Kapitel 5

Mehrdimensionale Suchstrukturen

5.1 Problemstellung

Ein Sekundär-Index ist in der Lage, alle Records mit $x_1 \leq a \leq x_2$ zu finden. Nun heißt die Aufgabe: Finde alle Records mit $x_1 \leq a_1 \leq x_2$ und $y_1 \leq a_2 \leq y_2, \ldots$

Beispiel für mehrdimensionale Bereichsabfrage:
Gesucht sind alle Personen mit der Eigenschaft

Alter	zwischen 20 und 30 Jahre alt
Einkommen	zwischen 2000 und 3000 DM
PLZ	zwischen 40000 und 50000

Im folgenden betrachten wir (wegen der einfacheren Veranschaulichung) nur den 2-dimensionalen Fall. Diese Technik ist auf beliebige Dimensionen verallgemeinerbar.

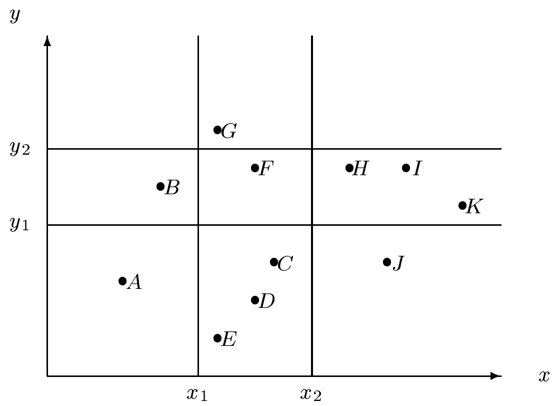

Abbildung 5.1: Fläche mit Datenpunkten

Abbildung 5-1 zeigt eine zweidimensionale Fläche mit Datenpunkten sowie ein Query-Rechteck, gegeben durch vier Geraden.

Die Aufgabe besteht darin, alle Punkte zu ermitteln, die im Rechteck liegen. Hierzu bieten sich zwei naheliegende Möglichkeiten an:

- Projektion durchführen auf x oder y mit binärer Suche über vorhandenen Index, danach sequentiell durchsuchen, d.h. zunächst werden G, F, C, D, E ermittelt, danach bleibt F übrig
- Projektion durchführen auf x und Projektion durchführen auf y, anschließend Durchschnitt bilden.

Es ist offensichtlich, daß trotz kleiner Trefferzahl ggf. lange Laufzeiten auftreten können. Dagegen ist für die 1-dimensionale Suche bekannt: Der Aufwand beträgt $O(k + \log n)$ bei k Treffern in einem Suchbaum mit n Knoten.

5.2 k-d-Baum

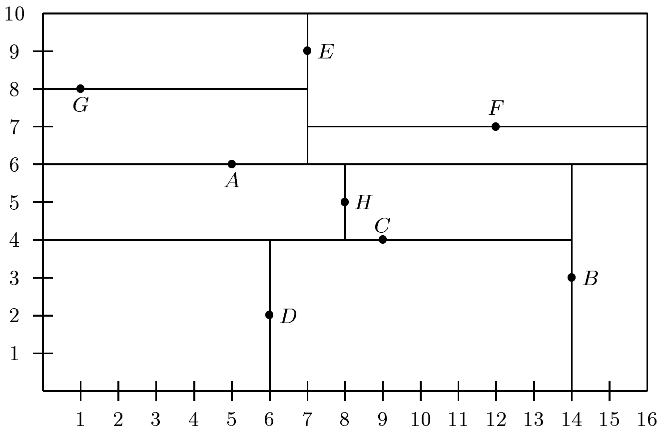

Abbildung 5.2: Durch die Datenpunkte A,B,C,D,E,F,G,H partitionierte Fläche

Eine Verallgemeinerung eines binären Suchbaums mit einem Sortierschlüssel bildet der *k-d-Baum* mit k-dimensionalem Sortierschlüssel. Er verwaltet eine Menge von mehrdimensionalen Datenpunkten, wie z.B. Abbildung 5.2 für den 2-dimensionalen Fall zeigt. In der homogenen Variante enthält jeder Baumknoten ein komplettes Datenrecord und zwei Zeiger auf den linken und rechten Sohn (Abbildung 5.3). In der inhomogenen Variante enhält jeder Baumknoten nur einen Schlüssel und die Blätter verweisen auf die Datenrecords (Abbildung 5.4). In beiden Fällen werden die Werte der einzelnen Attribute abwechselnd auf jeder Ebene des Baumes zur Diskriminierung verwendet. Es handelt sich um eine statische Struktur; die Operationen Löschen und die Durchführung einer Balancierung sind sehr aufwendig.

Im 2-dimensionalen Fall gilt für jeden Knoten mit Schlüssel $[x/y]$:

	im linken Sohn	im rechten Sohn
auf ungerader Ebene	alle Schlüssel $\leq x$	alle Schlüssel $> x$
auf gerader Ebene	alle Schlüssel $\leq y$	alle Schlüssel $> y$

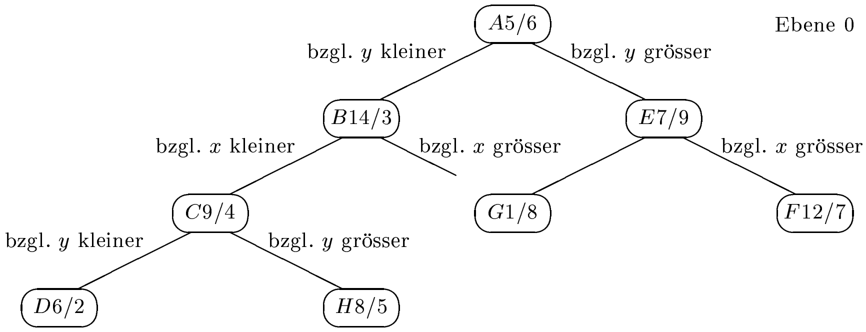

Abbildung 5.3: 2-d-Baum (homogen) zu den Datenpunkten A,B,C,D,E,F,G,H

Die Operationen auf einem $2-d$-Baum laufen analog zum binärem Baum ab:

- **Insert**:
  Suche mit Schlüssel $[x/y]$ unter Abwechslung der Dimension die Stelle, wo der $[x/y]$-Knoten sein müßte und hänge ihn dort ein.
- **Exakt Match** (z.B. finde Record [15/5]):
  Suche mit Schlüssel $[x/y]$ unter Abwechslung der Dimension bis zu der Stelle, wo der $[x/y]$-Knoten sein müßte.
- **Partial Match** (z.B. finde alle Records mit $x = 7$):
  An den Knoten, an denen nicht bzgl. x diskriminiert wird, steige in beide Söhne ab; an den Knoten, an denen bzgl. x diskriminiert wird, steige in den zutreffenden Teilbaum ab.
- **Range-Query** (z.B. finde alle Records $[x, y]$ mit $7 \leq x \leq 13, 5 \leq y \leq 8$):
  An den Knoten, an denen die Diskriminatorlinie das Suchrechteck schneidet, steige in beide Söhne ab, sonst steige in den zutreffenden Sohn ab. Beobachtung: Laufzeit $k + \log n$ Schritte bei k Treffern!
- **Best-Match** (z.B. finde nächstgelegenes Record zu $x = 7, y = 3$):
  Dies entspricht einer Range-Query, wobei statt eines Suchrechtecks jetzt ein Suchkreis mit Radius gemäß Distanzfunktion vorliegt. Während der Baumtraversierung schrumpft der Suchradius. Diese Strategie ist erweiterbar auf k-best-Matches.

Bei der inhomogenen Variante enthalten die inneren Knoten je nach Ebene die Schlüsselinformation der zuständigen Dimension sowie Sohnzeiger auf weitere innere Knoten. Nur die Blätter verweisen auf Datenblöcke der Hauptdatei, die jeweils mehrere Datenrecords aufnehmen können. Auch die inneren Knoten werden zu Blöcken zusammengefaßt, wie auf Abbildung 5.5 zu sehen ist. In Abbildung 5.4 befinden sich z.B. die Datenrecords *C, B* und *D* in einem Block.

Abbildung 5.6 zeigt, daß neben der oben beschriebenen 2-d-Baum-Strategie eine weitere Möglichkeit existiert, den Datenraum zu partitionieren. Dies führt zu den sogenannten *Gitterverfahren.*

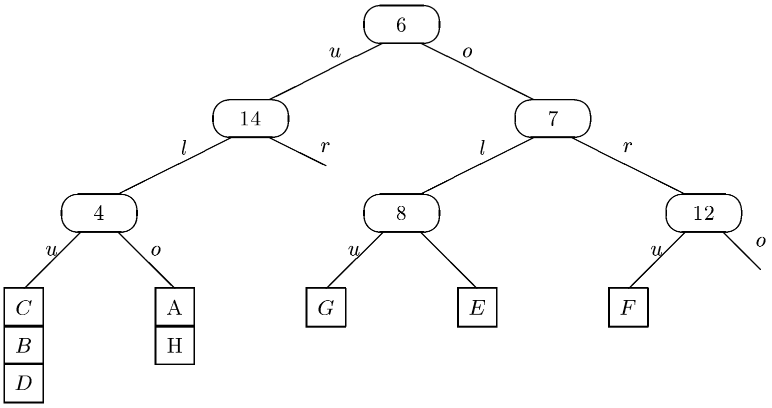

Abbildung 5.4: 2-d-Baum (inhomogen) zu den Datenpunkten A,B,C,D,E,F,G,H

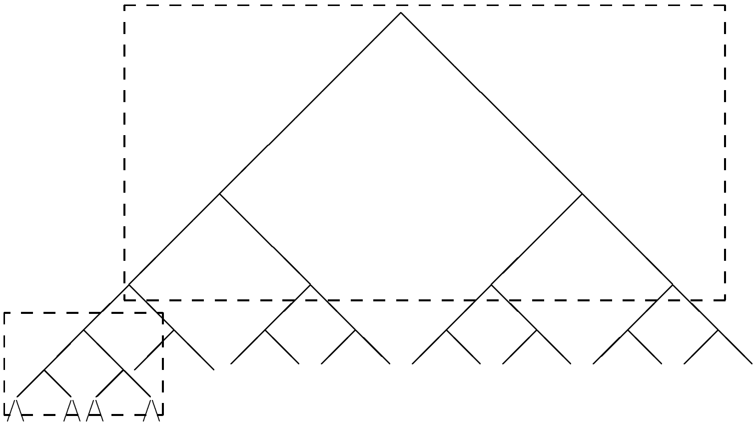

Abbildung 5.5: Zusammenfassung von je 7 inneren Knoten auf einem Index-Block

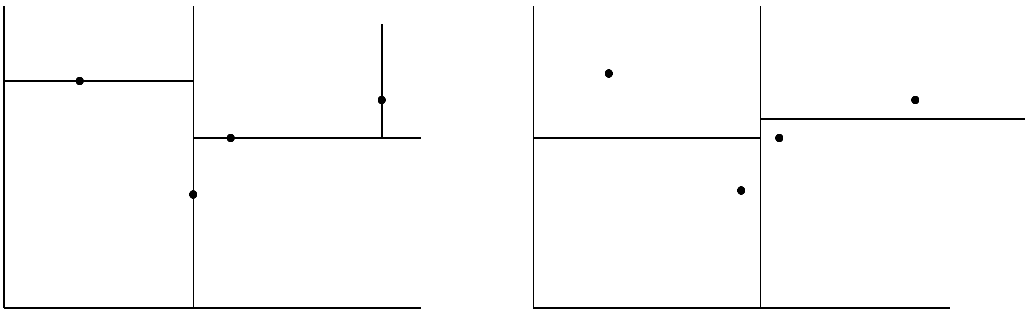

Abbildung 5.6: Partitionierungsmöglichkeiten des Raumes

5.3 Gitterverfahren mit konstanter Gittergröße

Gitterverfahren, die mit konstanter Gittergröße arbeiten, teilen den Datenraum in Quadrate fester Größe auf. Abbildung 5.7 zeigt eine Anordnung von 24 Datenblöcken, die jeweils eine feste Anzahl von Datenrecords aufnehmen können. Über einen Index werden die Blöcke erreicht. Diese statische Partitionierung lastet die Datenblöcke natürlich nur bei einer Gleichverteilung wirtschaftlich aus und erlaubt bei Ballungsgebieten keinen effizienten Zugriff.

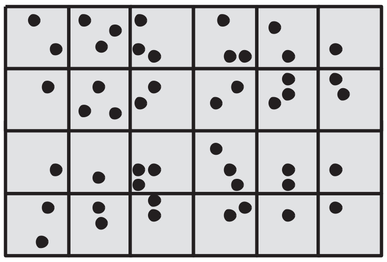

Abbildung 5.7: Partitionierung mit fester Gittergröße

5.4 Grid File

Als Alternative zu den Verfahren mit fester Gittergröße stellten Hinrichs und Nievergelt im Jahre 1981 das *Grid File* vor, welches auch bei dynamisch sich änderndem Datenbestand eine *2-Platten-Zugriffsgarantie* gibt.

Erreicht wird dies (bei k-dimensionalen Tupeln) durch

- k Skalen zum Einstieg ins Grid-Directory (im Hauptspeicher)
- Grid-Direktory zum Finden der Bucket-Nr. (im Hintergrundspeicher)
- Buckets für Datensätze (im Hintergrundspeicher)

Zur einfacheren Veranschaulichung beschreiben wir die Technik für Dimension $k = 2$. Verwendet werden dabei

- **zwei eindimensionale Skalen**,
  welche die momentane Unterteilung der X- bzw. Y-Achse enthalten:

  ```
  var X: array [0..max_x] of attribut_wert_x;
  var Y: array [0..max_y] of attribut_wert_y;
  ```

- **ein 2-dimensionales Grid-Directory**,
  welches Verweise auf die Datenblöcke enthält:

  ```
  var G: array [0..max_x - 1, 0..max_y - 1] of pointer;
  ```

D.h. `G[i, j]` enthält eine Bucketadresse, in der ein rechteckiger Teilbereich der Datenpunkte abgespeichert ist. Zum Beispiel sind alle Punkte mit $30 < x \leq 40, 2050 < y \leq 2500$ im Bucket mit Adresse `G[1,2]` zu finden (in Abbildung 5.8 gestrichelt umrandet). Achtung: mehrere Gitterzellen können im selben Bucket liegen.

- **mehrere Buckets**,
  welche jeweils eine maximale Zahl von Datenrecords aufnehmen können.

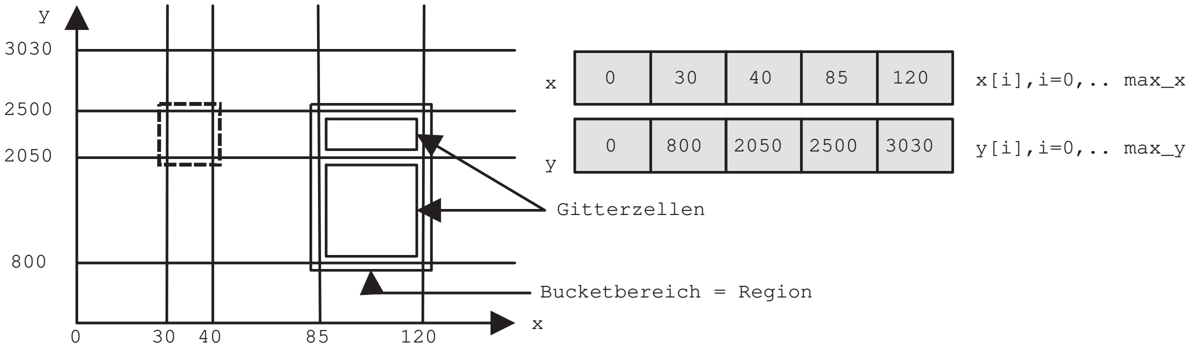

Abbildung 5.8: Skalen und resultierende Gitterzellen

Beispiel für ein Lookup mit $x = 100, y = 1000$:
Suche in Skala x den letzten Eintrag $< x$. Er habe den Index $i = 3$.
Suche in Skala y den letzten Eintrag $< y$. Er habe den Index $j = 1$.
Lade den Teil des Grid-Directory in den Hauptspeicher, der $G[3, 1]$ enthält.
Lade Bucket mit Adresse $G[3, 1]$.

Beispiel für den Zugriff auf das Bucket-Directory:
Vorhanden seien 1.000.000 Datentupel, jeweils 4 passen in einen Block. Die X- und die Y-Achse habe jeweils 500 Unterteilungen. Daraus ergeben sich 250.000 Einträge für das Bucket-Directory G. Bei 4 Bytes pro Zeiger und 1024 Bytes pro Block passen 250 Zeiger auf einen Directory-Block. Also gibt es 1000 Directory-Blöcke. D.h. $G[i, j]$ findet sich auf Block $2 \cdot j$ als i-te Adresse, falls $i < 250$ und befindet sich auf Block $2 \cdot j + 1$ als $(i - 250)$-te Adresse, falls $i \geq 250$

Bei einer *range query*, gegeben durch ein Suchrechteck, werden zunächst alle Gitterzellen bestimmt, die in Frage kommen, und dann die zugehörenden Buckets eingelesen.

5.5 Aufspalten und Mischen beim Grid File

Die grundsätzliche Idee besteht darin, bei sich änderndem Datenbestand durch Modifikation der Skalen die Größen der Gitterzellen anzupassen.

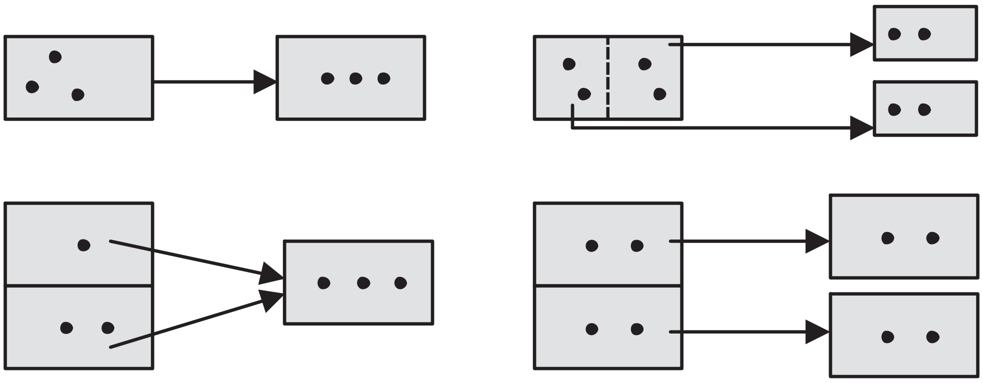

Abbildung 5.9: Konsequenzen eines Bucket-Überlauf (mit und ohne Gitterverfeinerung)

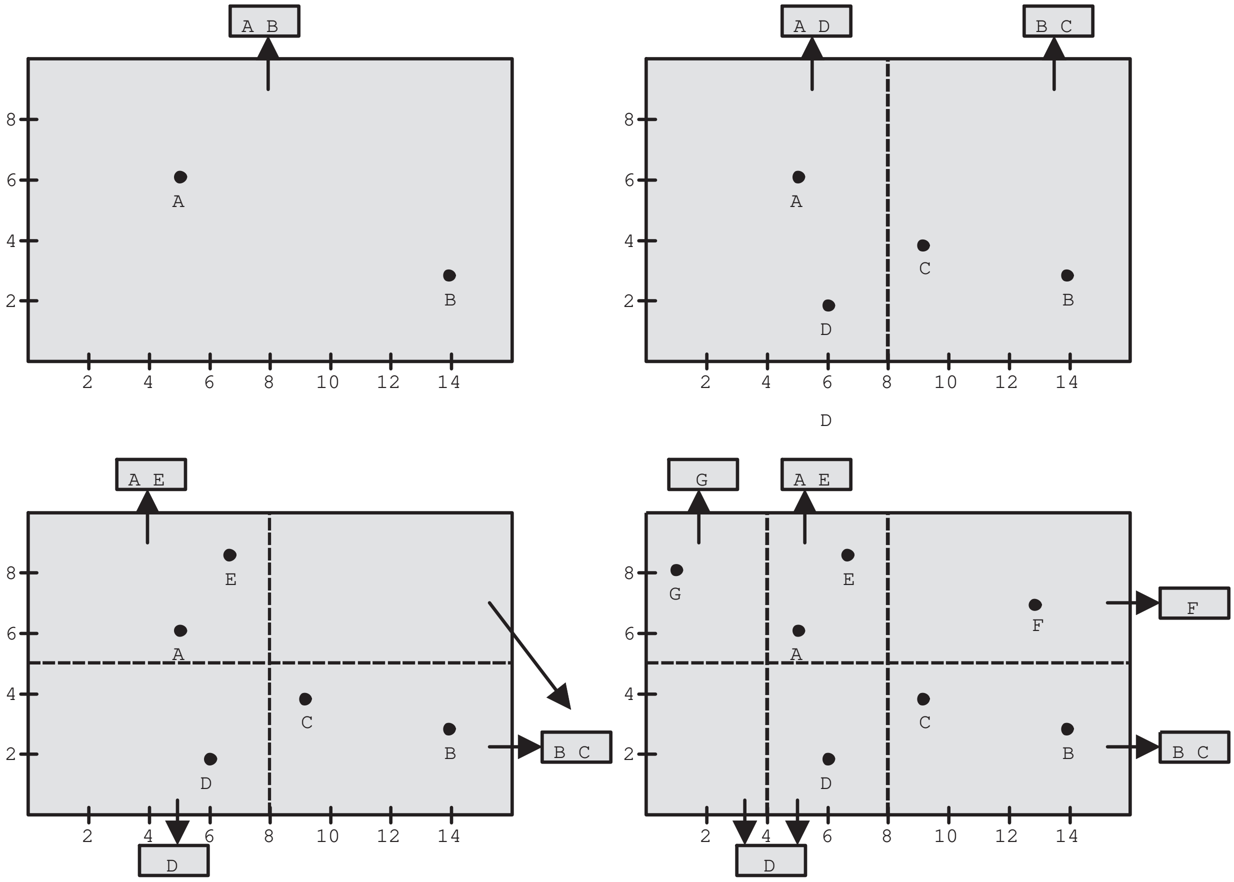

Abbildung 5.10: Aufspalten der Regionen für Datenpunkte A, B, C, D, E, F, G

Aufspalten von Regionen

Der Überlauf eines Buckets, dessen Region aus einer Zelle besteht, verursacht eine Gitterverfeinerung, die gemäß einer *Splitting Policy* organisiert wird. Im wesentlichen wird unter Abwechslung der Dimension die Region halbiert. Dieser Sachverhalt wird in der oberen Hälfte von Abbildung 5.9 demonstriert unter der Annahme, daß drei Datenrecords in ein Datenbucket passen. In der unteren Hälfte von Abbildung 5.9 ist zu sehen, daß bei Überlauf eines Buckets, dessen Region aus mehreren Gitterzellen besteht, keine Gitterverfeinerung erforderlich ist.

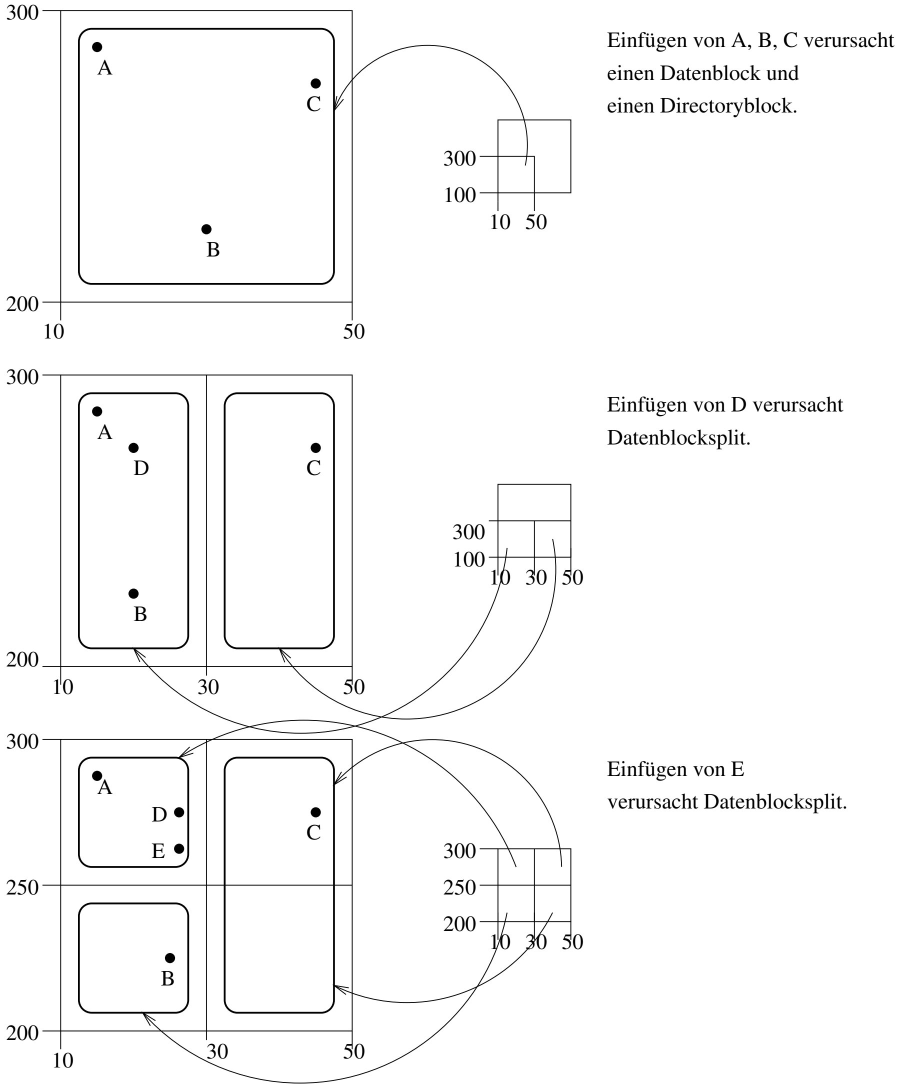

Abbildung 5.11: Dynamik des Grid File beim Einfügen der Datenpunkte A,B,C,D,E

Abbildung 5.10 zeigt die durch das sukzessive Einfügen in ein Grid File entwickelte Dynamik. Es handelt sich dabei um die in Abbildung 4.14 verwendeten Datenpunkte A, B, C, D, E, F, G. In dem Beispiel wird angenommen, daß 2 Datenrecords in einen Datenblock passen. Bei überlaufendem Datenblock wird die Region halbiert, wobei die Dimension abwechselt. Schließlich hat das Grid-Directory 6 Zeiger auf insgesamt 5 Datenblöcke. Die x-Skala hat drei Einträge, die y-Skala hat zwei Einträge.

Zu der dynamischen Anpassung der Skalen und Datenblöcke kommt noch die Buchhaltung der Directory-Blöcke. Dies wird in der Abbildung 5.11 demonstriert anhand der (neu positionierten) Datenpunkte A, B, C, D, E. Von den Directory-Blöcken wird angenommen, daß

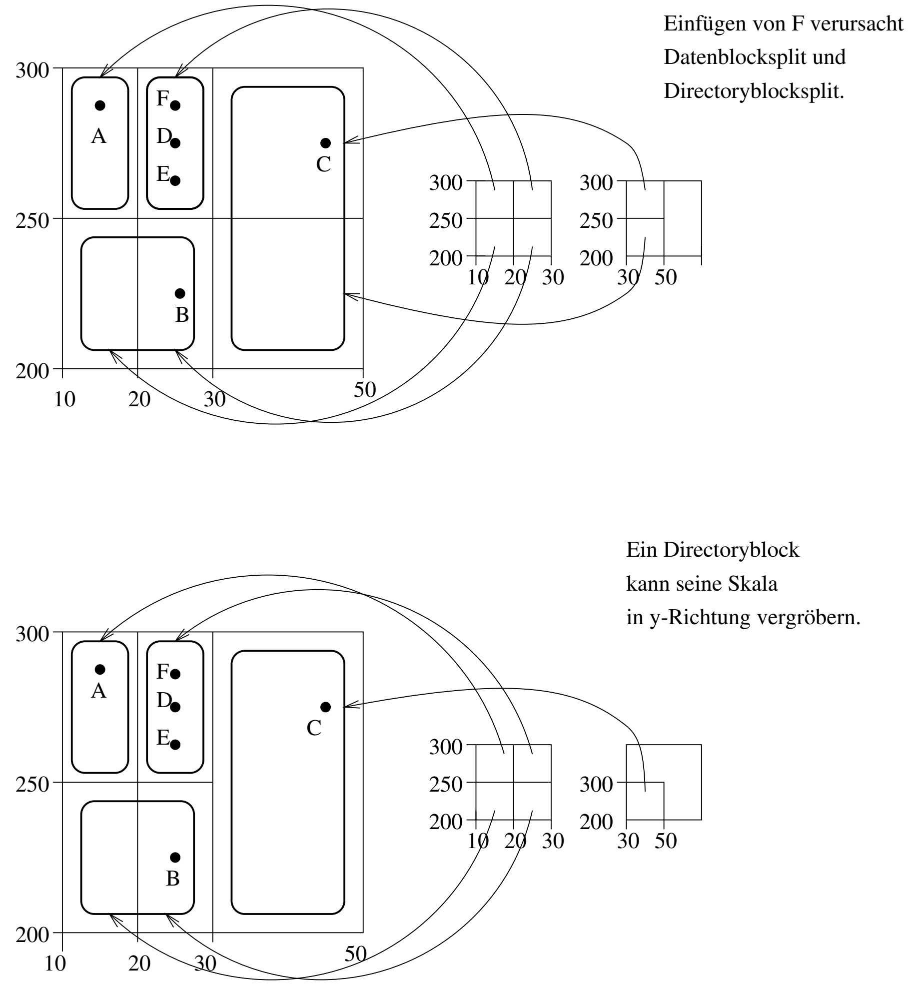

Abbildung 5.12: Vergröberung des Grid Directory nach Aufspalten

sie vier Adressen speichern können, in einen Datenblock mögen drei Datenrecords passen. Grundsätzlich erfolgt der Einstieg in den zuständigen Directory-Block über das sogenannte *Root-Directory*, welches im Hauptspeicher mit vergröberten Skalen liegt. Die durch das Einfügen verursachte Aufspaltung eines Datenblocks und die dadurch ausgelösten Verfeinerungen der Skalen ziehen auch Erweiterungen im Directory-Block nach. Abbildung 5.12 zeigt, wie beim Überlauf eines Directory-Blockes dieser halbiert und auf zwei Blöcke verteilt wird. Dabei kommt es zu einer Vergröberung der Skala.

Mischen von Regionen

Die beim Expandieren erzeugte Neustrukturierung bedarf einer Umordnung, wenn der Datenbestand schrumpft, denn nach dem Entfernen von Datenrecords können Datenblöcke mit zu geringer Auslastung entstehen, welche dann zusammengefaßt werden sollten. Die *Merging Policy* legt den Mischpartner und den Zeitpunkt des Mischens fest:

- Mischpartner zu einem Bucket X kann nur ein Bucket Y sein, wenn die Vereinigung der beiden Bucketregionen ein Rechteck bilden (Abbildung 5.13). Grund: Zur effizienten

Bearbeitung von Range-Queries sind nur rechteckige Gitter sinnvoll!

- Das Mischen wird ausgelöst, wenn ein Bucket höchstens zu 30 % belegt ist und wenn das vereinigte Bucket höchstens zu 70 % belegt sein würde (um erneutes Splitten zu vermeiden)

Abbildung 5.13: Zusammenfassung von Regionen

5.6 Verwaltung geometrischer Objekte

In der bisherigen Anwendung repräsentierten die Datenpunkte im k-dimensionale Raum k-stellige Attributkombinationen. Wir wollen jetzt mithilfe der Datenpunkte geometrische Objekte darstellen und einfache geometrische Anfragen realisieren.

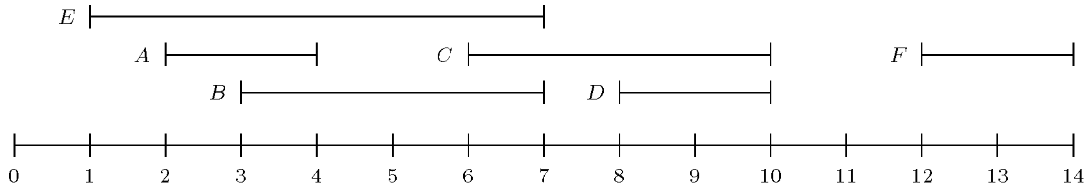

Abbildung 5.14: Intervalle A,B,C,D,E,F über der Zahlengeraden

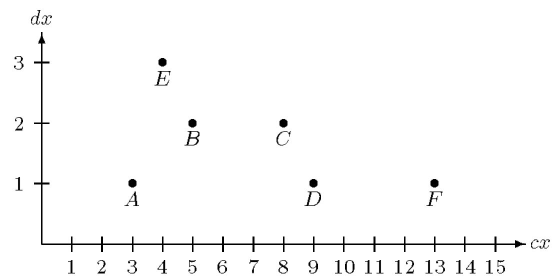

Abbildung 5.15: Repräsentation von Intervallen durch Punkte

Abbildung 5.14 zeigt eine Ansammlung von Intervallen, die zu verwalten seien. Die Intervalle sollen durch Punkte im mehrdimensionalen Raum dargestellt werden. Wenn alle Intervalle durch ihre Anfangs- und Endpunkte repräsentiert würden, kämen sie auf der Datenfläche nur oberhalb der 45-Grad-Geraden zu liegen.

Abbildung 5.15 präsentiert eine wirtschaftlichere Verteilung, indem jede Gerade durch ihren Mittelpunkt und ihre halbe Länge repräsentiert wird.

Typische Queries an die Intervall-Sammlung lauten:

- Gegeben Punkt P, finde alle Intervalle, die ihn enthalten.
- Gegeben Intervall I, finde alle Intervalle, die es schneidet.

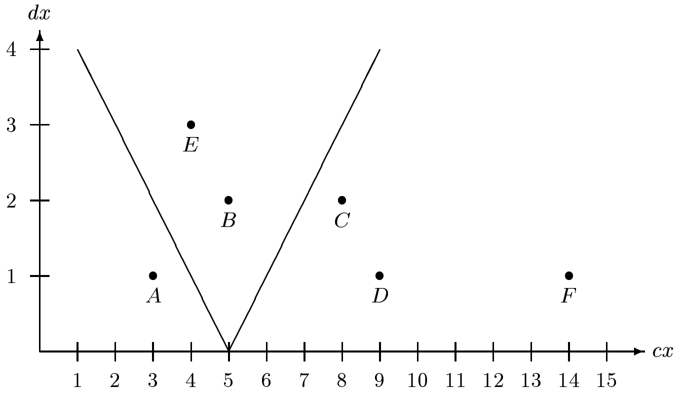

Abbildung 5.16: Abfragekegel zu Punkt p=5

Abbildung 5.16 zeigt den kegelförmigen Abfragebereich zum Query-Punkt p=5, in dem alle Intervalle (repräsentiert durch Punkte) liegen, die den Punkt p enthalten. Grundlage ist die Überlegung, daß ein Punkt P genau dann im Intervall mit Mitte cx und halber Länge dx enhalten ist, wenn gilt: $cx - dx \leq p \leq cx + dx$

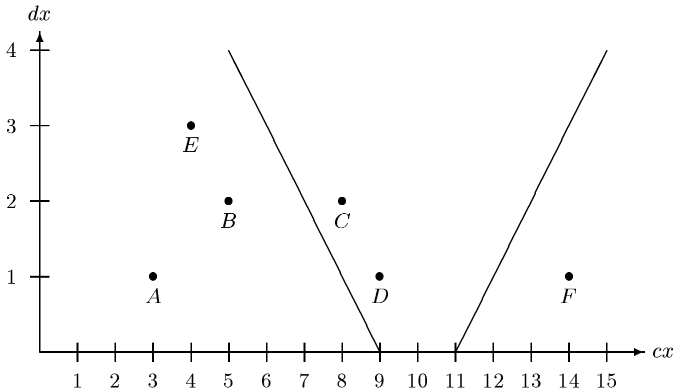

Abbildung 5.17: Abfragekegel zu Intervall mit p=10 und d=1

Abbildung 5.17 zeigt den kegelförmigen Abfragebereich zu dem Query-Intervall mit Mittelpunkt p=10 und halber Länge d=1, in dem alle Intervalle (repäsentiert durch Punkte) liegen, die das Query-Intervall schneiden. Grundlage ist die Überlegung, daß ein Intervall mit Mitte p und halber Länge d genau dann ein Intervall mit Mitte cx und halber Länge dx schneidet, wenn gilt: $cx - dx \leq p + d$ und $cx + dx \geq p - d$

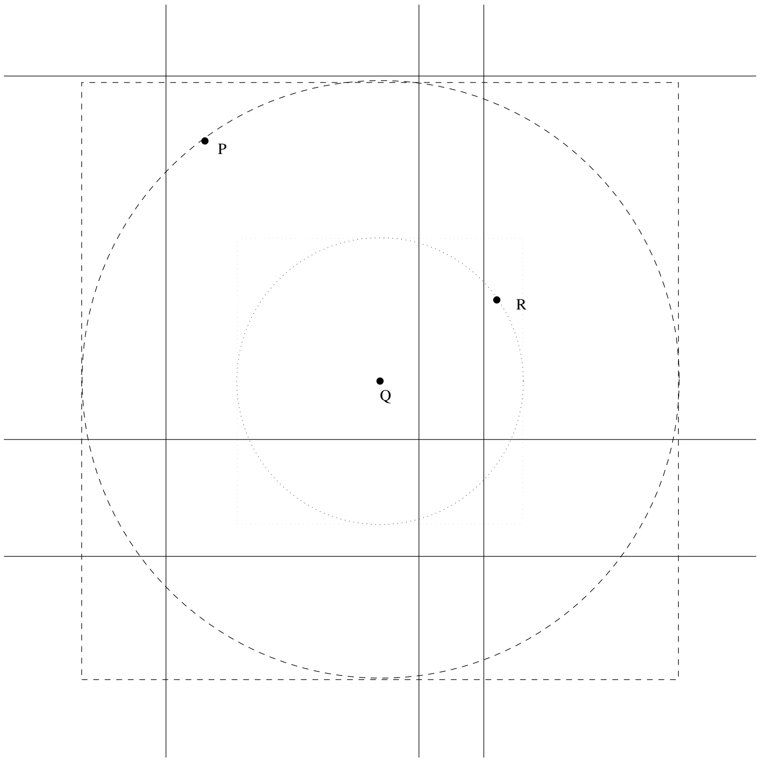

Abbildung 5.18: Nearest-Neighbor-Suche zu Query-Punkt Q

Abbildung 5.18 zeigt die Vorgehensweise bei der Bestimmung des nächstgelegenen Nachbarn (englisch: *nearest neighbor*). Suche zunächst auf dem Datenblock, der für den Query-Point Q zuständig ist, den nächstgelegenen Punkt P. Bilde eine *Range-Query* mit Quadrat um den Kreis um Q mit Radius $|P - Q|$. Schränke Quadratgröße weiter ein, falls nähere Punkte gefunden werden.

Die erwähnten Techniken lassen sich auf höherdimensionierte Geometrie-Objekte wie Rechtecke oder Quader erweitern. Zum Beispiel bietet sich zur Verwaltung von orthogonalen Rechtecken in der Ebene folgende Möglichkeit an: Ein Rechteck wird repräsentiert als ein Punkt im 4-dimensionalen Raum, gebildet durch die beiden 2-dimensionalen Punkte für horizontale bzw. vertikale Lage. Zu einem Query-Rechteck, bestehend aus horizontalem Intervall P und vertikalem Intervall Q, lassen sich die schneidenden Rechtecke finden im Durchschnitt der beiden kegelförmigen Abfragebereiche zu den Intervallen P und Q.

Kapitel 6

Das Relationale Modell

6.1 Definition

Gegeben sind n nicht notwendigerweise unterschiedliche *Wertebereiche* (auch *Domänen* genannt) $D_1, \ldots, D_n$, welche nur *atomare* Werte enthalten, die nicht strukturiert sind, z.B. Zahlen oder Strings.

Eine Relation R ist definiert als Teilmenge des kartesischen Produkts der n Domänen:

$$R \subseteq D_1 \times D_2 \times \ldots \times D_n$$

Es wird unterschieden zwischen dem *Schema* einer Relation, gegeben durch die n Domänen und der aktuellen *Ausprägung* (Instanz). Ein Element der Menge R wird als Tupel bezeichnet, dessen *Stelligkeit* sich aus dem Relationenschema ergibt. Wir bezeichnen mit $\mathbf{sch}(R)$ oder mit $\mathcal{R} = A_1, \ldots,\ A_n$ die Menge der Attribute und mit R die aktuelle Ausprägung. Mit $\mathbf{dom}(A)$ bezeichnen wird die Domäne eines Attributs A. Also gilt

$$R \subseteq \mathrm{dom}(A_1) \times \mathrm{dom}(A_2) \times \ldots \times \mathrm{dom}(A_n)$$

Im Datenbankbereich müssen die Domänen außer einem Typ noch einen Namen haben. Wir werden Relationenschemata daher durch eine Folge von Bezeichner/Wertebereich - Tupeln spezifizieren, z.B.

Telefonbuch : { [Name : string, Adresse: string, <u>TelefonNr : integer</u>] }

Hierbei wird in den eckigen Klammern [...] angegeben, wie die Tupel aufgebaut sind, d.h. welche Attribute vorhanden sind und welchen Wertebereich sie haben. Ein Schlüsselkandidat wird unterstrichen. Die geschweiften Klammern { ... } sollen ausdrücken, daß es sich bei einer Relationenausprägung um eine Menge von Tupeln handelt. Zur Vereinfachung wird der Wertebereich auch manchmal weggelassen:

Telefonbuch : { [Name, Adresse, <u>TelefonNr</u>] }

6.2 Umsetzung in ein relationales Schema

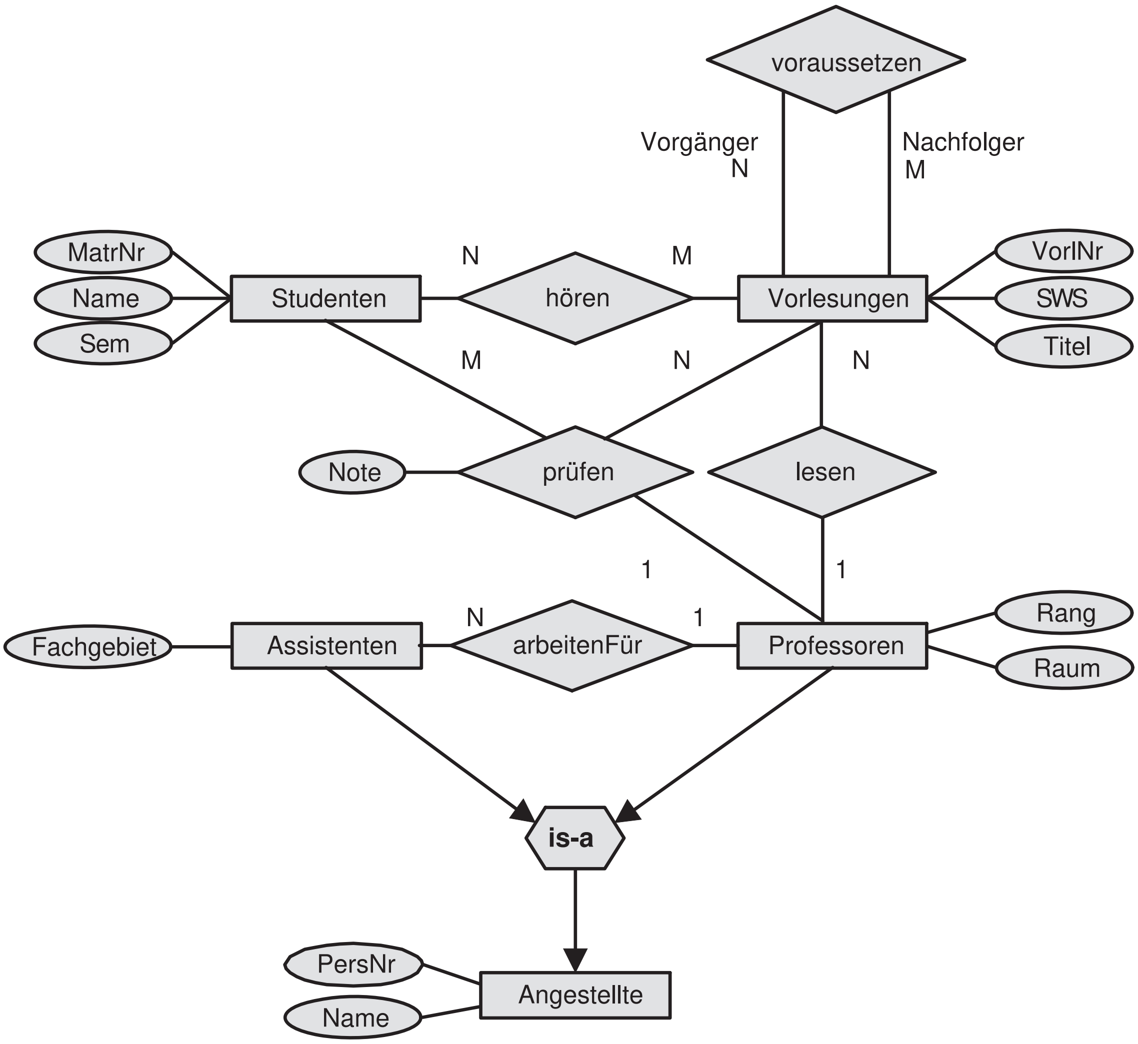

Abbildung 6.1: Konzeptuelles Schema der Universität

Das ER-Modell besitzt zwei grundlegende Strukturierungskonzepte:

- Entity-Typen
- Relationship-Typen

Abbildung 6.1 zeigt ein ER-Diagramm zum Universitätsbetrieb. Zunächst wollen wir die Generalisierung ignorieren, da es im relationalen Modell keine unmittelbare Umsetzung gibt. Dann verbleiben vier Entity-Typen, die auf folgende Schemata abgebildet werden:

$$
\begin{array}{rcl}
\text{Studenten} & : & \{[\ \underline{\text{MatrNr : integer}}, \text{ Name : string, Semester : integer}]\ \} \\
\text{Vorlesungen} & : & \{[\ \underline{\text{VorlNr : integer}}, \text{ Titel : string, SWS : integer}]\ \} \\
\text{Professoren} & : & \{[\ \underline{\text{PersNr : integer}}, \text{ Name : string, Rang : string, Raum : integer}]\ \} \\
\text{Assistenten} & : & \{[\ \underline{\text{PersNr : integer}}, \text{ Name : string, Fachgebiet : string}]\ \}
\end{array}
$$

Bei der relationalen Darstellung von Beziehungen richten wir im *Initial*-Entwurf für jeden Beziehungstyp eine eigene Relation ein. Später kann davon ein Teil wieder eliminiert werden. Grundsätzlich entsteht das Relationenschema durch die Folge aller Schlüssel, die an der Beziehung beteiligt sind sowie ggf. weitere Attribute der Beziehung. Dabei kann es notwendig sein, einige der Attribute umzubenennen. Die Schlüsselattribute für die referierten Entity-Typen nennt man *Fremdschlüssel*.

Für das Universitätsschema entstehen aus den Relationships die folgenden Schemata:

$$
\begin{array}{rcl}
\text{hören} & : & \{[\ \underline{\text{MatrNr : integer}}, \underline{\text{VorlNr : integer}}]\ \} \\
\text{lesen} & : & \{[\ \text{PersNr : integer}, \underline{\text{VorlNr : integer}}]\ \} \\
\text{arbeitenFür} & : & \{[\ \underline{\text{AssiPersNr : integer}}, \text{ ProfPersNr : integer}]\ \} \\
\text{voraussetzen} & : & \{[\ \underline{\text{Vorgänger : integer}}, \underline{\text{Nachfolger : integer}}]\ \} \\
\text{prüfen} & : & \{[\ \underline{\text{MatrNr : integer}}, \underline{\text{VorlNr : integer}}, \text{ PersNr : integer, Note : decimal}]\ \}
\end{array}
$$

Unterstrichen sind jeweils die Schlüssel der Relation, eine *minimale* Menge von Attributen, deren Werte die Tupel eindeutig identifizieren.

Da die Relation *hören* eine $N : M$-Beziehung darstellt, sind sowohl die Vorlesungsnummern als auch die Matrikelnummern alleine keine Schlüssel, wohl aber ihre Kombination.

Bei der Relation *lesen* liegt eine 1:N-Beziehung vor, da jeder Vorlesung genau ein Dozent zugeordnet ist mit der partiellen Funktion

$$lesen : Vorlesungen \rightarrow Professoren$$

Also ist für die Relation *lesen* bereits das Attribut *VorlNr* ein Schlüsselkandidat, für die Relation *arbeitenFür* bildet die *AssiPersNr* einen Schlüssel.

Bei der Relation *prüfen* liegt wiederum eine partielle Abbildung vor:

$$prüfen : Studenten \times Vorlesungen \rightarrow Professoren$$

Sie verlangt, daß *MatrNr* und *VorlNr* zusammen den Schlüssel bilden.

6.3 Verfeinerung des relationalen Schemas

Das im Initialentwurf erzeugte relationale Schema läßt sich verfeinern, indem einige der 1 : 1-, 1 : N- oder N : 1-Beziehungen eliminiert werden. Dabei dürfen nur Relationen mit gleichem Schlüssel zusammengefaßt werden.

Nach dieser Regel können von den drei Relationen

Vorlesungen : {[VorlNr : integer, Titel : string, SWS : integer] }
Professoren : {[PersNr : integer, Name : string, Rang : string, Raum : integer] }
lesen : {[PersNr : integer, VorlNr : integer] }

die Relationen *Vorlesungen* und *lesen* zusammengefaßt werden. Somit verbleiben im Schema

Vorlesungen : {[VorlNr : integer, Titel : string, SWS : integer, gelesenVon : integer] }
Professoren : {[PersNr : integer, Name : string, Rang : string, Raum : integer] }

Das Zusammenlegen von Relationen mit unterschiedlichen Schlüsseln erzeugt eine Redundanz von Teilen der gespeicherten Information. Beispielsweise speichert die (unsinnige) Relation

Professoren' : {[PersNr, liestVorl, Name, Rang, Raum] }

zu jeder von einem Professor gehaltenen Vorlesung seinen Namen, seinen Rang und sein Dienstzimmer:

Professoren´				
PersNrN	liestVorl	Name	Rang	Raum
2125	5041	Sokrates	C4	226
2125	5049	Sokrates	C4	226
2125	4052	Sokrates	C4	226

Bei 1 : 1-Beziehungen gibt es zwei Möglichkeiten, die ursprünglich entworfene Relation mit den beteiligten Entity-Typen zusammenzufassen.

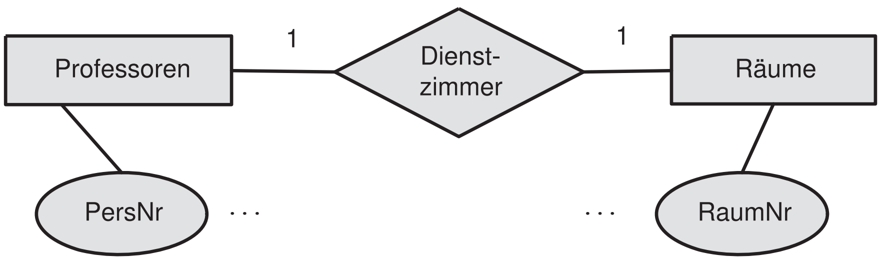

Abbildung 6.2: Beispiel einer 1:1-Beziehung

Abbildung 6.2 zeigt eine mögliche Modellierung für die Unterbringung von Professoren in Räumen als 1 : 1-Beziehung. Die hierzu gehörenden Relationen heißen

Professoren : {[PersNr, Name, Rang] }
Räume : {[RaumNr, Größe, Lage] }
Dienstzimmer : {[PersNr, RaumNr] }

Da *Professoren* und *Dienstzimmer* denselben Schlüssel haben, kann zusammengefaßt werden zu

Professoren : {[PersNr, Name, Rang, Raum] }
Räume : {[RaumNr, Größe, Lage] }

Da das Attribut *RaumNr* innerhalb der Relation *Dienstzimmer* ebenfalls einen Schlüssel bildet, könnten als Alternative auch die Relationen *Dienstzimmer* und *Räume* zusammengefaßt werden:

Professoren : {[PersNr, Name, Rang] }
Räume : {[RaumNr, Größe, Lage, ProfPersNr] }

Diese Modellierung hat allerdings den Nachteil, daß viele Tupel einen sogenannten Nullwert für das Attribut *ProfPersNr* haben, da nur wenige Räume als Dienstzimmer von Professoren genutzt werden.

Die in Abbildung 6.1 gezeigte Generalisierung von *Assistenten* und *Professoren* zu *Angestellte* könnte wie folgt durch drei Relationen dargestellt werden:

Angestellte : {[PersNr, Name] }
Professoren : {[PersNr, Rang, Raum] }
Assistenten : {[PersNr, Fachgebiet] }

Hierdurch wird allerdings die Information zu einem Professor, wie z.B.

[2125, Sokrates, C4, 226]

auf zwei Tupel aufgeteilt:

[2125, Sokrates] und [2125, C4, 226]

Um die vollständige Information zu erhalten, müssen diese beiden Relationen verbunden werden (Join).

Tabelle 6.1 zeigt eine Beispiel-Ausprägung der Universitäts-Datenbasis. Das zugrundeliegende Schema enthält folgende Relationen:

Studenten : {[MatrNr : integer, Name : string, Semester : integer] }
Vorlesungen : {[VorlNr : integer, Titel : string, SWS : integer, gelesenVon : integer] }
Professoren : {[PersNr : integer, Name : string, Rang : string, Raum : integer] }
Assistenten : {[PersNr : integer, Name : string, Fachgebiet : string, Boss : integer] }
hören : {[MatNr : integer, VorlNr : integer] }
voraussetzen : {[Vorgänger : integer, Nachfolger : integer] }
prüfen : {[MatrNr : integer, VorlNr : integer, PersNr : integer, Note :decimal] }

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Studenten		
MatrNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12
26120	Fichte	10
26830	Aristoxenos	8
27550	Schopenhauer	6
28106	Carnap	3
29120	Theophrastos	2
29555	Feuerbach	2

Vorlesungen			
VorlNr	Titel	SWS	gelesenVon
5001	Grundzüge	4	2137
5041	Ethik	4	2125
5043	Erkenntnistheorie	3	2126
5049	Mäeutik	2	2125
4052	Logik	4	2125
5052	Wissenschaftstheorie	3	2126
5216	Bioethik	2	2126
5259	Der Wiener Kreis	2	2133
5022	Glaube und Wissen	2	2134
4630	Die 3 Kritiken	4	2137

voraussetzen	
Vorgänger	Nachfolger
5001	5041
5001	5043
5001	5049
5041	5216
5043	5052
5041	5052
5052	5259

hören	
MatrNr	VorlNr
26120	5001
27550	5001
27550	4052
28106	5041
28106	5052
28106	5216
28106	5259
29120	5001
29120	5041
29120	5049
29555	5022
25403	5022
29555	5001

Assistenten			
PersNr	Name	Fachgebiet	Boss
3002	Platon	Ideenlehre	2125
3003	Aristoteles	Syllogistik	2125
3004	Wittgenstein	Sprachtheorie	2126
3005	Rhetikus	Planetenbewegung	2127
3006	Newton	Keplersche Gesetze	2127
3007	Spinoza	Gott und Natur	2134

prüfen			
MatrNr	VorlNr	PersNr	Note
28106	5001	2126	1
25403	5041	2125	2
27550	4630	2137	2

Abbildung 6.3: Beispielausprägung der Universitäts-Datenbank

Zur Modellierung von schwachen Entity-Typen betrachten wir Abbildung 6.4, in der mittels der Relation *liegt_in* der schwache Entitity-Typ *Räume* dem Entity-Typ *Gebäude* untergeordnet wurde.

Wegen der $1 : N$-Beziehung zwischen *Gebäude* und *Räume* kann die Beziehung *liegt_in* verlagert werden in die Relation Räume:

$$\text{Räume} : \{[\ \underline{\text{GebNr}, \text{RaumNr}}, \text{Größe}]\ \}$$

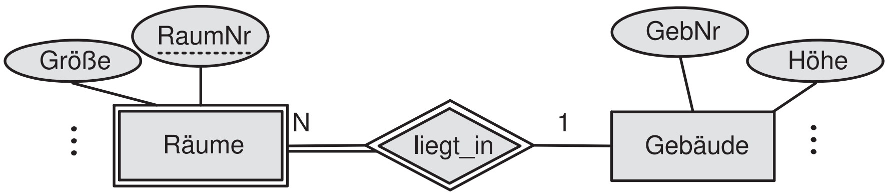

Abbildung 6.4: Schwacher Entity-Typ

Ein Beziehung *bewohnt* zwischen *Professoren* und *Räume* benötigt als Fremdschlüssel zum einen die Personalnummer des Professors und zum anderen die Kombination von Gebäude-Nummer und Raum-Nummer:

$$\text{bewohnt} : \{[\,\underline{\text{PersNr, GebNr, RaumNr}}\,]\}$$

Da *bewohnt* eine 1 : 1-Beziehung darstellt, kann sie durch einen Fremdschlüssel beim Professor realisiert werden. Ist die beim *Gebäude* hinterlegte Information eher gering, käme auch, wie im Universitätsschema in Abbildung 6.1 gezeigt, ein Attribut *Raum* bei den *Professoren* infrage.

6.4 Abfragesprachen

Es gibt verschiedene Konzepte für formale Sprachen zur Formulierung einer Anfrage (Query) an ein relationales Datenbanksystem:

- Relationenalgebra (prozedural):
  Verknüpft konstruktiv die vorhandenen Relationen durch Operatoren wie $\cup, \cap, \ldots$:
- Relationenkalkül (deklarativ):
  Beschreibt Eigenschaften des gewünschten Ergebnisses mit Hilfe einer Formel der Prädikatenlogik 1. Stufe unter Verwendung von $\wedge, \vee, \neg, \exists, \forall$.
- SQL (kommerziell):
  Stellt eine in Umgangssprache gegossene Mischung aus Relationenalgebra und Relationenkalkül dar.
- Query by Example (für Analphabeten):
  Verlangt vom Anwender das Ausfüllen eines Gerüstes mit Beispiel-Einträgen.

6.5 Relationenalgebra

Die Operanden der Sprache sind Relationen. Als unabhängige Operatoren gibt es *Selektion, Projektion, Vereinigung, Mengendifferenz, Kartesisches Produkt, Umbenennung*; daraus lassen sich weitere Operatoren *Verbund, Durchschnitt, Division* ableiten.

Selektion :

Es werden diejenigen Tupel ausgewählt, die das *Selektionsprädikat* erfüllen. Die Selektion wird mit σ bezeichnet und hat das Selektionsprädikat als Subskript.

Die Projektion

$$\sigma_{Semester>10}(Studenten)$$

liefert als Ergebnis

$\sigma_{Semester>10}(Studenten)$		
MatNr	Name	Semester
24002	Xenokrates	18
25403	Jonas	12

Das Selektionsprädikat wird beschrieben durch eine Formel F mit folgenden Bestandteilen:

- Attributnamen der Argumentrelation R oder Konstanten als Operanden
- arithmetische Vergleichsoperatoren $< \; = \; > \; \leq \; \neq \; \geq$
- logische Operatoren: $\wedge \; \vee \; \neg$ (und oder nicht)

Projektion :

Bei der Projektion werden Spalten der Argumentrelation extrahiert. Das Operatorsymbol lautet Π, die gewünschten Attribute werden im Subskript aufgelistet:

$$\Pi_{Rang}(Professoren)$$

liefert als Ergebnis

$\Pi_{Rang}(Professoren)$
Rang
C4
C3

Die Attributmenge wird üblicherweise nicht unter Verwendung von Mengenklammern sondern als durch Kommata getrennte Sequenz gegeben. Achtung: Da das Ergebnis wiederum eine Relation ist, existieren per definitionem keine Duplikate ! In der Praxis müssen sie dennoch algorithmisch entfernt werden.

Vereinigung :

Zwei Relationen mit gleichem Schema können durch die Vereinigung, symbolisiert durch $\cup$, zusammengefaßt werden. Beispiel:

$$\Pi_{PersNr,Name}(Assistenten) \cup \Pi_{PersNr,Name}(Professoren)$$

Mengendifferenz :

Für zwei Relationen R und S mit gleichem Schema ist die Mengendifferenz $R - S$ definiert als die Menge der Tupel, die in R aber nicht in S vorkommen. Beispiel

$$\Pi_{MatrNr}(Studenten) \; - \; \Pi_{MatrNr}(prüfen)$$

liefert die Matrikelnummern derjenigen Studenten, die noch nicht geprüft wurden.

Kartesisches Produkt :

Das kartesische Produkt (Kreuzprodukt) zweier Relationen R und S wird mit $R \times S$ bezeichnet und enthält alle möglichen Paare von Tupeln aus R und S. Das Schema der Ergebnisrelation, also $\mathbf{sch}(R \times S)$, ist die Vereinigung der Attribute aus $\mathbf{sch}(R)$ und $\mathbf{sch}(S)$.

Das Kreuzprodukt von *Professoren* und *hören* hat 6 Attribute und enthält 91 $(= 7 \cdot 13)$ Tupel.

Professoren × hören					
Professoren				hören	
PersNr	name	Rang	Raum	MatNr	VorlNr
2125	Sokrates	C4	226	26120	5001
...	...	...	...	...	...
2125	Sokrates	C4	226	29555	5001
...	...	...	...	...	...
2137	Kant	C4	7	29555	5001

Haben beide Argumentrelationen ein gleichnamiges Attribut A, so kann dies durch Voranstellung des Relationennamen R in der Form $R.A$ identifiziert werden.

Umbenennung von Relationen und Attributen :

Zum Umbenennen von Relationen und Attributen wird der Operator ρ verwendet, wobei im Subskript entweder der neue Relationenname steht oder die Kombination von neuen und altem Attributnamen durch einen Linkspfeil getrennt. Beispiele:

$$\rho_{Dozenten}(Professoren)$$

$$\rho_{Zimmer \leftarrow Raum}(Professoren)$$

Eine Umbenennung kann dann erforderlich werden, wenn durch das kartesische Produkt Relationen mit identischen Attributnamen kombiniert werden sollen.

Als Beispiel betrachten wir das Problem, die Vorgänger der Vorgänger der Vorlesung mit der Nummer 5216 herausfinden. Hierzu ist ein kartesisches Produkt der Tabelle mit sich selbst erforderlich, nachdem zuvor die Spalten umbenannt worden sind:

$$\Pi_{V1.Vorgänger}(\sigma_{V2.Nachfolger=5216 \wedge V1.Nachfolger=V2.Vorgänger}$$

$$(\rho_{V1}(voraussetzen) \times \rho_{V2}(voraussetzen)))$$

Die konstruierte Tabelle hat vier Spalten und enthält das Lösungstupel mit dem Wert 5001 als Vorgänger von 5041, welches wiederum der Vorgänger von 5216 ist:

V1		V2	
Vorgänger	Nachfolger	Vorgänger	Nachfolger
5001	5041	5001	5041
...	...	...	...
5001	5041	5041	5216
...	...	...	...
5052	5259	5052	5259

Natürlicher Verbund (Join) :

Der sogenannte *natürliche Verbund* zweier Relationen R und S wird mit $R \bowtie S$ gebildet. Wenn R insgesamt $m+k$ Attribute $A_1, \ldots, A_m, B_1, \ldots, B_k$ und S insgesamt $n+k$ Attribute $B_1, \ldots, B_k, C_1, \ldots, C_n$ hat, dann hat $R \bowtie S$ die Stelligkeit $m+k+n$. Hierbei wird vorausgesetzt, daß die Attribute A_i und C_j jeweils paarweise verschieden sind. Das Ergebnis von $R \bowtie S$ ist definiert als

$$R \bowtie S \;:=\; \Pi_{A_1,\ldots,A_m,R.B_1,\ldots,R.B_k,C_1,\ldots,C_n}(\sigma_{R.B_1=S.B_1 \wedge \ldots \wedge R.B_k=S.B_k}(R \times S))$$

Es wird also das kartesische Produkt gebildet, aus dem nur diejenigen Tupel selektiert werden, deren Attributwerte für gleichbenannte Attribute der beiden Argumentrelationen gleich sind. Diese gleichbenannten Attribute werden in das Ergebnis nur einmal übernommen.

Die Verknüpfung der *Studenten* mit ihren *Vorlesungen* geschieht durch

$$(Studenten \bowtie h\ddot{o}ren) \bowtie Vorlesungen$$

Das Ergebnis ist eine 7-stellige Relation:

$(Studenten \bowtie h\ddot{o}ren) \bowtie Vorlesungen$						
MatrNr	Name	Semester	VorlNr	Titel	SWS	gelesenVon
26120	Fichte	10	5001	Grundzüge	4	2137
25403	Jonas	12	5022	Glaube und Wissen	2	2137
28106	Carnap	3	4052	Wissenschaftstheorie	3	2126
...	...	...	...	...	...	...

Da der Join-Operator assoziativ ist, können wir auch auf die Klammerung verzichten und einfach schreiben

$$Studenten \bowtie h\ddot{o}ren \bowtie Vorlesungen$$

Wenn zwei Relationen verbunden werden sollen bzgl. zweier Attribute, die zwar die gleiche Bedeutung aber unterschiedliche Benennungen haben, so müssen diese vor dem Join mit dem ρ-Operator umbenannt werden. Zum Beispiel liefert

$$Vorlesungen \bowtie \rho_{gelesenVon \leftarrow PersNr}(Professoren)$$

die Relation $\{[VorlNr, Titel, SWS, gelesenVon, Name, Rang, Raum]\}$

Allgemeiner Join :

Beim natürlichen Verbund müssen die Werte der gleichbenannten Attribute übereinstimmen. Der allgemeine Join-Operator, auch *Theta-Join* genannt, erlaubt die Spezifikation eines beliebigen Join-Prädikats θ. Ein Theta-Join $R \bowtie_\theta S$ über der Relation R mit den Attributen $A_1, A_2, \ldots, A_n$ und der Relation S mit den Attributen $B_1, B_2, \ldots, B_n$ verlangt die Einhaltung des Prädikats θ, beispielsweise in der Form

$$R \bowtie_{A_1 < B_1 \wedge A_2 = B_2 \wedge A_3 < B_5} S$$

Das Ergebnis ist eine $n + m$-stellige Relation und läßt sich auch als Kombination von Kreuzprodukt und Selektion schreiben:

$$R \bowtie_\theta S := \sigma_\theta(R \times S)$$

Wenn in der Universitätsdatenbank die *Professoren* und die *Assistenten* um das Attribut *Gehalt* erweitert würden, so könnten wir diejenigen Professoren ermitteln, deren zugeordnete Assistenten mehr als sie selbst verdienen:

$$Professoren \bowtie_{Professoren.Gehalt < Assistenten.Gehalt \wedge Boss = Professoren.PersNr} Assistenten$$

Die bisher genannten Join-Operatoren werden auch innere Joins genannt (*inner join*). Bei ihnen gehen diejenigen Tupel der Argumentrelationen verloren, die keinen Join-Partner gefunden haben. Bei den äußeren Join-Operatoren (*outer joins*) werden - je nach Typ des Joins - auch partnerlose Tupel gerettet:

- left outer join: Die Tupel der linken Argumentrelation bleiben erhalten
- right outer join: Die Tupel der rechten Argumentrelation bleiben erhalten
- full outer join: Die Tupel beider Argumentrelationen bleiben erhalten

Somit lassen sich zu zwei Relationen L und R insgesamt vier verschiedene Joins konstruieren:

L		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2

R		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2

inner Join				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1

left outer join				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	-	-

right outer Join				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
-	-	c_3	d_2	e_2

outer Join				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_2	-	-
-	-	c_3	d_2	e_2

Mengendurchschnitt :

Der Mengendurchschnitt (Operatorsymbol $\cap$) ist anwendbar auf zwei Argumentrelationen mit gleichem Schema. Im Ergebnis liegen alle Tupel, die in beiden Argumentrelationen liegen. Beispiel:

$$\Pi_{PersNr}(\rho_{PersNr \leftarrow gelesenVon}(Vorlesungen)) \cap \Pi_{PersNr}(\sigma_{Rang=C4}(Professoren))$$

liefert die Personalnummer aller C4-Professoren, die mindestens eine Vorlesung halten.

Der Mengendurchschnitt läßt sich mithilfe der Mengendifferenz bilden:

$$R \cap S \;=\; R \;-\; (R \;-\; S)$$

Division Sei R eine r-stellige Relation, sei S eine s-stellige Relation, deren Attributmenge in R enthalten ist.

Mit der Division

$$Q \;:=\; R \;\div\; S \;:=\; \{t = t_1, t_2, \ldots, t_{r-s} \mid \forall U \in S \;:\; tu \in R\}$$

sind alle die Anfangsstücke von R gemeint, zu denen sämtliche Verlängerungen mit Tupeln aus S in der Relation R liegen. Beispiel:

$$\begin{array}{|c|c|} \hline \multicolumn{2}{|c|}{H} \\ \hline M & V \\ \hline m_1 & v_1 \\ m_1 & v_2 \\ m_1 & v_3 \\ m_2 & v_2 \\ m_2 & v_3 \\ \hline \end{array} \;\div\; \begin{array}{|c|} \hline L \\ \hline V \\ \hline v_1 \\ v_2 \\ \hline \end{array} \;=\; \begin{array}{|c|} \hline H \div L \\ \hline M \\ \hline m_1 \\ \hline \end{array}$$

Die Division von R durch S läßt sich schrittweise mithilfe der unabhängigen Operatoren nachvollziehen (zur Vereinfachung werden hier die Attribute statt über ihre Namen über ihren Index projiziert):

$T := \pi_{1,\ldots,r-s}(R)$	alle Anfangsstücke
$T \times S$	diese kombiniert mit allen Verlängerungen aus S
$(T \times S) \backslash R$	davon nur solche, die nicht in R sind
$V := \pi_{1,\ldots,r-s}((T \times S) \backslash R)$	davon die Anfangsstücke
$T \backslash V$	davon das Komplement

6.6 Relationenkalkül

Ausdrücke in der Relationenalgebra spezifizieren konstruktiv, wie das Ergebnis der Anfrage zu berechnen ist. Demgegenüber ist der *Relationenkalkül* stärker *deklarativ* orientiert. Er beruht auf dem mathematischen Prädikatenkalkül erster Stufe, der quantifizierte Variablen zuläßt. Es gibt zwei unterschiedliche, aber gleichmächtige Ausprägungen des Relationenkalküls:

- Der relationale Tupelkalkül
- Der relationale Domänenkalkül

6.7 Der relationale Tupelkalkül

Im *relationen Tupelkalkül* hat eine Anfrage die generische Form

$$\{t \mid P(t)\}$$

wobei t eine sogenannte Tupelvariable ist und P ist ein Prädikat, das erfüllt sein muß, damit t in das Ergebnis aufgenommen werden kann. Das Prädikat P wird formuliert unter Verwendung von $\vee, \wedge, \neg, \forall, \exists, \Rightarrow$.

Alle C4-Professoren:

$$\{p \mid p \in Professoren \wedge p.Rang = 'C4'\}$$

Alle Professorennamen zusammen mit den Personalnummern ihrer Assistenten:

$$\{p.Name, a.PersNr \mid p \in Professoren \wedge a \in Assistenten \wedge p.PersNr = a.Boss\}$$

Alle diejenigen Studenten, die sämtliche vierstündigen Vorlesungen gehört haben:

$$\{s \mid s \in Studenten \wedge \forall v \in Vorlesungen(v.SWS = 4 \Rightarrow$$

$$\exists h \in hören(h.VorlNr = v.VorlNr \wedge h.MatrNr = s.MatrNr))\}$$

Für die Äquivalenz des Tupelkalküls mit der Relationenalgebra ist es wichtig, sich auf sogenannte *sichere* Ausdrücke zu beschränken, d.h. Ausdrücke, deren Ergebnis wiederum eine Teilmenge der Domäne ist. Zum Beispiel ist der Ausdruck

$$\{n \mid \neg(n \in Professoren)\}$$

nicht sicher, da er unendlich viele Tupel enthält, die nicht in der Relation *Professoren* enthalten sind.

6.8 Der relationale Domänenkalkül

Im *relationalen Domänenkalkül* werden Variable nicht an Tupel, sondern an Domänen, d.h. Wertemengen von Attributen, gebunden. Eine Anfrage hat folgende generische Struktur:

$$\{[v_1, v_2, \ldots, v_n] \mid P(v_1, v_2, \ldots, v_n)\}$$

Hierbei sind die v_i Domänenvariablen, die einen Attributwert repräsentieren. P ist eine Formel der Prädikatenlogik 1. Stufe mit den freien Variablen $v_1, v_2, \ldots, v_n$.

Join-Bedingungen können implizit durch die Verwendung derselben Domänenvariable spezifiziert werden. Beispiel:

Alle Professorennamen zusammen mit den Personalnummern ihrer Assistenten:

$$\{[n, a] \mid \exists p, r, t([p, n, r, t]) \in Professoren$$

$$\wedge \, \exists v, w([a, v, w, p] \in Assistenten))\}$$

Wegen des Existenz- und Allquantors ist die Definition des *sicheren Ausdruckes* etwas aufwendiger als beim Tupelkalkül. Da sich diese Quantoren beim Tupelkalkül immer auf Tupel einer vorhandenen Relation bezogen, war automatisch sichergestellt, daß das Ergebnis eine endliche Menge war.

Kapitel 14

Sicherheit

In diesem Kapitel geht es um den Schutz gegen absichtliche Beschädigung oder Enthüllung von sensiblen Daten. Abbildung 14.1 zeigt die hierarchische Kapselung verschiedenster Maßnahmen.

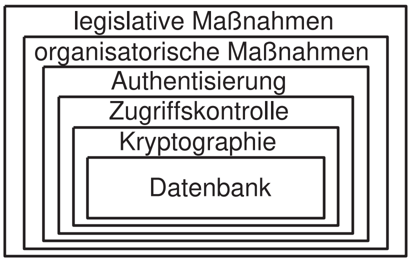

Abbildung 14.1: Ebenen des Datenschutzes

14.1 Legislative Maßnahmen

Im *Gesetz zum Schutz vor Mißbrauch personenbezogener Daten bei der Datenverarbeitung* ist festgelegt, welche Daten in welchem Umfang schutzbedürftig sind.

14.2 Organisatorische Maßnahmen

Darunter fallen Maßnahmen, um den persönlichen Zugang zum Computer zu regeln:

- bauliche Maßnahmen
- Pförtner
- Ausweiskontrolle
- Diebstahlsicherung

Verkäufe					
VerkDatum	Filiale	Produkt	Anzahl	Kunde	Verkäufer
30-Jul-96	Passau	1347	1	4711	825
...	...	...	...	...	...

Filialen			
Filialenkennung	Land	Bezirk	...
Passau	D	Bayern	...
...	...	...	...

Kunden			
KundenNr	Name	wiealt	...
4711	Kemper	38	...
...	...	...	...

Verkäufer					
VerkäuferNr	Name	Fachgebiet	Manager	wiealt	...
825	Handyman	Elektronik	119	23	...
...	...	...	...	...	...

Zeit								
Datum	Tag	Monat	Jahr	Quartal	KW	Wochentag	Saison	...
...	...	...	...	...	...	...	...	...
30-Jul-96	30	Juli	1996	3	31	Dienstag	Hochsommer	...
...	...	...	...	...	...	...	...	...
23-Dec-97	27	Dezember	1997	4	52	Dienstag	Weihnachten	...
...	...	...	...	...	...	...	...	...

Produkte					
ProduktNr	Produkttyp	Produktgruppe	Produkthauptgruppe	Hersteller	...
1347	Handy	Mobiltelekom	Telekom	Siemens	...
...	...	...	...	...	...

Abbildung 16.3: Ausprägung des Sternschemas in einem Handelsunternehmen

tabelle *Zeit* vielleicht 1.000 Einträge (für die letzen drei Jahre) aufweist. Abbildung 16.3 zeigt eine mögliche Ausprägung.

Die Dimensionstabellen sind in der Regel nicht normalisiert. Zum Beispiel gelten in der Tabelle *Produkte* folgende funktionale Abhängigkeiten: $ProduktNr \rightarrow Produkttyp$, $Produkttyp \rightarrow Produktgruppe$ und $Produktgruppe \rightarrow Produkthauptgruppe$. In der *Zeit*-Dimension lassen sich alle Attribute aus dem Schlüsselattribut *Datum* ableiten. Trotzdem ist die explizite Speicherung dieser Dimension sinnvoll, da Abfragen nach Verkäufen in bestimmten Quartalen oder an bestimmten Wochentagen dadurch effizienter durchgeführt werden können.

Die Verletzung der Normalformen in den Dimensionstabellen ist bei Decision-Support-Systemen nicht so gravierend, da die Daten nur selten verändert werden und da der durch die Redundanz verursachte erhöhte Speicherbedarf bei den relativ kleinen Dimensionstabellen im Vergleich zu der großen (normalisierten) Faktentabelle nicht so sehr ins Gewicht fällt.

16.2 Star Join

Das Sternschema führt bei typischen Abfragen zu sogenannten *Star Joins*:

> Welche Handys (d.h. von welchen Herstellern) haben junge Kunden in den bayrischen Filialen zu Weihnachten 1996 gekauft ?

```
select sum(v.Anzahl), p.Hersteller
from  Verkäufe v, Filialen f, Produkte p, Zeit z, Kunden k
where z.Saison = 'Weihnachten' and z.Jahr = 1996 and k.wiealt < 30
and   p.Produkttyp = 'Handy' and f.Bezirk = 'Bayern'
and   v.VerkDatum = z.Datum and v.Produkt = p.ProduktNr
and   v.Filiale = f.Filialenkennung and v.Kunde = k.KundenNr
group by Hersteller;
```

16.3 Roll-Up/Drill-Down-Anfragen

Der Verdichtungsgrad bei einer SQL-Anfrage wird durch die **group by**-Klausel gesteuert. Werden mehr Attribute in die **group by**-Klausel aufgenommen, spricht man von einem *drill down*. Werden weniger Attribute in die **group by**-Klausel aufgenommen, spricht man von einem *roll up*.

> Wieviel Handys wurden von welchem Hersteller in welchem Jahr verkauft ?

```
select Hersteller, Jahr, sum(Anzahl)
from   Verkäufe v, Produkte p, Zeit z
where  v.Produkt = p.ProduktNr
and    v.VerkDatum = z.Datum
and    p.Produkttyp = 'Handy'
group  by p.Hersteller, z.Jahr;
```

Das Ergebnis wird in der linken Tabelle von Abbildung 16.4 gezeigt. In der Tabelle rechts oben bzw. rechts unten finden sich zwei Verdichtungen.

Durch das Weglassen der Herstellerangabe aus der **group by**-Klausel (und der **select**-Klausel) entsteht ein *roll up* entlang der Dimension *p.Hersteller*:

> Wieviel Handys wurden in welchem Jahr verkauft ?

```
select Jahr, sum(Anzahl)
from   Verkäufe v, Produkte p, Zeit z
where  v.Produkt = p.ProduktNr
and    v.VerkDatum = z.Datum
and    p.Produkttyp = 'Handy'
group  by z.Jahr;
```