Gunter Saake
Kai-Uwe Sattler
Andreas Heuer

Datenbanken

Konzepte und Sprachen

Sechste Auflage

Dieses pdf-Kapitel ist eine kostenlose Ergänzung zum oben genannten Buch, das 2018 bei MITP erschienen ist.

Bitte beachten: Verweise auf Umgebungen B-X beziehen sich auf Teile innerhalb dieses Download-Kapitels. Verweise, die mit Kapitelnummern beginnen, wie 11-31, beziehen sich auf Umgebungen innerhalb des obigen Lehrbuches. Verweise auf Seiten vor 738 beziehen sich ebenfalls auf das zugrundeliegende Lehrbuch.

Literaturhinweise in diesen pdf-Kapiteln beziehen sich auf ein erweitertes Literaturverzeichnis zum Lehrbuch, das auch als kostenlose Ergänzung an derselben Stelle zum Download bereitsteht.

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Bei der Herstellung des Werkes haben wir uns zukunftsbewusst für umweltverträgliche und wiederverwertbare Materialen entschieden. Der Inhalt ist auf elementar chlorfreiem Papier gedruckt.

ISBN 978-3-95845-776-8
6. Auflage 2018

www.mitp.de
E-Mail: mitp-verlag@sigloch.de

Telefon: +49 7953/7189-079
Telefax: +49 7953/7189-082

Lektorat: Sabine Janatschek, Ernst-Heinrich Pröfener
Sprachkorrektorat: Jürgen Dubau, Astrid Langen
Covergestaltung: Christian Kalkert, www.kalkert.de
Bildnachweis Cover: iStock.com/Shawn Hempel | fotolia.com/karpenko_ilia
Satz: Gunter Saake, Magdeburg; Kai-Uwe Sattler, Ilmenau; Andreas Heuer, Rostock
Druck: Westermann Druck Zwickau GmbH

B.1 Historische Modelle

Das Netzwerkmodell und das hierarchische Datenmodell werden auch als Datenbankmodelle der ersten Generation bezeichnet. Sie sind echte Datenbankmodelle im Sinne unserer Begriffsbildung und sind weitgehend unabhängig vom Typsystem einer bestimmten Programmiersprache. Im Vergleich zu modernen Datenbankmodellen wie dem relationalen haben sie allerdings erhebliche Schwächen im Bereich der Datenunabhängigkeit und der Abstraktion von konkreten Speicherstrukturen.

Wir werden im Folgenden kurz beide Modelle beschreiben, um danach auf den Entwurf von Datenbeschreibungen in diesen Modellen ausgehend vom ER-Modell einzugehen. Abschließend skizzieren wir die Anwendungsprogrammierung in derartigen Modellen.

B.1.1 Das Netzwerkmodell

Das *Netzwerkmodell* wurde 1971 von dem Normungsausschuss CODASYL-DBTG festgelegt und ist auch als CODASYL-Datenbankmodell bekannt. Vereinfacht ausgedrückt entspricht das Netzwerkmodell dem ER-Modell mit einigen Einschränkungen: Beziehungstypen sind ausschließlich zweistellig und funktional („binär und 1:n“), und Beziehungstypen haben keine Attribute. Diese Einschränkungen waren unter anderem motiviert durch eine günstigere Implementierung. Genauer betrachtet weicht die Semantik insofern vom ER-Modell ab, dass im Netzwerkmodell durchgängig eine *Listensemantik* anstelle einer Mengensemantik verwendet wird.

Tabelle B.1 stellt die Konzepte und Begriffe des ER-Modells den Entsprechungen im Netzwerkmodell gegenüber. Sofern zutreffend, werden auch die entsprechenden Konzepte des Relationenmodells den neuen Begriffen gegenübergestellt.

ER-Modell	**Relationenmodell**	**Netzwerkmodell**
Entity	Tupel	Logical Record
Entity-Typ	Relationenschema	Record-Typ
Attribut	Attribut	Feld
binärer 1:n-Beziehungstyp	Relationenschema oder Fremdschlüsselattribut	Link oder auch *Set-Typ*

Tabelle B.1: Begriffe des Netzwerkmodells

B.1.1.1 Netzwerkschema

Ein *Netzwerkschema* ist ein gerichteter Graph mit der Menge der Record-Typen als Knoten und den Set-Typen als Kanten. Die Kantenmenge wird dadurch bestimmt, dass (E_1, E_2) eine Kante im Netzwerkschema definiert, falls E_1 und E_2 in einer n:1-Beziehung stehen. Die gerichtete Kante geht in Richtung der Funktion der funktionalen Beziehung: Am Pfeilende steht der Record aus E_1, der mit mehreren anderen aus E_2 in Beziehung steht. *Vorsicht:* Die Pfeilrichtung wird von verschiedenen Autoren unterschiedlich gehandhabt!

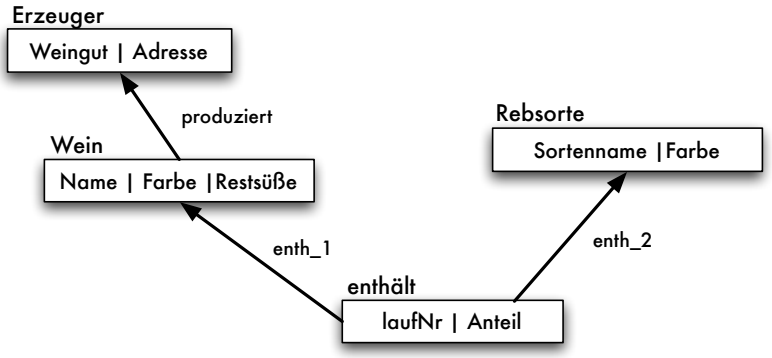

Abbildung B.1: Beispielschema im Netzwerkmodell

Abbildung B.1 zeigt eine kleine Beispielmodellierung im Netzwerkmodell. Das Beispiel zeigt, dass eine n:m-Beziehung des ER-Modells, hier die Beziehung `enthält` zwischen `Wein` und `Rebsorte`, in einen neuen Record-Typ plus zwei Set-Typen `enth_1` und `enth_2` aufgelöst werden muss. Die funktionale Beziehung `produziert` hingegen kann direkt übernommen werden.

Abbildung B.2 zeigt eine Datenausprägung zu dem Beispielschema in Abbildung B.1. Gezeigt werden zwei Erzeuger, drei Weine sowie drei Rebsorten. Die `enthält`-Beziehung ist durch drei Records dargestellt. Die verschiedenen Set-Typen sind durch unterschiedliche Pfeiltypen (durchgezogen, gestrichelt, gepunktet) unterschieden. Beispielsweise sind für das Weingut Helena zwei Weine gespeichert (ringförmige Verkettung mit durchgezogener Linie), und einer dieser Weine (Red Dream) enthält zwei Rebensorten (gestrichelte Linie). Nicht alle Set-Ausprägungen sind dargestellt, um das Bild übersichtlich zu halten.

Für eine konkrete Set-Ausprägung spricht man von dem *Besitzer*, engl. *Owner*, und den *Teilnehmern*, engl. *Members*. Da es sich um eine funktionale Beziehung handelt, gibt es jeweils genau einen Owner und beliebig viele (hier ist auch die Anzahl 0 enthalten!) Members für eine konkrete Set-Ausprägung.

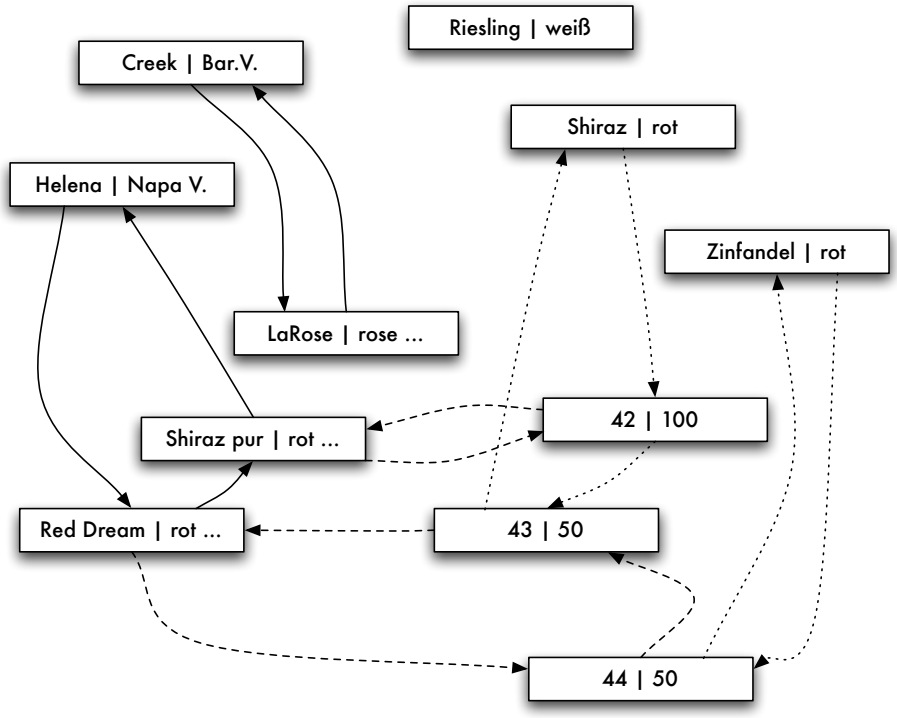

Abbildung B.2: Netzwerk von Datensätzen als Beispielausprägung im Netzwerkmodell

Um über alle Datensätze eines Record-Typs iterieren zu können, wird für jeden Record-Typ eine spezielle Set-Ausprägung angelegt, bei der der Owner als `System` bezeichnet wird, und in der alle Datensätze als Members fungieren.

Dem Konzept der Set-Ausprägung liegt eine stark implementierungsnahe Betrachtungsweise zugrunde. Im Prinzip wird physisch für jeden Record ein Zeigerfeld für die Verkettung mit dem Owner `System` vorgesehen sowie ein weiteres Feld für jeden Set-Typ, an dem der Record-Typ beteiligt ist. Eine Set-Ausprägung wird durch eine ringförmig verkettete Liste über diese Felder realisiert.

B.1.1.2 Simulation einer allgemeinen Beziehung

Das Netzwerkmodell kennt nur zweistellige funktionale Beziehungen. Wird eine Datenbank mittels des ER-Modells entworfen, stellt sich nun die Frage, wie beliebige k-stellige Beziehungen des ER-Modells auf derartige funktionale Beziehungen abgebildet werden können.

Eine k-stellige Beziehung kann wie folgt aufgelöst werden: Gegeben sei eine Relationship R zwischen den Entity-Typen $E_1, \ldots, E_k$. Zusätzlich zu den Record-Typen $T_1, \ldots, T_k$ wird ein neuer Record-Typ TR mit nur einem Attribut erzeugt. Dieser neue Record-Typ wird auch als *Kett-Record-Typ* bezeichnet. Falls R selber Attribute hatte, werden diese ebenfalls T zugeordnet. Danach erzeuge man Links $L_1, \ldots, L_k$ jeweils von TR nach $T_1, \ldots, T_k$. Abbildung B.3 verdeutlicht diese Vorgehensweise.

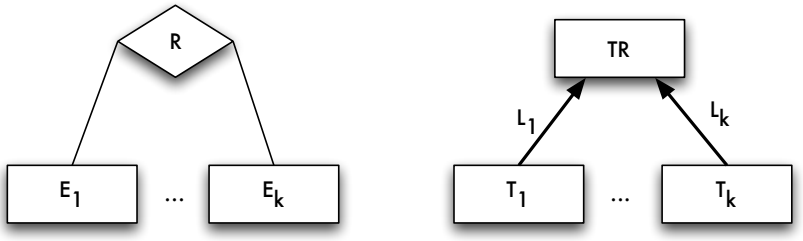

Abbildung B.3: Abbildung einer k-stelligen Beziehung des ER-Modells im Netzwerkmodell

B.1.2 Das hierarchische Modell

Das *hierarchische Datenbankmodell* wurde von IBM 1969 mit dem System IMS eingeführt und ist das kommerziell erfolgreichste Datenbankmodell der ersten Generation. Noch auf Jahrzehnte hinaus werden große Datenbestände, deren Fundamente in den 70er Jahren gelegt wurden, hierarchisch organisiert sein.

Eine *Hierarchie* ist ein Netzwerkschema, das ein Wald ist („Menge von Bäumen"). Die Links zeigen jeweils vom Nachfolger zum Vorgänger. Eine Hierarchie kann natürlich keine allgemeinen Beziehungen darstellen, so dass sogenannte „Virtual Records" (Zeiger) eingeführt werden müssen, um die Baumstruktur zu durchbrechen. Abbildung B.4 zeigt die Umsetzung des Netzwerkschemas aus Abb. B.1 in ein derartiges hierarchisches Schema.

Die Datenbankausprägungen im hierarchischen Datenmodell können als hierarchisch aufgebaute Dateien aufgefasst werden oder alternativ als Bäume, in denen die Söhne jeweils sequentiell verzeigert sind. Abbildung B.5 zeigt im Überblick eine derartige Speicherstruktur. Die oberste Ebene wird wie im Netzwerkmodell sequentiell verkettet. Im Beispiel hat die oberste Ebene zwei Unterhierarchien, wobei die zweite wiederum unterstrukturiert ist.

Für die Darstellung in Abbildung B.5 wurde eine Verkettung analog zum Netzwerkmodell gewählt. Die interne Realisierung folgt nicht unbedingt dieser

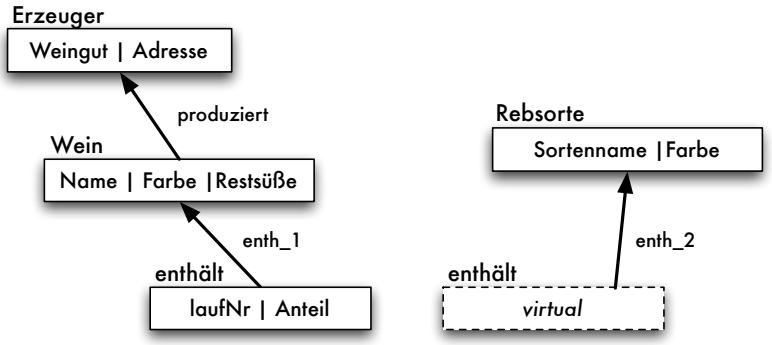

Abbildung B.4: Umsetzung des Beispielschemas in das hierarchische Datenmodell

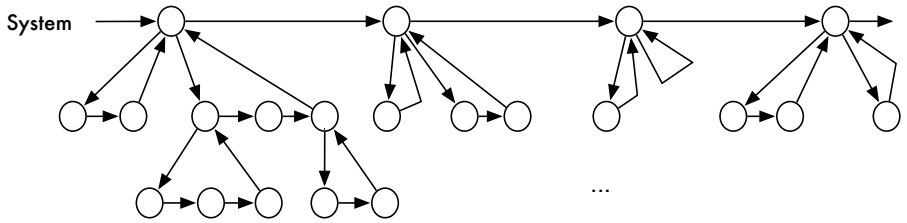

Abbildung B.5: Skizzierung der Speicherstrukturen im hierarchischen Datenmodell

Darstellung, sondern kann auch auf sequentieller Abspeicherung in einer Datei beruhen.

Die Operationen in Hierarchien sind noch einfacher als im Netzwerkmodell: Die wesentliche Operation ist der Durchlauf durch einen Baum *ausschließlich* von oben nach unten bzw. bei Söhnen oder Töchtern einer Ebene von links nach rechts. Diese Operationen lassen sich sehr effizient implementieren.

B.1.3 ER-Abbildung auf das Netzwerkmodell

Ein Entity-Typ wird auf einen Record-Typ mit allen Attributen als Feldern abgebildet. Ein Beziehungstyp kann auf einen Link oder Set-Typ des Netzwerkmodells nur in folgendem Spezialfall abgebildet werden:

- die Beziehung ist eine 1:n-Beziehung,
- die Beziehung ist binär,

ER-Konzept	wird abgebildet auf Netzwerkkonzept
Entity-Typ E_i	Record-Typ R_i
Attribute von E_i	Felder von R_i
Beziehungstyp	Set-Typ
dessen Attribute	–; evtl. Kett-Record-Typ oder bei Original-Record-Typ
1:n	Standard-Set-Typ
1:1	Standard-Set-Typ; Zusatzbedingung nicht darstellbar
m:n	Kett-Record-Typ
IST-Beziehung	Standard-Set-Typ; Zusatzbedingung nicht darstellbar

Bezeichnungen:
E_1, E_2: an Beziehung beteiligte Entity-Typen

Tabelle B.2: Abbildung eines ER-Schemas auf ein Netzwerkschema

- die Beziehung hat keine eigenen Attribute.

Ist die Beziehung eine m:n-Beziehung oder ist sie nicht binär, so ist ein Ausweg über die bereits eingeführten Kett-Record-Typen möglich. 1:1-Beziehungen kann man wie 1:n-Beziehungen auf einen Set-Typ abbilden, muss jedoch die Zusatzbedingung im Datenbankanwendungsprogramm überwachen.

IST-Beziehungen werden ebenfalls wie 1:n-Beziehungen behandelt, die zwei Zusatzbedingungen (es sind eigentlich gleiche Entities gemeint, die Beziehung ist eigentlich 1:1) müssen ebenfalls im Programm überwacht werden.

Hat eine Beziehung Attribute, so gibt es zu ihrer Darstellung folgende Möglichkeiten:

- Im Fall einer m:n-Beziehung können die Attribute dem neu eingeführten Kett-Record-Typ zugeordnet werden.

- Bei einer 1:n-Beziehung (also einer Owner-Member-Beziehung) kann das Attribut der Beziehung beim Record-Typ der Member-Seite zusätzlich aufgenommen werden. Bei optionalen Beziehungen hat dies den Nachteil, dass das Attribut nicht sinnvoll belegt werden kann.

  Nehmen wir beispielsweise das zusätzliche Attribut `Produktionsbeginn` in die `produziert`-Beziehung des Beispiels mit auf, so kann dieses im Netzwerkmodell im Record-Typ des zugehörigen Member-Typs, dem `Wein`, aufgenommen werden.

Eine Übersicht über die Abbildungen gibt Tabelle B.2.

Als Beispiel für die Umsetzung einer mehrstelligen Beziehung mit einem Kett-Record-Typ betrachten wir die mehrstellige Beziehung `Empfiehlt`. Das Netzwerkschema dazu ist in Abbildung B.6 abgebildet.

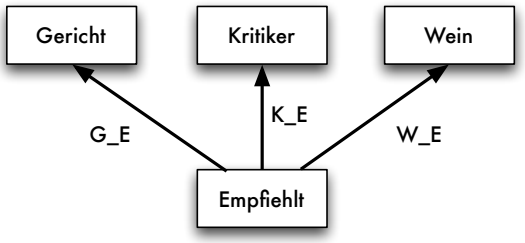

Abbildung B.6: Die dreistellige `Empfiehlt`*-Beziehung als Netzwerkschema*

B.1.4 ER-Abbildung auf das hierarchische Modell

Da das hierarchische Modell eine Einschränkung des Netzwerkmodells ist, können wir hierarchische Schemata am besten aus einem Netzwerkschema heraus entwickeln. Aus dem Netzwerk müssen wir also eine Menge von Bäumen ableiten.

Die Links im Netzwerkmodell zeigen von den Nachfolgern (den Member-Typen) zu den Vorgängern (den Owner-Typen). Als Wurzel eines Schemabaumes im hierarchischen Modell eignen sich nun die Knoten im Netzwerk, die nicht mehr Member eines anderen Owner-Typs sind, also die Knoten, die keine auslaufenden Kanten mehr haben. Diese Knoten werden jeweils Wurzeln der Schemabäume, die Nachfolger bleiben die Nachfolger im Netzwerkschema.

◂**Beispiel B-1**▸ Wir starten beim Netzwerkschema für eine m:n-Beziehung `WirdAngebaut` zwischen `Rebsorten` und `Anbaugebieten` mit einer Abbildung in das Netzwerkmodell. Wurzeln der Schemabäume werden die Knoten ohne auslaufende Kanten, also `Rebsorte` und `Anbaugebiet`. Das vorläufige Schema ist in Abbildung B.7 aufgeführt. □

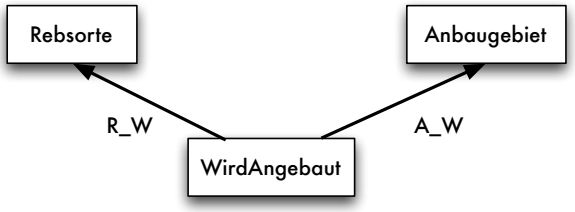

Abbildung B.7: Vorläufiges Schema für die Transformation in das hierarchische Modell

Dieser vorläufige Entwurf muss nun in zwei Punkten noch verbessert werden:

- Knoten mit mehr als einer auslaufenden Kante müssen derzeit mehreren Bäumen zugeordnet werden und zerstören deshalb die Baumstruktur. Sie werden durch virtuelle Record-Typen getrennt, um die Baumstruktur zu erreichen.
- Gerade m:n-Beziehungen können nun nachträglich noch nachgebessert und optimiert werden, indem zwei wechselseitige virtuelle Record-Typen eingeführt werden.

◂**Beispiel B-2**▸ Beim vorläufigen Schema aus Abbildung B.7 muss nun ein virtueller `WirdAngebaut`-Knoten eingeführt werden. Hinter diesem virtuellen Record-Typ verbergen sich Zeiger auf den `WirdAngebaut`-Record-Typ. Das resultierende Schema aus Abbildung B.8 ist nun ein echtes hierarchisches Schema.

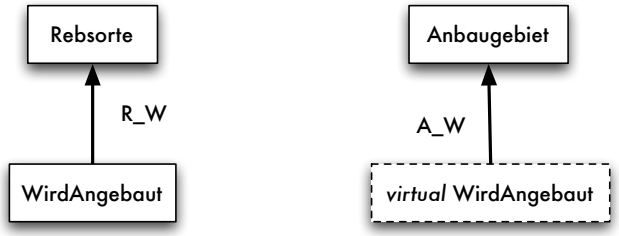

Abbildung B.8: Hierarchisches Schema durch Einführung virtueller Record-Typen

Der Zugriff ist nun aber nicht symmetrisch. Daher wird bei einer m:n-Beziehung besser beiden Seiten als Nachfolger ein virtueller Record-Typ zugeordnet, der jeweils auf die Wurzel der anderen Seite dieser m:n-Beziehung zeigt. Die optimierte hierarchische Darstellung ist in Abbildung B.9 enthalten. □

B.1.5 Anwendungsprogrammierung in den historischen Modellen

Als Vertreter der navigierenden Ansätze betrachten wir die DML des Netzwerkmodells, die im CODASYL-Normungsvorschlag festgelegt worden ist, sowie die entsprechenden Sprachkonstrukte für das hierarchische Datenmodell. Auch für

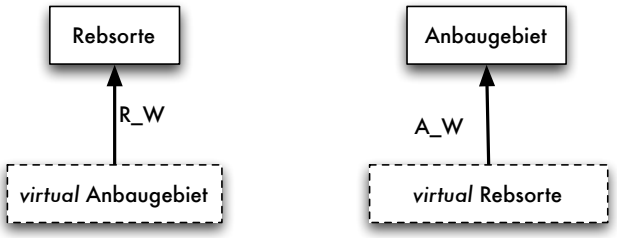

Abbildung B.9: Für m:n-Beziehungen optimiertes hierarchisches Schema

andere Datenmodelle wurden verwandte Sprachvorschläge gemacht, die sich in den prinzipiellen Konstrukten nicht wesentlich unterscheiden.

B.1.5.1 Datenmanipulation im Netzwerkmodell

Das Netzwerkmodell bietet im Gegensatz zum Relationenmodell keine separate Sprache wie SQL an, die bei der Anwendungsprogrammierung mit einer Programmiersprache verbunden werden müsste. Stattdessen werden alle Programme in einer sogenannten Wirtssprache, auch Host-Sprache, geschrieben (im CODASYL-Vorschlag die Sprache COBOL), die um spezielle DML-Kommandos erweitert wird. Es handelt sich insbesondere um die Kommandos **`find`** zum Suchen und Positionieren, **`get`** zum Transfer eines Datensatzes in das Anwendungsprogramm und **`store`** für den umgekehrten Transfer.

Zu jedem Anwendungsprogramm gehört eine sogenannte *User Working Area*, kurz UWA, die unter anderem Positionszeiger auf Datensätze im Datennetzwerk beinhaltet. Die UWA ist in Abbildung B.10 skizziert.

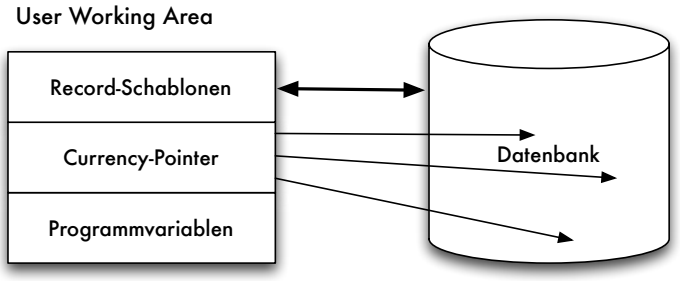

Abbildung B.10: User Working Area (UWA) im Netzwerkmodell

In der UWA sind neben den Positionszeigern vor allem die sogenannten Record-Schablonen definiert, die den Datenaustauschbereich bei **store**- und **get**-Kommandos bilden. Die erwähnten Positionszeiger werden als *Currency Pointer* bezeichnet und sind Zeiger auf Records im Datennetzwerk, die vom DBS automatisch verwaltet werden. Definiert sind jeweils die folgenden Zeiger:

- *Current of run-unit:* Letzter Record, auf den im Programm zugegriffen wurde.
- *Current of record type:* Für jeden Record-Typ T wird auf den zuletzt zugegriffenen Record mit **current of** T verwiesen.
- *Current of set type:* Für jeden Set-Typ S wird auf den zuletzt zugegriffenen Record (Owner oder Member) mit **current of** S verwiesen.

Navigation in der Datenbank

Das Lesen eines Records aus der Datenbank in die Record-Schablone erfolgt in zwei Stufen: Mittels des **find**-Befehls wird der gesuchte Record lokalisiert und zum **current of run-unit**. Das **get**-Kommando kopiert den unter **current of run-unit** stehenden Record in die passende Record-Schablone der UWA, von wo die Daten durch das Anwendungsprogramm gelesen werden können.

Der Zugriff kann dabei über einen sogenannten Datenbankschlüssel, der tatsächlich die physische Adresse eines Records in der Datenbank ist, oder über einen berechneten Schlüssel, den **calc**-Schlüssel, erfolgen. Dieser wiederum definiert einen Hashwert eines Records über ein beliebiges Feld (oder auch mehrere Felder).

In den Suchanweisungen werden konzeptuelle Zugriffe mit den Zugriffen über interne Datenstrukturen vermischt, wie auch im gesamten Netzwerkmodell die Trennung der Ebenen nicht sehr ausgeprägt ist. Folgerichtig gibt es eine ganze Reihe unterschiedlicher Formen der **find**-Anweisung. Wir präsentieren hier nicht alle in der Netzwerk-DDL angebotenen Sprachkonstrukte, sondern beschränken uns auf ausgewählte Konstrukte, um einen Eindruck von der Sprache zu geben.

1. Für einen gegebenen Datenbankschlüssel kann der zugehörige Record direkt mit der folgenden Anweisung gefunden werden:

   find *x* **record by database key** *y*

2. Eine analoge Anweisung existiert für einen vorgegebenen **calc**-Schlüssel, etwa einen aus mehreren Attributwerten berechneten Hashwert. Der Hashwert wird aus den entsprechenden Feldern der Record-Schablone berechnet, die vorher geeignet mit Daten besetzt werden müssen:

```
find x record by calc-key
```

3. Die vorige Anweisung findet genau einen (abhängig von der Speicherreihenfolge) Eintrag für den gegebenen berechneten Schlüssel. Die folgende Anweisung bestimmt für einen gegebenen **calc**-Schlüssel alle Records dazu nacheinander in der durch die Speicherung vorgegebenen Reihenfolge.

   ```
   find duplicate x record by calc-key
   ```

4. Mit den folgenden Anweisungen ist es möglich, alle Elemente (Members) einer Set Occurrence zu durchlaufen:

   ```
   find owner of current x set;
   find next y record in current x set
   ```

   Mittels der Angabe **owner is system** in der Datendefinition ist ein sequentieller Durchlauf über alle Elemente eines Record-Typs möglich.

5. In einer Set-Ausprägung können alle Members gefunden werden, die spezielle Werte in bestimmten Feldern aufweisen.

6. Ferner ist es möglich, den Owner eines Records bezüglich eines gegebenen Sets zu finden.

7. Weiterhin ist es möglich, den **current of** *T* für einen Record-Typ *T* (ebenfalls für Set-Typ *S*) zu bestimmen und ihn zum **current of run-unit** zu machen.

Diese Suchoperationen bilden ebenfalls die Grundlage für die Datenmanipulation, indem sie die *Currency Pointer* setzen.

Datenmanipulation

Entsprechend den Basisoperationen der Datenmanipulation im Relationenmodell werden auch im Netzwerkmodell Operationen zum Einfügen, Löschen und Modifizieren von Datenbankelementen angeboten. Im Gegensatz zum Relationenmodell ist im Netzwerkmodell die Reihenfolge der Speicherung relevant; den Operationen müssen somit entsprechende Parameterangaben mitgegeben werden.

Die erwähnten drei Basisoperationen zur Datenbankmodifikation werden im Netzwerkmodell wie folgt bezeichnet:

- Das Einfügen wird als **store** bezeichnet. Die **store**-Operation gibt es für Record-Typen wie auch für Set-Ausprägungen. Ein **store** für einen Record-Typ ist ein Transfer der Wertebelegung der entsprechenden Record-Schablone vom Anwendungsprogramm in das Datenbanknetzwerk.

- Das Löschen wird für Record-Typen als **delete** bezeichnet. Das Herausnehmen aus einer Set-Ausprägung wird als **remove** bezeichnet.

- Das Ändern von Attributen erfolgt mittels **modify**.

All diese Operationen werden jeweils bezüglich des aktuellen *current of run-unit* ausgeführt. Bei expliziter Forderung kontrolliert das Datenbankmanagementsystem auch Existenzbedingungen, die den Fremdschlüsselbedingungen des Relationenmodells entsprechen.

Beim Einfügen in eine Set-Ausprägung wird die zu wählende Set-Ausprägung des neuen Elements wie folgt bestimmt; wurde in der Set-Deklaration die Angabe **insertion is automatic** angegeben, bestehen zwei Möglichkeiten:

- Das Einfügen erfolgt an der Position des aktuellen *Currency Pointers* des Set-Typs:

  set selection is thru current of *x* **set**

- Der Owner der Set-Ausprägung wird anhand eines berechneten **calc**-Schlüssels bestimmt:

  set selection is thru owner
  using *Feldliste für* **calc**-*Schlüssel*

Erfolgte hingegen die Angabe **insertion is manual**, muss die Einfügeposition wie folgt durch die *Currency Pointer* explizit bestimmt werden:

insert *x* **into** *y*

Hierbei ist *x* ein Record-Typ und *y* ein Set-Typ.

Beim Löschen eines Records mittels der **delete**-Operation müssen die von dem betreffenden Record abhängigen Set-Ausprägungen berücksichtigt werden, falls der betreffende Record Owner von nicht-leeren Set-Ausprägungen ist. Bei der Angabe

delete *x*

wird die Löschung verweigert, falls *x* eine nicht-leere Set-Ausprägung besitzt. Die Angabe

delete *x* **all**

hingegen führt zur Löschung aller Set-Ausprägungen (auch rekursiv). Dies entspricht dem kaskadierenden Löschen bei referentieller Integrität in relationalen Datenbanken.

B.1.5.2 Datenmanipulation im hierarchischen Modell

Konzeptionell ist die Anwendungsprogrammierung im hierarchischen Datenmodell sehr ähnlich zum Vorgehen im Netzwerkmodell, nur dass die Navigation in der Datenbank eng an die hierarchische Baumstruktur gekoppelt ist.

Die am weitesten verbreitete Manipulationssprache für das hierarchische Datenmodell ist die Sprache *DL/I* des IMS-Systems. Wie beim CODASYL-Ansatz für das Netzwerkmodell gibt es in DL/I die Konzepte der *User Working Area*, *Record-Schablonen*, *Current Record* für jeden Record-Typ sowie einen *Current Parent* für Records innerhalb der Baumstruktur. Wir stellen nicht die Original-DL/I-Syntax vor, die auf Prozeduraufrufen mit Parametern zur Operationskennung basiert, sondern präsentieren eine an den CODASYL-Vorschlag angelehnte Notation.

Das **get**-Kommando ermöglicht das Navigieren innerhalb der hierarchisch angeordneten Datensätze. Die folgenden Notationen erlauben den Durchlauf durch die Hierarchieebenen:

```
(1) get unique x [ where Bedingungen ];
(2) get next x [ where Bedingungen ];
(3) get next within parent;
```

Die Variante `(1)` greift direkt auf einen Datensatz zu, indem ein eindeutiger Pfad von der Wurzel bis zum Datensatz angegeben wird. Die Auswahl auf einer Ebene kann etwa durch ein identifizierendes Attribut erfolgen. Die Variante `(2)` navigiert durch die Hierarchie der Baumstruktur folgend von links nach rechts. Abbildung B.11 zeigt eine Hierarchie von Datensätzen.

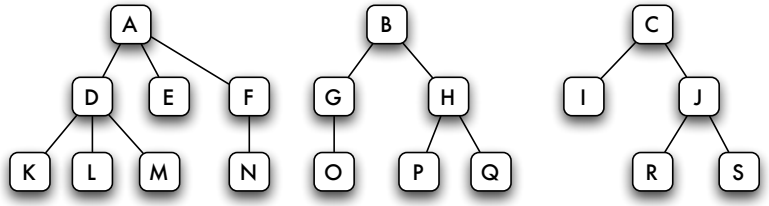

Abbildung B.11: Baumstruktur im hierarchischen Datenmodell

Die Abarbeitungsreihenfolge mit **get next** für die Hierarchie in Abbildung B.11 ist in Abbildung B.12 angegeben.

Diese Variante `(2)` ist nicht auf eine Hierarchieebene beschränkt; das Erreichen des jeweils letzten Sohnes eines Knotens kann allerdings explizit abgefragt werden. Die Variante `(3)` kürzt dies ab, indem das Weitersetzen auf die Söhne eines Knotens beschränkt bleibt.

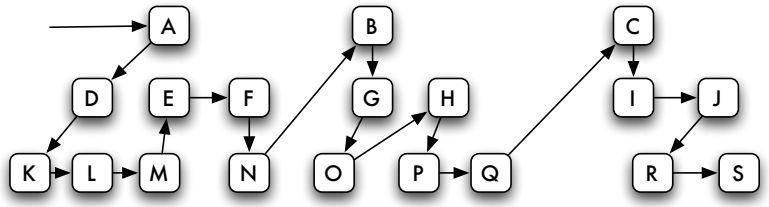

Abbildung B.12: Entsprechende Abarbeitungsreihenfolge mit dem **`get next`**-*Kommando*

Beim Einfügen mittels **`insert`** x wird der Inhalt der Record-Schablone zum Typ x Nachfolger des *Current Records*, der vom passenden Parent-Typ sein muss. Die Einfügereihenfolge wird durch die Optionen **`leftmost`**, **`rightmost`** bzw. **`order by`** festgelegt. Der jeweilige *Current Record* kann über eine **`where`**-Klausel spezifiziert werden.

Für das Löschen und Ändern mittels **`delete`** und **`replace`** wird zunächst mit **`get hold`** (anstelle eines normalen **`get`** im obigen Sinne) der aktuelle *Current Record* bestimmt und für die Änderung gesperrt. Das Sperren von Datenbankelementen verhindert Probleme beim synchronen Ändern im Mehrbenutzerbetrieb. Für Einzelheiten verweisen wir auf die entsprechenden Kapitel von [SSH11]. Ansonsten entsprechen die Operationen den korrespondierenden Anweisungen im Netzwerkmodell.

B.1.6 Zusammenfassung

Vor dem Siegeszug der relationalen Datenbanken waren Datenbankmodelle verbreitet, die stark an Speicherungstechniken angelehnt waren und somit die Datenunabhängigkeit wenig unterstützten. Das Netzwerkmodell speichert Daten als (ringförmige) verkettete Datensätze, während das hierarchische Datenmodell an hierarchisch aufgebauten Dateiformaten angelehnt ist. Beschreibungen in diesen Datenmodellen können aus ER-Schemata abgeleitet werden. Die Nutzung derartiger Datenbanken kennt keine deskriptiven Anfragesprachen sondern nutzt stattdessen operationale Programmierung (Verfolgung von Zeigern bzw. Iterieren mit hierarchischem Cursor).

Eine Übersicht über die in diesem Kapitel eingeführten Begriffe und deren Bedeutung geben wir in Tabelle B.3.

Begriff	Informale Bedeutung
Netzwerkdatenmodell	Datenbank als verzeigerte Daten-Records; nur funktionale Beziehungstypen
Kett-Record	künstlicher Record-Typ zur Realisierung von beliebigen Beziehungstypen
hierarchisches Datenmodell	Datenbank als Menge hierarchisch aufgebauter Dateien

Tabelle B.3: Wichtige Begriffe bei historischen Datenmodellen

B.1.7 Vertiefende Literatur

Die navigierenden DMLs des hierarchischen und Netzwerkmodells werden in älteren Datenbankbüchern, etwa in [Dat90], noch ausführlich beschrieben.

B.1.8 Übungsaufgaben

Übung B-1 Modellieren Sie die Wein-Anwendung im Netzwerk- und im hierarchischen Modell. □

Übung B-2 Gesucht sind alle Rosé-Weine im Anbaugebiet Hessen aus maximal zwei Rebsorten. Skizzieren Sie ein CODASYL-Programm in Pseudocode, das diese Anfrage beantwortet. □

Übung B-3 Löschen Sie den Wein Red Dream aus der Netzwerkdatenbank. □