen letztlich nur interaktiv wirklich sinnvoll entschieden werden kann. Es wird daher nur eine *Prüfung* der Schlüssel- und Fremdschlüssel-Bedingungen vorgenommen. Verletzt ein Update eine Bedingung, wird es zurückgewiesen.

Dafür bietet die DML in Punkto physischer Datensicherheit mehr, indem sie ein **Commit-Protokoll** implementiert, das es erlaubt, nach Störungen während des Updates zu einem konsistenten Datenbankzustand zurückzukehren, indem nach einem „Synchronisationspunkt“ erfolgte Änderungen am Datenbestand automatisch rückgängig gemacht werden.

3.3.2 Die interaktive Update-Schnittstelle

Die interaktive Update-Schnittstelle erreicht man unter dem Menüpunkt „Dialogkomponente“. Wählt man dort „Interaktives Update“, so kann man zunächst aus einer Liste die gewünschte Relation auswählen. Es werden übrigens nur die Benutzerrelationen angezeigt, die des DD/D aus naheliegenden Gründen nicht.

Anschließend erscheint das Fenstersystem der **Vorselektion**. Sie bietet die Möglichkeit, die Menge der im folgenden angezeigten Tupel von vornherein einzuschränken, um auch bei großen Datenmengen die Übersicht zu behalten. Die Vorselektion erfolgt durch Eingabe einer Selektionsbedingung, die sich aus einfachen Bedingungen an die einzelnen Attribute zusammensetzt.

Die Bedingungen für die einzelnen Attribute werden konjunktiv, d.h. durch logisches „und“, miteinander verknüpft. Dadurch können hier zwar nicht alle denkbaren Selektionsbedingungen formuliert werden, aber für den angestrebten Zweck ist das wohl tolerierbar. Zusätzlich zu den von RELAX angebotenen Vergleichsoperatoren steht hier noch der Operator `ZWI` zur Verfügung, der überprüft, ob ein Attributwert *zwischen* den beiden angegebenen konstanten Werten liegt. Die Vorselektion wird zur Auswertung in eine RELAX-Anfrage übersetzt.

Anschließend wird die vorselektierte Relation angezeigt und kann nun bearbeitet werden. Man kann alle oder einzelne Tupel markieren, die markierten Tupel löschen oder nacheinander ändern, oder man kann neue Tupel eingeben. Alle Aktionen erfolgen fenstergestützt, die Bedienung wird jeweils am Bildschirm erläutert. Außerdem stehen Hilfstexte zur Verfügung.

Treten Fremdschlüssel-Verletzungen auf, so wird zunächst angegeben, welches Tupel sie verursacht, und es werden die möglichen Folgeaktionen aufgeführt. Per Tastendruck kann der Benutzer dann über die Behandlung des „Fehlers“ entscheiden. Welche Möglichkeiten grundsätzlich bestehen, wurde bereits weiter oben erläutert. **Achtung:** Wenn man sich nicht gerade dafür entscheidet, alle Konflikte durch Eingabe fehlender Tupel oder radikales Löschen vorhandener zu lösen, finden implizite „Rollbacks“ statt, d.h.: Alle bis dahin durchgeführten Änderungen werden rückgängig gemacht; die Datenbank sieht hinterher (äußerlich) genauso aus, als wäre nichts geschehen.

Nach dem Abschluß eines Updates, hier auch als „**Transaktion**“ bezeichnet, listet IRIS wieder die *vorselektierte* Basisrelation auf, der Benutzer kann dann die nächste Transaktion starten. Zu beachten ist, daß durch die Vorselektion evtl. auch Tupel ausgeblendet werden, die soeben erst eingegeben wurden. Dies trifft genau für die neu eingegebenen Tupel zu, die die Selektionsbedingung nicht erfüllen.

3.3.3 Die DML-Schnittstelle

Die DML [bib90] umfaßt Kommandos zum Einfügen (`INSERT`), Löschen (`DELETE`) und Ändern (`MODIFY`) von Tupeln. Nach `INSERT` wird genau ein Tupel durch Angabe seiner sämtlichen Attributwerte in der Form „*attname* `:=` *attwert*“ eingegeben, die Reihenfolge der Attribute ist beliebig. `DELETE` und `MODIFY` haben jeweils drei Varianten: Bei der ersten wird durch „`KEY = ...`“ genau ein Tupel durch Angabe eines Schlüssels spezifiziert, bei der zweiten eine Menge von Tupeln durch eine Selektionsbedingung, wie sie von RELAX bekannt ist, und bei

der dritten durch das Schlüssel Wort „`ALL`“ alle Tupel der Relation. `DELETE` löscht die so spezifizierten Tupel; `MODIFY` ändert deren Attributwerte, wobei die neuen Attributwerte wie bei `INSERT` anzugeben sind (s.o.), zusätzlich aber auch RELAX-Spaltenerweiterungsausdrücke zum Berechnen der neuen Attributwerte aus den alten angewendet werden können. Bei jedem Kommando ist zusätzlich der Name der betroffenen Relation anzugeben.

Ein Update mit Hilfe der DML-Schnittstelle läuft in der Regel wie folgt ab:

Zunächst ist ein DML-„Programm“ zu erstellen und in der Datei `UPDATE.DAT` zu speichern. In diesem Programm dürfen beliebige DML-Statements, die auf beliebigen (aber existierenden!) Relationen arbeiten, hintereinander ausgeführt werden. Danach muß die Syntax des Programms überprüft und das Programm für die interne Auswertung der DML-Statements übersetzt werden. Anschließend kann die Transaktion, die das Programm abarbeitet, gestartet werden. Gleichzeitig wird ein File `ROLLBACK.DAT` mit „entgegengesetzten“ DML-Kommandos aufgebaut, das zum Rückgängigmachen der Änderungen gebraucht wird. Achtung: Schlüssel- und Fremdschlüssel-Bedingungen werden nur *überprüft*, aber es erfolgt *keine* Fremdschlüsselverfolgung!

Per „Commit“ werden alle bisher durchgeführten Änderungen „bestätigt“. Sie können danach nicht mehr automatisch rückgängig gemacht werden. Das File `ROLLBACK.DAT` ist dann leer oder völlig gelöscht. Mit „Rollback“ können die letzten, noch nicht per „Commit“ bestätigten Änderungen rückgängig gemacht werden. Wird eine Transaktion abgebrochen, so wird anschließend ein Rollback ausgeführt.

Das DML-Menü bietet weiterhin die Möglichkeit, Metadaten, d.h. Informationen über alle mit der DML änderbaren Benutzerrelationen, auszugeben. Die Ausgabe erfolgt in das File `RELINFO.DAT` und umfaßt zu jeder Relation ihren Namen, ihre Attribute mit deren Domänen sowie die Schlüsselmengen.

Mit dem Verlassen der DML-Schnittstelle werden alle noch unbestätigten Updates automatisch „committed“. Ein automatisches „Rollback“ findet statt, wenn beim Starten von IRIS eine nichtleere Datei `ROLLBACK.DAT` vorgefunden wird. Dies wird als Indiz dafür angesehen, daß IRIS durch einen Systemfehler oder Stromausfall nicht mit der Bearbeitung aller Änderungen fertig wurde. Anhand des Files wird dann zuerst ein konsistenter Datenbank-Zustand wiederhergestellt (Recovery).

Hundertprozentige Sicherheit können wir bei IRIS konzeptbedingt *nicht* garantieren, und zwar ist daran das im Kapitel „Interne Ebene“ vorgestellte Puffer-Konzept schuld: Alle Änderungen am Datenbestand werden zunächst auf dem Puffer im Hauptspeicher durchgeführt, aber eben *nur* dort. Auf Platte geschrieben werden sie erst, wenn die betreffende Seite aus dem Puffer entfernt werden muß, um einer anderen Platz zu machen, oder wenn die gesamte Transaktion abgeschlossen wird. Zwischenzeitlich ist es also durchaus möglich, daß – etwa durch einen Stromausfall oder einen Hardware-Fehler – das System zusammenbricht, bevor alle Änderungen auf Platte geschrieben wurden. IRIS betrachtet aber schon die Änderung im Puffer als maßgeblich für den Erfolg eines Updates. Ein Log-Protokoll oder Schattenspeicher-Konzept könnte hier Abhilfe schaffen.

3.4 Ausgabe von Relationen

Die interaktive Ausgabe von Relationen ist in IRIS nur auf dem Bildschirm möglich, nicht etwa auf dem Drucker oder auf Platte. Für diese Zwecke steht aber IRIS-PASCAL zur Verfügung. Auf dieser Basis könnte z.B. auch ein Report-Generator o.ä. implementiert werden.

Die Ausgabe einer Relation erfolgt wie gewohnt in tabellarischer Form, wobei die Attributnamen als Spaltenüberschriften dienen und jene, die den Primärschlüssel bilden, farbig hervorgehoben werden. Da die Relation meistens breiter und höher als der Bildschirm ist, kann sowohl vertikal als auch horizontal gescrollt werden.

Interessant ist die **Ausgabe genesteter Relationen**, wie sie RELAX oder NERO liefern: In der Spaltenüberschrift genesteter Attribute wird außer dem Nestnamen auch dessen Struktur, d.h. die Namen der darin enthaltenen Attribute, angezeigt.

Anfangs werden zu jedem Nest nur die jeweils ersten Teiltupel angezeigt, so daß die Relation erst einmal „flach“ erscheint. Der Benutzer kann aber jedes Nest anwählen und per Tastendruck öffnen. Es erscheint dann ein neues Fenster, das *alle* Teiltupel dieses Nests anzeigt, die zum zuletzt aktuellen Tupel der Gesamtrelation gehören. In diesem Fenster kann wiederum vertikal gescrollt werden, das zuletzt aktuelle Teiltupel wird nach dem Schließen des Fensters als Repräsentant des ganzen Nests gewählt.

3.5 Sichten

3.5.1 Begriffsklärung

Unter (Benutzer-) Sichten (engl.: **Views**) versteht man Relationen, die mit Hilfe einer Anfrage aus gegebenen Basisrelationen oder anderen Sichten konstruiert sind und, da sie im Prinzip keine neuen Informationen enthalten, nicht explizit als Relation gespeichert werden (es sei denn zu Optimierungszwecken: kontrollierte Redundanz durch „Materialisierung“). Der Inhalt einer solchen virtuellen Relation wird also erst im Bedarfsfall berechnet; gespeichert wird nur die Berechnungsvorschrift. Die erwähnten Basisrelationen sind diejenigen Relationen, die tatsächlich als solche auf Platte gehalten werden.

Abzugrenzen ist der Begriff der Sicht von dem des **Snapshot**: Letzterer stellt einen Auszug aus anderen Relationen dar, der zu einem bestimmten Zeitpunkt mittels einer Anfrage aus diesen gewonnen und danach abgespeichert wurde. Ein Snapshot repräsentiert also lediglich den Zustand zu diesem einen Zeitpunkt, während eine Sicht immer einen Auszug aus dem *aktuellen* Zustand der Datenbank darstellt.

Ein Problem besteht bei Sichten hinsichtlich ihres Update-Verhaltens: Es ist immer noch nicht abschließend geklärt, unter welchen Voraussetzungen Sichten änderbar sind, so daß die Updates auf den Sichten auch intern in integre Updates auf den Basisrelationen übersetzt werden können. Überlegungen hierzu, die die Basis für View-Updates in IRIS bilden sollten, finden sich in [Win88].

3.5.2 Sichten in IRIS

Die Definition einer Sicht kann in IRIS an zwei Stellen erfolgen:

Erstens unter der Bezeichnung „Definitionskomponente externe Ebene“; dort ist auch ihre Änderung möglich, wohlgemerkt der Definition, nicht des Inhalts.

Die zweite Definitionsmöglichkeit besteht darin, daß sich nach jeder RELAX-Anfrage, die keine Zuweisung enthält, an die Ausgabe des Resultats die Frage anschließt, ob diese Anfrage als Sicht gespeichert werden soll oder nicht. Gibt man hier einen Namen an, so wird die vorhergegangene Anfrage als Definition der Sicht verwendet.

Sichten sind also immer durch RELAX-Ausdrücke definiert, weshalb sie auch in RELAX, NERO und IRIS-PASCAL genauso verwendet werden können wie Basisrelationen.

Die Speicherung einer Sicht erfolgt so, daß zunächst ihre Existenz und einige Zusatzinformationen im DD/D vermerkt wird. Ihre Definition selbst läßt sich leider nicht in einer DD/D-Relation unterbringen, da dafür Strings beliebiger Länge als Datentyp verfügbar sein müßten. Daher wird die Definition jeder einzelnen Sicht in einem eigenen File `*.VIE` abgelegt, wobei `*` für den Namen der Sicht steht.

Kapitel 4

Konzeptuelle Ebene

Die Aufgabe der Konzeptuellen Ebene eines DBMS besteht darin, die abstrakten und anwendungsunabhängigen Konzepte des zugrundeliegenden Datenbankmodells — hier also des Relationenmodells — zu implementieren. Dazu bedient sie sich der Dienste der Internen Ebene und stellt ihrerseits Leistungen für die Externe Ebene und für systeminterne Zwecke bereit.

Zur Konzeptuellen Ebene gehört neben den verschiedenen Möglichkeiten zur Definition des Datenbankschemas die Verwaltung der Metadaten im Data Dictionary. Die implementierten Konzepte umfassen die Relationenalgebra, der wir vollständigkeitshalber den Relationenkalkül gegenüberstellen, und die Abstrakten Datentypen (ADTs), durch die IRIS gewissermaßen erweiterbar wird.

4.1 Schemadefinition und -evolution

IRIS bietet grundsätzlich zwei verschiedene Möglichkeiten zur Definition und späteren Änderung von Relationenschemata. Die erste wird als *freie* Definition bzw. Änderung bezeichnet. Hierbei ist der Benutzer[1] allein für die Güte des Datenbankschemas verantwortlich. Die andere Möglichkeit nutzt eine als IRIS-SYNTHI bezeichnete Variante des Synthese-Verfahrens aus der Datenbankentwurfs-Theorie, um unter Ausnutzung semantischer Integritätsbedingungen ein Datenbankschema zu erzeugen, das bestimmte Normalisierungseigenschaften aufweist.

4.1.1 Freie Schemadefinition und -änderung

Wie schon gesagt, handelt es sich hierbei um ein Datenbankentwurfs-„Werkzeug“, das die eigentliche Arbeit und damit die Verantwortung komplett auf den Benutzer abwälzt. Für einfache Anwendungen reicht dies jedoch vollkommen aus, andererseits muß auch eine Möglichkeit bestehen, in bestimmten Fällen den Synthese-Algorithmus umgehen zu können.

Jedes Relationenschema wird bei dieser Komponente durch Eingabe des Namens der Relation sowie der Namen und Datentypen (**Domänen**) der **Attribute** definiert. Da IRIS die Menge

[1] Es handelt sich strenggenommen um eine Aufgabe des DBA.

aller Attribute, das sog. „Universum“, als global ansieht, müssen alle Attribute mit gleichem Namen auch vom gleichen Typ sein. Danach sind die Attribute auszuwählen, die den Primärschlüssel bilden. Für jeden weiteren Schüssel ist diese Prozedur zu wiederholen.

Nach den Schlüsseln werden die sogenannten **Sekundärschlüssel** eingegeben. Es handelt sich dabei *nicht* um Schlüssel im Sinne einer Integritätsbedingung, sondern um Attributmengen, für die besondere Zugriffsunterstützungen (siehe „Interne Ebene“) vorgesehen sind. Als letzter Schritt erfolgt dann noch die Festlegung, welche Art von **Zugriffsunterstützung** für diese Relation aufgebaut werden soll.

Schließlich wird die Relation ins DD/D eingetragen, und im `DATAF` wird eine erste Seite für sie reserviert.

Unter dem Menüpunkt „Datenbankdefinition ändern frei“ bestehen nach Angabe der zu bearbeitenden Relation die folgenden Auswahlmöglichkeiten:

1. **Hinzufügen eines Attributs**: Bei `STRING`-Domänen werden als Werte Punkte („.“) entsprechend der Länge des Strings eingetragen, bei numerischen Domänen Nullen.
2. **Löschen eines Attributs**: Zu beachten ist, daß Schlüsselattribute nicht gelöscht werden dürfen. Daher wird keine Eliminierung von Duplikaten erforderlich.
3. **Ändern der Domäne eines Attributs**: Hierbei darf nur eine „größere“ Domäne gewählt werden. Die Änderung der Domäne wird automatisch auch für die anderen Relationen, die dieses Attribut enthalten, durchgeführt.
4. **Ersetzen eines Schlüssels**: Zunächst wird der zu ersetzende Schlüssel gelöscht (s.u.) und danach der neue hinzugefügt (s.u.). Technisch werden also nur Zugriffspfade abgebaut und wieder neu aufgebaut.
5. **Einführung weiterer Schlüssel**: Es können nur Schlüssel neu eingefügt werden, die die Tupel der Relation eindeutig bestimmen und nicht in einem bereits existierenden Schlüssel enthalten sind.
6. **Löschen eines Schlüssels**: Das Löschen eines Schlüssels ist nur möglich, falls mehr als ein Schlüssel für die aktuelle Relation existiert. Somit entfällt auch hier die Duplikat Eliminierung.
7. **Löschen des Schemas**, d.h. einer Relation mit allen ihren Zugriffspfaden.
8. **Liste aller möglichen Relationennamen ausgeben**.

Genaueres über die Änderung bestehender Datenbankschemata „on line“ ist in [kg88] nachzulesen.

Mit dieser Komponente erzeugte Relationen sind im DD/D mit der Bemerkung „manuell“ gekennzeichnet. Sie können nicht in URQUEL-Anfragen verwendet werden.

Alle Änderungen werden sofort umgesetzt und ins DD/D eingetragen, und zwar sowohl in das „verzeigerte“ Hauptspeicher-Dictionary (siehe 4.2.2) als auch in die DD/D-Seiten des `DATAF`(siehe 4.2.1).

4.1.2 Algorithmische Schemadefinition und -änderung

Diese Komponente umfaßt zum einen das Synthese-Verfahren IRIS-SYNTHI, das für den Datenbankentwurf mit funktionalen Abhängigkeiten zuständig ist, und zum anderen ein Pendant dazu, das mit mehrwertigen Abhängigkeiten arbeitet und via Dekomposition ein Schema in Vierter Normalform (4NF) liefert (s.u.).

Das Gebiet der Datenbankentwurfs-Theorie ist eines der am intensivsten untersuchten Teilgebiete der Theorie relationaler Datenbanken. Durch die mathematische Fundierung der Relationen bot es sich dazu auch regelrecht an. Im Laufe der Zeit wurden verschiedene sogenannte „Normalformen“ ersonnen, die alle dem Zweck dienen, eine Modellierung der Daten zu ermöglichen, die erstens weitestgehend redundanzfrei und damit speicherplatzsparend und frei von Update-Anomalien ist, zweitens trotz der Aufteilung der Informationen auf mehrere Relationen exakt denselben Informationsgehalt garantiert und drittens sämtliche semantischen Integritätsbedingungen erhält.

Große Bedeutung hat vor allem die sog. „Dritte Normalform“ (3NF) erlangt, daneben die Ling-Tompa-Kameda-Normalform (LTK-NF). Relationenschemata, die diesen Normalformen und einigen weiteren Eigenschaften genügen, können grundsätzlich mit zwei verschiedenen Methoden erzeugt werden: Durch *Dekomposition* großer, redundanter Schemata in mehrere kleine und durch *Synthese* einzelner Attribute zu passenden Relationenschemata.

Wie der Name schon andeutet, ist der IRIS-SYNTHI-Algorithmus ein Vertreter der zweiten Gruppe. Die Überlegungen, die zu seiner Entwicklung geführt haben, sind in [Heu86] veröffentlicht. Hier soll noch einmal kurz zusammengefaßt werden, welche Eigenschaften der Synthese-Algorithmus für die von ihm erzeugten Datenbankschemata garantiert:

- vollständige Charakterisierung der vorgegebenen FD-Menge durch das erzeugte Datenbankschema (Abhängigkeitstreue)
- Dritte Normalform: Ein Relationenschema ist in Dritter Normalform, wenn jedes Attribut atomar ist (erste Normalform), und jedes Attribut entweder Teil eines Schlüssels (Primattribut) ist oder nicht-transitiv von einem Schlüssel abhängt. Ein Datenbankschema ist in Dritter Normalform, wenn alle seine Relationenschemata es sind.
- Minimalität des Datenbankschemas im Sinne einer minimalen Anzahl von Relationen unter Beibehaltung der übrigen Qualitäten
- Projektions-Verbund-Treue: Eine Zerlegung der Attributmenge einer Relation in Teilmengen, deren Vereinigung wieder die Attributmenge ergibt, heißt projektions-verbund-treu, falls der Verbund der durch Projektion der Ursprungsrelation auf die einzelnen Teilmengen entstandenen Relationen wieder die ursprüngliche Relation ergibt.

Datenbankschemata, die diese Eigenschaften haben, werden als vollständig und verbundtreu bezeichnet. Darüber hinaus liefert SYNTHI sogar noch eine Verschärfung der Dritten Normalform, nämlich die o.g. Ling-Tompa-Kameda-Normalform. Ein Datenbankschema ist in LTK-NF, wenn es in 3NF ist und kein Relationenschema überflüssige Attribute enthält, die ohne Verlust funktionaler Abhängigkeiten aus dem Schema entfernt werden könnten.)

Eine detaillierte Dokumentation der SYNTHI-Implementierung enthält [Cza87]. Eine grobe Beschreibung der Implementierung findet sich auch in [SH85], wo allerdings der Schwerpunkt eher auf theoretischen als auf technischen Aspekten liegt.

Des weiteren ist es in IRIS möglich, aus Synthese-Ergebnissen mit Hilfe des Dekompositionsalgorithmus von Lien [Lie81] neue Datenbankschemata zu berechnen, die der Vierten Normalform genügen: Ein Relationenschema ist in Vierter Normalform, wenn alle zwischen seinen Attributen bestehenden nichttrivalen mehrwertigen Abhängigkeiten $X \rightarrow\rightarrow Y$ so beschaffen sind, daß die Attributmenge X einen Schlüssel umfaßt [Ull88]. Relationenschemata, die obige Bedingung nicht erfüllen, sind geeignet aufzuteilen.

4.2 Data Dictionary und Datenbank-Files

Das Data Dictionary / Directory (DD/D) dient zur Verwaltung der Strukturinformationen oder Metadaten eines relationalen Systems in relationaler Form. Es bildet also selbst eine Menge von Relationen, in denen die Informationen über alle vorhandenen Relationen gespeichert werden. Insbesondere enthält das DD/D seine eigene Beschreibung.

Bei IRIS umfaßt das DD/D neben der Beschreibung von Attributen, Relationen, Zugehörigkeiten von Attributen zu Relationen, Schlüsseln, Zugriffspfaden usw. auch die Statistik-Relationen der Definitionskomponente der Internen Ebene (siehe Abschnitt 5.3). Im Anhang findet sich eine ausführliche Beschreibung der DD/D-Relationen und ihrer Bedeutung.

4.2.1 `DATAF` und externes `MINDIC`

Da das DD/D die Strukturdaten der gesamten Datenbank enthält, sind für IRIS Zugriffe auf das Daten-File `DATAF` nur nach vorheriger „Konsultation“ des DD/D möglich. Das gilt von dem Moment an, in dem IRIS gestartet wird. Da nun aber das DD/D seine eigene Beschreibung enthält und wie alle Relationen im `DATAF` abgelegt wird, ist eigentlich die Kenntnis des DD/D nötig, um sich aus dem `DATAF` Informationen über das DD/D zu holen.

Deswegen war es nicht zu umgehen, einige Grundbestandteile des DD/D im IRIS-Programm fest zu „verdrahten“. Dies betrifft etwa die Struktur der Relation `Relation`. An dieser Stelle war es also unvermeidbar, die angestrebte Speicherung aller Informationen in und über die Datenbank in relationaler Form innerhalb der Datenbank zu durchbrechen.

Neben dem `DATAF` gibt es ein weiteres File namens `MINDIC` (im folgenden als „**externes** `MINDIC`“ bezeichnet), das genau wie das `DATAF` aufgebaut ist, aber nur ein leeres DD/D enthält, also ein DD/D, das nur sich selbst, aber keine Benutzerrelationen beschreibt. Das `MINDIC` dient allein dem Zweck, beim Anlegen einer neuen Datenbank ein initiales DD/D zur Verfügung zu haben.

Ein Update des DD/D ist immer dann erforderlich, wenn in einer der Definitionkomponenten (DKE, DIE) Änderungen vorgenommen werden. Alle Informationen, die in diesen Komponenten behandelt werden, stammen nämlich aus dem DD/D. Änderungen, die vom Benutzer initiiert werden, können allerdings nur die Struktur der Benutzerrelationen betreffen, nicht die des DD/D selbst.

Änderungen am DD/D-Inhalt werden – wie alle Updates – sofort auf die Platte bzw. in den Puffer geschrieben, so daß das DD/D im `DATAF` immer konsistent ist. Gleichzeitig werden die Änderungen auch im Hauptspeicher-DD/D eingetragen, das im nächsten Abschnitt eingeführt wird.

4.2.2 Hauptspeicher-DD/D und internes `MINDIC`

Beim Starten von IRIS wird zur Beschleunigung aller folgenden Operationen eine Kopie des DD/D in Form einer verzeigerten Datenstruktur im Hauptspeicher (Long Heap) angelegt. Im Prinzip wäre das nicht erforderlich, da alle benötigten Informationen ja durch Anfragen an das DD/D im `DATAF` ermittelt werden könnten, wozu die „fest verdrahteten" Kenntnisse der elementarsten DD/D-Relationen reichen. Wenn aber für jeden Zugriff auf eine Relation, der ja zwangsläufig erst einmal einen Zugriff auf das DD/D erfordert, erst einmal das DD/D von der Platte geholt werden müßte, wäre das zu ineffizient. Daher lohnt sich der zusätzliche Aufwand für die Unterhaltung der DD/D-Kopie durchaus, da man, bedingt durch die wesentlich höhere Geschwindigkeit des Speichermediums, mit einem erfreulicheren Zeitverhalten belohnt wird.

Relationenalgebraische Anfragen erzeugen in jedem Auswertungsschritt Ergebnisse in Form temporärer Relationen. Außerdem führen Umbenennungen und Spaltenerweiterungen zur Erzeugung temporärer Attribute. Da die Lebenszeit dieser „Metaobjekte" nur sehr kurz ist und ihre Existenz keinen Einfluß auf die Konsistenz des `DATAF` für den Fall hätte, daß ausgerechnet in dieser Zeit ein Systemfehler auftritt, wird darauf verzichtet, sie im DD/D einzutragen. Der Zeitaufwand hierfür würde in keinem vernünftigen Verhältnis zum Nutzen stehen, zumal nach Abschluß der Anfrage oder der Update-Operation die gerade eingetragenen „Neuigkeiten" ohnehin wieder zu löschen wären. Stattdessen werden diese Informationen *nur* im Hauptspeicher gehalten. (Der *Inhalt* der temporären Relationen steht natürlich im `DATAF`.)

Zur weiteren Beschleunigung wird im Hauptspeicher das sog. **interne** `MINDIC` unterhalten. Es handelt sich dabei um ein Array fester Größe, in dem die minimal erforderlichen Informationen für die Behandlung temporärer Relationen gespeichert werden: Relationen-Identifier sowie Anzahl und Domänen der Attribute.

4.3 Relationenalgebra und Relationenkalkül

Die **Relationenalgebra** ist eine Menge von unären und binären Operationen, die aus Relationen wieder Relationen erzeugen, was die mathematische Klassifizierung als Algebra rechtfertigt. Dazu gehören neben den aus der Mengentheorie bekannten Operationen Vereinigung, Durchschnitt und Differenz weitere, speziell auf Relationen zugeschnittene Operationen, die als Projektion, Selektion und natürlicher Verbund (Join) bezeichnet werden, sowie die Umbenennung. (Je nach Autor wird statt des Join-Operators auch das kartesische Produkt verwendet.)

Der **Relationenkalkül** stellt eine Einschränkung des Prädikatenkalküls erster Ordnung dar. Es handelt sich um eine formelmäßig aufgebaute, rein deskriptive Beschreibung von Anfragen, indem eine Tupelmenge durch Angabe ihrer Eigenschaften spezifiziert wird. Man gibt,

vereinfacht ausgedrückt, Bedingungen für die Herkunft von (Teil-) Tupeln und für Attributwerte an und verknüpft diese Bedingungen mit logischen Operatoren. Je nach Art der verwendeten Variablen unterscheidet man noch zwischen Tupel- und Domänen-Relationenkalkül. [Mai83, Ull88]

Von der Relationenalgebra zeigte Codd, daß sie zum zuerst entwickelten **Relationenkalkül** äquivalent in dem Sinne ist, daß alle Anfragen, die im Relationenkalkül — genauer: in einem eingeschränkten, „sicheren" Kalkül — gestellt werden können, auch mit Hilfe der Relationenalgebra formuliert werden können und umgekehrt. Diese Eigenschaft wird als **relationale Vollständigkeit** bezeichnet und ist eine Art Untergrenze für die Mächtigkeit jeder vernünftigen Anfragesprache.

Dem Relationenkalkül und der Relationenalgebra gemeinsam ist, daß beide keine Möglichkeit vorsehen, Operationen auf den verwendeten Domänen („**Domänenalgebra**") in Anfragen einzubauen. Solche Fähigkeiten braucht man beispielsweise, um die Artikel in einem Warenlager aus der Datenbank zu ermitteln, bei denen der aktuelle Bestand einen vorgegebenen Mindestbestand um höchstens 10% überschreitet, die also demnächst nachzubestellen wären.

Anfragesprachen wie SQL (SQL/DS, DB2, ...) oder QUEL (INGRES) bieten dem Benutzer solche Fähigkeiten an, die in der Praxis auch enorme Bedeutung haben. Daher war *ein* Ziel bei der Entwicklung einer universell verwendbaren Anfragesprache RELAX für IRIS, hier mithalten zu können. Gleichzeitig sollten jedoch die konzeptionellen Schwächen von SQL vermieden werden, so daß diese Sprache als Leitbild nicht in Betracht kam. So ist bei der Formulierung von Anfragen in SQL schnell festzustellen, daß dieser Sprache die Orthogonalität fehlt, worunter man – informal ausgedrückt – die Möglichkeit versteht, alle angebotenen Konstrukte in beliebiger Reihenfolge ineinander zu verschachteln oder nebeneinander anzuwenden, zumindest soweit sinnvoll. Bei SQL gibt es etwa die Einschränkung, den Vereinigungs-Operator `UNION` nur auf der äußersten Stufe einer Anfrage verwenden zu dürfen. Ausführlich werden die Schwächen von SQL in [Dat89] offengelegt. Die neue Fassung der SQL-Norm, SQL2 genannt, enthält diesbezügliche Verbesserungen, wird aber in kommerziellen Produkten noch nicht vollständig unterstützt.

Die Operationen der Relationenalgebra sind so implementiert, daß zunächst versucht wird, aus dem Anfragetext einen Operatorbaum zu konstruieren, wobei eventuelle syntaktische oder semantische Fehler aufgedeckt werden. War die Anfrage korrekt, wird der Operatorbaum anschließend nach algebraischen Gesetzen (Kommutativität, Assoziativität, ...) optimiert mit dem Ziel, die Anzahl der Operationen und der „anzufassenden" Tupel zu minimieren, also etwa Selektionen so früh wie möglich und Joins so spät wie möglich durchzuführen und die Zahl der Joins zu minimieren. Zu dieser „**algebraischen Optimierung**" kommt noch die „**interne Optimierung**", die die Ausnutzung von Zugriffspfaden bei Relationen betrifft.

Die Gesetze, nach denen die algebraische Optimierung erfolgen soll, sind in IRIS als „Regeln" implementiert, die sich jeweils auf eine einzelne oder zwei aufeinanderfolgende Operationen beziehen. Diese Regeln, die Umordnungen der Knoten des Anfragebaumes darstellen, können mit Hilfe eines Regeleditors (`REGEL_ED.EXE` [lr92]) als PASCAL-Quelltext eingegeben, gespeichert und jederzeit geändert werden. Die Anwendbarkeit jeder Regel kann durch Angabe von Bedingungen eingeschränkt werden. Zusätzlich ist zu jeder Regel ihre Priorität anzugeben, wodurch eine Reihenfolge der Regelanwendungen bei der Optimierung definiert wird. Die gespeicherten Regeln können aktiviert und deaktiviert werden, d.h. der Administrator bzw. – hier – Implementierer kann entscheiden, ob eine Regel für die Optimierung benutzt wird

oder nicht.

Nach der algebraischen Optimierung wird der Operatorbaum in pre-order-Reihenfolge abgearbeitet. Beginnend bei den Blättern des Baumes (sie repräsentieren die Basisrelationen) wird jede relationenalgebraische Operation einzeln ausgewertet und erzeugt eine temporäre Zwischenrelation. Hierbei werden existierende Sortierungen der jeweiligen Ausgangsrelationen ausgenutzt; ist die Ausgangsrelation nicht (passend) sortiert, so wird zunächst der für die Operation nötige Sortierlauf durchgeführt. Die Sortierungen der Ausgangsrelationen werden soweit wie möglich für die Ergebnisrelation beibehalten. Ebenso werden Schlüssel der Ausgangsrelationen auf die Ergebnisrelationen übertragen, sofern sie ihre Gültigkeit behalten. Zur internen Optimierung trägt neben der Ausnutzung von Sortierungen und Zugriffspfaden auch die Ausnutzung der Selektivitäten der einzelnen Attribute bei der Auswertung von Selektionen bei. Umbenennungen und Spaltenerweiterungen erzeugen außerdem temporäre Attribute.

Eine gruppierte Auswertung mehrerer Operationen wäre zwar prinzipiell möglich, ist aber nicht implementiert.

4.4 Abstrakte Datentypen

Unter einem Abstrakten Datentyp (Abstract Data Type, ADT) versteht man eine Menge zulässiger Werte, also eine Domäne, mit einer Menge darauf zulässiger Operationen.

Ein klassisches **Beispiel** dafür ist der Typ `Datum` (kalendarisches Datum). Zwei Daten zu multiplizieren, erscheint wenig sinnvoll. Addieren von Daten ist zumindest erklärungsbedürftig: Für ein `Datum` x und eine `Integer`-Konstante n wäre „$x+n$“ eine denkbare Schreibweise für das `Datum` n Tage nach dem Tag x (analog für $x - n$). Die Differenz zweier Daten zu bilden, scheint ebenfalls vernünftig, genauso wie die Ermittlung des auf x folgenden Tages usw. Die Menge aller zulässigen Daten ist außerdem eingeschränkt: Es gibt weder einen 32.13.2005 noch einen 29.02.1989. Die Menge der erlaubten Werte bildet also zusammen mit den Operationen $naechster\_tag(d_1)$, $dauer(d_1, d_2)$ sowie den Operatoren $d_1 + n$ und $d_1 - n$ für erlaubte Werte d_1, d_2 den ADT `Datum`.

In IRIS wurden bisher exemplarisch die ADT's `Datum` und `Zeit` implementiert [gim89].

Einem Attribut wird bei der Definition einer Relation in der DKE seine Domäne zugeordnet; die Domänen „Datum“ und „Zeit“ sind dort gleichberechtigt neben den numerischen Standardtypen und Zeichenketten aufgeführt. In RELAX-Anfragen — und daher auch in LDML und IRIS-Pascal — können beide Typen in Selektionsbedingungen auftreten, wobei außer einer korrekten Schreibweise von Konstanten und der Inkompatibilität mit allen anderen Typen hinsichtlich Vergleichen keinerlei nennenswerte Restriktionen zu beachten sind. Die üblichen Vergleichsoperatoren sind voll anwendbar, da beide Typen eine natürliche (oder kanonische) Ordnung besitzen.

Interessant ist auch die Verwendung von `Datum` und `Zeit` in Spaltenerweiterungen, da auch hier auf die typspezifischen Operationen zugegriffen werden kann:

- Addition und Subtraktion: Erlaubt sind Addition und Subtraktion eines `Datums` und

einer `Integer`-Konstanten einerseits, wodurch man ein neues `Datum` erhält, und andererseits zweier `Zeit`-Werte zu einem neuen `Zeit`-Wert.

- `MIN, MAX`: Diese Aggregat-Funktionen sind auch auf Attributen vom Typ `Datum` oder `Zeit` definiert und liefern ein Attribut desselben Typs.
- `NAECHSTER_TAG(`x`)` für ein `Datum` x, liefert als Ergebnis wieder ein `Datum`.
- `DAUER(`x, y`)`: Hiermit wird die zeitliche Differenz zwischen zwei Daten oder Zeitpunkten ermittelt. Im Falle zweier Daten wird ein `Integer`-Wert zurückgegeben, der die Anzahl der Tage zwischen den Daten repräsentiert. Im Falle zweier Zeiten wird die Differenz ebenfalls als Wert vom Typ `Zeit` ausgedrückt.
- `KOMP1(`x`), KOMP2(`x`), KOMP3(`x`)`: Da beide ADTs logisch als Tripel elementarer Werte dargestellt werden, kann man hiermit auf ihre Komponenten zugreifen. Sie werden, der internen Repräsentation entsprechend, als `Integer`-Zahlen geliefert.

`Datum` und `Zeit` werden intern in ganze Zahlen (`integer4`) umgerechnet und gespeichert, wobei die kanonische Ordnung der Typen beibehalten wird, so daß

$$\texttt{date\_to\_int}(x) < \texttt{date\_to\_int}(y)$$

genau dann gilt, wenn x kalendarisch vor y steht (analog für Zeiten).

Kapitel 5

Interne Ebene

Wie eingangs beschrieben, dient die interne Ebene dazu, die zur Realisierung von Anfragen oder Updates notwendigen Zugriffe auf die physikalischen Daten durchzuführen. Es sind also Anforderungen wie etwa „Schreibe folgendes Tupel in die Relation A“ oder „Hole das nächste Tupel aus Relation B“ umzusetzen in die entsprechenden Aktionen auf dem Datenspeicher, im Normalfall also auf der Festplatte.

Die interne Ebene ist somit dafür zuständig, die Daten in geeigneter Form in Plattendateien unterzubringen, wobei „geeignet“ bedeutet, daß der Zugriff auf die Daten möglichst schnell erfolgen soll. Da die Festplatte im Vergleich zum Hauptspeicher eine ganze Größenordnung langsamer arbeitet und Plattenzugriffe immer ganze Blöcke von Bytes, genannt „Seiten“, betreffen, sind also die Daten geschickt so auf die Seiten zu verteilen, daß alle Aktionen mit möglichst wenigen Plattenzugriffen auskommen. Die Hilfsmittel dazu sind einerseits eine gut durchdachte Speicherorganisation, andererseits spezielle „Zugriffsunterstützungen“ oder „Zugriffspfade“.

5.1 IRIS-Speicherorganisation

5.1.1 Plattenspeicher und Puffer

Die Organisation des Plattenspeichers inclusive der Pufferverwaltung ist bei IRIS Aufgabe der UFO-Komponente innerhalb der Transformationskomponente der internen Ebene (INTRA). UFO bedeutet „Universelle File-Organisation“. Für den IRIS-Benutzer ist UFO „transparent“, er wird mit dieser Komponente nur insofern konfrontiert, als ihre Dienstleistungen die Arbeit der übrigen Komponenten erst ermöglichen. Einen groben Überblick über die Funktionen von UFO gibt [Heu87].

Das `DATAF`

Das `DATAF` („Data File“) ist die Datei auf der Festplatte, in der IRIS alle vom System in relationaler Form verwalteten Daten speichert. Das sind sowohl die Daten in den benutzerdefinierten Relationen als auch die „Metadaten“ des DD/D. Einige wenige Systemdaten, die der Benutzer nie zu sehen bekommt, werden in der Datei `FREI` gespeichert (s.u.!).

Grundlage für die Speicherung von Daten auf der Festplatte ist natürlich die Handhabung derselben durch das Betriebssystem: DOS teilt die Festplatte in „Seiten“ zu jeweils 512 Bytes auf. Zugriffe erfolgen immer seitenweise.

Das `DATAF` ist so organisiert, daß jede nichtleere Seite nur Tupel derselben Relation enthält. Die Seiten einer Relation sind untereinander durch Zeiger verbunden. Genauer: Neber der von DOS verwalteten Verkettung *aller* Seiten des `DATAF` zu einer Datei nimmt IRIS eine „innere“ Verkettung dieser Seiten in mehrere, disjunkte Ketten vor, die jeweils eine Relation repräsentieren. Somit ist die Konsistenz des `DATAF` unabhängig von der physikalischen Verteilung seiner Seiten auf der Platte.

Die Nummer der ersten Seite jeder Relation ist im DD/D als Relationen-Identifier eingetragen. Die Identifier der DD/D-Relationen, die ja immer vorhanden sind, sind im Programm „eingebrannt“. Diese Maßnahme war erforderlich, um überhaupt das `DATAF` (bzw. vorher das externe `MINDIC` als „leeres“ `DATAF`) lesen zu können.

Beim Löschen temporärer Relationen oder einzelner Tupel bzw. bei der Reorganisation von Indexseiten entstehende leere Seite werden nicht aus dem `DATAF` entfernt und an DOS zurückgegeben, sondern einbehalten und wiederverwendet, ehe man neue Seiten anfordert. Man muß sich also merken, welche Seiten des `DATAF` leer sind und bei Bedarf neu vergeben werden können. Die leeren Seiten sind – wie die Seiten einer Relation mit Identifier 0 – untereinander verkettet, so daß es wiederum reicht, sich die *erste* freie Seite zu merken. Diesem Zweck dient die Datei `FREI`, und ursprünglich diente sie auch *nur* dazu – daher der Name. Inzwischen werden in ihr auch noch andere systeminterne Informationen abgelegt. Aus der Behandlung freiwerdender Seiten ergibt sich, daß das `DATAF` zur Laufzeit von IRIS nur wachsen kann, aber nie kleiner wird.

Das Pufferkonzept

Aufgrund der großen Geschwindigkeitsdifferenz zwischen Arbeitsspeicher und Festplatte und aufgrund der Beobachtung, daß aufeinanderfolgende Plattenzugriffe häufig dieselbe Seite betreffen, war es naheliegend, IRIS durch Einführung eines Puffers für Plattenseiten zu beschleunigen. (Heute würde man von einem Platten-Cache sprechen.)

Dies funktioniert so, daß zunächst ein Bereich im Hauptspeicher reserviert wird, der eine festgelegte Anzahl von Plattenseiten aufnehmen kann. (vgl. „Paging“ bei Betriebssystemen!) Wie groß diese Anzahl ist, wird vor dem Start im Konfigurationsfile `IRIS.SYS` eingetragen; maximal sind 248 Seiten möglich.

Soll auf eine Plattenseite zugegriffen werden, so wird zuerst geprüft, ob sie schon im Puffer vorhanden ist. Wenn ja, kann sofort der gewünschte Schreib- oder Lesezugriff auf dem Puffer erfolgen. Ist sie nicht vorhanden, wird zunächst die komplette Seite von der Platte in den Puffer geladen. Dabei ist zu klären, ob noch ein freier Pufferplatz existiert, oder ob erst eine andere Seite aus dem Puffer entfernt werden muß. Im ersten Fall kann man sofort die benötigte Seite an den freien Platz laden; im zweiten Fall ist auch noch festzustellen, ob der Inhalt der „alten“ Seite verändert wurde oder nicht. Falls ja, muß sie zuerst zurückgespeichert werden, ansonsten reicht einfaches Überschreiben. Die Auswahl der aus dem Puffer zu entfernenden Seite erfolgt prioritätsgesteuert, um gegenüber einer reinen LRU-Strategie[1] ein

[1] LRU ... least recently used. Diese Strategie entfernt die am längsten nicht benutzte Seite.

günstigeres Verhalten zu erreichen.

Eine Seite kann im Puffer „verankert“ werden, um sie gegen Austausch zu schützen, wenn feststeht, daß sie in allernächster Zeit mehrfach gebraucht wird. Dies beschleunigt dann den Zugriff noch mehr, da der ganze Verwaltungsaufwand entfällt und die Seite direkt über die Nummer des von ihr belegten Pufferplatzes angesprochen werden kann.

Insgesamt bewirkt die Pufferung eine spürbare Beschleunigung, da deutlich weniger „echte“ Plattenzugriffe erforderlich werden. Der nicht eben geringe Verwaltungsaufwand fällt nicht so sehr ins Gewicht, da er im wesentlichen Aktionen im Hauptspeicher erfordert.

EMS und Puffer

Wie alle Dinge im Leben hat auch der IRIS-Puffer zwei Seiten. Einerseits wird IRIS durch ihn zwar schneller, aber andererseits kostet er Speicherplatz. Soll der Puffer nun aus Geschwindigkeitsgründen größer werden, kann es im direkt adressierbaren Arbeitsspeicher, der unter PC/MS-DOS ja auf 640 kB begrenzt ist, eng werden.

IRIS testet daher auf Rechnern, die mit mehr als 1 MB Arbeitsspeicher ausgerüstet sind, ob EMS verfügbar ist, und nutzt ihn ggf., indem der Puffer zunächst im bis zu 64 kB großen EMS-Fenster (zwischen 640 kB und 1 MB gelegen) angelegt wird. Erst wenn dieser Platz nicht mehr ausreicht (oder EMS nicht verfügbar ist), legt IRIS den Puffer unterhalb der 640 kB-Grenze an.

5.1.2 Organisation des Hauptspeichers

Overlay-Technik

Da im Laufe der Entwicklung die `IRIS.EXE`-Datei zu groß für den unter DOS auf 640 kB beschränkten Arbeitsspeicher wurde, mußten die Overlay-Fähigkeiten des Microsoft-Linkers genutzt werden. Dies bedeutet, daß Code-Module, die voneinander unabhängig sind und somit nicht gleichzeitig im Speicher vorliegen müssen, denselben Speicherbereich belegen dürfen. Es wird dann jeweils nur ein Modul von der Festplatte in den Arbeitsspeicher geladen und bei Bedarf gegen einen anderen Modul ausgetauscht.

Der IRIS-Treiber

Auch die Overlay-Technik reichte schließlich nicht mehr aus, um IRIS weiter wachsen zu lassen. IRIS-PASCAL erforderte ein völlig neues Konzept. Da IRIS-PASCAL und die interaktiven IRIS-Komponenten nicht gleichzeitig betrieben werden konnten, aber andererseits zum Teil dieselben Dateien und Variablen benutzen, war eine Art „Hyper-Overlay“ zu bewerkstelligen, so daß sich entweder IRIS oder IRIS-PASCAL im Speicher befinden.

Die Lösung besteht in `IPEXEC.EXE`, dem „IRIS-Treiber“. Dieses Programm verankert sich zuerst selbst im Hauptspeicher, lädt danach `IRIS.EXE` und startet dieses. Wird in IRIS dann die „Anwendungsdefinition“ aufgerufen, wird IRIS beendet und stattdessen IRIS-PASCAL geladen und gestartet. Will der Benutzer IRIS-PASCAL wieder verlassen, wird vom Treiber

wieder IRIS geladen und gestartet, usw. Die Besonderheit an diesem Verfahren ist, daß zwischendurch niemals die Kontrolle an DOS zurückgegeben wird, wie es etwa bei einer Realisierung als Batch-File geschehen würde.

Später wurde `IPEXEC` insofern abgeändert, als auch IRIS-SYNTHI und Teile der Update-Schnittstelle als selbständige Einheiten „ausgeklinkt“ wurden und nur noch über den Treiber aktiviert werden können.

Auch dieses Konzept bleibt für den Benutzer weitgehend transparent. Er merkt nur insofern etwas davon, als er IRIS entweder direkt als `IRIS.EXE` oder indirekt über `IPEXEC.EXE` aufrufen kann und je nachdem Zugriff auf bestimmte Komponenten hat oder nicht.

5.2 Zugriffspfade

5.2.1 Sinn und Zweck von Zugriffspfaden

Unter Zugriffspfaden oder auch Zugriffsunterstützungen versteht man im Datenbank-Bereich Mechanismen, die die in den Daten codierte Information selbst nutzen, um den Zugriff auf sie zu beschleunigen. Konkret sollen Zugriffspfade Selektionen und Verbunde beschleunigen, bei denen es ja darum geht, aus den Tupeln einer Relation diejenigen herauszusuchen, deren Attributwerte bestimmten Bedingungen (Gleichungen und Ungleichungen in (fast) beliebiger Kombination) genügen.

Man unterscheidet dabei zwischen:

- „**exact match queries**“, bei denen alle Attributwerte oder zumindest die eines Schlüssels vorgegeben sind und also höchstens ein passendes Tupel existieren kann,
- „**partial match queries**“, bei denen einige Attributwerte vorgegeben sind, aber noch kein Schlüssel, so daß prinzipiell mehrere Tupel zu suchen sind,
- sowie „**range queries**“, bei denen nur Bereiche gegeben sind, in denen bestimmte Attributwerte zu liegen haben.

Für die verschiedenen Anfragetypen eignen sich die verschiedenen Zugriffspfade in unterschiedlichem Maße, so daß die Wahl der Zugriffspfade einer Relation sich danach richten sollte, welche Arten von Anfragen besonders häufig vorkommen und daher besonders zu unterstützen sind. Um die dafür benötigten Informationen erheben zu können, wurde in IRIS eine Statistik-Komponente eingebaut, die wir weiter unten vorstellen werden.

Allgemeine Informationen über Zugriffspfade enthält [Fuc84], das Konzept für IRIS wird in [Fuc85] vorgestellt.

5.2.2 Anlegen und Ändern von Zugriffspfaden

Die Wahl der Zugriffspfade einer Relation ist Aufgabe des DBA, die zuständige IRIS-Komponente ist die Definitionskomponente der internen Ebene (DIE). Zugriffspfade können

einerseits bei der Definition einer neuen Relation festgelegt werden, andererseits besteht *jederzeit* die Möglichkeit, Zugriffspfade zu existierenden Relationen on-line anzulegen, zu ändern oder zu entfernen.

5.2.3 Primitive Speicherung

Die einfachste denkbare Form, die Tupel einer Relation zu speichern, dürfte wohl darin bestehen, sie in der gleichen Reihenfolge zu speichern, in der sie vom Benutzer eingegeben wurden. Dieses Vorgehen ist zwar einfach zu implementieren, aber alles andere als schnell. Wird nämlich ein bestimmtes Tupel gesucht, so müssen nacheinander im Schnitt die Hälfte aller Tupel überprüft werden. Noch schlimmer ist das Einfügeverhalten: Um festzustellen, ob das einzutragende Tupel wirklich noch nicht vorhanden ist, müssen alle vorhandenen Tupel gelesen werden.

Etwas besser ist dann schon eine Sortierung, die wenigstens binäres Suchen erlaubt, was die Komplexität des Suchverfahrens schon auf $O(\log n)$ senken kann, wenn n die Zahl der Tupel bezeichnet. Nachteil: Eine Relation kann immer nur nach einem Kriterium sortiert werden. Wird nach anderen als den zur Sortierung verwendeten Attributen oder auch nur nach einem Teil davon gesucht, ist der Nutzeffekt der Sortierung bereits dahin. Außerdem ist binäres Suchen ein Verfahren, das eher für das Suchen im Hauptspeicher geeignet ist und weniger für externe Speichermedien wie die Festplatte. Trotzdem wird auch in IRIS davon Gebrauch gemacht, allerdings in abgewandelter Form.

Wie die Sortierung von Relationen in IRIS durchgeführt wird, ist in [hw86] erläutert. Es handelt sich um ein zweistufiges Verfahren, das – in der Nomenklatur von [Wir83] – als „ausgeglichenes n-Weg-Mischen“ bezeichnet werden kann [Heu87].

5.2.4 Indexsequentielle Organisation

Vereinfacht ausgedrückt, werden bei dieser Art der Zugriffsunterstützung die Tupel einer Relation nach den Attributen eines Schlüssels sortiert gespeichert (s.o.). Außerdem wird ein Inhaltsverzeichnis (Index) angelegt, aus dem IRIS entnehmen kann, wo welches Tupel steht, ohne alle Tupel einzeln ansehen zu müssen.

Ein Konzept, dem die indexsequentielle Organisation ähnelt, ist der B-Baum. (vgl. z.B. [Wir83]) Wie bei diesem entspricht das Suchen bei indexsequentieller Organisation dem Traversieren eines Baumes, wobei an jedem Knoten mehrere Verzweigungsmöglichkeiten bestehen. Je nachdem, zwischen welchen der im Knoten enthaltenen „Eckwerte“ der gesuchte Wert liegt, wird auf dem zugehörigen Ast weitergesucht, bis ein Blatt erreicht wurde. Die Blätter verweisen dann direkt auf die Seiten des `DATAF`, die den eingekreisten Teil der durchsuchten Relation enthalten.

Um aber sachlich genau zu sein: Bei den in IRIS verwendeten Bäumen handelt es sich insofern *nicht* um B-Bäume, als auf einige charakteristische Eigenschaften dieser Datenstruktur verzichtet wurde: Erstens ist die Höhe eines „echten“ B-Baumes dynamisch, d.h. der Baum wächst und schrumpft zur Laufzeit, indem durch Teilen oder Zusammenlegen von Knoten unter bestimmten Bedingungen neue „Etagen“ (Levels) entstehen oder wieder verschwinden. Dadurch erreicht man dann zweitens, daß alle Knoten bzw. Seiten des Baumes, evtl.

mit Ausnahme der Wurzel, immer mindestens zur Hälfte belegt sind, und drittens ist der B-Baum immer ausgeglichen (balanced). Alle diese Dinge gelten nicht für die indexsequentielle Organisation bei IRIS.

Der Vorteil dieser Art von Bäumen, die ja im Prinzip zunächst eine Verfeinerung des binären Suchens darstellen, besteht darin, daß ihre Struktur voll auf die Verwendung als Sekundärspeicher-Suchverfahren zugeschnitten ist und es erlaubt, mit extrem wenigen *Seiten*zugriffen ans Ziel zu kommen, auch wenn die indizierte Relation sehr umfangreich ist. Diesen Vorteil erkauft man allerdings mit einem großen Zusatzaufwand bei Updates, da sich die Änderungen an den Daten auch im Index wiederspiegeln müssen und in bestimmten Fällen mit einer größeren Reorganisation desselben verbunden sind.

Der Index wird von IRIS wie eine eigene Relation verwaltet, die die unterstützten Attribute der ursprünglichen Relation sowie das Attribut „Seitennummer“ enthält. In der DD/D-Relation `Relation` wird sie mit `Bemerkung = Index` eingetragen. Allerdings sind solche Relationen für den Benutzer nicht zugreifbar, sondern können nur vom System gelesen und geändert werden.

Indexsequentielle Zugriffspfade sind zur Unterstützung von „exact match queries“ und „range queries“ geeignet.

5.2.5 Indiziert-nichtsequentielle Organisation

Auch dieses Verfahren baut auf B-Bäumen als „Inhaltsverzeichnis“ auf. Allerdings ist die indizierte Relation nicht nach den im Baum unterstützten Attributen sortiert, so daß die Blätter eine Stufe tiefer liegen und „kreuz und quer“ auf Tupel der Relation verweisen müssen. Der Sinn dieser Konstruktion liegt darin, daß eine Relation immer nur in *einer* Weise sortiert werden kann, es aber sinnvoll ist, einen weiteren Index anzulegen, wenn auch andere Attribute häufig für Selektionen verwendet werden. Es handelt sich hierbei also um zusätzliche Zugriffspfade, die mit (anderen) Dateioganisationsformen kombiniert werden können.

Hinsichtlich der Realisierung gilt für die indiziert-nichtsequentielle Organisationsform dasselbe, was im vorigen Abschnitt über die indexsequentielle Organisation gesagt wurde: „Echte“ B-Bäume liegen auch hier nicht vor.

Das Verfahren ist vorteilhaft für „exact match“ und „partial match queries“.

5.2.6 kdB-Bäume

Ein kdB-Baum ist ein Baum, dessen innere Knoten aus je einer „Bereichsseite“ und dessen Blätter aus je einer „Satzseite“ bestehen. Eine Bereichsseite enthält eine Menge von Knoten (Schnittelemente), von denen jeder einen Bereich von Datensätzen anhand der Werte eines Attributes in zwei (idealerweise gleichmächtige) disjunkte Unterbereiche aufteilt. Im Gegensatz zu B-Bäumen wird also bei kdB-Bäumen bei jedem Knoten in nur zwei Richtungen verzweigt, indem geprüft wird, ob der gesuchte Attributwert größer oder kleiner als der im Knoten gespeicherte Wert ist. Die beiden Nachfolgeknoten sind entweder wieder Schnittelemente dieser Bereichsseite oder sog. Adreßelemente, die auf je einen Nachfolger dieser

Bereichsseite verweisen. Nachfolger können sowohl weitere Bereichsseiten als auch Satzseiten sein, die den durch den jeweiligen Ast des kdB-Baumes eingeschränkten Bereich von Datensätzen enthalten.

kdB-Bäume sind mehr-(k-)dimensional, d.h. verschiedene Knoten können (und sollten) den aktuellen Bereich anhand der Werte unterschiedlicher Attribute teilen. Zum einen kann so unter Ausnutzung der Selektivitäten der Attributwerte das Schnittattribut so gewählt werden, daß die beiden Unterbereiche gleich groß sind, zum anderen werden somit eine Indizierung nach mehreren Attributen und mehrdimensionales Suchen ermöglicht.

Genaueres zu kdB-Bäumen, speziell zu ihrer Implementierung in IRIS, findet man in [mp87].

kdB-Bäume eignen sich besonders für die Unterstützung von „partial match queries“, durch die bereichsorientierte Zerlegung des Datenbestandes aber auch für „range queries“.

5.2.7 Mehrdimensionales Hashing

Hashing bedeutet im Grundsatz, zunächst eine feste Anzahl von Speicherplätzen für Tupel zu reservieren und anschließend mittels einer sogenannten Hash-Funktion aus den Attributwerten des gesuchten bzw. einzutragenden Tupels den Speicherplatz zu errechnen, auf den dieses Tupel gehört. Die Kunst besteht dabei darin, die Tupel möglichst gleichmäßig auf den Speicherbereich zu verteilen. Außerdem ist eine Kollisionsbehandlung für den Fall vorzunehmen, daß die Hash-Funktion für zwei verschiedene Tupel denselben Funktionswert liefert. Dies kann in der Erzeugung von einer oder mehreren Überlauflisten bestehen oder aber in der Bereitstellung einer ganzen Folge von Hash-Funktionen, die dann jeweils einen anderen Speicherplatz errechnen.

Abweichend davon darf bei Hashing als Organisationsform für den Externspeicher die Zahl der reservierten Speicherplätze *nicht* fest vorgegeben, sondern dynamisch sein. Das bedeutet gleichzeitig, daß auch die Überlaufbehandlung durch Listen ungeeignet ist. Das Hash-Verfahren für IRIS verfährt so, daß im Kollisionsfall die Zahl der verwalteten Speicherplätze verdoppelt wird.

In IRIS wurde ein *mehrdimensionales* Hash-Verfahren implementiert [nsw89]. Das heißt, daß diese Art der Zugriffsunterstützung nicht nur für jeweils *ein* Attribut einer Relation verfügbar ist, sondern daß *mehrere* Attribute gleichzeitig unterstützt werden können. Die Hash-Adresse eines Tupels wird dann aus den Werten aller dieser Attribute errechnet.

Eine weitere Besonderheit besteht darin, daß zwei Varianten der Hash-Unterstützung angeboten werden: Eine **ordnungserhaltende** und eine **nicht ordnungserhaltende**. Ersteres bedeutet, daß die Sortierung der Tupel nach ihren Attributwerten durch die Hash-Funktion erhalten und nicht „durcheinandergebracht“ wird; letzteres bedeutet, daß bei der Berechnung der Hash-Adressen darauf keine Rücksicht genommen wird. Beide Varianten haben in bestimmten Fällen Vor- und in anderen Nachteile, worauf wir hier aber nicht näher eingehen möchten.

Bei „exact match queries“ ist Hashing in der Leistung unschlagbar, da ein Tupel im Prinzip mit einem einzigen „Griff“ gefunden wird — „im Prinzip“ deshalb, weil bei IRIS zur Ermittlung der tatsächlichen Adresse eines Tupels zunächst in einem Hash-Index nachgeschaut werden muß, der im Hauptspeicher steht. Dieser verweist auf einen Eintrag in einer

Hash-Tabelle, die als Relation im `DATAF` steht, und erst nach deren Konsultation kann dann der Zugriff auf das gesuchte Tupel erfolgen. Es werden also tatsächlich *zwei* Seitenzugriffe benötigt. Trotzdem ist das im Normalfall weniger, als ein wie auch immer gearteter Baum als Index brauchen würde. Erkaufen muß man diese Geschwindigkeit allerdings mit einer schlechteren Speicherauslastung, da zwangsläufig auch Speicherplätze im `DATAF` leer bleiben.

5.3 Die Statistikkomponente

Zweck der Definitonskomponente der Internen Ebene (DIE) ist es, die von IRIS angebotenen Zugriffsunterstützungen durch gezielte Sammlung von Informationen über die erfolgten Zugriffe und ggf. entsprechende Änderung der eingestellten Zugriffspfade optimal nutzen zu können.

Dazu umfaßt die DIE eine Statistik-Komponente, die zunächst Informationen über die in Anfragen angesprochenen Relationen und Attribute sowie über die dabei verwendeten relationenalgebraischen Operationen sammelt. Der DBA als Verantwortlicher für den effizienten Einsatz des Datenbanksystems soll daraus ermitteln können, ob die zur Zeit gewählten Zugriffspfade bereits die optimale Unterstützung bieten, oder ob eine andere Unterstützungsmethode den Bedürfnissen der Benutzer eher entspräche.

Die Schemata der Statistik-Relationen finden sich im Anhang bei den DD/D-Relationen, denen sie organisatorisch angegliedert sind.

Die interaktive Schnittstelle zur Statistikkomponente bietet folgenden Funktionsumfang:

1. Anzeige der **Tupel- und Seitenanzahl von Relationen**
2. Anzeige von **Selektivitäten und Varianzen einzelner Attribute:** Es handelt sich hier um Größen, die ausdrücken, wie gut ein Attribut geeignet ist, bei einer Selektion die Anzahl der Tupel im Ergebnis zu reduzieren.
3. **Statistik über relationenalgebraische Anfragen:** Anzeige der statistischen Werte über die Verwendung bestimmter Gruppen der relationenalgebraischen Operationen (Joins; Mengenoperationen; Projektionen und Selektionen), und zwar für bestimmte oder für alle Relationen.
4. **Basisorganisations- und Zugriffspfadänderungen auf Relationen:** Die Indexstruktur einer Relation kann angezeigt und durch Änderung der Basisorganisation, Neuerstellung oder Variation eines Index veränderten Bedürfnissen angepaßt werden.
5. **Änderung statistischer Größen:** Hier werden Tupel- und Seitenanzahl einer oder aller Relation(en) aktualisiert. Außerdem kann die Anfragestatistik ein- und ausgeschaltet sowie initialisiert werden.

Letzte Worte

Dieser Abschlußbericht basiert z.T. auf den Zwischenberichten [Heu85c,Heu87], der „Metadokumentation“ [hm92] sowie den in Anhang B aufgeführten Einzeldokumentationen.

An der Konzeption und Implementierung waren folgende Mitglieder der Projektgruppe Relationale Daten-/Objektbanksysteme beteiligt:

Prof. Dr. Klaus Ecker, Dr. Andreas Heuer, Dr. Jürgen Fuchs, Prof. Dr. Volkert Brosda, Dr. Jutta Göers, Holger Riedel,
Joachim Bassendowski, Norbert Bibo, Ralph Busse, Petra Constapel, Martin Dolezyk, Hartmut Garke, Jutta Gimpel, Charlotte Grünsch, Torsten Grust, Bernadette Hillebrand, Olaf Hübsch, Carsten Hörner, Bernd Kaack, Betül Kaygi, Michael Kern, Beate Klingebiel, Joachim Kröger, Wolfgang Kurz, Thomas Lemmen, Gunner Linde-Göers, Stefan Manegold, Annette Meyer, Bernd Mußmann, Astrid Niemeyer, Silvia Peter, Stefan Piening, Michael Post, Michael Röbbecke, Dirk Röder, Karl-Christian Rudolph, Tijen Sahin, Dr. Peter Sander, Martin Schwätzer, Detlef Schuckert, Steffen Sohrweide, Rolf Terei, Jens Thamm, Thomas Weik, Uwe Wiebking, Axel Wiechel, Dorothea Winkler, Christina Worf.

Ihnen allen sei für ihren Einsatz, ihre Anregungen, aber auch für ihre Kritik gedankt, ohne die das Projekt nicht denkbar gewesen wäre.

Der Leser, der bis hierhin gelangt ist und mit gerunzelter Stirn nun aufblickt, weil vielleicht des Vollendeten zu wenig ihm erschien, möge ein Letztes noch bedenken:

> *Tadeln ist leicht, erschaffen so schwer; Ihr Tadler des Schwachen, habt Ihr das Treffliche denn auch zu belohnen ein Herz?*
>
> (Goethe, Xenien)

Anhang A
Externe Literatur

[AB86] S. Abiteboul, N. Bidoit: *Non First Normal Form Relations: An Algebra Allowing Data Restructuring;* Journal of Computer and System Sciences, Vol. 33, 361-393, 1986.

[ANS75] ANSI X3/SPARC Study Group on Data Base Management Systems: *Interim Report;* FDT (ACM SIGMOD Record) 7 (2), 1975.

[Ban86] F. Bancilhon: *Naive Evaluation of Recursively Defined Relations;* in Michael L. Brodie, John Mylopoulos (Eds.): *On Knowledge Based Management Systems,* Ch. 15; Springer, Topics in Information Systems, 1986.

[BM77] R.S. Boyer, J.S. Moore: *A fast string searching algorithm;* Communications of the ACM 20, 10, 1977.

[Bro89] Volkert Brosda: *Zugriffs-Konstruktions-Regeln: Ein Konzept zur Erweiterung relationaler Datenbankanfragesprachen für den Einsatz in speziellen Anwendungen der Wissensverarbeitung;* Dissertation, TU Clausthal, Institut für Informatik, 1989.

[BV85] Volkert Brosda, G. Vossen: *Updating a Relational Database through a Universal Schema Interface;* ACM PODS, 4, 1985.

[Cod70] E.F. Codd: *A Relational Model of Data for Large Shared Data Banks;* Communications of the ACM, Vol. 13, No. 6, Juni 1970.

[Cza87] Detlef Czaja: *Zusammenhang zwischen der LTK-Normalform und FD-basierten bzw. attribut-basierten Datenbankentwurfsalgorithmen;* Diplomarbeit, TU Clausthal, Institut für Informatik, 1987.

[Dat83] C.J. Date: *An Introduction to Database Systems Vol.2;* Addison-Wesley, 1983.

[Dat89] C.J. Date: *A Guide to the SQL Standard;* Addison Wesley, 1989.

[Dat90] C.J. Date: *An Introduction to Database Systems Vol.1;* Addison-Wesley, 1990.

[Fuc84] Jürgen Fuchs: *Interne Ebene von Datenbanksystemen: Grundlegende Datenorganisationsformen und ihre Operationen;* Diplomarbeit, TU Clausthal, Institut für Informatik, 1984.

[Fuc85] Jürgen Fuchs: *Die interne Ebene von IRIS;* in [Heu85].

[Heu85] Andreas Heuer (Hrsg.): *Workshop über relationale Datenbanken*, Informatik-Bericht 85/1, TU Clausthal, Institut für Informatik, 1985.

[Heu85a] Andreas Heuer: *Systematischer Überblick über bekannte und neue Fensterfunktionen;* in [Heu85].

[Heu85b] Andreas Heuer: *Überblick über das IRIS-Projekt;* in [Heu85].

[Heu86] Andreas Heuer: *Theorie der IRIS-SYNTHI-Synthese;* Informatik-Bericht 86/2, TU Clausthal, Institut für Informatik, Juli 1986.

[Heu87] Andreas Heuer: *Das IRIS-Projekt - Ein Systemüberblick;* Informatik-Bericht 87/1, TU Clausthal, Institut für Informatik; 1987.

[Heu88] Andreas Heuer: *Exakte Charakterisierung eines semantischen Datenmodells und seiner Operationen durch relationale Konzepte;* Dissertation, TU Clausthal, Institut für Informatik, 1988.

[HH91] Carsten Hörner, Andreas Heuer: *EXTREM — The structural part of an object-oriented database model;* Informatik-Bericht 91/5, TU Clausthal, Institut für Informatik, Oktober 1991.

[Kel85] A. Keller: *Algorithms for Translating View Updates to Database Updates for Views Involving Selections, Projections and Joins;* ACM PODS, 4, 1985.

[Kla92] Theo Klaustal: *Das OSCAR-Projekt;* Informatik-Bericht 92/2, TU Clausthal, Institut für Informatik; 1992.

[LH85] Gunner Linde, Andreas Heuer: *Clausthaler URQUEL gegen Dortmunder DURST (oder: Zwei UR-Schnittstellen im Vergleich);* in [Heu85].

[Lie81] Y. E. Lien: *Hierarchical Schemata for Relational Databases;* ACM Transactions on Database Systems, Vol. 6, No. 1, March 1981, pp. 48-69.

[Mai83] David Maier: *The Theory of Relational Databases;* Computer Science Press, Rockville, MD, 1983.

[Sag83] Y. Sagiv: *Quadratic Algorithms for Minimizing Joins in Restricted Relational Expressions;* SIAM Journal of Computation, 12, 2, 1983.

[SH85] Peter Sander, Andreas Heuer: *Vergleich verschiedener Synthese-Algorithmen;* in [Heu85].

[Ull88] J.D. Ullman: *Principles of Database and Knowledge-Base Systems;* Computer Science Press, Rockville, MD; 1988.

[VGF86] Dirk Van Gucht, Patrick C. Fischer: *Some Classes of Multilevel Relational Structures;* Proc. Int. Conf. on Principles of Database Systems, 5: 60-69, 1986.

[Vos87] Gottfried Vossen: *Datenmodelle, Datenbanksprachen und Datenbank-Management-Systeme;* Addison-Wesley Deutschland, 1987.

[Win88] Dorothea Winkler: *Über die Abbildbarkeit von Aufdatierungsoperationen auf relationenalgebraischen Ausdrücken in elementare Operationen auf Basisrelationen;* Diplomarbeit, TU Clausthal, Institut für Informatik, 1988.

[Wir83] Niklaus Wirth: *Algorithmen und Datenstrukturen;* Teubner, Stuttgart, 1983.

Anhang B
Interne Literatur

[bib90] Norbert Bibo: *Insert – Delete – Modify: Update-DML-Benutzeranleitung;* April 1990.

[bus91] Ralph Busse: *Beschreibung der Auswertung von Regel-Anfragen unter NaLogo!;* Mai 1991.
(z.T. neu in [kk92])

[dw88] Martin Dolezyk, Uwe Wiebking: *Das RELAX-Buch;* SWP-Bericht, 1988.

[gim89] Jutta Gimpel: *Dokumentation zur Einbettung abstrakter Datentypen in das IRIS-System;* 1989.

[gk88] Jutta Gimpel, T. Betül Kaygi: *Ändern der Definitionskomponente konzeptuelle Ebene bei manuell erzeugten Relationenschemata;* SWP-Bericht, März 1988. (vgl. [kg88])

[gk90] Charlotte E. Ch. Grünsch, Beate Klingebiel: *Eine NERO-Schnittstelle (Die Units: NSCAN und NTRAFO);* SWP-Bericht, 1990.

[gt92] Torsten Grust, Jens Thamm: *NERO++ (A better NERO) Benutzerdokumentation;* SWP-Bericht, Juni 1992.

[hm92] Carsten Hörner, Stefan Manegold: *A Guide to the IRIS Docs — Die Meta-Dokumentation;* Juli 1992.

[hw86] Olaf Hübsch, Dorothea Winkler: *Sortieren;* September 1986.

[kg88] T. Betül Kaygi, Jutta Gimpel: *Benutzerhandbuch zu Änderungen am bestehenden Schema;* Dezember 88. (vgl. [gk88])

[kk92] Michael M. Kern, Joachim Kröger: *Das Softwarepraktikum NaLogo!^{opt};* Januar 1992.

[lr92] Thomas Lemmen, Karl-Christian Rudolph: *Das Regel_Ed-Buch;* SWP-Bericht, Juni 1992.

[lst?] Gunnar Linde, Steffen Sohrweide, Rolf Terei: *Handbuch IRIS-URQUEL.*

[mp87] Bernd Mußmann, Stefan Piening: *Implementierung einer Pufferverwaltung und eines kdB-Baumes als mehrdimensionaler Zugriffspfadstruktur innerhalb des relationalen Datenbanksystem IRIS;* SWP-Bericht, Dezember 1987.

[nsw89] Astrid Niemeyer, Tijen Sahin, Christina Worf: *Dokumentation des Software-Praktikums HASH;* November 1989.

[rr88] Holger Riedel, Michael Röbbecke: *Dokumentation des SWP LDML (Implementation einer regelbasierten Datenbankanfragesprache);* SWP-Bericht, Dezember 1988.

Anhang C
IRIS-Menüstruktur

1. Definitionskomponenten

1. Konzeptuelle Ebene *(DKE)*

    1. Datenbankdefinition algorithmisch neu *(IRIS-SYNTHI)*
        1. Start SYNTHESE
            1. Universum und funktionale Abhängigkeiten bearbeiten
                1. Vorhandene Daten einladen
                2. Universum von Hand bearbeiten
                3. Funktionale Abhängigkeiten von Hand bearbeiten
                4. Algorithmisches Ändern von Universum und FD's
                    1. Attribut einfügen
                    2. Attribut löschen
                    3. Funktionale Abhängigkeit einfügen
                    4. Funktionale Abhängigkeit löschen
                5. Universum und FD's am Bildschirm betrachten
                6. Universum und FD's drucken
                7. Daten für spätere Verwendung speichern
            2. Berechnung ausführen
            3. Schlüsselwahl und Ergebnis abspeichern
            4. Ergebnis der Berechnung am Bildschirm betrachten
            5. Synthese-Ergebnis drucken
        2. 4NF-Dekomposition nach Lien
        3. Eintragen SYNTHESE
        4. Eintragen Zusatzinformationen *(1)*
        5. DBA-Tools *(1)*
    2. Datenbankdefinition frei neu
    3. Datenbankdefinition ändern algorithmisch
        1. 4NF-Dekomposition nach Lien *(1) s.o.!*
    4. Datenbankdefinition ändern frei
        1. Hinzufügen eines Attributs
        2. Löschen eines Attributs
        3. Ändern der Domäne eines Attributs
        4. Ersetzen eines Schlüssels

5. Einführung weiterer Schlüssel
6. Löschen eines Schlüssels
7. Löschen des Schemas *(Löschen der gesamten Relation!)*
8. Liste der möglichen Relationennamen ausgeben

5. DD/D-Information *(1)*

2. Interne Ebene *(DIE + Statistik)*

1. Tupel- und Seitenanzahl von Relationen
2. Selektivitäten und Varianzen einzelner Attribute
3. Aufdatierung statistischer Größen
   1. für alle Relationen aktualisieren
   2. einer ausgewählten Relation aktualisieren
   3. Anfrage-Statistik einschalten
   4. Anfrage-Statistik ausschalten
   5. Anfrage-Statistik initialisieren
4. Statistik über relationenalgebraische Anfragen
   1. Joins zwischen 2 Relationen
   2. Vereinigungen, Differenz- und Durchschnittsbildungen zwischen 2 Relationen
   3. Projektion und Selektion auf Relationen
5. Basisorganisations- und Zugriffspfadänderungen auf Relationen
   1. Index neu erstellen
   2. Index variieren
   3. Basisorganisation ändern
   4. Indexstruktur zu einer Relation anzeigen

3. Externe Ebene *(DEE)*

1. Sichten definieren
2. Sichten ändern
3. DD/D-Information *(1)*

2. Anwendungskomponenten

1. Datendefinition *(1)*

2. Anwendungsdefinition

1. Workfile erzeugen
2. Workfile ändern
3. Editor aufrufen
4. IRIS-PASCAL-Precompiler
5. Compiler & Linker
   1. Compiler PAS1

    2. Compiler PAS2
    3. Linkfile erzeugen
    4. Linkfile ändern
    5. Linker aufrufen
  6. Programm starten

3. Formulardefinition / Interaktionsdefinition *(1)*

3. IQL

1. RELAX *(→ Editor)*

2. NERO *(→ Editor)*

3. URQUEL

4. LDML

    1. Regeln eingeben, ändern oder löschen
    2. LDML-Anfragen stellen

5. Update – DML

    1. UPDATE.DAT bearbeiten
    2. Syntax überprüfen
    3. Übersetzen
    4. Transaktion starten
    5. Commit
    6. Rollback
    7. Abbrechen
    8. Metadaten ausgeben

6. NaLogo!

    1. Regelauswertung
    2. Regelverwaltung
        1. Regeln aus DB einlesen
            1. Regel zu Regelnummer einlesen
            2. Regeln zu Regelnamen einlesen
            3. Selektiv aus DB auswählen
            4. Alle Regeln einlesen
            5. Regel ansehen
        2. Regeln einfügen
        3. Regeln ändern
        4. Regeln speichern

5. Regeln löschen

4. Dialogkomponente

1. Standard-Aufdatierungsroutinen *(1)*

2. Interaktives Update *(→ Vorselektion)*

5. Test

1. Datenbasis erzeugen *(neue Datenbank!)*

2. Kopie der Tastatureingabe ausschalten

3. Datenbasis einschleusen *(alte Update-Komponente)*

    1. Datenbasis listen *(Relation „Relation“)*
    2. Tests für EXTREM-Schemata
        1. Test
    4. Relation einschleusen *(zwecks Update)*
        1. Schema listen
        2. Tupel anlegen *(eingeben)*
        3. Tupel listen
        4. Sequentielles Löschen *(mit FS-Test)*
        5. Mehrdimensionales Löschen *(ohne FS-Test)*
        6. Mehrdimensionales Suchen

Bemerkungen:

(1) ... nicht vorhanden (mit oder ohne Meldung)

Anhang D
Struktur des Data Dictionary / Directory

Im folgenden werden die Relationen des Data-Dictionary / Directory (DD/D) und der daran angeschlossenen Statistik-Komponente der Definitionskomponente Interne Ebene (DIE) schematisch aufgelistet. Dadurch soll vor allem *Benutzern* die Möglichkeit gegeben werden, sich über den Aufbau des DD/D zu informieren. Einige Details wie Attribut- oder Domänen-Identifier werden daher hier nicht dargestellt, im Bedarfsfall dürfte aber die vorliegende Darstellung den daran Interessierten in die Lage versetzen, sich diese Informationen aus den DD/D-Relationen zu besorgen.

Folgendes Schema wird für die Darstellung verwendet:

(Rel_Id) **Rel_Name** — Bemerkung:

Att_Name1	Att_Name2	...
Dom_Name1 [Länge1]	Dom_Name2 [Länge2]	... ...

„Bemerkung“ ist die in der DD/D-Relation „Relation“ verwendete Klassifizierung der einzelnen Relationen nach ihrem Zweck. Werden Attributnamen unterstrichen, so bedeutet dies, daß die betreffenden Attribute zum Primärschlüssel der Relation gehören. Weitere Schlüssel sind für die DD/D-Relationen zur Zeit nicht definiert. Die Angabe „Länge“ wird nur im Bedarfsfall eingetragen: Sie ist nur bei Strings erforderlich, bei allen anderen Domänen liegt sie fest. (s.u.!)

D.1 Eigentliches Data Dictionary

(1) **Relation** — DD/D:

Relname	Attributanzahl	Rel_Id	Bemerkung
Stringshort 30	Integer	Integer	Stringshort 30

Welche Relation (Name) verbirgt sich hinter welchem Relationen-Identifier, wieviele Attribute hat sie, und wie kann man ihren Zweck kurz charakterisieren? Neben den verschiedenen „Bemerkungen“ der DD/D-Relationen treten v.a. die folgenden auf:

manuell: Kennzeichnet Benutzerrelationen, die ohne algorithmische Unterstützung definiert wurden.

syntheseergebnis: Benutzerrelationen, die mit der SYNTHI-Komponente erzeugt wurden. (Voraussetzung für die Verwendung mit URQUEL)

extrem: Relationen, die zur Übersetzung eines EXTREM-Schemas in flache Relationen gehören.

(2) **Attribut** — DD/D:

Att_ID	Att_Name
Integer	Stringshort 30

Welches Attribut (Name) wird durch welchen Attribut-Identifier identifiziert?

(3) **Rel_Att** — DD/D:

Rel_Id	Att_Id	Rel_Pos
Integer	Integer	Integer

Welches Attribut kommt in welcher Relation vor, und an welcher Position steht es im Relationenschema?

(4) **Domain** — DD/D:

Dom_Id	Dom_Name	Laenge
Integer	Stringshort 30	Integer

Welcher Identifier repräsentiert welche Domäne, und mit welcher Länge (in Bytes) werden ihre Werte gespeichert?

(5) **Att_Dom** — DD/D:

Att_Id	Dom_Id
Integer	Integer

Welches Attribut hat welche Domäne?

(6) **Schluessel** — DD/D:

Rel_Id	Vektor	Schluessel_Art
Integer	Stringshort 30	Stringshort 30

„Vektor“ ist ein boole'scher Vektor, dessen i-te Komponente angibt, ob das im Relationenschema an i-ter Stelle stehende Attribut zum Schlüssel der Relation gehört. (1 bedeutet TRUE, also Zugehörigkeit.) „Schluessel_Art“ ist entweder „Primaerschluessel“ oder „Schluessel“. (Sekundärschlüssel gehören nicht hierher!)

(7) **Organisation** — DD/D:

Rel_Id	Ordnung
Integer	Stringshort 30

Diese Relation beschreibt die sequentielle Organisationsform (Sortierung) der Relationen. „Ordnung“ ist ein String aus Zeichen '0' ...'9', seine Länge ist die Attributanzahl der Relation. Ein Eintrag i an j-ter Stelle bedeutet, daß das im Relationenschema an j-ter Stelle stehende Attribut als i-tes Sortierkriterium verwendet wird.

(8) **Index** — DD/D:

Index_Id	Rel_Id	X_Vektor	Org_Art
Integer	Integer	Stringshort 30	Integer

„Index_Id“ ist der Identifier einer Indexrelation zur Relation mit Identifier „Rel_Id“. „X_Vektor“ ist ein boole'scher Vektor, bei dem ein Eintrag 1 an i-ter Stelle bedeutet, daß das i-te Attribut im Schema der indizierten Relation durch den Index unterstützt wird. Dieses Attribut ist also auch ein Attribut der Indexrelation. „Org_Art“ codiert die Art der Zugriffsunterstützung, also „index-sequentiell“, „indiziert nicht-sequentiell“, usw.:

Org_Art	Bedeutung
0	sortiert/unsortiert, keine Zugriffsunterstützung
1	indexsequentiell oder indiziert-nichtsequentiell
2	kdB-Baum
3	Hash-Unterstützung, ordnungsbewahrend
4	Hash-Unterstützung, nicht ordnungsbewahrend

(17) **Sichten** — DD/D:

Sichtname	Gueltig
Stringshort 8	Stringshort 1

Hier werden die Namen aller definierten Sichten gesammelt. „Gueltig“ ist ‘J’, wenn die betreffende Sicht noch verwendet werden darf, sonst ‘N’. (Eine Sicht kann durch Änderungen am Datenbankschema ungültig werden.)

(18) **Sicht_Rel** — DD/D:

Sichtname	Relationen
Stringshort 8	Stringshort 30

Diese Relation enthält zu jeder definierten Sicht die Menge der in ihr verwendeten Basisrelationen (mit Namen, nicht mit Identifiern!).

D.2 Statistik-Komponente

(11) **Stat_Tup_Seit** — Statistik:

Rel_Id	Tupelanz	Seitenanz	Datum
Integer	Integer	Integer	Stringshort 18

Die Relation enthält zu jeder Relation Tupel- und Seitenanzahl sowie den Zeitpunkt (Datum und Uhrzeit) der Ermittlung dieser Angaben.

(12) **Stat_Skt_Var** — Statistik:

Rel_Id	Att_Id	Selekt	Varianz
Integer	Integer	Rational	Rational

Hier werden zu den einzelnen Attributen jeder Relation ihre Selektivitäten und Varianzen gespeichert.

(13) **Stat_Join** — Statistik:

Rel_Id_A	Rel_Id_B	Maske_A	Maske_A	Join
Integer	Integer	Stringshort 30	Stringshort 30	Integer

„Join“ gibt an, wie oft bei den statistisch erfaßten Anfragen die Relationen mit Identifiern „Rel_Id_A“ und „Rel_Id_B“ miteinander verknüpft wurden. „Maske_A“ und „Maske_B“ sind boole'sche Vektoren, bei denen ein Eintrag 1 an Stelle i bedeutet, daß das i-te Attribut der ersten bzw. zweiten Relation Joinattribut war.

(14) **Stat_Sig_Pi** — Statistik:

Rel_Id	Sigma	Pi	Maske
Integer	Integer	Integer	Stringshort 30

„Pi“ gibt an, wie oft auf auf die in „Maske“ mit einer 1 gekennzeichneten Attribute der Relation mit Identifier „Rel_Id“ projiziert wurde, „Sigma“ enthält die Anzahl der Selektionen nach diesen Attributen.

(15) **Stat_VDD** — Statistik:

Rel_Id_A	Rel_Id_B	V_D_D
Integer	Integer	Integer

„V_D_D“ enthält die Anzahl der Durchschnitts-, Vereinigungs- oder Differenzoperationen mit den beiden Relationen „Rel_Id_A“ und „Rel_Id_B“.

D.3 Verwaltungs-Relationen

(9) **Reststrings** — Halde:

Ueberlaufstring
Stringlong

Diese Relation sollte einmal zur Verwaltung von Longstrings dienen.

(10) **KdB-Varianz** — Streubereich:

Rel_Id	Rel_Pos	Bereich
Integer	Integer	Doublerational

„Bereich“ ist die Varianz der Werte des durch „Rel_Pos“ angegebenen Attributs der Relation „Rel_Id“.

(16) **Hashverwaltung** — Hash:

Relid	RLevel	Anzseite
Integer	Integer	Stringshort 30

„RLevel“ ist gleich dem logarithmus dualis der Seitenanzahl der unterstützten Relation. „Anzseite“ ist die Seitenanzahl der zugehörigen Hashtabelle.

D.4 Domänen und ihre Längen

Die folgende Tabelle enthält die Namen der in IRIS zur Zeit vorgesehenen Domänen sowie die Länge, mit der ihre Werte im `DATAF` gespeichert werden.

Domäne	**Länge (in Bytes)**
Integer	2
Doubleinteger	4
Rational	4
Doublerational	8
Stringshort	1–30 ([2])
Stringlong	([3])
Datum	4 ([4])
Zeit	4 ([4])

[2] Längere Strings sind zur Zeit aus technischen Gründen nicht möglich.
[3] Ebenfalls zur Zeit nicht verfügbar.
[4] Speicherung als Doubleinteger.

D.5 LMDL/NaLogo!/NaLogo!^{opt}—DD/D

Im folgenden werden die Relationen schematisch aufgelistet, um die das Data-Dictionary / Directory (DD/D) für bestimmte Komponenten (LDML/NaLogo!/NaLogo!^{opt}) erweitert wird.

Attliste — manuell:

Reg_Nr	Zugriff_Pos	Att	Att_Neu	Konstante
Integer	Integer	Shortstring 30	Shortstring 30	Shortstring 30

Bed_Att_Att — manuell:

Reg_Nr	Vgl_Op	Att_Eins	Att_Zwei	Bed_Nr
Integer	Shortstring 30	Shortstring 30	Shortstring 30	Integer

Bed_Att_Konst — manuell:

Reg_Nr	Vgl_Op	Att	Konstant
Integer	Shortstring 30	Shortstring 30	Shortstring 30

Bed_Nest — manuell:

Bed_Nr	Att_Eins	Att_Zwei
Integer	Shortstring 30	Shortstring 30

Intrel — manuell:

Rel_Name	Att	Att_Pos
Shortstring 30	Shortstring 30	Integer

Intrel_Att — manuell:

Att	Att_Typ	Att_Laenge
Shortstring 30	Integer	Integer

Intrel_Erweit — manuell:

Att	Reg_Nr	Erweit
Shortstring 30	Integer	Shortstring 30

Intrel_Nest — manuell:

Nest	Att
Shortstring 30	Shortstring 30

Regel — manuell:

Reg_Nr	Reg_Name
Integer	Shortstring 30

Regel_Nest — manuell:

Reg_Nr	Nest	Nest_Att
Integer	Shortstring 30	Shortstring 30

Sel_Dep — manuell:

Sd_Nr	Rel_Name	Sd
Integer	Shortstring 30	Integer

Sel_Dep_Att — manuell:

Sd_Nr	Att_Name
Integer	Shortstring 30

Zugriff — manuell:

Reg_Nr	Zugriff_Name	Zugriff_Pos
Integer	Shortstring 30	Integer

Anhang E
Beispiel-Datenbankschema

Die Beispiele in diesem Bericht beziehen sich, sofern nicht anders angegeben, auf eine Datenbank mit folgenden Relationen:

Urlauber:

Urlaubernr	Name	Vorname	Wohnort
Integer4	Stringshort	Stringshort	Stringshort

Die potentiellen Kunden von „IRIS-Tours“.

Hobbies:

Urlaubernr	Hobby
Integer4	Stringshort

Jeder Kunde *kann* ein oder mehrere Hobbies haben, denen er im Urlaub gern nachgehen möchte.

Urlaubsorte:

Ort	Angebot
Stringshort	Stringshort

Urlaubsorte und ihre Freizeitmöglichkeiten, potentiell mehrere je Ort.

Verbindung:

Art	Start	Ziel
Stringshort	Stringshort	Stringshort

Direkte Verkehrsverbindungen mit Angabe des Transportmittels.