3.3.4 Prozedurale Programmierung und SQL

Während in den letzten Abschnitten die grundlegenden Eigenschaften von SQL erläutert wurden, soll hier noch eine interessante Erweiterung vorgestellt werden. Hierbei handelt es sich um die prozeduralen bzw. imperativen Sprachelemente, welche sich in vielen kommerziell verfügbaren SQL-Dialekten befinden. Im Einzelnen werden folgende Themen behandelt:

- Prozeduren und Funktionen
- Trigger
- Cursor

Während die bis jetzt behandelten Spracheigenschaften im Wesentlichen in allen kommerziell verfügbaren SQL-Dialekten zu finden sind, ist bei den prozeduralen Elementen weder im Umfang noch in der Syntax eine Einheitlichkeit gegeben. Deswegen wird die Diskussion ab jetzt auf den SQL-Dialekt PL/SQL von Oracle eingeschränkt. PL/SQL zeichnet sich durch seine zahlreichen Erweiterungen in der imperativen Programmierung aus. Die Syntax lehnt sich an die Programmiersprache ADA an. Selbst fortgeschrittene Konzepte wie Module oder Ausnahmen sind in PL/SQL realisiert worden.

Prozeduren und Funktionen („Stored Procedures“)

In der Datenbank gespeicherte Prozeduren bieten die Möglichkeit, dem Anwender neben den generischen Operationen zum Einfügen, Ändern und Löschen an das Anwendungsgebiet angepasste Operationen zur Verfügung zu stellen. Welche Vorzüge besitzen Prozeduren gegenüber einer Folge von generischen SQL-Anweisungen?

- Prozeduren können bei mehrmaliger Verwendung einmal definiert werden und dann wiederholt aufgerufen werden. Prozeduren in PL/SQL können Ein- und Ausgabeparameter benutzen.

- Bedingt durch die Trennung der Definition und Benutzung einer Prozedur kann eine Datenbank eine Prozedur vorverarbeiten, indem sie die Definition analysiert und einen Zugriffsplan bestimmt. Im Abschnitt über Optimierung von Abfragen (Abschnitt 3.4) wird der Begriff eines Zugriffsplans ausführlich behandelt.

- Die Trennung von Definition und Benutzung bietet einen weiteren Vorteil: Die Definition der Prozedur wird als Bestandteil der Datenbank verwaltet. Daher muss beim Aufruf einer Prozedur nur der Prozeduraufruf selbst über das Netzwerk übertragen werden, da die Definition der Prozedur der Datenbank schon vorliegt. Somit kann die Benutzung von Prozeduren und Funktionen auch eine Reduktion der Übertragungszeit und eine Ersparnis von Bandbreite bewirken.

- Wenn man den Zugriff auf die Daten über generische Funktionen verbietet, aber den Aufruf über bestimmte Prozeduren und Funktionen erlaubt, so kann man erreichen, dass der Anwender nur bestimmte Änderungen im Datenbestand durchführen kann. Auch ist es dann möglich, weitere Konsistenzprü-

fungen in den Prozeduren zu definieren, die dann garantiert bei jeder Datenänderung beachtet werden.

Wann werden nun Prozeduren eingesetzt? Zum einen werden Prozeduren dann eingesetzt, wenn die generischen Operationen (UPDATE, INSERT) nur in Kombination mit weiteren Befehlen aufgerufen werden sollen. Zum zweiten können die Prozeduren bei wiederkehrender Benutzung eine Beschleunigung bewirken, da bei der Definition einer Prozedur das DBMS diese schon vorverarbeiten kann, und dann später bei der Ausführung auf dieser Vorverarbeitung aufbauen kann.

Syntax von Prozeduren
Prozeduren besitzen die folgende Form:

```
CREATE PROCEDURE <Prozedurname>(<Parameterliste>)
IS
BEGIN
  <Anweisungen>
END
```

Die Definition von Funktionen (Prozeduren mit einem ausgezeichneten Rückgabeparameter) ist analog und sei hier auch kurz wiedergegeben:

```
CREATE FUNCTION <Funktionsname>(<Parameterliste>)
RETURN <Typ des Rückgabewertes>
IS
BEGIN
  <Anweisungen>
END
```

Beispiel
Falls bei einem Angestellten bei einer Zuordnung zu einem Projekt auch gleichzeitig die Vergütung geändert werden soll, so bietet die folgende Prozedur die Möglichkeit, diese Aktionen zu einer Einheit zusammenzufassen.

```
CREATE PROCEDURE AssignToProject
   (Angestellter IN INTEGER,
     Projekt IN INTEGER,
     proz_Arbeit IN INTEGER,
    Gehaltsaufschlag IN INTEGER)

IS

BEGIN

  INSERT INTO ANG_PRO
    VALUES (Projekt, Angestellter, proz_Arbeit);

  UPDATE ANGEST
    SET Gehalt = Gehalt + Gehaltsaufschlag
    WHERE ANGNR = Angestellter;

END;
```

Trigger

Trigger besitzen eine große Ähnlichkeit zu parameterlosen Prozeduren. Ebenso wie diese stellen sie eine Folge von Anweisungen dar, die als Einheit ausgeführt werden. Im Gegensatz zu einer Prozedur wird ein Trigger aber nicht explizit aufgerufen, sondern in der Definition des Triggers wird zusätzlich noch angegeben, bei welchen Ereignissen der Trigger ausgeführt werden soll. Dies sind vor allem Änderungen am Datenbestand, die über eine Insert, Update oder Delete Anweisung angestoßen wurden. Hierbei kann man in PL/SQL neben der Art der Anweisung, welche die Änderung hervorgerufen hat (INSERT, UPDATE oder DELETE), auch festlegen, ob der Trigger vor oder nach der eigentlichen Operation ausgeführt werden soll. Zusätzlich lässt sich definieren, ob der Trigger nur einmal pro Operation oder einmal pro geänderter Zeile ausgeführt werden soll.

Syntax von Trigger

```
CREATE TRIGGER <Triggername> [BEFORE|AFTER]
  [INSERT, UPDATE, DELETE] ON <Tabellenname>
  [FOR EACH ROW]
BEGIN
   <Anweisungen>
END;
```

Cursor

Cursor sind ein Hilfsmittel, um bei der imperativen Programmierung, welche tupel-orientiert ist, einen Zugriff auf die Ergebnisse der SQL-Queries, welche mengenorientiert sind, zu erhalten (siehe auch Abschnitt 3.3.3). Zunächst wird der Cursor definiert, dann in der OPEN-Anweisung geöffnet, und anschließend werden die Elemente der Ergebnismenge in einer Folge von FETCH-Anweisungen durchlaufen und bearbeitet. Mit dem FOUND-Attribut des Cursors kann man feststellen, ob das FETCH noch ein weiteres Element geliefert hat oder ob die Ergebnismenge bereits vollständig durchlaufen wurde.

Beispiel

Der folgende Trigger prüft, ob bei den Einfügungen in der Tabelle ANG_PRO berücksichtigt wurde, dass der prozentuale Anteil an der Arbeitszeit zwischen 0 und 100 Prozent liegt. Zusätzlich prüft der Trigger, ob auch die Summe der Arbeitszeit eines jeden Angestellten in diesem Bereich liegt.

Die Raise-Anweisung in dem Beispiel bietet die Möglichkeit, eine Ausnahme zu werfen (sie ist somit vergleichbar zu der throw Anweisung in der Programmiersprache Java).

```
CREATE TRIGGER ANGPROCHECK BEFORE INSERT ON
        ANG_PRO

FOR EACH ROW

DECLARE

  CURSOR Arbeitszeit_Cursor IS
    SELECT SUM(PROZ_ARB)
    FROM ANG_PRO
    GROUP BY ANGNR;

   Arbeitszeit INTEGER;

BEGIN

  IF :new.PROZ_ARB < 0 OR :new.PROZ_ARB > 100
        -- :new.PROZ_ARB ist der Wert des Attributs PROZ_ARB im
        -- neu einzufügenden Tupel
    THEN
    RAISE INVALID INSERT;

  ELSE

    OPEN Arbeitszeit_Cursor;
    FETCH Arbeitszeit_Cursor INTO Arbeitszeit;

    WHILE Arbeitszeit_Cursor%FOUND
    LOOP
      IF Arbeitszeit < 0 OR Arbeitszeit > 100 THEN
        CLOSE Arbeitszeit_Cursor;
        RAISE INVALID Arbeitszeit-Summe;
      ELSE
        FETCH Arbeitszeit_Cursor INTO Arbeitszeit;
      END IF;
    END LOOP;

    CLOSE Arbeitszeit_Cursor;
  END IF;
END
```

3.4 Optimierung von Abfragen

Wie wir gesehen haben, erlauben es relationale Abfragesprachen, sehr komplexe Abfragen zu stellen. Da der Benutzer über die Systemabläufe und die physische Datenorganisation nichts weiß, werden viele Abfragen so formuliert sein, dass die naive Ausführung sehr viel Zeit in Anspruch nehmen würde.

Beispiel 3.44:
Man betrachte die Relationenschemata R(A,D) und S(B,C); gegeben sei dann der Ausdruck $\sigma_{C=10}(R \bowtie_{A=B} S)$.

Es soll also der Verbund von R und S mit der Bedingung R.A = S.B gebildet und aus der entstehenden Relation jene Tupel selektiert werden, für die S.C = 10 ist. Beantworten wir die Abfrage naiv (d.h. nach der vorgegebenen Klammerung), so bilden wir zuerst den Verbund von R und S, dann durchsuchen wir die entstehende Relation nach Tupeln mit C = 10. Selbst wenn wir ein extrem effizientes Verfahren zur Bildung von Joins haben, so benötigen wir O(#S + #R) Schritte, wobei wir als Schritt den Zugriff auf ein Tupel ansehen (meist werden sehr viel mehr Schritte benötigt). Anschließend benötigen wir ebenso viele Schritte wie der Verbund Tupel besitzt, um die Selektion durchzuführen.

#S = Zahl der Tupel in S

Was könnte man besser machen?

Man könnte offensichtlich mit der Selektion von S beginnen, dann mit der sehr viel kleineren Relation S' den Join durchführen. Dieses Vorgehen ist vor allem dann dem naiven Vorgehen deutlich überlegen, wenn nur wenige Tupel aus S den Wert 10 besitzen. In diesem Falle wäre nämlich die bei weitem aufwendigste Operation, nämlich der Verbund, wegen #S' << #S entschieden schneller auszuführen als der Verbund von R und S. Schließlich könnte der besonders günstige Fall vorliegen, dass für C ein Sekundärindex existiert.

Dieses einfache Beispiel zeigt, dass die Art der Abarbeitung einer Abfrage entscheidend ist für den notwendigen Aufwand. Eine günstige Ausführungsstrategie kann gegenüber einer naiven Vorgehensweise Größenordnungen in der Ausführungszeit einsparen. Der Query-Prozessor (also der für die Abfrageverarbeitung zuständige Teil des DBMS) muss deshalb auf jeden Fall in gewissem Umfang *Abfrageoptimierung* betreiben.

Query-Prozessor

Abfrageoptimierung ist ein sehr komplexes Problem - und in Wirklichkeit optimieren die Systeme nicht, sondern ermitteln lediglich eine mit großer Wahrscheinlichkeit gute Ausführungsstrategie. Schon das kleine Beispiel zeigt, welche Faktoren bei der Optimierung zu berücksichtigen sind:

- Wie kann die Abfrage umformuliert werden, so dass sie äquivalent zur ursprünglichen Abfrage ist: das System muss erkennen, dass
  $\sigma_{C=10}(R \bowtie_{A=B} S)$ äquivalent ist zu $R \bowtie_{A=B} (\sigma_{C=10}(S))$.

- Welche statistischen Daten liegen vor (wie viele Tupel in jeder Relation, Prozentsatz der Tupel in S mit C = 10, usw.)?
- Sind die Tupel einer Relation sortiert?
- Welche Zugriffspfade gibt es (können für die Selektion oder den Verbund Sekundärindexe oder Verbindungen ausgenutzt werden)?

Algebraische Optimierung

Wir betrachten die Relationenalgebra. Für andere Sprachen gelten die folgenden Ausführungen sinngemäß.

Ein wesentlicher Teil der Optimierung von Abfragen besteht darin, algebraische Ausdrücke umzuformulieren, so dass die Abarbeitung entsprechend den Regeln der Relationenalgebra günstiger wird. Wir wollen im Folgenden einige heuristische Regeln dafür angeben, wie ein Ausdruck umgeformt werden sollte, so dass seine Abarbeitung in aller Regel effizienter wird.

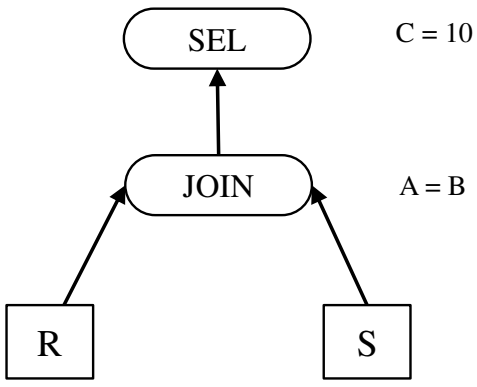
Wir betrachten den *Operatorbaum* einer Abfrage. Der Operatorbaum zeigt die zu bearbeitenden Relationen, die auszuführenden Operationen sowie die Präzedenzen zwischen den Operationen.

Operatorbaum

Der Operatorbaum für den Ausdruck

$$\sigma_{C=10} (R \bowtie_{A=B} S)$$

ist

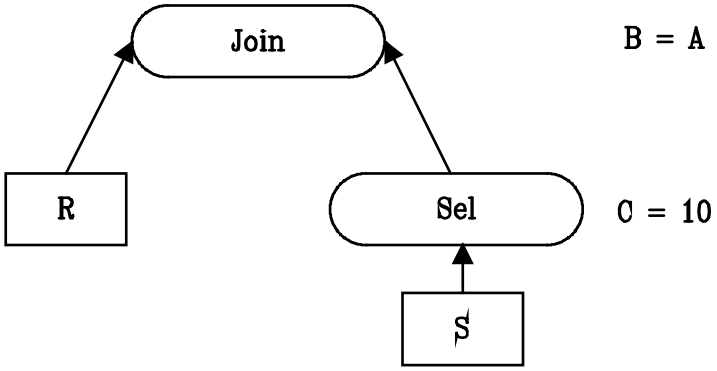


Die Rechtecke stellen Relationen, die Ellipsen Operationen dar. Der Operatorbaum zeigt: die JOIN-Operation ist auf den Relationen R und S auszuführen, danach eine Selektionsoperation auf dem durch den JOIN entstandenen Zwischenergebnis.

Die äquivalente Abfrage

$$R \bowtie_{A=B} (\sigma_{C=10} (S))$$

hat den folgenden Operatorbaum:



Einige *heuristische Regeln für die algebraische Optimierung* sind die folgenden: Optimierungsregeln

1. Selektionen auf dem gleichen Operanden werden zu komplexen Selektionen zusammengefasst.

2. Selektionen werden soweit als möglich zu den Blattknoten des Operatorbaumes verschoben. Mit anderen Worten: Selektionen werden möglichst früh ausgeführt.

   Beispiel 3.45:

   $$\sigma_{B=10} (\sigma_{C=20} (S) \bowtie_{A=B} R)$$

   wird zu

   $$\sigma_{C=20 \wedge B=10} (S) \bowtie_{A=B} R).$$

   Durch diese Umformung wird ein Selektionsdurchlauf für den Verbund (wegen $\sigma_{B=10}$) vermieden; $\sigma_{C=20 \wedge B=10} (S)$ kann mit einem Durchlauf bearbeitet werden.

   Alle Selektionen, die sich auf jeweils eine Relation beziehen, werden vor dem Join ausgeführt.

Der Zeitpunkt der Selektion ist deshalb so wichtig, weil eine Selektion in den meisten Fällen die Anzahl der weiter zu betrachtenden Tupel ganz erheblich reduziert. Nachfolgende Operationen arbeiten dadurch auf kleineren Tupelmengen. Dies ist vor allem wesentlich für die Join-Operation, bei der ja zwei Tupelmengen (Relationen) miteinander zu verknüpfen sind. Im ungünstigsten

Falle kann die Komplexität der Join-Operation von der Ordnung O(#R * #S) sein.

3. Projektionen, die keine Eliminierung von Duplikaten erfordern, werden so früh wie möglich, jedoch nicht vor einer Selektion durchgeführt. Sie werden soweit wie möglich zu den Blättern des Operatorbaumes verschoben.
   Eine Projektion erfordert dann keine Elimination von Duplikaten, wenn dabei zumindest ein Schlüssel des Relationenschemas erhalten bleibt. Ist

   R(A,B,C,D)

   ein Relationenschema mit (A,B) Schlüssel, so kann

   $\pi_{B, C, D}(R)$

   identische Tupel enthalten, die dann entfernt werden müssen. Da das Auffinden identischer Tupel im Allgemeinen aufwendig ist, wird in diesem Falle die Projektion möglichst spät (d.h. meist auf kleinen Relationen) durchgeführt.

4. Projektionen, die eine Eliminierung von Duplikaten erfordern, sind also soweit als möglich zur Wurzel des Operatorbaumes zu verschieben.

5. Suche gemeinsame Teilbäume des Operatorbaums. Wenn das Ergebnis des gemeinsamen Teilausdruckes eine Relation ist, die vom Sekundärspeicher in sehr viel kürzerer Zeit gelesen werden kann, als zu ihrer Berechnung notwendig ist, so lohnt es sich, diese Zwischenrelation nur einmal zu berechnen und abzuspeichern (Hier kommt es also nicht auf eine Umstrukturierung der Abfrage an, sondern auf das Erkennen gleicher Teilbäume).

Diese Regeln sind, wie gesagt, nur heuristischer Natur. Sie führen in den meisten Anwendungsfällen zu guten Ergebnissen. Es lassen sich jedoch immer Gegenbeispiele finden, bei denen diese Regeln keine Verbesserung, sondern vielleicht sogar eine Verschlechterung mit sich bringen.

Übung 3.18:
Gegeben seien wieder die Relationenschemata ANGEST, ANG_PRO und PROJEKT aus Kurseinheit 2. Versuchen Sie unter Anwendung der angegebenen heuristischen Regeln für die algebraische Optimierung, den folgenden Ausdruck durch geeignetes Umformen zu optimieren. Zeichnen Sie auch die entsprechenden Operatorbäume (vorher, nachher).

$$
\begin{array}{l}
\pi_{\text{WOHNORT, PNR}} \\
(\quad \sigma_{\text{ABTNR = 5}} \\
\quad (\quad \sigma_{\text{PNR < 30}} \\
\quad\quad (\quad \sigma_{\text{WOHNORT='HAGEN'}} \\
\quad\quad\quad (\quad \sigma_{\text{BERUF='INGENIEUR'}} \\
\quad\quad\quad\quad (\quad \rho_{\text{ANGEST.ANGNR} \leftarrow \text{ANGNR}} \\
\quad\quad\quad\quad\quad (\quad \pi_{\text{ANGNR, WOHNORT, BERUF, ABTNR}} (\text{ANGEST}) \\
\quad\quad\quad\quad\quad) \\
\quad\quad\quad\quad\quad \bowtie_{\text{ANGEST.ANGNR = ANGNR}} \text{ANG\_PRO} \\
\quad\quad\quad\quad) \\
\quad\quad\quad) \\
\quad\quad\quad \bowtie_{\text{ANGNR = P\_LEITER}} \text{PROJEKT} \\
\quad\quad) \\
\quad) \\
)
\end{array}
$$

Optimierung auf der physischen Ebene

Neben der Umformung algebraischer Ausdrücke ist zur Ermittlung einer günstigen Abfragestrategie natürlich die physische Organisation der Daten zu berücksichtigen.

Es geht jetzt auch darum, wie einzelne Operationen günstig ausgeführt werden können, wobei Zugriffspfade und statistische Größen eine wesentliche Rolle spielen. Wir wollen diese Fragen am Beispiel der Verbund-Operation kurz diskutieren.

Betrachten wir wieder die Relationenschemata R(A,D), S(B,C). Wir wollen den Verbund ($S \bowtie_{A=B} R$) erstellen. Die primitivste Lösung wäre sicher die, durch beide Relationen in der folgenden Weise hindurchzulaufen: für jedes Tupel in R mit Wert A = a werden alle Tupel in S gesucht, für die B = a. Dieser Algorithmus ist für große Relationen offensichtlich sehr ineffizient. Zwei bessere Möglichkeiten sind die folgenden:

- **Ausnutzung von Sekundärindexen**

Ein Sekundärindex ist eine Datenstruktur, mit welcher der Zugriff auf einzelne Datensätze einer Menge von Datensätzen beschleunigt wird. Hierbei wird auf die physische Speicherung der Originaldaten kein Einfluss genommen (sekundärer Zugriffspfad), sondern nur eine weitere Datenstruktur aufgebaut. Diese Datenstruktur beinhaltet ein Verzeichnis von Wertpaaren (s, p). s ist die Ausprägung des Merkmals, für welches der Index aufgebaut wird, p ist der Ort des Datensatzes, welcher diese Merkmalsausprägung besitzt.

Beispiel
Die Tabelle Angestellter kennt das Attribut Name. Da häufig nach allen Personen gesucht wird, welche einen bestimmten Namen besitzen, kann es sich hier lohnen, einen Sekundärindex bzgl. des Merkmals Name aufzubauen.

Da bei einer Änderung des Datenbestandes (in der Regel) auch der Sekundärindex angepasst werden muss, ist die Entscheidung, einen Sekundärindex einzusetzen, gegen die erhöhten Kosten bei der Änderung des Datenbestandes abzuwägen.

Eine genauere Erläuterung zu dem Thema physische Datenorganisation können sie im Kurs Datenbanksysteme II, Kurseinheit 3 bzw. im Kurs Datenbanksysteme, Kurseinheit 6, finden.

Existiert ein Index für B, so ist für jedes Tupel in R sofort die zugehörige Tupelmenge in S feststellbar - und damit die Tupelmenge des Verbundes konstruierbar. Existiert kein Index für A oder B, so lohnt es sich im allgemeinen, einen solchen Index für die gerade auszuführende Verbundoperation anzulegen. Mit diesem Verfahren wird die Zahl der notwendigen Zugriffe auf Tupel im günstigsten Fall linear zur Größe der beiden Relationen: ein Durchlauf zum Anlegen des Index, ein Durchlauf durch R und dabei für jedes Tupel $r \in R$ die notwendigen Zugriffe auf $s \in S$, insgesamt #S Zugriffe auf S.

Beachten Sie, dass die Zahl der Zugriffe auf Tupel im Allgemeinen nicht gleich ist zur Zahl der Zugriffe auf Blöcke des Sekundärspeichers. Es kann etwa der Fall auftreten, dass ein Block mehrmals gelesen werden muss. Ferner kann der Index so groß werden, dass er nicht vollständig im Arbeitsspeicher gehalten werden kann.

- **Sortieren von R nach A und von S nach B**

Zum Sortieren von n Tupeln werden bei Verwendung effizienter Sortierverfahren in der Größenordnung (n log n) Schritte benötigt. Das anschließende Erstellen des Verbundes ist dann sehr schnell: sind R und S sortiert, so werden sie einfach parallel durchlaufen, wobei jeweils auf gleiche Werte für A und B zu prüfen ist.

3.5 Implementierung eines relationalen Datenbanksystems

In diesem Kapitel wollen wir kurz diskutieren, wie ein relationales Datenbanksystem implementiert werden kann, d.h. wie ein Softwaresystem aufzubauen ist, das die beschriebenen Strukturen und Funktionen eines relationalen Systems realisiert. Große Softwaresysteme werden typischerweise in Schichten realisiert, wobei jede Schicht wohldefinierte Strukturen und Operationen für die nächsthöhere Schicht anbietet und dabei von Details der darunterliegenden Ebene abstrahiert.

Aufgrund der sehr komplexen Funktionen von Datenbanksystemen ist es keineswegs einfach, eine geeignete Schichtung zu finden. Es geht bei der Schichtenbildung nicht nur um die Abbildung von Relationen auf speicherbezogene Datenstrukturen, sondern auch um die geeignete Anordnung der funktionalen Komponenten

- Abfrageoptimierer,
- Autorisierung und Integritätskontrolle,
- Recovery sowie
- Synchronisation.

Die prinzipielle Schichtung für ein relationales System ist in Bild 3.3 dargestellt. Wir beschränken uns dabei auf das Wesentliche, auf Verfeinerungen und auf die unterschiedlichen Ansätze kommerzieller Systeme gehen wir nicht ein.
Das Thema Synchronisation wird ausführlich in Kurseinheit 1, Recovery in Kurseinheit 2 des Kurses „Datenbanken II“ behandelt. Wie Daten physisch auf Datenspeichern verwaltet werden, erläutern wir in Kurseinheit 3 des Kurses „Datenbanken II“. Im Kurs Datenbanksysteme finden Sie die entsprechenden Themen in den Kurseinheiten 4 bis 7.

Systempuffer-Manager

Fordert ein Anwendungsprogramm Daten aus der Datenbank an, so führt dies nicht zwangsläufig zu einem physischen Transport von Seiten (*pages*) vom Externspeicher in den Arbeitsspeicher. Vielmehr werden möglichst viele einmal aus der Datenbank gelesene Seiten im Arbeitsspeicher gehalten. Ziel ist es dabei, die Chance möglichst groß zu halten, dass eine Seite, auf die zugegriffen werden muss, sich bereits im Arbeitsspeicher befindet. Dies ist die Aufgabe des Systempuffer-Managers.

Systempuffer-Manager

Der Datentransport zwischen Arbeitsspeicher und Externspeicher geschieht in Einheiten von Seiten fester Größe. Der Systempuffer-Manager stellt den höheren Schichten des DBMS die Seiten zur Verfügung, auf denen sich die benötigten Daten befinden. Wird eine Seite benötigt, so übergibt der Systempuffer-Manager die entsprechende Arbeitsspeicheradresse an die anfordernde Komponente; befindet sich die benötigte Seite noch nicht im Systempuffer, so ruft er sie vom Betriebssystem (Externspeicherverwaltung) ab und platziert sie im Systempuffer.

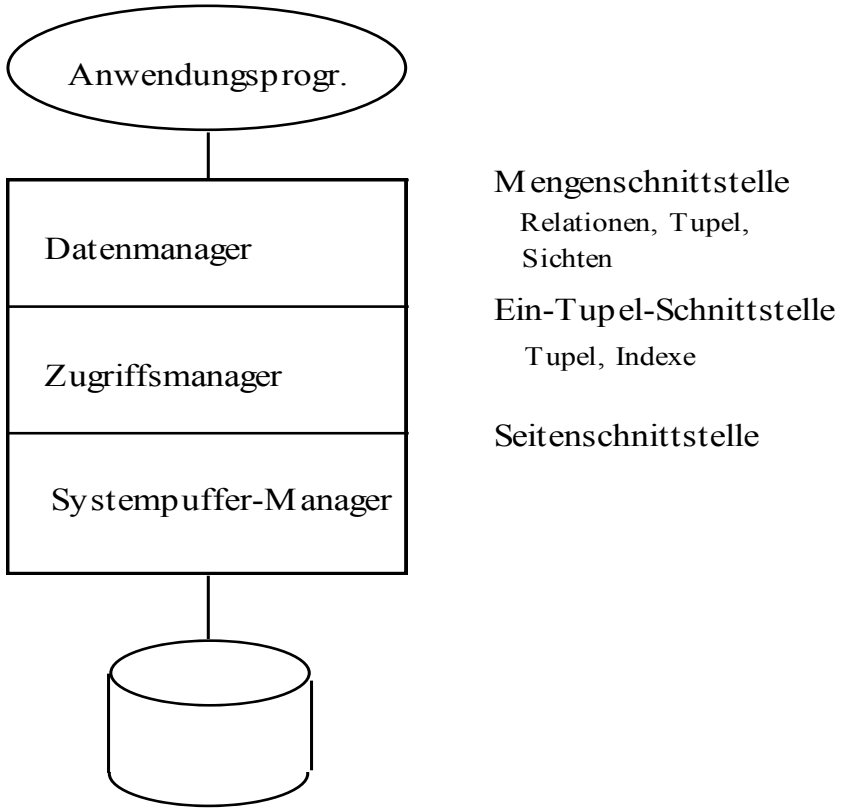


Bild 3.3: Softwareschichten eines DBMS

Wie bei virtuellen Speichern (s. Kurs *Betriebssysteme*) tritt auch hier das Problem auf, welche Seite überschrieben werden darf oder zurückzuschreiben ist, wenn eine neue Seite angefordert wird, im Systempuffer aber kein Platz mehr vorhanden ist. Die Seitenverwaltung in Datenbanksystemen muss jedoch zusätzliche Anforderungen erfüllen:

pinned pages

- Im Zusammenspiel mit der Recovery (Wiederherstellung nach Fehlern) dürfen Seiten nicht beliebig in die Datenbank geschrieben werden. Solche Seiten nennen wir *pinned* (festgeheftet). Beispielsweise ist Recovery nach einem Fehler sehr einfach, wenn die betroffenen Seiten noch nicht in die Datenbank zurückgeschrieben wurden.

forced output

- Ebenfalls im Zusammenhang mit der Recovery müssen gelegentlich Seiten auf den Externspeicher zurückgeschrieben werden, obwohl der Platz im Systempuffer gar nicht benötigt wird. Diese Situation nennt man *forced output* (zwangsweises Schreiben).

Zugriffsmanager

Der Zugriffsmanager stellt eine Schnittstelle zur Verfügung, in der einzelne Tupel und *logische* Zugriffspfade angesprochen werden können (Ein-Tupel-

Schnittstelle). Wie die Zugriffspfade physisch implementiert sind und wie Tupel intern als Sätze dargestellt werden, weiß nur der Zugriffsmanager selbst.

Die wesentlichen Objekte dieser Ein-Tupel-Schnittstelle sind typischerweise

Objekte der Ein-Tupel-Schnittstelle

- Tupel und
- Indexe.

Indexe definieren eine logische Ordnung der Tupel einer Relation, so dass man über den Index auf ein erstes, zweites usw. Tupel entsprechend dieser Ordnung zugreifen kann. Hierfür müssen vom Zugriffsmanager entsprechende Operatoren zur Verfügung gestellt werden:

- Zugriff auf Tupel mit gegebenen Attributwerten
- Zugriff auf Tupel aufgrund ihrer Position in einem Zugriffspfad (FIND NEXT: finde das nächste Tupel bzgl. dieses Zugriffspfades)
- Bereitstellen eines Tupels in einem Übergabebereich
- Einfügen, Löschen eines Tupels
- Verändern von Attributwerten eines Tupels

Wenn für die Suche benötigte Seiten im Systempuffer nicht vorhanden sind, so verlangt der Zugriffsmanager deren Beschaffung vom Systempuffer-Manager.

Operatoren für Schemata

Zu diesen Operatoren auf Tupeln kommen solche für das Lesen von Schemaeinträgen, für das Anlegen und Löschen solcher Einträge (d.h. z.B. das Anlegen einer neuen Relation oder das Löschen einer vorhandenen), das Anlegen und Löschen von Zugriffspfaden, usw.

Operatoren für das Transaktionsmanagement

Die Ein-Tupel-Schnittstelle muss ferner geeignete Operatoren für das *Transaktionsmanagement* zur Verfügung stellen. Der Transaktionsmanager sorgt dafür, dass parallel laufende Anwendungsprogramme sich nicht gegenseitig stören. Dieser Punkt wird im Kapitel 4 im Kurs Datenbanken II ausführlich behandelt.

Datenmanager

Objekte und Operatoren des Datenmanagers

Der Datenmanager bietet eine relationale Schnittstelle der Art, wie wir sie in den vorangegangenen Abschnitten kennengelernt haben. Die Schnittstelle zwischen Anwendungsprogramm und Datensystem ist *mengenorientiert*, d.h. die Objekte dieser Schnittstelle sind Relationen und Tupel, die Operatoren sind relationale Sprachen, etwa SQL. Die Realisierung von Relationen und die Zugriffspfade sind nicht sichtbar.

Abfrageoptimierung

Zugriffskontrolle Integritätskontrolle

Aufgabe des Datenmanagers ist es somit, Anweisungen der relationalen Sprache in entsprechende Aufrufe des Zugriffsmanagers umzusetzen, also durch Objekte und Operatoren der Ein-Tupel Schnittstelle auszudrücken. Neben der Übersetzung und Optimierung der Benutzerabfragen muss der Datenmanager die Zugriffsberechtigungen der Benutzer zu allen in den Anweisungen referenzierten Daten überprüfen und die Integritätskontrollen durchführen.

3.6 Der Datenbank-Entwurfsprozess

In diesem Kapitel betrachten wir den Entwurf des Datenbank-Schemas. Der Datenbank-Entwurf ist in der Regel Teil eines umfassenderen Prozesses, in dem ein Informationssystem (etwa eines Unternehmens) entwickelt wird. Hierauf gehen wir kurz im ersten Abschnitt ein (3.6.1), bevor wir anschließend die verschiedenen Phasen des eigentlichen Datenbank-Entwurfs beleuchten (3.6.2). Ein Großteil der Entwurfsarbeiten wird unabhängig vom konkreten Datenmodell und von der konkreten Datenbank durchgeführt und mündet in der Erstellung eines semantischen Datenmodells, etwa in Form eines Entity-Relationship (ER) Diagramms. Dieses muss anschließend in ein relationales Datenbank-Schema umgewandelt werden. Die entsprechenden Regeln werden in Abschnitt 3.6.3 zusammengefasst.

Wenn man im Entwurfsprozess ein konzeptuelles Schema als ER Diagramm erstellt, so ergibt sich in der Regel ein vernünftiges relationales Datenbank-Schema, das verkürzt dadurch gekennzeichnet ist, dass eine Relation ein Entity der realen Welt darstellt. In der Relationen-Theorie gibt es eine Vielzahl von Untersuchungen über die Frage, welche Eigenschaften für ein gutes relationales Datenbankschema wichtig sind. In 3.6.4 zeigen wir zunächst auf, welche Probleme ein ungünstiges relationales Schema bewirken kann, und kommen dann über die funktionalen Abhängigkeiten (3.6.5) zur Definition verschiedener Normalformen (3.6.6). Relationen-Schemata mit unerwünschten Eigenschaften können durch Zerlegungen in bessere Relationen-Schemata überführt werden; wie solche Zerlegungen aussehen müssen, sehen wir in Abschnitt 3.6.7. Die Theorie der Normalformen gibt uns ein wichtiges Mittel an die Hand, mit dem wir relationale Datenbankschemata überprüfen können, die aus höheren Modellierungssprachen abgeleitet oder gar automatisch generiert worden sind.

3.6.1 Datenbank-Entwurf als Teil der Entwicklung von Informationssystemen

Der Entwurf kleiner Datenbank-Schemata kann mit den heute verfügbaren Desktop-Tools sehr einfach vonstattengehen: Relationen und Queries werden mit graphischer Unterstützung am Bildschirm definiert, und kleine Applikationen können damit scheinbar „on the fly“ erstellt werden. Häufig wird jedoch die Komplexität eines Datenbank-Schemas unterschätzt, und bereits nach wenigen Änderungen oder Ergänzungen wird es zunehmend schwerer, die Zusammenhänge zwischen den Relationen zu überschauen. Da ein enger Zusammenhang zu den Applikationen gegeben ist, die mit den Daten in der Datenbank arbeiten, ist in der Entwurfs-

phase mit häufigen Änderungen und Erweiterungen zu rechnen. Daher ist ein systematischer Entwurf der Datenbank eine Notwendigkeit, und es muss von vornherein der Kontext berücksichtigt werden, in dem die Daten benutzt werden.

Die Datenbank ist in der Regel Bestandteil eines größeren Systems, das alle Ressourcen für die Erfassung, Verwaltung, Nutzung und Verteilung der Daten eines Unternehmens umfasst. Daher wird auch der Entwurfsprozess der Datenbank als Bestandteil eines größeren Prozesses gesehen, des Software-Entwurfsprozesses für das Informationssystem des Unternehmens. Der Lebenszyklus eines solchen Informationssystems umfasst dabei die folgenden Phasen /ELNA02/:

1. Machbarkeitsanalyse
   In dieser Phase wird untersucht, ob das Vorhaben wirtschaftlich und technisch durchführbar ist, welche Anwendungsbereiche damit abgedeckt werden können und wie das Verhältnis der Kosten zum erwarteten Nutzen aussieht.

2. Erfassung und Analyse der Anforderungen
   In dieser Phase geht es darum, die Anforderungen an das System zu ermitteln. Dies geschieht vor allem in Interaktion mit den zukünftigen Nutzern des Systems.

3. Entwurf
   In dieser Phase werden sowohl die Anwendungssysteme als auch das Datenbanksystem entworfen. Dabei ist es wichtig, die Zugriffe der Anwendungen auf die Datenbank zu spezifizieren.

4. Implementierung
   Hier werden die Anwendungs- und Datenbank-Programme erstellt und getestet.

5. Validierung und Abnahmetests
   In dieser Phase wird untersucht, ob das System die gestellten Benutzeranforderungen und die festgelegten Leistungskriterien erfüllt.

6. Installation, Betrieb, Wartung
   Nach der Installation auf dem Zielsystem kann das System in Betrieb genommen werden. Dazu kann eine Umstellung von einem Vorgängersystem auf das neue System gehören, was insbesondere die Konvertierung bereits vorhandener Daten auf die neue Struktur einschließen kann. Zur Systemwartung zählt insbesondere die Korrektur von Programm-Fehlern. Weiterhin entstehen im Laufe des Betriebs üblicherweise neue Anforderungen, die gemäß den oben beschriebenen Phasen umzusetzen und in das laufende System zu integrieren sind.

Es ist klar, dass diese Phasen nicht sequentiell durchlaufen werden, sondern dass es viele Rückkoppelungen und Schleifen und parallele Arbeiten in den verschie-

denen Phasen gibt. Im Folgenden interessiert uns insbesondere die Phase 3, der Entwurf, und hier insbesondere der Entwurf des Datenbank-Schemas.

3.6.2 Der konzeptuelle Datenbank-Entwurf

Der Datenbank-Entwurf verfolgt die folgenden Ziele:

- Erfüllung der Anforderungen, die an die Dateninhalte gestellt werden
- Bereitstellung einer natürlichen und verständlichen Strukturierung der Informationen
- Unterstützung der Verarbeitungsanforderungen (Reaktionszeit, Verarbeitungszeit, etc.).

Das Ergebnis des Entwurfsprozesses ist ein Datenbank-Schema, das die Basis für die Implementierung der Datenbank darstellt.

Der gesamte Datenbank-Entwurfsprozess wird häufig in die folgenden Phasen unterteilt:

1. Erfassung und Analyse der Anforderungen
2. Konzeptueller Datenbankentwurf
3. Wahl des DBMS
4. Abbildung des Entwurfes auf das Datenmodell (auch logischer Entwurf genannt)
5. Physischer Datenbank-Entwurf
6. Implementierung des Datenbanksystems und Tuning

In diesem Kapitel werden wir uns vor allem mit den Schritten 2 und 4 befassen. Der Datenbank-Entwurf ist das Thema dieses Abschnitts, die Abbildung des (ER-) Entwurfes auf das relationale Datenmodel ist das Thema des nächsten Abschnitts. Auf den physischen Entwurf und das Tuning gehen wir in Kapitel 7 des Kurses „Datenbanken II“ ein.

Das Ziel des konzeptuellen Entwurfs ist es, ein Datenbank-Schema zu erstellen, das unabhängig von einem bestimmten Datenbanksystem ist und deshalb durch eine konzeptuelle Modellierungssprache dargestellt wird, etwa durch das Entity-Relationship Modell (siehe Kurseinheit 1). Daneben werden in dieser Phase auch die grundlegenden Transaktionen definiert, durch die Anwendungen auf die Datenbank zugreifen können.

Die Erstellung eines DBMS-unabhängigen konzeptuellen Modells hat eine Reihe von Vorteilen:

- Die Modelle von Datenbanksystemen haben typischerweise spezielle Einschränkungen, die den konzeptuellen Entwurf nicht beeinflussen sollten.
- Das konzeptuelle Schema bleibt stabil, selbst wenn sich das eingesetzte Datenbanksystem ändern sollte (oder es erst in einem späteren Schritt, der Da-

tenbankauswahl, festgelegt wird). Dadurch ist eine größere Flexibilität bei der späteren Implementierung und bei eventuellen Plattformwechseln gegeben.
- Das konzeptuelle Modell stellt eine wichtige Kommunikationsplattform zwischen Datenbank-Designern, Anwendungsentwicklern und Benutzern dar. Daher ist es wichtig, dieses Modell auf eine Weise zu präsentieren, die auch für Laien verständlich und nachvollziehbar ist. Dies ist durch semantische Modelle wie das ER-Modell gegeben, insbesondere durch die graphische Darstellung.

Beim Entwurf des konzeptuellen Modells werden die Komponenten des Schemas, d.h. Entities, Beziehungstypen, Attribute, etc. identifiziert. Weiterhin werden Spezialisierungs- und Generalisierungs-Hierarchien erstellt sowie Schlüsselattribute, Kardinalitäten von Beziehungen und weitere Constraints definiert. Dies geschieht auf der Basis der Anforderungen in Interaktion mit den Fachexperten (u.a. die zukünftigen Nutzer des Systems).

Bei der Erstellung eines komplexen Datenbankschemas kann man zwei prinzipielle Ansätze unterscheiden:

1. Zentraler Schema-Entwurf
2. View-Integration

Beim ersten Ansatz wird die Integration der verschiedenen Anforderungen aus der Phase 1 vor dem Schema-Entwurf durchgeführt. Damit entsteht eine Gesamt-Anforderungsmenge, die als Basis für den Entwurf des integrierten Schemas dient. Anschließend werden die externen Views definiert, die den Anforderungen der einzelnen Benutzergruppen und Anwendungen gerecht werden.

Beim View-Integrations-Ansatz werden die Anforderungen der einzelnen Anwendungen direkt in konzeptuelle Schemata umgesetzt, d.h. pro Anwendung entsteht ein Teilschema. Anschließend werden diese Teilschemata integriert zu einem globalen konzeptuellen Schema für die gesamte Datenbank. Die Teilschemata können anschließend als externe Views an das globale Schema angekoppelt werden. Ein Vorteil des View-Integrations-Ansatzes besteht darin, dass die Modellierung selbst dezentral erfolgen kann und erst in der anschließenden Integrationsphase die verschiedenen Sichten zusammengefasst werden müssen.

Die Integrationsphase beinhaltet etwa die folgenden Aktivitäten:

- Identifizierung von gemeinsamen Konzepten

  Die Teilschemata sind zwar unabhängig voneinander entworfen worden, sie bilden aber in vielen Bereichen dieselben Konzepte ab – natürlich jeweils aus einer unterschiedlichen Sicht. Um diese Darstellungen zusammenführen zu können, muss zunächst festgestellt werden, welche Objekte in den unterschiedlichen Teilschemata dieselben Real-Welt-Konzepte darstellen. Es ist damit zu rechnen, dass diese Objekte nicht immer auf dieselbe Art und Weise modelliert worden sind. Beispiele für mögliche Unterschiede:

  - Unterschiedliche Benennungen zur Beschreibung desselben Konzeptes (Synonyme), aber auch dieselbe Benennung für unterschiedliche Konzepte (Homonyme).
  - Modellierung eines Konzeptes durch unterschiedliche Modellierungskonstrukte, z.B. als Entity-Typ in Teilschema 1 und als Attribut in Teilschema 2.
  - Zuordnung von unterschiedlichen Wertemengen, z.B. bei Benutzung von unterschiedlichen Einheiten für Attribute.
  - Definition unterschiedlicher Constraints; so kann eine Beziehung in Teilschema 1 etwa als 1:n Beziehung, in Teilschema 2 aber als n:m Beziehung definiert sein.

- Bei Vorliegen von derartigen Unterschieden müssen Teilschemata modifiziert werden, um sie untereinander konform zu machen.

- Mischen von Teilschemata: Sind die übereinstimmenden Konzepte gleichartig modelliert worden, so können die Teilschemata in ein globales Schema integriert werden, indem jedes Konzept nur noch einmal dargestellt wird. Dies ist sicher der komplexeste Schritt und erfordert eine intensive Zusammenarbeit der verschiedenen Benutzergruppen zur Lösung von Konflikten.

- Schließlich kann das entstandene globale Schema umstrukturiert werden, um Redundanzen und unnötige Komplexität zu eliminieren.

Parallel zum Entwurf des Datenbank-Schemas werden die Transaktionen spezifiziert, mit denen die Anwendungen auf die Datenbank zugreifen können. Dazu werden verschiedene Methoden der Prozess-Modellierung benutzt, von denen z.B. in UML (Unified Modelling Language, eine Sammlung von Modellierungsmechanismen für das Software Engineering, siehe z.B. /BABA11/) eine große Anzahl angeboten werden. Wichtig für den Entwurfsprozess ist es, zu erkennen, welche Daten durch welche Transaktionen gelesen bzw. geschrieben werden. Dadurch wird zum einen klar, ob alle notwendigen Informationen tatsächlich im Schema berücksichtigt worden sind, zum anderen werden dadurch Informationen gewonnen, welche Daten in welchen Zusammenhängen mit anderen Daten benutzt werden, was wichtig für den physischen Datenbank-Entwurf ist.

3.6.3 Abbildung von ER-Diagrammen in relationale Schemata

Nachdem das konzeptuelle Schema als ER-Modell vorliegt, muss es auf das Modell des gewählten Datenbanksystems abgebildet werden. Im Folgenden gehen wir auf die wesentlichen Regeln für die Transformation von ER-Diagrammen in relationale Datenbankschemata ein.

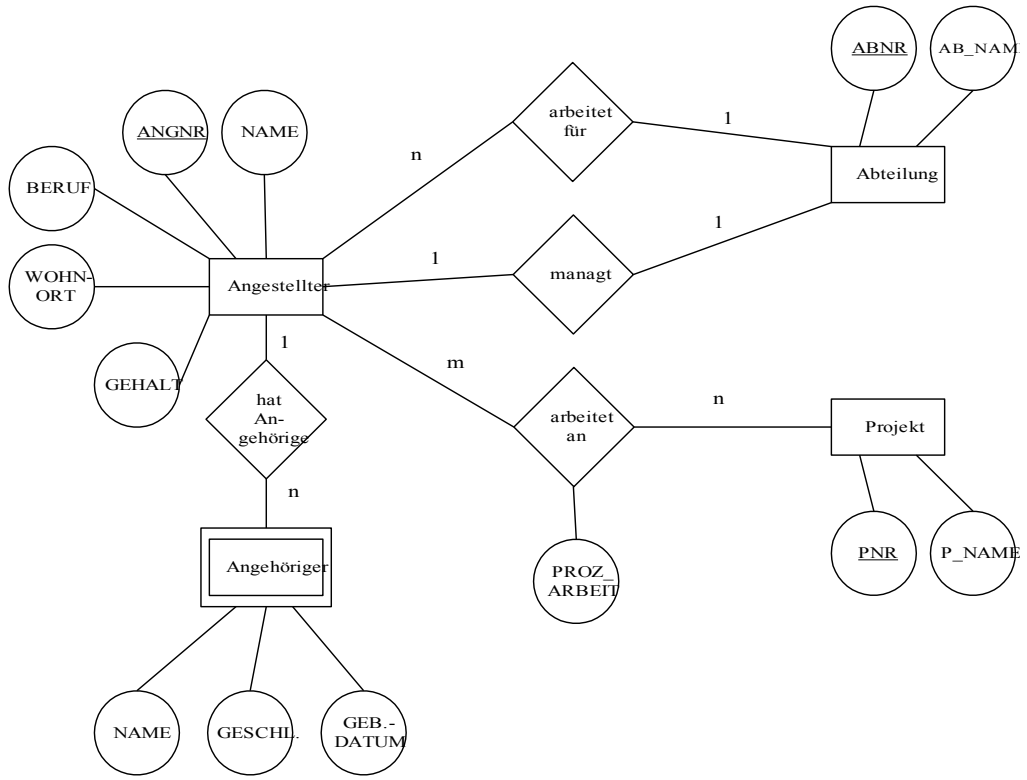


Bild 3.4: ER-Diagramm

Transformation starker Entity-Typen
Jeder starke Entity-Typ A wird in eine Relation bzw. Tabelle RA überführt, die die Schlüsselattribute und die Nichtschlüsselattribute des entsprechenden Entity-Typs A enthält. Der Name der Relation bzw. der Tabelle RA entspricht dem Namen des Entity-Typen A. Die Attribute des Entity-Typs A werden die Attribute der Relation bzw. die Spaltennamen der entsprechenden Tabelle RA. Transformiert man die starken Entity-Typen des obigen Bildes, so erhält man:

$R_{ANGESTELLTER}$(ANGNR, NAME, BERUF, WOHNORT, GEHALT)
$R_{PROJEKT}$(PNR, P_NAME)
$R_{ABTEILUNG}$(ABNR, AB_NAME)

Transformation von n:m-Beziehungstypen
Jeder n:m-Beziehungstyp $\boldsymbol{X}$ zwischen zwei Entity-Typen $\boldsymbol{A}$ und $\boldsymbol{B}$ wird in eine eigene Relation bzw. Tabelle $\boldsymbol{R_X}$ überführt. Die Primärschlüssel der Relationen $\boldsymbol{R_A}$ und $\boldsymbol{R_B}$ werden als Fremdschlüssel in $\boldsymbol{R_X}$ eingesetzt, wobei deren Kombination den Primärschlüssel von $\boldsymbol{R_X}$ bildet. Die übrigen Attribute von $\boldsymbol{X}$ werden ebenfalls in $\boldsymbol{R_X}$ übernommen. Daraus ergibt sich für das zuvor dargestellte ER-Diagramm:

$R_{arbeitet\ an}$(ANGNR, PNR, PROZ_ARBEIT)

Transformation von 1:n-Beziehungstypen
Bei dieser Art von Beziehungstypen muss keine neue zusätzliche Relation eingeführt werden. Stattdessen wird der Primärschlüssel der Relation, die dem Entity-

Typen auf der 1-Seite entspricht, als Fremdschlüssel mit in die Relation aufgenommen, die dem Entity-Typen auf der n-Seite entspricht.

$R_{ANGESTELLTER}$(ANGNR, NAME, BERUF, WOHNORT, GEHALT, **ABNR**)

Transformation von 1:1-Beziehungstypen
Diese Transformation wird ähnlich wie die der 1:n-Beziehungstypen gehandhabt. Auch hier muss keine neue zusätzliche Relation eingeführt werden. Seien ***A*** und ***B*** zwei Entity-Typen. Dann kann man entweder den Primärschlüssel der Relation R_A in R_B als Fremdschlüssel einfügen, um die Beziehung auszudrücken, oder umgekehrt den Primärschlüssel der Relation R_B in R_A.

$R_{ANGESTELLTER}$(ANGNR, NAME, BERUF, WOHNORT, GEHALT, **LEITET_ABT**)
oder alternativ
$R_{ABTEILUNG}$(ABNR, AB_NAME, **ABT_LEITER**)

In den Relationen $R_{ANGESTELLTER}$ und $R_{ABTEILUNG}$ wurden die eingeführten Fremdschlüssel ABNR in LEITET_ABT sowie ANGNR in ABT_LEITER umbenannt. Mit dieser Attributumbenennung wird klarer, welche Rolle die eingefügten Attribute in den jeweiligen Relationen spielen. Die jeweiligen Domänen der Attribute bleiben natürlich gleich. Die zweite Transformationsalternative ist in diesem Fall die sinnvollere, da man davon ausgehen kann, dass es für jede Abteilung einen Angestellten gibt, der diese managt. Hätte man die Abteilungsnummer in $R_{ANGESTELLTER}$ als Fremdschlüssel aufgenommen (LEITET_ABT), um die 1:1-Beziehung darzustellen, so hätte fast jedes Tupel in LEITET_ABT einen Null-Wert, da nicht jeder Angestellte eine Abteilung managt.

Transformation schwacher Entity-Typen
Schwache Entity-Typen werden ebenfalls auf Relationen abgebildet. Jeder schwache Entity-Typ ***A*** wird in eine Relation R_A überführt, die zunächst alle Attribute des Entity-Typen ***A*** enthält. Kann dieser Entity-Typ ***A*** nur durch einen anderen starken Entity-Typen ***B*** identifiziert werden, man spricht in diesem Zusammenhang auch vom Owner-Typen ***B***, so wird der Primärschlüssel der Relation R_B als Fremdschlüssel in R_A eingefügt. Dieser sowie der partielle Schlüssel des schwachen Entity-Typen ***A*** bilden den Primärschlüssel von R_A. (Unter dem partiellen Schlüssel eines schwachen Entity-Typen versteht man die Menge der Attribute, die ein Entity innerhalb der Menge der schwachen Entitäten, die zu einem Owner gehören, identifiziert.)

$R_{ANGEHÖRIGER}$(NAME, ANGNR, GESCHLECHT, GEBURTSDATUM)

In ähnlicher Weise kann man mehrwertige Attribute, wie sie im ER-Modell vorkommen können, behandeln. Ein Beispiel ist, dass es für eine Wohnung mehrere Telefonnummern geben kann.

Transformation mehrwertiger Beziehungstypen

Bei den zuvor betrachteten Beziehungstypen sind wir immer davon ausgegangen, dass diese binärer Natur (n=2) sind. Mehrwertige Beziehungstypen, (n>2), werden grundsätzlich in eine eigene Relation überführt. Sei ***X*** ein Beziehungstyp sowie ***A1***, ..., ***An*** die n beteiligten Entity-Typen. Die Primärschlüssel der n Relationen R_{A1}, ..., R_{An} werden als Fremdschlüssel in die neue Relation R_X eingeführt, ebenso werden alle Attribute des Beziehungstypen ***X*** in die neue Relation R_X übernommen. Der Primärschlüssel von R_X entspricht der Kombination aller Fremdschlüssel, die auf die Relationen R_{A1}, ..., R_{An} verweisen.

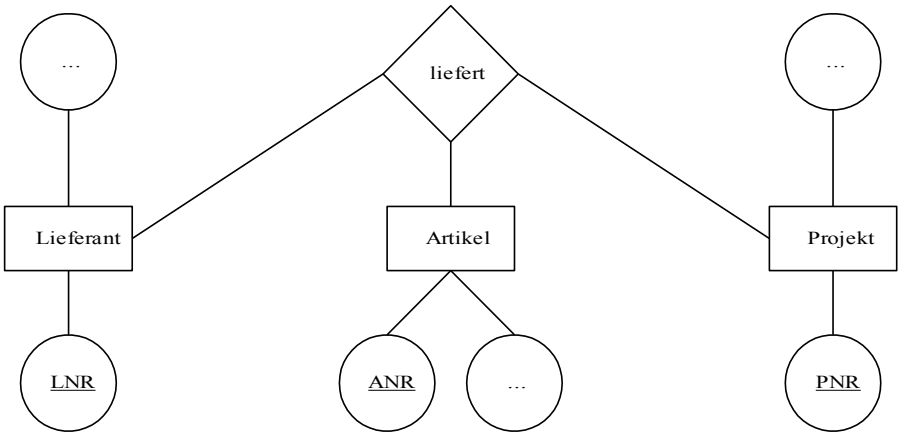


Bild 3.5: ER-Diagramm

Für das obige Diagramm ergibt sich somit.

$R_{liefert}$(LNR, ANR, PNR)

Transformation von Supertyp/Subtyp-Hierarchien

Für diesen Fall existieren mehrere Transformationsmöglichkeiten, die in Abhängigkeit vom späteren Zugriffsverhalten auf die Datenbank angewendet werden. Eine dieser Transformationsmethoden ist die folgende.

Sei der Entity-Typ ***A*** der Supertyp der beiden Entity-Typen ***B*** und ***C***. Alle an dieser Hierarchie beteiligten Entity-Typen werden in eigene Relationen R_A, R_B bzw. R_C überführt. Jede dieser Relationen enthält als Attribute nur die Eigenschaften, die auf der entsprechenden Stufe der Typhierarchie definiert sind und die an die darunterliegenden Subtypen weitervererbt werden können. Die Relationen enthalten nicht die von oben geerbten Attribute. Die Relationen R_B und R_C haben als Primärschlüssel den Primärschlüssel der Relation R_A, wobei diese auch als Fremdschlüssel agieren um auf R_A zu verweisen. Das folgende Beispiel zeigt eine derartige Umsetzung.

Beispiel 3.46:

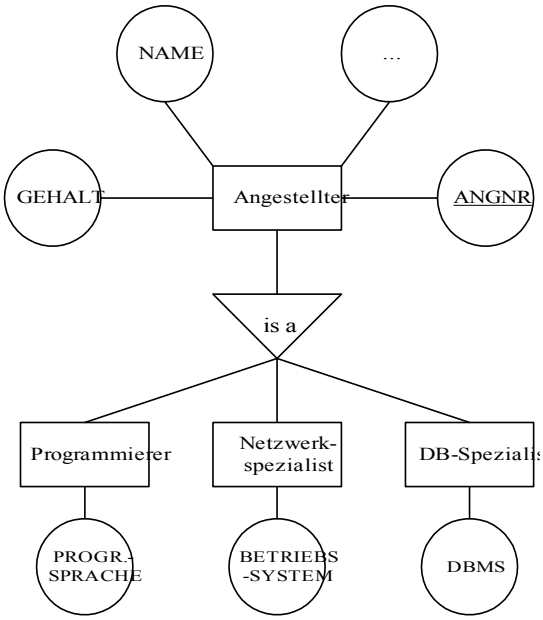


Bild 3.6: ER-Diagramm

$R_{ANGESTELLTER}$(ANGNR, NAME, GEHALT, ...)
$R_{PROGRAMMIERER}$(ANGNR, PROGRAMMIERSPRACHE)
$R_{NETZWERKSPEZIALIST}$(ANGNR, BETRIEBSSYSTEM)
$R_{DB\text{-}SPEZIALIST}$(ANGNR, DBMS)

Je nach Anwendung kann es aber auch günstiger sein, folgende Methode anzuwenden.

Sei der Entity-Typ ***A*** der Supertyp der beiden Enitity-Typen ***B*** und ***C***. Im Gegensatz zur zuvor vorgestellten Transformationsmethode werden nicht alle an dieser Hierarchie beteiligten Entity-Typen in eigene Relationen überführt. Lediglich der Supertyp wird in eine Relation R_A transformiert, wobei in diese zusätzlich die Attribute der Subtypen ***B*** und ***C*** integriert werden.
Für Beispiel 3.46 ergibt sich somit die Relation

$R_{ANGESTELLTER}$(ANGNR, NAME, GEHALT, ...,
PROGRAMMIERSPRACHE,
BETRIEBSSYSTEM, DBMS)

Aus den vorgestellten Einzeltransformationen entwickelt sich im Rahmen eines Datenbankentwurfprojektes nach und nach ein Gesamtschema. Ein minimales Gesamtschema für das ER-Diagramm in Bild 3.4 ist das Folgende:

$R_{ANGESTELLTER}$(ANGNR, NAME, BERUF, WOHNORT, GEHALT, ABNR)
$R_{PROJEKT}$(PNR, P_NAME)
$R_{ABTEILUNG}$(ABNR, AB_NAME, ABT_LEITER)
$R_{ANGEHÖRIGER}$(NAME, ANGNR, GESCHLECHT, GEBURTSDATUM)
$R_{arbeitet\ an}$(ANGNR, PNR, PROZ_ARBEIT)

Übung 3.19:
Gegeben sei das folgende ER-Diagramm, das einen rekursiven Beziehungstyp darstellt. Schlagen Sie eine geeignete Transformation vor.

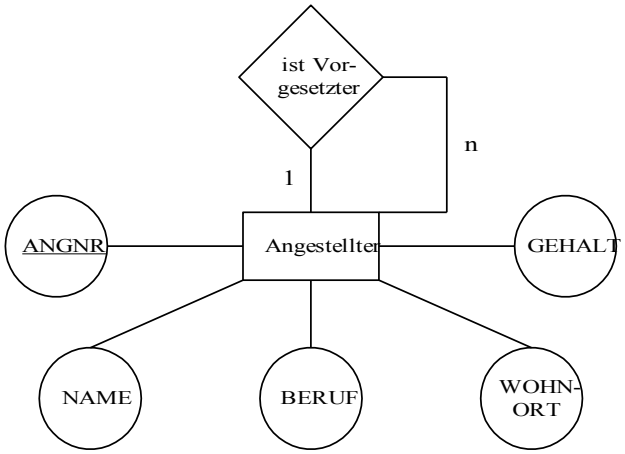


Bild 3.7: ER-Diagramm

3.6.4 Mögliche Anomalien eines Relationen-Schemas

Über das konzeptuelle Modell und die Abbildung auf das relationale Modell ist nun ein relationales Datenbankschema entstanden. Wir werden im Folgenden diskutieren, welche Eigenschaften für ein relationales Datenbankschema wichtig sind. Im Bereich relationaler Systeme gibt es für diese Aufgabe eine Reihe von Methoden und Richtlinien; in der Literatur, z.B. /TEOR94/ u. /DUHA89/, finden sich mannigfaltige theoretische Untersuchungen zu diesem Thema. Diese Sichtweise auf das Entwurfsproblem bei relationalen Datenbanken ist durchaus auf andere Datenmodelle übertragbar.

Der zentrale Gesichtspunkt beim Entwurf eines Datenbankschemas ergibt sich unmittelbar aus der allgemeinen Diskussion über konzeptuelle Schemata:

> Ein Relationenschema sollte gerade einen Entity-Typ oder einen Beziehungstyp beschreiben.

Mit anderen Worten: eine Relation sollte Informationen beinhalten, die tatsächlich auch logisch zusammengehört. Ein Beispiel möge diese sehr allgemeine Forderung verdeutlichen. Betrachten wir das Buchungssystem eines Zeitschriftenverlages, das Auskunft über die Zeitschriftenabonnements seiner Kunden gibt. Es

werden der Kunde, seine Adresse, die Zeitschriften, die er abonniert, und die entsprechenden Preise festgehalten.

ABONNEMENT(KUNDE, ADRESSE, ZEITSCHRIFT, PREIS)
{KUNDE, ZEITSCHRIFT} sei einziger Schlüssel

Dieses Relationenschema weist bei genauerem Hinsehen folgende Schwächen auf:

Anomalien

1. *Einfüge-Anomalie (Insertion Anomaly)*:

   Soll ein potentieller Kunde aufgenommen werden, der sich bisher noch nicht endgültig für ein Abonnement entscheiden konnte, so ist dies nicht möglich. Man könnte zwar auf die Idee kommen, für ZEITSCHRIFT einen speziellen Null-Wert einzusetzen. {KUNDE, ZEITSCHRIFT} ist aber der einzige Schlüssel der Relation, und es kann schwierig oder unmöglich sein, Tupel mit Null-Werten in Schlüsseln zu verwalten oder wieder aufzufinden.

2. *Lösch-Anomalie (Deletion Anomaly)*:

   Dies ist gerade das umgekehrte Problem zu der unter 1 erwähnten Einfüge-Anomalie. Löschen wir in ABONNEMENT einen Kunden, der der letzte Kunde ist, der eine gewisse Zeitschrift abonniert, so gehen alle Informationen über die Existenz dieser Zeitschrift verloren.

3. *Änderungs-Anomalie (Update Anomaly)*:

   Ändert sich der Preis einer Zeitschrift, so muss die ganze Relation durchsucht werden, und alle entsprechenden Einträge müssen abgeändert werden. Obwohl nur ein einziger Tatbestand geändert wird, müssen bei dem gegebenen Relationenschema mehrere Änderungen vorgenommen werden.

Alle diese Schwierigkeiten rühren daher, dass in der Relation ABONNEMENT Informationen über zwei Entity-Typen (Kunden und Zeitschriften) sowie deren Beziehung zueinander (Abonnement) festgehalten wird. Die Adresse eines Kunden hat nichts mit der gelieferten Zeitschrift zu tun. Für dieses Beispiel werden die oben genannten Probleme vermieden, wenn wir statt ABONNEMENT die Relationenschemata

KUNDE(KUNDE, ADRESSE)
ZEITSCHRIFT (ZEITSCHRIFT, PREIS)
ABONNEMENT (KUNDE, ZEITSCHRIFT)

definieren. Die Adresse wird jetzt in KUNDE für jeden Kunden genau einmal aufgeführt. Wir können die Adresse eines potentiellen Kunden aufnehmen, auch wenn er noch keine Zeitschrift abonniert hat. Wir können einen Kunden löschen, ohne dass die Gefahr ungewollten Informationsverlustes über Zeitschriften be-

steht. Und wir können einen Preis in der gewünschten einfachen Weise ändern. Als ER-Modellierer sehen Sie auch sofort, dass eine vernünftige Modellierung der Entity-Typen Kunde und Zeitschrift mit einem Beziehungstyp Abonnement und der Anwendung der oben angeführten Überführungsregeln genau zu diesem Ergebnis führt.

Theorie des Schemaentwurfs

Das Hauptanliegen der Theorie zum Entwurf relationaler Datenbankschemata ist es, Kriterien für Schemata anzugeben, die keine der oben genannten oder ähnliche Anomalien aufweisen. Wir werden im Folgenden mit der Betrachtung verschiedener Normalformen einige Ansätze zu diesem Problem diskutieren. Grundlage all dieser Ansätze bildet das Konzept der funktionalen Abhängigkeiten.

3.6.5 Funktionale Abhängigkeit

Für unsere folgenden Überlegungen müssen wir Information über die Bedeutung der Daten, deren Semantik, heranziehen. Im gegebenen Zusammenhang sind solche Eigenschaften von Interesse, die Aussagen über allgemein geltende Beziehungen zwischen Attributen machen. Das mit Abstand wichtigste Konzept zur Charakterisierung solcher Beziehungen ist das der funktionalen Abhängigkeit zwischen Attributen einer Relation.

Zur Sprechweise: Sei $R(A_1, A_2, ..., A_n)$ ein Relationenschema und X eine Teilmenge von $\{A_1, A_2, ..., A_n\}$. Betrachten wir ein Tupel r einer Relation vom Typ R, so hat jedes Attribut in der Menge X einen Wert, und wir sprechen abkürzend vom Wert von X für dieses Tupel.

Funktionale Abhängigkeit (functional dependency)

Definition funktionale Abhängigkeit

Sei $R(A_1, A_2, ..., A_n)$ ein Relationenschema und X und Y Teilmengen von $\{A_1, A_2, ..., A_n\}$. Dann ist *Y funktional abhängig von X*, geschrieben $X \rightarrow Y$, wenn es keine Relation vom Typ R geben kann, in der zwei Tupel denselben Wert für X, aber verschiedene Werte für Y haben.

Natürlich kann die Aussage „es gibt keine Relation vom Typ R mit einer bestimmten Eigenschaft“ nur aus der betrachteten Realwelt abgeleitet werden. Es handelt sich hier um Aussagen, die für alle möglichen Relationen eines Relationenschemas R gelten, und die deshalb keinesfalls aus der Betrachtung eines gerade vorhandenen Wertes für R ableitbar sind.

Ist also in zwei Tupeln der Wert für X derselbe, so ist auch der Wert für Y derselbe, wenn $X \rightarrow Y$ gilt. Umgekehrt können sehr wohl Tupel mit unterschiedlichen Werten für X aber gleichen Werten für Y auftreten.

Beispiel 3.47:
Betrachten wir ein Liefersystem, das Auskunft über den Wareneinkauf eines Obstgeschäftes gibt. Es enthält Informationen über den Lieferanten, dessen Adresse, die von ihm gelieferten Artikel, deren Anbaugebiet und Qualität.

LIEFER (LIEFERANT, ADRESSE, ARTIKEL, ANBAUGEBIET,
QUALITÄT).

Dann gilt beispielsweise:

Beziehungen zwischen Attributen	**Funktionale Abhängigkeiten**
jeder Lieferant hat genau eine Adresse	LIEFERANT→ADRESSE
gleiche Artikel eines Lieferanten kommen aus demselben Anbaugebiet	{LIEFERANT,ARTIKEL}→ ANBAUGEBIET
jedes Anbaugebiet ist nur durch einen Lieferanten vertreten	ANBAUGEBIET→LIEFERANT
gleiche Anbaugebiete produzieren dieselbe Qualität	ANBAUGEBIET→QUALITÄT

Zu LIEFER gibt es also die Menge

F = {LIEFERANT→ADRESSE,
{LIEFERANT,ARTIKEL}→ANBAUGEBIET,
ANBAUGEBIET→LIEFERANT,
ANBAUGEBIET→QUALITÄT}

von funktionalen Abhängigkeiten.

Statt von einer „Menge funktionaler Abhängigkeiten“ (functional dependencies) sprechen wir im Folgenden auch einfach von einer Fd-Menge.

Eine Fd-Menge F besitzt die Eigenschaft, neue funktionale Abhängigkeiten zu implizieren. Genauer sagt man *F impliziert X→Y*, wenn X→Y in allen Relationen eines Relationenschemas gültig ist, in denen auch F gültig ist.

Closure von F

Die Menge F^+ aller funktionalen Abhängigkeiten, die von F impliziert werden, heißt *Closure von F*.

Im Beispiel 3.47 wird die funktionale Abhängigkeit {LIEFERANT, ARTIKEL}→QUALITÄT von der zum Relationenschema LIEFER gehörenden Fd-Menge F impliziert. Dies ist leicht nachzuweisen, indem man zeigt, dass es nicht zwei Tupel geben kann, die auf den {LIEFERANT,ARTIKEL}-Werten, nicht aber auf den QUALITÄT-Werten übereinstimmen.

Zwar ist es möglich, die Menge F^+ explizit zu konstruieren, aber dies ist für unsere Zwecke nicht sinnvoll. Wesentlich interessanter ist die Frage, ob eine funktio-

nale Abhängigkeit in F^+ enthalten ist oder nicht. Diese Frage kann man, wie der folgende Satz zeigt, ohne die vollständige Ermittlung von F^+ beantworten.

Satz:

$X \rightarrow Y \in F^+$ genau dann, wenn $Y \subseteq cl_F(X)$, wobei $cl_F(X)$ wie folgt berechnet wird:

(i) initialisiere $cl_F(X) := \{X\}$

(ii) wenn $W \rightarrow Z \in F$ und $W \subseteq cl_F(X)$
dann $cl_F(X) := cl_F(X) \cup \{Z\}$

(iii) wiederhole (ii) solange, bis kein weiteres Attribut zu $cl_F(X)$ hinzugefügt werden kann.

Ob {LIEFERANT, ARTIKEL}$\rightarrow$QUALITÄT $\in F^+$ lässt sich somit auf das folgende Problem zurückführen:

{QUALITÄT } $\subseteq cl_F(\{LIEFERANT, ARTIKEL\})$?

$cl_F(\{LIEFERANT, ARTIKEL\})$ berechnet sich dabei schrittweise wie folgt: (cl^i_F bezeichne die im i-ten Schritt erreichte Menge von Attributen):

$cl^1_F = \{LIEFERANT, ARTIKEL\}$ (Initialisierung)
$cl^2_F = cl^1_F \cup \{ADRESSE\}$ (1. Regel)
$cl^3_F = cl^2_F \cup \{ANBAUGEBIET\}$ (2. Regel)
$cl^4_F = cl^3_F \cup \{QUALITÄT\}$ (4. Regel)
➔ $cl_F(\{LIEFERANT, ARTIKEL\}) = \{LIEFERANT, ADRESSE, ARTIKEL, ANBAUGEBIET, QUALITÄT\}$,
d.h.: {QUALITÄT } $\subseteq cl_F(\{LIEFERANT, ARTIKEL\})$

Das Problem, ob zwei Fd-Mengen F und H die gleiche Closure besitzen (also $F^+ = H^+$) kann unter Verwendung des obigen Satzes ebenfalls sehr effizient beantwortet werden; denn

$F^+ \subseteq H^+$ genau dann, wenn für alle $X \rightarrow Y \in F$ gilt : $X \rightarrow Y \in H^+$.

Übung 3.20:
Beweisen Sie:

Seien F,H zwei Fd-Mengen, dann gilt $F^+ \subseteq H^+$ genau dann, wenn für alle $X \rightarrow Y \in F$ gilt $X \rightarrow Y \in H^+$.
Hinweis: $F \subseteq H^+ \Rightarrow cl_F(X) \subseteq cl_H(X)$.
Benutzen Sie den Satz aus diesem Abschnitt und den obigen Hinweis.

3.6.6 Normalisierung

Um die eingangs genannten Anomalien mehr oder weniger vollständig zu vermeiden, wurden verschiedene Normalformen für relationale Datenbankschemata vor-

geschlagen und Methoden entwickelt, gegebene Schemata in solche umzuwandeln, die in einer bestimmten Normalform sind. Alle Normalformen sind auf der Basis der funktionalen Abhängigkeit definiert, einige berücksichtigen zusätzlich andere Arten von Abhängigkeiten, auf die wir jedoch nicht eingehen werden.

Für die Normalformen spielen der Begriff des Schlüssels und der vollen funktionalen Abhängigkeit eine wesentliche Rolle.

voll funktional abhängig

Für eine Fd-Menge F und eine funktionale Abhängigkeit $X \rightarrow Y \in F^+$ heißt Y *voll funktional abhängig von X*, genau dann wenn es keine echte Teilmenge X' von X gibt, so dass $X' \rightarrow Y \in F^+$.

Beispiel 3.48:
In Beispiel 3.47 ist ADRESSE voll funktional abhängig von LIEFERANT, aber nicht von {LIEFERANT, ARTIKEL}.

Schlüssel

In der bisher eingeführten Terminologie können wir den Schlüssel wie folgt definieren:

X ist *Schlüssel* von $\{A_1, ..., A_n\}$ genau dann, wenn $X \rightarrow \{A_1, ..., A_n\} \in F^+$ und $\{A_1, ..., A_n\}$ ist voll funktional abhängig von X.

Schlüsselattribut, Nichtschlüsselattribut

Wir nennen A ein *Schlüsselattribut* des Relationenschemas R, wenn A Element irgendeines Schlüssels von R ist; andernfalls nennen wir A *Nichtschlüsselattribut*.

Beispiel 3.49:
In LIEFER(LIEFERANT, ADRESSE, ARTIKEL, ANBAUGEBIET, QUALITÄT) mit den Fd's

LIEFERANT→ADRESSE,
{LIEFERANT,ARTIKEL}→ANBAUGEBIET,
ANBAUGEBIET→LIEFERANT,
ANBAUGEBIET→QUALITÄT

ist {LIEFERANT, ARTIKEL} Schlüssel, also ist z.B. ARTIKEL ein Schlüsselattribut. QUALITÄT ist ein Nichtschlüsselattribut.

erste Normalform

Erste Normalform (1NF):
Die Relationen, wie wir sie definiert haben, sind bereits in *erster Normalform*. Damit ist nichts anderes gemeint, als dass die Werte der Wertebereiche jedes Attributes unteilbare Werte sind und nicht ihrerseits wieder aus Mengen oder Tupeln bestehen.

zweite Normalform

Zweite Normalform (2NF):
Eine Relation R ist in *zweiter Normalform*, wenn jedes Nichtschlüsselattribut A von R voll funktional abhängig von jedem Schlüssel X von R ist.

Beispiel 3.50:
Betrachten wir wieder das Relationenschema LIEFER(LIEFERANT, ADRESSE, ARTIKEL, ANBAUGEBIET, QUALITÄT) mit den funktionalen Abhängigkeiten:

$F = \{LIEFERANT \rightarrow ADRESSE,$
$\{LIEFERANT, ARTIKEL\} \rightarrow ANBAUGEBIET,$
$ANBAUGEBIET \rightarrow LIEFERANT,\ ANBAUGEBIET \rightarrow QUALITÄT\}$

LIEFER ist nicht in zweiter Normalform, da das Nichtschlüsselattribut ADRESSE nicht voll funktional abhängig vom Schlüssel {LIEFERANT, ARTIKEL} ist; denn es gilt sowohl $\{LIEFERANT, ARTIKEL\} \rightarrow ADRESSE \in F^+$ als auch $(LIEFERANT \rightarrow ADRESSE) \in F^+$

Aus der Tatsache, dass LIEFER nicht in zweiter Normalform ist, resultiert eine Lösch-Anomalie, denn wird eine Lieferung eines Lieferanten gelöscht, der nur ein Teil liefert, so geht auch die gesamte Information über ihn verloren.

Der Normalisierungsschritt von 1NF zu 2NF besteht darin, in den Relationen partielle (nicht voll funktionale) Abhängigkeiten zu eliminieren. Dies geschieht dadurch, dass wir die 1NF Relation zerlegen. Dazu werden die partiell abhängigen Attribute in eine neue Relation übertragen, zusammen mit einer Kopie der Attribute, von denen sie abhängig sind.

In unserem Beispiel wird das Attribut ADRESSE aus der Relation LIEFER herausgenommen und zusammen mit seinem bestimmenden Attribut in eine neue Relation LF-ADRESSE (LIEFERANT, ADRESSE) übertragen. Damit verbleiben in LIEFER (LIEFERANT, ARTIKEL, ANBAUGEBIET, QUALITÄT). Somit haben wir Informationen über den Lieferanten in eine eigene Tabelle ausgelagert. Die verbleibende Tabelle enthält im Wesentlichen Informationen über die Artikel. Deshalb benennen wir sie um zu LF-ARTIKEL. Beide Relationen sind in 2. Normalform.

Die Relation LF-ARTIKEL weist aber z.B. noch eine weitere Änderungs-Anomalie auf: Löschen wir den letzten Artikel aus einem bestimmten Anbaugebiet, so geht die Information über die Qualität dieses Anbaugebietes verloren. Diese Anomalie wird dadurch hervorgerufen, dass QUALITÄT nur indirekt vom Schlüssel abhängig ist. Somit ist mit der zweiten Normalform unser Entwurfsziel - Vermeidung von Anomalien – noch nicht erreicht.

Daher wird die dritte Normalform eingeführt, um derartige transitive Abhängigkeiten von Nichtschlüsselattributen zu eliminieren. Es gibt eine Reihe unterschiedlicher, inhaltlich jedoch gleicher Definitionen der dritten Normalform. Wir wählen die folgende:

dritte Normalform

Dritte Normalform:
Ein Relationenschema R mit Fd-Menge F ist in *dritter Normalform*, wenn für alle $X \rightarrow A \in F^+$ mit $A \notin X$ gilt: X enthält einen Schlüssel für R oder A ist Schlüsselattribut.

Anders formuliert, die dritte Normalform ist verletzt, wenn es eine funktionale Abhängigkeit $X \rightarrow A$ gibt, so dass X Nichtschlüssel und A Nichtschlüsselattribut ist.

Beispiel 3.51:
Man betrachte das Relationenschema

LF-ARTIKEL (LIEFERANT, ARTIKEL, ANBAUGEBIET, QUALITÄT)

mit den funktionalen Abhängigkeiten
{LIEFERANT, ARTIKEL}→ANBAUGEBIET,
ANBAUGEBIET→LIEFERANT,
ANBAUGEBIET→QUALITÄT.

Das Relationenschema LF-ARTIKEL mit Schlüssel {LIEFERANT, ARTIKEL} ist wegen der funktionalen Abhängigkeit ANBAUGEBIET→QUALITÄT nicht in dritter Normalform, denn es gilt keine der geforderten Bedingungen: ANBAUGEBIET enthält keinen Schlüssel und QUALITÄT ist kein Schlüsselattribut. Betrachten wir ein Tupel von LF-ARTIKEL als Beschreibung eines Entity. Das Entity wird identifiziert durch {LIEFERANT, ARTIKEL}. Aber QUALITÄT beinhaltet eine Eigenschaft, die nicht dem Entity als Ganzes zukommt, sondern sich nur auf ANBAUGEBIET bezieht. Die Definition der dritten Normalform verbietet dies für Nichtschlüsselattribute (wie QUALITÄT).

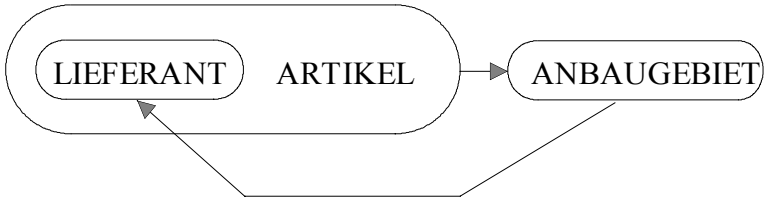
Eine Relation, die in 2NF vorliegt, kann wiederum durch Zerlegung in 3NF überführt werden. Es werden die Attribute aus der Relation herausgenommen, die transitiv abhängig von einem Nichtschlüssel-Attribut sind. Sie werden zusammen mit einer Kopie des bestimmenden Attributes in eine neue Relation aufgenommen.

In unserem Beispiel wird das Attribut QUALITÄT aus der Relation LF-ARTIKEL herausgenommen und bildet zusammen mit einer Kopie des Attributes ANBAUGEBIET die neue Relation
LIEF-QUAL (ANBAUGEBIET, QUALITÄT).
Die verbleibenden Attribute bilden die neue Relation
LF-ART (LIEFERANT, ARTIKEL, ANBAUGEBIET)

Jede Relation in dritter Normalform ist auch in zweiter Normalform; denn eine Verletzung der zweiten Normalform führt zu einer funktionalen Abhängigkeit zwischen einem Nichtschlüssel und einem Nichtschlüsselattribut.

Die dritte Normalform reicht zur vollständigen Vermeidung von Anomalien nicht aus. Das folgende Beispiel zeigt, dass auch Abhängigkeiten zwischen einem Nichtschlüssel und einem Schlüsselattribut zu Anomalien führen. In diesem Fall beinhaltet das Schlüsselattribut eine Eigenschaft, die nicht dem Entity als Ganzes zukommt, sondern sich nur auf die Attribute des Nichtschlüssels bezieht. Das heißt, die Relation ist zwar in dritter Normalform, stellt aber nicht einen Entity-Typ dar.

Beispiel 3.52:
Die Relation LF-ART (LIEFERANT, ARTIKEL, ANBAUGEBIET) mit den Fd's {LIEFERANT, ARTIKEL}→ANBAUGEBIET und ANBAUGEBIET→LIEFERANT ist in dritter Normalform. {LIEFERANT, ARTIKEL} bestimmt das Anbaugebiet. Aus dem Anbaugebiet ist nur der Lieferant, nicht jedoch der Artikel ableitbar. Die folgende Skizze zeigt die funktionalen Abhängigkeiten:



Schlüssel der Relation LF-ART sind {LIEFERANT, ARTIKEL} und {ANBAUGEBIET, ARTIKEL}. Ein mögliches Problem mit dieser Relation ist das folgende: da LIEFERANT zu einem Schlüssel gehört, können wir nicht ohne weiteres einen ARTIKEL und ein bestimmtes ANBAUGEBIET speichern, wenn wir keinen Lieferanten kennen, der diesen ARTIKEL liefert.

Die folgende Normalform berücksichtigt auch Beziehungen zwischen Schlüsselattributen.

BCNF
Boyce-Codd Normalform

Boyce-Codd Normalform (BCNF):
Eine Relation R ist in *Boyce-Codd Normalform*, wenn für jede funktionale Abhängigkeit $X \rightarrow A \in F^+$, $A \notin X$ gilt: X enthält einen Schlüssel für R.

Beispiel 3.53:
In Beispiel 3.52 ist LF-ART nicht in BCNF, weil es die funktionale Abhängigkeit ANBAUGEBIET→LIEFERANT gibt, und ANBAUGEBIET keinen Schlüssel enthält. Um die Boyce-Codd Normalform zu erreichen, müßten wir die Relation zerlegen, z.B. in die beiden Relationen LF-LA (LIEFERANT, ANBAUGEBIET) und LF-AA (ARTIKEL, ANBAUGEBIET).

Jede Relation in Boyce-Codd Normalform ist auch in dritter Normalform; der Unterschied zur dritten Normalform besteht darin, dass nun auch funktionale Abhängigkeiten zwischen Nichtschlüsseln und Schlüsselattributen verboten sind.

Der Fall, dass eine Relation in 3NF nicht in BCNF ist, kann nur unter bestimmten Bedingungen vorkommen: Es müssen mehrere zusammengesetzte Schlüssel in der Relationen existieren, oder es gibt überlappende Schlüssel, d.h. die Schlüssel teilen sich ein Schlüsselattribut. Im obigen Beispiel haben wir zwei zusammengesetzte Schlüssel, die sich im gemeinsamen Schlüsselattribut ARTIKEL überlappen.

Wenn Sie die letzte Zerlegung genau betrachten, fällt Ihnen vielleicht auf, dass im resultierenden Relationenschema eine funktionale Abhängigkeit verloren gegangen ist: Die funktionale Abhängigkeit ANBAUGEBIET $\rightarrow$ LIEFERANT ist in der Relation LF-LA erhalten. Damit können wir die Aussage „Zu jedem Anbaugebiet gibt es genau einen Lieferanten“ festhalten. Die 2. funktionale Abhängigkeit {LIEFERANT, ARTIKEL}$\rightarrow$ANBAUGEBIET ist jedoch nicht mehr darstellbar, da es keine Relation mehr gibt, die LIEFERANT und ARTIKEL in diesen Zusammenhang stellt. Unser Relationenschema verliert damit die Fähigkeit, die Aussage „Ein Lieferant liefert einen Artikel immer nur aus einem Anbaugebiet“ über funktionale Abhängigkeiten auszudrücken. Somit kann es manchmal Sinn machen, die Normalisierung bei der 3. Normalform zu stoppen, da bei den Zerlegungen zur Erlangung der 3. Normalform die funktionalen Abhängigkeiten erhalten bleiben (man spricht dann von *fd-treuen Zerlegungen*). Wie wir gesehen haben, ist die zur Erlangung der BCNF benötigte Zerlegung nicht immer fd-treu.

Aus dem Relationschema, das sich durch diesen Zerlegungsprozess ergibt, läßt sich die Ausgangsrelation durch Join-Operationen wiederherstellen. Daher spricht man von verlustfreien Zerlegungen. Zerlegungen sind immer dann verlustfreu, wenn das Attribut, über das der natürliche Join für die Wiederherstellung der ursprünglichen Relation durchgeführt wird, ein Schlüssel einer der beiden Teilrelationen ist. Der interessierte Leser kann mehr dazu lesen in /O'NEIL94/.

Anmerkung: Um die Verletzung der dritten Normalform oder der Boyce-Codd-Normalform zu testen, reicht es nach der Definition nicht aus, nur die Fd-Menge F zu berücksichtigen, stattdessen müssen alle funktionalen Abhängigkeiten aus F^+ betrachtet werden. Man kann jedoch zeigen, dass die Betrachtung von F ausreicht, wenn F in eine Menge funktionaler Abhängigkeiten umgeformt wird, die nur ein Attribut auf der rechten Seite haben (eine solche äquivalente Umformung ist immer möglich.).

Neben den funktionalen Abhängigkeiten gibt es weitere Arten von Bedingungen (Integritätsbedingungen), die für Relationen gelten können, und die zur Definition weiterer Normalformen herangezogen werden. Wir verzichten darauf, diese Gedanken weiter zu verfolgen; der interessierte Leser sei beispielsweise auf /DATE04/, für weiterführende Studien etwa auf /ULL01/ verwiesen.

Übung 3.21:
Beispiel für die Normalformen eines Relationenschemas.
Gegeben sei das folgende Relationenschema

PROF-STUD (PROF-NR, STUD-NR, VORLESUNG, SCRIPT, ABSCHLUSS)

in erster Normalform. Eine Relation PROF-STUD zum obigen Schema habe folgendes Aussehen:

PROF-STUD	PROF-NR	STUD_NR	VORLESUNG	SCRIPT	ABSCHLUSS
	P1	S1	125	S_125	DIPL.-INFORM.
	P1	S2	225	S_225	DIPL.-MATH.
	P1	S4	225	S_225	DIPL.-INFORM.
	P2	S1	125	S_125	DIPL.-INFORM.
	P2	S2	125	S_125	DIPL.-MATH.
	P2	S3	226	S_226	DIPL.-MATH.
	P3	S2	123	S_123	DIPL.-MATH.
	P3	S4	123	S_123	DIPL.-INFORM.
	P4	S4	125	S_125	DIPL.-INFORM.
	P4	S5	125	S_125	DIPL.-INFORM.
	P5	S5	226	S_226	DIPL.-INFORM.

Es gelte:
a) Ein Student belege maximal 1 Vorlesung je Professor
b) Ein Professor kann mehrere Vorlesungen halten
c) Eine Vorlesung kann von mehreren Professoren gleichzeitig angeboten werden
d) Zu jeder Vorlesung gibt es ein Skript, egal von welchem Professor die Vorlesung gehalten wird
e) Ein Student kann nicht mehrere Abschlüsse gleichzeitig anstreben

Zeigen Sie, dass PROF-STUD weder in zweiter noch in dritter Normalform ist.

3.6.7 Zusammenfassung

Wir haben in diesem Abschnitt einen Blick auf den Entwurf von Datenbank-Schemata geworfen. Der Entwurf startet mit dem datenbank-unabhängigen konzeptuellen Schema, das gemäß vorgegebener Regeln in ein relationales Datenbank-Schema transformiert werden kann. Die vorangehenden Abschnitte haben sich mit Gütekriterien für relationale Datenbankschemata beschäftigt. Dabei stehen die Normalformen für das Ziel, möglichst nur einen Entity-Typ oder einen Beziehungstyp innerhalb eines Relationenschemas zu beschreiben, um so Anomalien zu vermeiden. In den Beispielen zu den Normalformen wurde gezeigt, dass dieses Ziel durch Zerlegen des Ausgangsschemas erreicht werden kann. Fd-Treue und Verlustlosigkeit garantieren, dass die entstehenden Zerlegungen dieselbe semantische Information enthalten wie die Ausgangsrelation. Wir haben gesehen,

dass die Vermeidung von Anomalien manchmal nur dadurch erreicht werden kann, dass einige funktionale Abhängigkeiten aus dem Schema verschwinden (vgl. Unverträglichkeit zwischen BCNF und Fd-Treue). Letztendlich muss die Datenbankadministration entscheiden, welches Ziel ihr in der gegebenen Anwendung wichtiger erscheint.

In den Beispielen zu den Normalformen steckte eine implizite Vorgehensweise zur Erzeugung von Normalformen: Wir haben überprüft, ob die Relationenschemata in der geforderten Normalform sind, und wenn nicht, dann wurde das betreffende Schema weiter zerlegt. Unter der Voraussetzung, dass bereits auf der Ebene des konzeptuellen Entwurfs mit semantischen Modellen modelliert wurde, ist dies ein durchaus praktikables Vorgehen, da bei Einhaltung von wichtigen Modellierungsregeln dabei bereits vernünftige relationale Datenbank-Schemata entstehen. Allerdings ist der Test auf dritte Normalform NP-hart. Aus diesem Grund wurden Algorithmen entwickelt, die eine Zerlegung liefern, die nachweislich in dritter Normalform ist. Der Datenbankadministrator kann diese Algorithmen verwenden, um bei Eingabe eines Anfangsschemas (das im Allgemeinen alle vorkommenden Attribute enthält) und der Menge der semantischen Bedingungen eine vollständige Zerlegung in dritter oder Boyce-Codd-Normalform zu erzeugen. Der interessierte Leser findet mehr Informationen dazu etwa in /MAI86/, /ULL01/ oder /BIM87/.

Literaturverzeichnis

/AST79/ Astrahan, M.M. et al.: System R: A Relational Approach to Data Management. ACM Trans. on Database Systems Vol. 12-5 May 1979), 43 - 48.

/BABA11/ Balzert, Heide; Balzert, Helmut: Lehrbuch der Objektmodellierung: Analyse und Entwurf mit der UML 2. Spektrum Akademischer Verlag (Taschenbuch), 2011)

/BIM87/ Biskup, J.; Meyer, R.: Design of Relational Database Schemes by Deleting Attributes in the Canonical Decomposition, JCSS, 35(1), 1987.

/CHAU98/ Chaudhuri, S.: An Overview of Query Optimization in Relational Systems. Proc. ACM Symp. on Principles of Database Systems. Seattle. 1998. 34-43.

/DATE04/ Date, C.J.: An Introduction to Database Systems. Pearson Addison Wesley, 2004, ISBN 0-321-19784-4.

/DUHA89/ Dutka, Alan F.; Hanson, Howard H.: Fundamentals of Data Normalization, Addison-Wesley, 1989.

/ELNA02/ Elmasri, R.; Navathe, S.B.: Grundlagen von Datenbanksystemen. 3. Auflage, Addison-Wesley, 2002, 1103 S..

/HÄR78/ Härder, T.: Implementierung von Datenbanksystemen. Hanser, 1978.

/MAI86/ Maier, D.: The Theory of Relational Databases, Computer Science Press, Rockville, Md. 1986.

/MITS95/ Mitschang, B.: Anfrageverarbeitung in Datenbanksystemen: Entwurfs- und Implementierungskonzepte. Reihe Datenbanksysteme. Vieweg. 1995.

/O'NEIL 94/ O´Neil, P.: Database : Principles – Programming – Performance. Morgan Kaufmann Publishers, California, 1994

/SQL/ ISO, IEC: Information technology – Database Languages – SQL 2. International Organization for Standardization ISO/IEC 9075/1992; American National Standard X3.135-1992, American National Standards Institute, New York, NY 10036, 11/1992

/TEOR94/ Teorey, Toby J.: Database Modeling & Design – The Fundamental Principles, 2nd Edition. Morgan Kaufmann Publishers, Inc., San Francisco California, 1994.

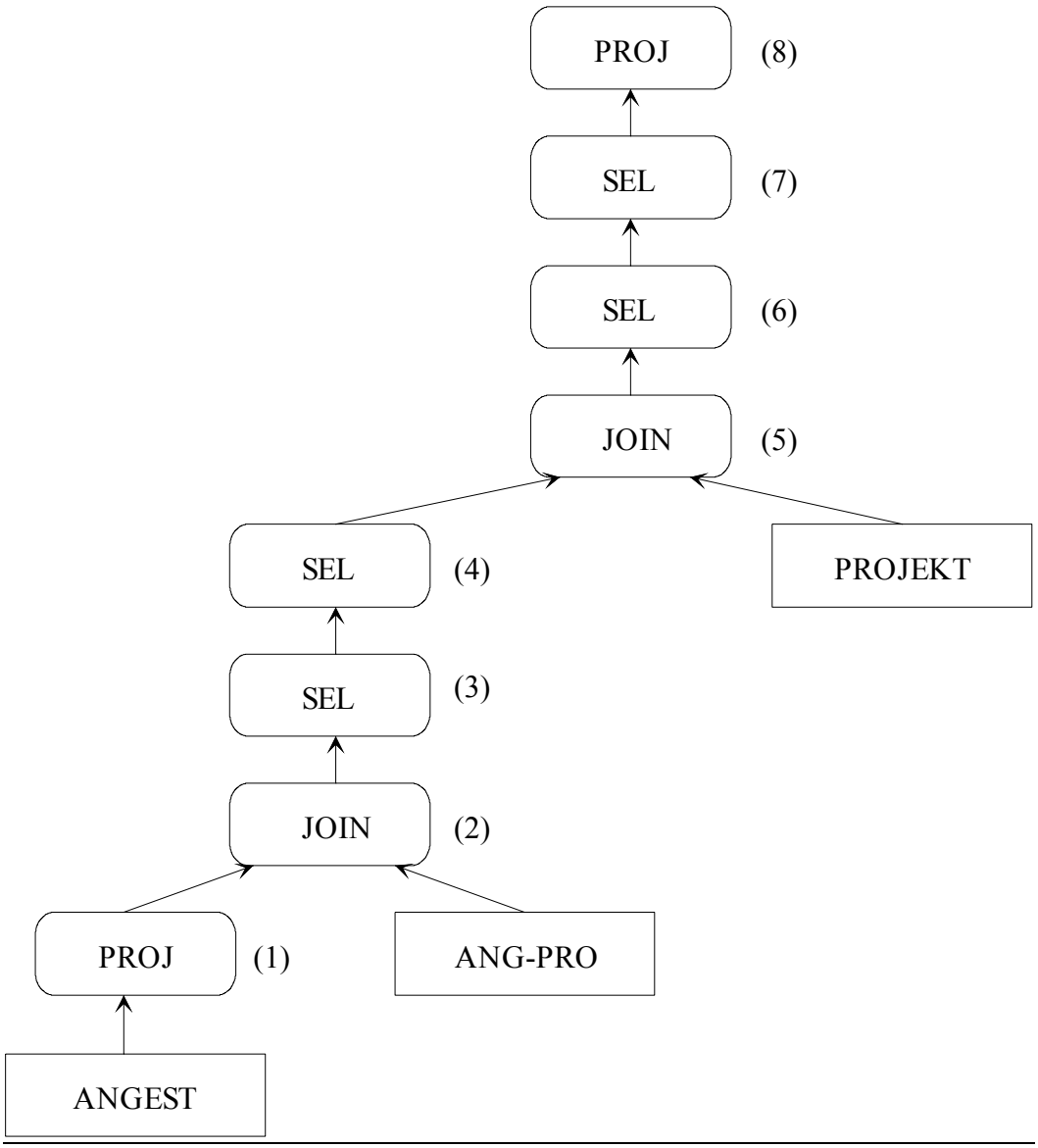
/ULL01/ Ullman, J.D.; Garcia-Molina, H.; Widom, J.: Database Systems: The Complete Book. Prentice Hall, 2001, ISBN 0-130-31995-3.

Datenbanksysteme
Kurseinheit 3
Lösungen zu den Übungsaufgaben

Übung 3.18:

$\pi_{\text{WOHNORT, PNR}}$	(8)
($\sigma_{\text{ABTNR = 5}}$	(7)
($\sigma_{\text{PNR < 30}}$	(6)
($\sigma_{\text{WOHNORT='HAGEN'}}$	(4)
($\sigma_{\text{BERUF='INGENIEUR'}}$	(3)
($\rho_{\text{ANGEST.ANGNR} \leftarrow \text{ANGNR}}$	
($\pi_{\text{ANGNR, WOHNORT, BERUF, ABTNR}}$ (ANGEST)	(1)
)	
$\bowtie_{\text{ANGEST.ANGNR = ANGNR}}$ ANG_PRO	(2)
)	
)	
$\bowtie_{\text{ANGNR = P\_LEITER}}$ PROJEKT	(5)
)	
)	
)	

Operatorbaum für den ursprünglichen Ausdruck:



Regel 1:
Die Selektionen (3) und (4) beziehen sich auf denselben Operanden (JOIN (2)); sie werden zusammengefasst zu (3'):
$\sigma_{\text{WOHNORT='HAGEN'} \wedge \text{ BERUF='INGENIEUR'}}$ (…),
Selektion (4) entfällt.

Regel 2:
Die Selektion (3') wird vor dem Join (2) ausgeführt; die Selektion (6) vor dem Join (5), Selektion (7) vor (2).

Regel 1: Sel(3') und (7) werden zu 3" zusammengefasst:

$\sigma_{\text{ABTNR = 5} \wedge \text{ WOHNORT='HAGEN'} \wedge \text{ BERUF='INGENIEUR'}}$ (…)

Regel 3:
Die Projektion (1) wird direkt nach der Selektion (3'') und (7) durchgeführt (keine Eliminierung von Duplikaten erforderlich).

$\pi_{\text{ANGNR, WOHNORT, BERUF, ABTNR}}$
($\sigma_{\text{ABTNR = 5} \wedge \text{ WOHNORT='HAGEN'} \wedge \text{ BERUF='INGENIEUR'}}$
($\rho_{\text{ANGEST.ANGNR} \leftarrow \text{ANGNR}}$
(ANGEST)
)
)

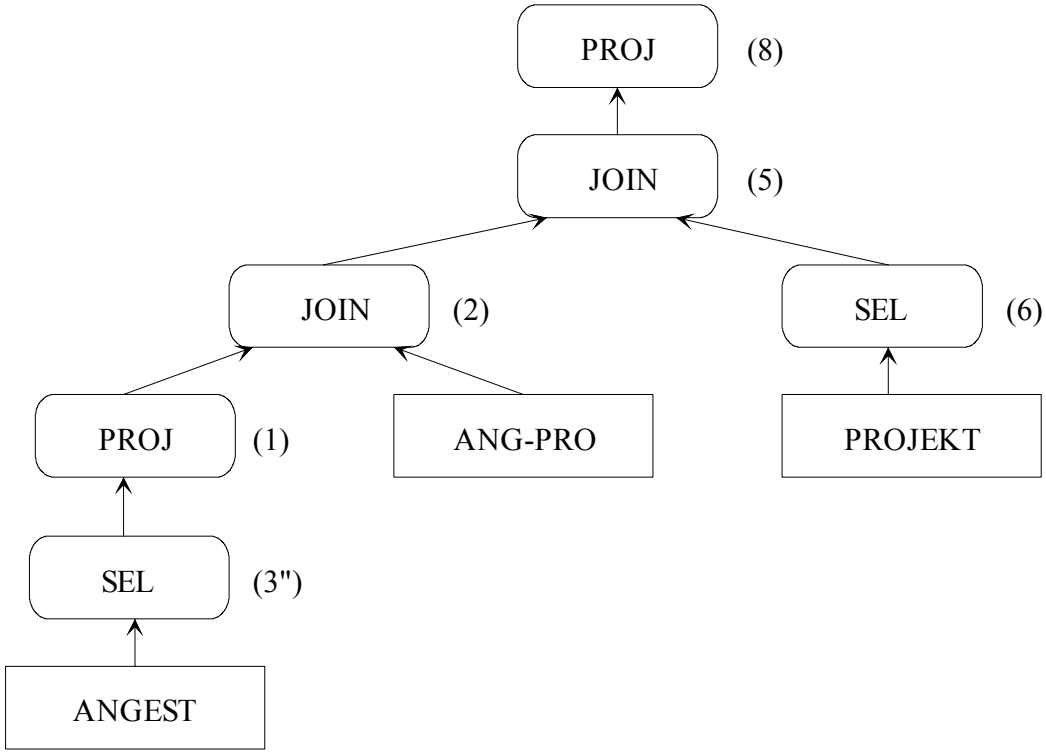
Regel 4:
Die Projektion (8) wird zum Schluss durchgeführt werden (Eliminierung von Duplikaten erforderlich).

Regel 5:
Keine gemeinsamen Operator-Teilbäume vorhanden.

Optimierter Ausdruck:

$\pi_{\text{WOHNORT, PNR}}$	(8)
($\pi_{\text{ANGNR, WOHNORT, BERUF, ABTNR}}$	(1)
($\sigma_{\text{ABTNR = 5} \wedge \text{ WOHNORT='HAGEN'} \wedge \text{ BERUF='INGENIEUR'}}$	(3'')
($\rho_{\text{ANGEST.ANGNR} \leftarrow \text{ANGNR}}$	
(ANGEST)	
)	
)	
$\bowtie_{\text{ANGEST.ANGNR = ANGNR}}$ ANG_PRO	(2)
$\bowtie_{\text{ANGNR = P\_LEITER}}$	(5)
$\sigma_{\text{PNR < 30}}$ (PROJEKT)	(6)
)	

Operatorbaum für den optimierten Ausdruck



Übung 3.19:

$R_{ANGESTELLTER}$(<u>ANGNR</u>, NAME, GEHALTB, BERUF, WOHNORT,
VORGESETZTER)

VORGESETZTER ist die Angestelltennummer des Vorgesetzten eines Angestellten.

Übung 3.20:

Beweis: Es gilt:
(i) $F \subseteq F^+$, denn F impliziert F.
(ii) $F \subseteq H^+ \Rightarrow cl_F(X) \subseteq cl_H(X)$,
f.a. X , laut Hinweis.

„$\Rightarrow$“ Vorauss: $F^+ \subseteq H^+$
$\Rightarrow$ $F \subseteq H^+$, wegen (i)
$\Leftrightarrow$ f.a. $X \rightarrow Y \in F$ gilt $X \rightarrow Y \in H^+$.

„$\Leftarrow$“ Vorauss.: $F \subseteq H^+$
Sei $X \rightarrow Y \in F^+$,
$\Rightarrow$ $Y \subseteq cl_F(X)$ nach Satz aus Abschnitt 3.6
$\Rightarrow$ $Y \subseteq cl_H(X)$ nach (ii) und Vorauss.
$\Rightarrow$ $X \rightarrow Y \in H^+$ nach Satz aus Abschnitt 3.6.2
also gilt $F^+ \subseteq H^+$.

Übung 3.21:

(a) Ermittlung der funktionalen Abhängigkeiten

{PROF-NR, STUD-NR}$\rightarrow$VORLESUNG
VORLESUNG$\rightarrow$SCRIPT
STUD-NR$\rightarrow$ABSCHLUSS

{PROF-NR, STUD-NR}$\rightarrow$SCRIPT

(b) Test auf 2. Normalform
Zunächst müssen wir für jede Teilmenge X der Attribute prüfen, ob alle anderen Attribute hiervon funktional abhängig sind, ob also gilt {PROF-NR, STUD-NR, VORLESUNG, SCRIPT, ABSCHLUSS}$\subseteq cl_F(X)$. Bei einer Menge von 5 Attributen wären dies 2^5=32 verschiedene Teilmengen und somit auch gleichviele Tests. Diese Anzahl kann man durch einige Überlegungen erheblich reduzieren: Die Attribute PROF-NR und STUD-NR kommen in keiner funktionalen Abhängigkeit auf der rechten Seite vor. Daher sind sie nur in $cl_F(X)$, wenn sie schon zu X gehören. Das heißt, jeder Schlüssel für die

gesamte Attributmenge muss PROF-NR und STUD-NR enthalten. Falls {PROF-NR, STUD-NR} bereits ein Schlüssel ist, gibt es keine anderen Schlüssel.

Da

$cl_F(\{\text{PROF-NR,STUD-NR}\})=$

$\{\text{PROF-NR, STUD-NR}\} \cup \{\text{VORLESUNG}\} \cup \{\text{SCRIPT}\} \cup \{\text{ABSCHLUSS}\}$

ist {PROF-NR, STUD-NR} der einzige Schlüssel. Das heißt, die Attribute VORLESUNG, SCRIPT und ABSCHLUSS sind Nichtschlüsselattribute.

Wir prüfen nun die funktionalen Abhängigkeiten der Nichtschlüsselattribute vom Schlüssel.

{PROF-NR,	STUD-NR}	→	VORLESUNG	
	STUD-NR	→	ABSCHLUSS	(*)
{PROF-NR,	STUD-NR}	→	SCRIPT	

Aus (*) folgt, dass PROF-STUD nicht in 2. Normalform und somit auch nicht in dritter Normalform ist.