Lernziele

Nach dem Durcharbeiten der vorliegenden Kurseinheit sollen Sie sich mit den sprachlichen und den benutzerbezogenen Aspekten betrieblicher Datenbankanwendungen soweit vertraut gemacht haben, dass Sie nicht nur

- die verschiedenen Arten von Datenbanksprachen,
- die verschiedenen Klassen von Datenbankbenutzern und
- die verschiedenen Formen der Benutzung von Datenbanken

charakterisieren können, sondern darüberhinaus auch einschätzen können, welche Formen der Datenbankbenutzung unter Verwendung welcher Arten von Datenbanksprachen für

- einen gelegentlichen Datenbankbenutzer,
- einen routinemäßigen Datenbankbenutzer und
- einen professionellen Datenbankbenutzer

prinzipiell in Frage kommen.

Außerdem sollen Sie zwei ausgewählte Datenbanksprachen, nämlich

- die Relationenalgebra und
- die Sprache SQL,

soweit kennengelernt haben, dass Sie die Wirkung der einzelnen Operationen bzw. Anweisungen dieser Sprachen sowie die Wirkung von aus Einzeloperationen zusammengesetzten Sprachkonstrukten - wie sie im Rahmen von Datenbankanwendungen und insbesondere von Datenbankabfragen auftreten - erläutern können. Was die Sprache SQL betrifft, sollen Sie darüberhinaus Anweisungen kennengelernt haben, die es gestatten,

- eine aus Relationen und Zugriffsindizes bestehende Datenbasis zu definieren,
- Daten in die Relationen und Indizes der Datenbasis einzugeben und
- Datenmanipulationen auf der Datenbasis auszuführen,

wobei sich diese Manipulationen auf

- das Einfügen von Daten (INSERT),
- das Ändern von Daten (UPDATE),
- das Löschen von Daten (DELETE) und
- das Auswählen von Daten (SELECT)

beziehen können.

Neben den genannten kenntnisorientierten Lernzielen sollen Sie auch ein verhaltensorientiertes Lernziel erreicht haben: Sie sollen in der Lage sein, auch komplexere Datenbankabfragen mit Hilfe der Relationenalgebra und mit Hilfe der Sprache SQL zu formulieren.

4 Datenbankbenutzung

In diesem Kapitel wird die Betrachtung von Datenbanksystemen auf der logischen Ebene weitergeführt und erweitert. Während im Kap. 3 die Frage im Vordergrund stand, wie man eine Datenbank strukturiert und entwickelt, geht es hier um den Umgang unterschiedlicher Benutzer mit einer Datenbank. Das Mittel für diesen Umgang stellen die in Kap. 4.1 angesprochenen Datenbanksprachen dar. Datenbanksprachen gestatten es den Benutzern, die in einer Datenbank abzulegenden Daten zu definieren und in einer Weise zu manipulieren, wie es der jeweilige Anwendungszweck erfordert.

Inhalt des Kapitels

Je nach Anwendungszweck kann die Manipulation einer Datenbank durch die Benutzer recht unterschiedlich ausfallen. In Kap. 4.2 werden unterschiedliche Benutzungsformen beschrieben und mit bestimmten Klassen von Benutzern in Verbindung gebracht. Außerdem wird gezeigt, dass sich für die einzelnen Benutzungsformen bestimmte Arten von Datenbanksprachen in besonderer Weise eignen.

Eingehender werden zwei Datenbanksprachen behandelt, die Relationenalgebra in Kap. 4.3 und die Sprache SQL in Kap. 6.1. Mit beiden Sprachen wird der herausragenden Bedeutung relationaler Datenbanksysteme Rechnung getragen. Die Relationenalgebra ist deshalb von Interesse, weil sie von ihrem Entwickler - dem Mathematiker CODD - speziell für den Zweck der Manipulation relationaler Datenbasen konzipiert wurde. Eine ganze Reihe der mit der Relationenalgebra formulierbaren Manipulationsmöglichkeiten wurden in anderen Datenbanksprachen nachgebildet. So auch in der Sprache SQL, die mittlerweile eine derart weite Verbreitung gefunden hat, dass sie als eine Standardsprache gelten kann. Über SQL-Grundkenntnisse sollte daher jeder Wirtschaftswissenschaftler verfügen, der sich mit datenbankgestützten Anwendungen im betrieblichen Bereich auseinandersetzt.

4.1 Datenbanksprachen

Ein grober Überblick über Datenbanksprachen wurde bereits in Kap. 2.1 gegeben. Unterschieden wurde nach Datenbeschreibungssprachen, Datenmanipulationssprachen und Speicherbeschreibungssprachen. Für die Benutzer einer Datenbank sind lediglich Datenbeschreibungs- und Datenmanipulationssprachen von Interesse. Diese Arten von Sprachen werden im vorliegenden Kapitel behandelt.

a) Datenbeschreibungssprachen

Zweck von Datenbeschreibungssprachen

Datenbeschreibungssprachen dienen der Definition und Beschreibung von Datenmodellen, vor allem auch von DBVS-gestützten Datenmodellen. Aus dieser allgemeinen Zweckbestimmung resultiert eine Reihe von speziellen Funktionen, welche die Schemadefinition bzw. die Definition von Datenstrukturen, die Datenkonsistenz, die Zugriffsberechtigung und - bei den meisten DBVS - auch die physische Datenorganisation betreffen.

Auf die genannten Funktionen von Datenbeschreibungssprachen wird nachfolgend eingegangen. Konkrete Anweisungen zur Datenbeschreibung werden in Kap. 4.4, im Zu-

sammenhang mit der Behandlung der Sprache SQL, dargestellt. Bereits an dieser Stelle sei darauf hingewiesen, dass sich der Sprachumfang von SQL sowohl auf Anweisungen zur Datendefinition als auch zur Datenmanipulation erstreckt. Die Sprache SQL fasst also die Funktionen von Datenbeschreibungssprachen und die Funktionen von Datenmanipulations-sprachen in einer Sprache zusammen. Nun aber zu den Funktionen von Datenbeschreibungssprachen.

(1) Schemadefinition bzw. Definition von Datenstrukturen

Gemeint sind hier das konzeptionelle Schema und die externen Schemata. Eine explizite Schemabeschreibung gestatten nur diejenigen DBVS, welche Schemata auch verwenden und als solche ausdrücklich bezeichnen. Hierzu gehören vor allem die am CODASYL-DBTG-Modell orientierten Systeme. Nicht alle diese Systeme unterscheiden zwischen dem konzeptionellen Schema und externen Schemata.

explizite und implizite Schemabeschreibung

Viele DBVS verwenden das Schemakonzept nicht oder nicht in ausdrücklicher Form. Ohne eine Schemabeschreibung werden in solchen Fällen mehrere voneinander unabhängige oder (teils) miteinander verbundene Datenstrukturen definiert.

In beiden Fällen erstreckt sich die Datendefinition auf Datenobjekte und Beziehungen zwischen Datenobjekten, meist in eigenständigen Sektionen. Beispielsweise kann die Beschreibung von Datenobjekten Angaben wie Name eines Objekts, Datentyp, Stellenanzahl usw. enthalten. Die Beschreibung von Datenobjekten und Beziehungen bildet den umfangreichsten Teil und den Kernbestandteil der Datenbeschreibung. Die übrigen Beschreibungsteile haben ergänzenden Charakter.

(2) Datenkonsistenz

Datenbeschreibungssprachen verfügen in der Regel über Sprachkomponenten, die auf die Gewährleistung der Datenkonsistenz abzielen. So lassen sich z.B. im Falle der in Kap. 4.4 beschriebenen Datenbeschreibungssprache des Systems R mittels der ASSERT-Anweisung Konsistenzbedingungen für ausgewählte Daten einer Datenbasis formulieren. In Kap. 6.1 wird der Gebrauch der ASSERT-Anweisung an einigen Beispielen demonstriert.

ASSERT-Anweisung

Die Anordnung von Konsistenzbedingungen in einer Datenbeschreibung kann unterschiedlich geregelt sein. Bei älteren DBVS werden die Konsistenzbedingungen meist in die Beschreibung der Datenstrukturen einbezogen. Fortgeschrittenere DBVS sehen dagegen eigenständige Sektionen für Konsistenzregeln vor.

(3) Zugriffsberechtigung

Üblicherweise erstreckt sich der Funktionsumfang einer Datenbeschreibungssprache auch auf die Festlegung von Zugangsrechten. Für die definierten Datenstrukturen können in Abhängigkeit von der Benutzerart Schreibrechte, Leserechte oder keine Zugangsrechte vorgesehen sein.

Zugangsrechte

In der Vergangenheit wurden bei vielen DBVS die Zugangsrechte auf der Ebene der externen Schemata geregelt. Eine solche Regelung ist deshalb ungünstig, weil einmal gewährte Privilegien kaum noch revidierbar sind. Sinnvoller ist es, die Zugangsrechte im konzeptionellen Schema festzulegen und damit der ausschließlichen Kontrolle durch den Datenbankadministrator zu unterwerfen.

(4) Physische Datenorganisation

Unerfreulicherweise verfügen nahezu alle Datenbeschreibungssprachen über Sprachmöglichkeiten, welche die physische Ebene betreffen. Beispielsweise gestatten diese Sprachkomponenten die Festlegung der Speicherungsform von Daten oder die Definition von Zugriffspfaden. Angaben zur physischen Datenorganisation sollten eigentlich dem internen Schema vorbehalten bleiben, da nur dann Änderungen der physischen Datenorganisation nicht auf die konzeptionelle Ebene durchschlagen.

Zugriffspfade

b) Datenmanipulationssprachen

Datenmanipulationssprachen gestatten die Manipulation von Datenbanken. Der Begriff "Manipulation" ist hierbei weit auszulegen. Er umfasst folgende Funktionen:

- Einrichten und Erzeugen einer Datenbank oder von Teilen einer Datenbank.
- Lesen von Daten einer Datenbank, wobei die zu lesenden Daten gegebenenfalls komplexe Auswahlkriterien erfüllen müssen.
- Mutieren von Daten einer Datenbank, d.h. Einfügen, Ändern oder Löschen von Daten; auch die zu mutierenden Daten müssen gegebenenfalls komplexe Auswahlkriterien erfüllen.
- Aufbereiten von auszugebenden Daten; für die Datenausgabe stehen teils auch leistungsfähige Reportgeneratoren zur Verfügung.

Begriff der Datenmanipulation

Die Menge der Datenmanipulationssprachen kann man nach unterschiedlichen Kriterien in Gruppen einteilen. Von Bedeutung ist vor allem die Unterscheidung nach selbständigen und eingebetteten sowie nach deskriptiven und prozeduralen Sprachen. Darüberhinaus existieren Sprachen, die speziell auf die Manipulation von relationalen Datenbanken zugeschnitten sind.

Arten von Datenmanipulationssprachen

Selbständige und eingebettete Datenmanipulationssprachen

Selbständige (engl. self containing) Datenmanipulationssprachen sind eigenständige Sprachen, die in der Regel über Sprachkomponenten für sämtliche der oben genannten Funktionen verfügen. Bei der Manipulation einer Datenbank mit Hilfe einer selbständigen Sprache ist man an keine andere Sprache gebunden. Meist stellen Abfragesprachen, d.h. Sprachen zur Formulierung von Abfragen an Datenbanken, selbständige Datenmanipulationssprachen dar.

Eingebettete (engl. embedded) Datenmanipulationssprachen können nur in Verbindung mit einer Programmiersprache benutzt werden. Die auch als Wirtssprache (engl. host language) bezeichnete Programmiersprache wird mittels der eingebetteten Sprache um

Wirtssprache

datenbankbezogene Anweisungen ergänzt. Daten, die beispielsweise mit Anweisungen einer eingebetteten Sprache aus einer Datenbank extrahiert werden, können nun unter Nutzung des vollen Funktionsumfangs der Wirtssprache verarbeitet werden.

Deskriptive und prozedurale Datenmanipulationssprachen

Bei deskriptiven Datenmanipulationssprachen stehen dem Benutzer Sprachelemente zur Spezifikation des gewünschten Ergebnisses einer Manipulation - also des "was" - zur Verfügung. Anzugeben ist lediglich, was mit welchen Daten geschehen soll. Oder anders ausgedrückt, welche Daten ausgegeben, geändert, gelöscht usw. werden sollen. Ein Algorithmus zur Durchführung der benannten Manipulationen wird nicht formuliert. Deskriptive Sprachen gestatten eine Manipulation von Daten auf einer höheren Sprachebene. Die Sprachen dieser Ebene zählt man zu den Programmiersprachen der 4. Generation. Typische Operationen auf Daten sind beispielsweise das Auswählen (Schlüsselwort SELECT), das Einfügen (Schlüsselwort INSERT), das Ändern (Schlüsselwort MODIFY) und das Entfernen (Schlüsselwort DELETE).

Sprachen der 4. Generation

Bei prozeduralen Datenmanipulationssprachen muss der Benutzer dagegen explizit den Algorithmus - also das "wie" - formulieren, mit dem das gewünschte Ergebnis erzeugt werden soll. Zu diesem Zweck kann er auf Sprachelemente zurückgreifen, die Operationen auf Datenobjekten oder Datenstrukturen darstellen. Typische Operationen auf Datenobjekten sind beispielsweise das Lesen (Schlüsselwort READ) und Schreiben (Schlüsselwort WRITE) auf Dateien. Mittels derartiger Operationen hat der Benutzer seine datenbankbezogenen Auswertungen und Anwendungen Schritt für Schritt zu programmieren. Prozedurale Datenmanipulationssprachen lassen sich etwa der Sprachebene von problemorientierten Programmiersprachen, also von Sprachen der 3. Generation, zuordnen.

Sprachen der 3. Generation

Sprachen für relationale Systeme

Speziell auf die Manipulation relationaler Systeme sind zwei Arten von Sprachen zugeschnitten, algebraische Sprachen und Prädikatenkalkülsprachen. Algebraische Sprachen basieren auf der Relationenalgebra und gestatten es dem Benutzer, erwünschte Ergebnisrelationen als Resultate von Mengenoperationen wie Vereinigung, Durchschnitt, Differenz usw. zu beschreiben. Bei der Benutzung von Prädikatenkalkülsprachen wird dagegen eine Ergebnisrelation durch Eigenschaften bzw. Prädikate beschrieben, welche die Elemente der Ergebnisrelation aufweisen müssen. Bei der Bildung von Prädikaten können sowohl logische Operatoren, z.B. UND, ODER und NICHT, als auch Vergleichsoperatoren, z.B. <, = und >, zur Anwendung kommen.

Mittels der bereits angesprochenen Kriterien "Sprachgeneration" und "Selbständigkeit" lassen sich die Datenmanipulationssprachen in vier Gruppen einteilen. Ein entsprechendes Einteilungsschema, das pro Gruppe ein Beispiel für eine zutreffende Sprache enthält, zeigt die Abb. 4.1.

Klassifizierung der Datenmanipulationssprachen

Bei der in Kap. 4.3 behandelten Relationenalgebra handelt es sich also um eine prozedurale, selbständige Datenmanipulationssprache, während die in Kap. 4.4 vorgestellte selbständige Sprache SQL deskriptiven Charakter besitzt. Als Beispiel für eine prozedurale eingebettete Datenmanipulationssprache wurde die von der CODASYL-DBTG

vorgeschlagene Spracherweiterung der Sprache COBOL angegeben und als Beispiel für eine

Sprachgeneration / Selbständigkeit	prozedurale Datenmanipulationssprachen	deskriptive Datenmanipulationssprachen
selbständige Datenmanipulationssprachen	relationale Sprachen (Relationenalgebra)	SQL (Structured Query Language)
eingebettete Datenmanipulationssprachen	CODASYL-DBTG/ COBOL	MODULA-2/R

Abb. 4.1. Klassifizierung von Datenmanipulationssprachen.

deskriptive, eingebettete Sprache die Einbettung der Sprache R in die Programmiersprache MODULA-2.

Übungsaufgabe 4.1

Betrachtet werde ein Benutzer, der stark variierende Abfragen an eine Datenbank richtet. Sollte dieser Benutzer eine prozedurale oder eher eine deskriptive Datenmanipulationssprache verwenden? Begründen Sie kurz Ihre Antwort.

4.2 Benutzungsformen und Benutzerklassen

Nicht alle Benutzer von Datenbanken haben gleichgelagerte Interessen. So existieren z.B. Benutzer, die gelegentlich Abfragen an eine Datenbank richten, und andererseits auch Benutzer, die regelmäßig umfangreiche Datenbankanwendungen abwickeln. Die Interessen von Datenbankbenutzern beeinflussen offensichtlich die Benutzungsform. Das vorliegende Kapitel beschäftigt sich mit dem Problemkreis der Datenbankbenutzung. Behandelt werden zunächst

Benutzerklassen und Benutzungsformen

- die unterschiedlichen Klassen von Datenbankbenutzern und
- die verschiedenen Formen der Datenbankbenutzung.

Anschließend werden zwei Benutzungsformen anhand je eines Beispiels ausführlicher dargestellt, nämlich

- die Ausführung parametrisierter Anwendungsprogramme und
- die datenbankgestützte Stapelverarbeitung.

a) Benutzerklassen

Benutzer:

Die Benutzer von Datenbanken lassen sich in drei Klassen einteilen: gelegentliche Benutzer, routinemäßige Datenbankanwender und Datenbankspezialisten.

gelegentliche

Gelegentliche Benutzer (engl. casual users) greifen fallweise auf die in einer Datenbank abgelegten Informationen zu, typischerweise in der Form der freien Abfrage. Diesem Benutzertyp gehören vor allem Vertreter des Management, Stabsangehörige, Revisoren usw. an, die als Nicht-Datenbankspezialisten unter Verwendung einer (möglichst) einfach zu erlernenden Abfragesprache mit einer Datenbank kommunizieren und auf diese Weise primär spontan auftretende Informationsbedürfnisse befriedigen.

routinemäßige

Routinemäßige Datenbankanwender (engl. parametic users) sind ebenfalls Nicht- Datenbankspezialisten, die allerdings - im Unterschied zu gelegentlichen Benutzern - regelmäßig datenbankgestützte Anwendungen abwickeln. Dieser Benutzertyp umschließt Fachleute der verschiedensten Anwendungsgebiete, die als Disponenten, Sachbearbeiter, Kundenberater usw. Geschäftsvorfälle meist mittels interaktiver Anwendungsprogramme bearbeiten. Den spezifischen Gegebenheiten eines Geschäftsvorfalles wird dabei durch die Eingabe von Parameterwerten Rechnung getragen.

professionelle

Datenbankspezialisten sind professionelle Datenbankbenutzer (engl. professional users), die für die Entwicklung von Datenbankanwendungen, die Einrichtung von Datenbanken und den Datenbankbetrieb verantwortlich zeichnen. Diese Klasse von Benutzern repräsentiert Datenbank-Programmierer und Datenbankadministratoren, also Personen, die - im Unterschied zu den beiden anderen Typen von Benutzern - mit einer Datenbank nicht als Anwender umgehen.

Übungsaufgabe 4.2

Betrachtet werden die eben erläuterten Klassen von Datenbankbenutzern. Geben Sie für jede dieser Klassen ein konkretes Beispiel für die Datenbankbenutzung an.

b) Benutzungsformen

drei Benutzungsformen

Bei dem Einsatz von Datenbanken können folgende Benutzungsformen unterschieden werden:

- freie Datenbankabfragen,
- interaktive Anwendungsprogramme mit vorgegebenen Abfragen,
- datenbankgestützte Stapelverarbeitung.

Den beiden erstgenannten interaktiven Benutzungsformen steht die Abwicklung von Stapelverarbeitungsprogrammen auf Datenbanken gegenüber.

Freie Datenbankabfrage

ad hoc-Abfragen

Eine freie Datenbankabfrage soll lediglich ein (spontan) aufgetretenes Informationsbedürfnis befriedigen. Die Abfrage wird vom Benutzer beim Eintritt des Informationsbedürfnisses formuliert. Keinesfalls ist sie ein vorformulierter Bestandteil eines (weiterverarbeitenden) Datenbank-Anwendungsprogramms. Der weiteren Erläuterung dieser Benutzungsform diene die Abb. 4.2.

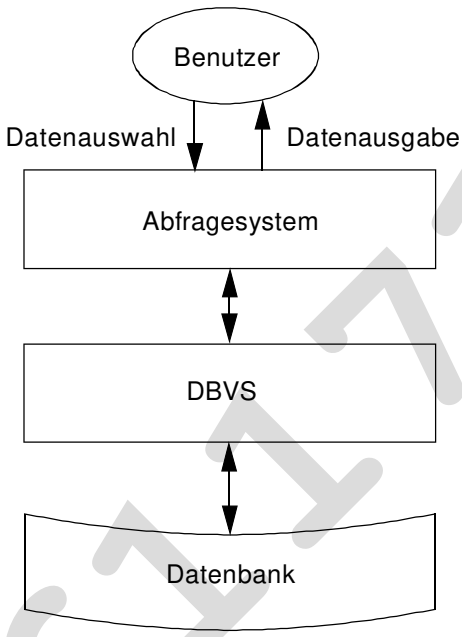


Abb. 4.2. Benutzungsform der freien Abfrage.

Laut Abb. 4.2 umfasst eine Abfrage zwei Teile, die Datenauswahl und die Datenausgabe (vgl. hierzu auch ZEHNDER 1985, S. 100). Die Datenauswahl repräsentiert den Frageteil einer Abfrage und die Datenausgabe den Antwortteil.

Datenauswahl

Bei der Datenauswahl geht es um die Selektion der Teilmenge von Daten einer Datenbank, welche den von dem Benutzer vorgegebenen Frage- bzw. Qualifikationskriterien genügt. Die Qualifikationskriterien beziehen sich auf die Inhalte der gespeicherten Daten. Je nach der Reichweite der Kriterien kann sich die relevante Teilmenge von Daten

- aus den Werten von einem oder mehreren Attributen,
- entnommen aus einer Entität oder mehreren Entitäten, welche
- einer oder mehreren Entitätsmengen bzw. Dateien entstammen,

zusammensetzen. Die zu erfüllenden Qualifikationskriterien bestimmen die Form der Abfrage. Unterscheiden lassen sich folgende Frageformen:

Frageformen

(1) Einzelfragen

Eine Einzelfrage bezieht sich stets auf eine Entitätsmenge bzw. Datei. Gesucht werden Attributwerte einer oder mehrerer Sätze der Datei. Beispiele für Einzelfragen sind:

Beispiele für Einzelfragen

- Welchen Wert besitzt das Attribut *TelefonNr* des in der Datei *Kundenstamm* verzeichneten Kunden mit der Kundennummer *307 512?*
- Wie lauten die Namen der in der Lieferantendatei verzeichneten Lieferanten, für die das Attribut *Ort* den Wert *Berlin* besitzt?

Während bei der ersten Frage genau ein Satz angesprochen wird, können bei der zweiten Frage mehrere Sätze relevant sein.

(2) Zusammengesetzte Fragen

Zusammengesetzte Fragen bestehen aus Kombinationen von Einzelfragen. Anders als Einzelfragen können sie sich auf mehrere Entitätsmengen bzw. Dateien beziehen. Den Unterschied zu Einzelfragen möge ein Beispiel verdeutlichen:

Beispiel für eine zusammengesetzte Frage

- Wie lauten die Teilenummern der in der Teilestrukturdatei verzeichneten Teile, welche als Unterteile in das Oberteil mit der Teilenummer *2905* eingehen UND welche von dem Lieferanten mit der Bezeichnung *Getriebeteile KG* geliefert werden?

Auszuwerten ist in diesem Fall ein Ausdruck, der aus zwei mit dem logischen Operator UND verknüpften Einzelfragen besteht. Es ist davon auszugehen, dass die Teilelieferanten nicht in der Teilestrukturdatei, sondern z.B. in der Teiledatei verzeichnet sind. Die Auswertung erstreckt sich daher über zwei Dateien.

(3) Abgestufte Fragen

Navigieren

Während Einzelfragen und zusammengesetzte Fragen vollständig formulierte Suchausdrücke darstellen, sind abgestufte Fragen a priori nicht ausformuliert. Vielmehr verkörpern sie eine Folge von Fragen und Antworten, die schrittweise und unter Verwertung von Zwischenergebnissen zu den gesuchten Informationen führt. Dieses fortschreitende Herumsuchen in einer Datenbank bezeichnet man auch als Navigieren.

Datenausgabe

Neben den Qualifikationskriterien enthält eine Abfrage regelmäßig auch Angaben zur Datenausgabe. Insbesondere zum Umfang der Ausgabe und zur Ausgabeform. Was den Ausgabeumfang betrifft, ist zu unterscheiden zwischen

- der unmittelbaren Ausgabe der auf eine Frage zutreffenden Daten und
- der Ausgabe von Informationen über die Anzahl der zutreffenden Daten.

Im zweiten Fall kann der Benutzer z.B. entscheiden, ob die zutreffenden Daten ausgegeben werden sollen, oder ob die gestellte Frage wegen der zu großen Anzahl der auszugebenden Daten präzisiert und erneut eingegeben werden soll. Angaben zur Ausgabeform betreffen die Anordnung der auszugebenden Daten auf dem Ausgabemedium. Denkbar sind z.B. sortierte oder unsortierte Datenanordnungen sowie bloße Aneinanderreihungen von Daten oder tabellarische Anordnungen.

Welche Gestaltungsmöglichkeiten dem Benutzer in Bezug auf die Datenauswahl und die Datenausgabe zur Verfügung stehen, hängt von dem verwendeten Abfragesystem - genauer: von der verwendeten Datenmanipulationssprache - ab. Neben einer Datenmanipulationssprache umschließt ein Abfragesystem als zweite Komponente einen Sprachübersetzer. Die Aufgabe des Sprachübersetzers besteht darin, Abfragen in eine Form zu übersetzen, die für das DBVS - das ja letztlich die Abfragen abwickelt - verständlich und ausführbar ist. Datenmanipulationssprachen, die speziell dem Zweck der freien Abfrage dienen, bezeichnet man auch als Abfragesprachen.

Übersetzung von Abfragen

Anwendungsprogramme mit vorgegebenen Abfragen

Auch bei dieser Benutzungsform treten die Datenauswahl und die Datenausgabe als charakteristische Elemente auf. Allerdings findet das Wechselspiel von Datenauswahl und Datenausgabe nicht unmittelbar zwischen dem Benutzer und dem Datenbanksystem statt, sondern zwischen einem Anwendungsprogramm und dem Datenbanksystem. Das Anwendungsprogramm sieht die Bearbeitung bestimmter Vorfälle vor. Beispielsweise das Erfassen von Kundenaufträgen, das Erstellen von Angeboten, das Aktualisieren von Stammdaten usw. Der im Zuge solcher Arbeitsabläufe auftretende Informationsbedarf wird durch vorgegebene Abfragen, die einen festen Bestandteil des Anwendungsprogramms bilden, befriedigt. Der von dem interaktiven Anwendungsprogramm geführte Benutzer trägt gegebenenfalls - auf Anforderung durch das Programm - durch die Eingabe von Parametern zur Anpassung der Abfragen an die Besonderheiten der jeweils bearbeiteten Vorfälle bei. Eine schematische Darstellung dieser Benutzungsform zeigt die Abb. 4.3.

vorgegebene Abfragen

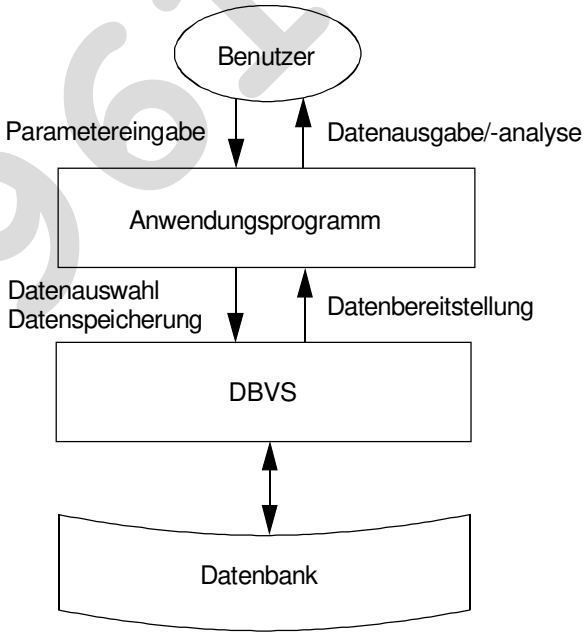


Abb. 4.3. Anwendungsprogramm mit vorgegebenen Datenbankabfragen.

Daten, die auf eine vorgegebene Abfrage zutreffen, stellt das DBVS dem Anwendungsprogramm zur Verfügung. Die bereitgestellten Daten können nun z.B. ausgegeben werden oder in einen Verarbeitungsprozess eingehen.

Anweisungsgruppen:

Grundsätzlich besteht ein Anwendungsprogramm mit vorgegebenen Abfragen aus zwei Gruppen von Anweisungen:

Steueranweisungen

- Anweisungen ohne Bezug zur Datenbank; sie dienen der Steuerung des gesamten Arbeitsablaufs, der Verarbeitung bereitgestellter Daten, der Ausgabe von Daten usw.

Datenbankoperationen

- Anweisungen mit Bezug zur Datenbank; sie repräsentieren die in den Abfragen und auch außerhalb von Abfragen - z.B. im Rahmen von Mutationen - auftretenden Datenbankoperationen wie Lesen, Einfügen, Ändern, Löschen usw.

Besonders deutlich lassen sich die beiden Gruppen im Fall der Verwendung einer eingebetteten Datenmanipulationssprache abgrenzen. Die Anweisungen ohne Datenbankbezug sind dann in der Wirtssprache und die Anweisungen mit Datenbankbezug in der eingebetteten Datenmanipulationssprache abgefasst.

Datenbankgestützte Stapelverarbeitung

Stapelverarbeitungsprogramm als Benutzer

Ein interagierender menschlicher Benutzer tritt bei dieser Benutzungsform nicht auf. Als benutzende Instanz fungiert vielmehr ein Stapelverarbeitungsprogramm. Es bearbeitet während eines Programmlaufes sukzessive eine Menge von angesammelten Geschäftsvorfällen. Die Bearbeitung eines Vorfalls erfordert in der Regel eine oder mehrere Datenbankoperationen, die - wie bei der zuvor beschriebenen Benutzungsform - dem Lesen, Einfügen, Ändern, Löschen usw. von Daten dienen können. Die Abb. 4.4 veranschaulicht die datenbankgestützte Stapelverarbeitung.

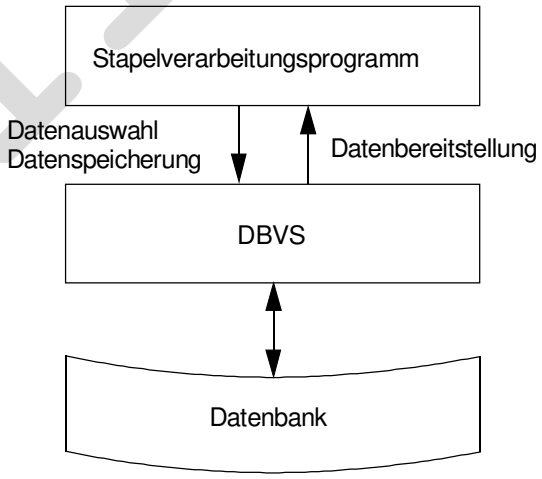


Abb. 4.4. Datenbankgestützte Stapelverarbeitung.

Ebenso wie Anwendungsprogramme mit vorgegebenen Abfragen umfassen datenbankorientierte Stapelverarbeitungsprogramme neben allgemeinen Anweisungen auch datenbankbezogene Anweisungen. Letztere dienen den bereits dargelegten Zwecken.

Bedeutung der Stapelverarbeitung

Früher, als das Gros der betrieblichen Informationsverarbeitungsaufgaben zentral im Batchbetrieb abgewickelt wurde, erfreute sich die datenbankgestützte Stapelverarbeitung einer weiten Verbreitung. Mit dem Aufkommen dezentraler, interaktiver Verarbeitungsformen hat sich das Bild geändert. Heute werden die meisten betrieblichen Datenbankanwendungen von Sachbearbeitern, Disponenten, Revisoren, Kundenberatern, Stabsmitarbeitern usw. interaktiv und unmittelbar am (dezentralen) Arbeitsplatz abgewickelt. Und

zunehmend beschränkt sich die datenbankgestützte Stapelverarbeitung auf Maßnahmen der Datenbankpflege wie Datenbankreorganisation und Datensicherung.

Übungsaufgabe 4.3

Nennen sie je ein typisches betriebliches Anwendungsbeispiel für die Benutzungsformen der freien Abfrage, der interaktiven Verarbeitung mit vorgegebenen Abfragen und der datenbankgestützten Stapelverarbeitung. Wählen Sie im zweiten Fall eine Datenbankanwendung, bei welcher der Benutzer einen oder mehrere Parameter einzugeben hat und benennen Sie den oder die Parameter.

Übungsaufgabe 4.4

In den Abbildungen 4.3 und 4.4 tritt je ein Pfeil auf, der neben "Datenauswahl" auch mit "Datenspeicherung" beschriftet ist. Erläutern Sie, was mit "Datenspeicherung" gemeint ist.

SQL-Abfrage

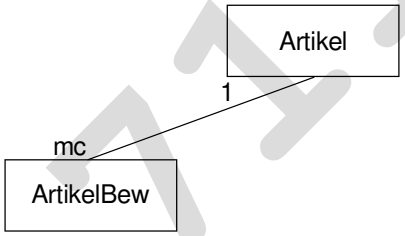
c) Beispiel für eine datenbankgestützte Anwendung mit vorgegebenen Abfragen

Ein vollständiges Anwendungsprogramm kann hier nicht angegeben werden, da es zu viel Platz beanspruchen würde. Wohl kann aber anhand eines Programmausschnitts, nämlich einer Prozedur, die Arbeitsweise eines Anwendungsprogramms mit vorgegebenen Datenbankabfragen demonstriert werden.

Als Hintergrund für das nachfolgend betrachtete Beispiel 4.1 dient das in Kap. 3.5.1 zur Erläuterung der Schritte des logischen Datenbankentwurfs verwendete Beispiel 3.6.

Beispiel 4.1

Betrachtet werde ein Ausschnitt aus dem in Beispiel 3.6 entwickelten global normalisierten Datenmodell, der aus zwei Entitätsmengen und einer 1-mc-Beziehung besteht:



Für den hier verfolgten Zweck genügt die Beschreibung der beiden Entitätsmengen mit folgenden, gegenüber Beispiel 3.6 vereinfachten Relationen:

Artikel(<u>ArtikelNr</u>, Bezeichnung, TechnDaten, Preis, Bestand, MindBestand,
DispoBestand, AuftrBestand)

ArtikelBew(<u>ArtikelNr</u>, <u>ProjektNr</u>, <u>BewDatum</u>, Menge, DispoMenge)

Vorgestellt wird in diesem Beispiel eine Prozedur *$liefere*, die dem Erfassen von Artikellieferungen dient. Artikellieferungen sind in der Relation *ArtikelBew* zu verbuchen. Ein Artikelzugang darf laut Datenmodell allerdings nur dann in die Relation *ArtikelBew* eingetragen werden, wenn zu diesem Artikel bereits ein Eintrag in der Relation *Artikel* existiert. Ist dies nicht der Fall, so muss zuerst der betroffene Artikel in die Relation *Artikel* eingefügt werden. Danach kann die Verbuchung des Artikelzugangs erfolgen.

Datenbankverwaltungssystem ZIM/SQL

Zur Formulierung der Prozedur *$liefere* wird das Datenbankverwaltungssystem ZIM/SQL verwendet. Der Begriff ZIM bezeichnet ein relationales Datenbankverwaltungssystem und zugleich auch eine Sprache, welche es gestattet, ZIM-Datenbanken aufzubauen und zu manipulieren. Eine Version des Systems ZIM kann in Verbindung mit der Datenmanipulationssprache SQL eingesetzt werden; die Bezeichnung dieser ZIM-Version lautet ZIM/SQL. Bei Verwendung von ZIM/SQL dürfen ZIM-Anweisungen und SQL-Anweisungen beliebig aufeinander folgen. Bei der nachfolgend dargestellten Prozedur werden SQL-Anweisungen für Datenbankoperationen und ZIM-Anweisungen für sonstige Operationen verwendet.

Die in ZIM/SQL abgefasste Prozedur *$liefere* lautet:

```
PROCEDURE $liefere
FORM SET EXIT Escape                          % Abbruchtaste der Maske
LET $TRANSMITKEY = " "                        % Initialisierung Abbruchtaste
WHILE $TRANSMITKEY <> "Escape"                % Solange keine Abbruchtaste
      FORM SELECT fArtikelBew                 % Formular anwählen
      FORM DISPLAY INPUT                      % Daten eingeben
      IF $TRANSMITKEY <> "Escape"                     % Falls kein Abbruch
       TRANSACTION                            % Beginn einer Transaktion
        SELECT * FROM Artikel WHERE \         % Zugriff zu Relation Artikel
           Artikel.ArtikelNr. = fArtikelBew.\
           ArtikelNr
        IF $SETCOUNT = 0                      % Falls Artikel nicht vorhanden
           FORM SELECT fArtikel               % Formular anwählen
           FORM DISPLAY INPUT                 % Daten eingeben
           INSERT INTO Artikel VALUES ( \     % Einfügen eines Artikels in die
             fArtikel.ArtikelNr,\             % Relation Artikel
             fArtikel.Bezeichnung, fArtikel.TechnDaten,\
             fArtikel.Preis, fArtikel.Bestand,\
             fArtikel.MindBestand, fArtikel.DispoBestand,\
             fArtikel.AuftrBestand)
        ENDIF                                 % $SETCOUNT
        INSERT INTO ArtikelBew VALUES ( \     % Einfügen eines Artikelzu-
           fArtikelBew.ArtikelNr,\            % gangs in die Relation
           fArtikelBew.ProjektNr,\            % ArtikelBew
           fArtikelBew.BewDatum,\
           fArtikelBew.Menge,\
           fArtikelBew.DispoMenge,\
       ENDTRANSACTION                         % Ende der Transaktion
      ENDIF                                   % $TRANSMITKEY
ENDWHILE
ENDPROCEDURE
```

SQL-Abfrage

Einfügen mit SQL

Einfügen mit SQL

Erläuterung des Beispiels

Zur Darstellungsform der Prozedur *$liefere* ist folgendes anzumerken:

- Die nur aus Großbuchstaben bestehenden Wörter sind Schlüsselwörter.
- Zu den meisten Anweisungen wurden Kommentare angegeben, die - gegebenenfalls auch in Folgezeilen - nach dem Zeichen "%" beginnen.
- Das Zeichen "\" zeigt an, dass eine Anweisung in der nächsten Zeile fortgesetzt wird.
- Im Unterschied zu den ZIM-Anweisungen wurden die SQL-Anweisungen mit einem Raster unterlegt.

Zum Inhalt der Prozedur *$liefere* sind ebenfalls einige Bemerkungen anzubringen:

- FORM SET EXIT Escape: Mit dieser Anweisung wird eine Abbruchtaste definiert, die jederzeit den Abbruch der Bearbeitung der aktuellen Bildschirmmaske gestattet.
- LET $TRANSMITKEY = " ": Die globale Variable $TRANSMITKEY, die der Übergabe eines Abbruchsignals dient, wird initialisiert.

- Der Block der folgenden WHILE-Schleife wird wiederholt ausgeführt solange kein Abbruchsignal eingegeben bzw. die Escape-Taste betätigt wird.
- FORM SELECT fArtikelBew: Es wird das Formular *fArtikelBew*, das für die Erfassung von Artikelbewegungen vorgesehen ist, angewählt.
- FORM DISPLAY INPUT: Das Formular bzw. die Erfassungsmaske wird auf dem Bildschirm angezeigt. Mittels des aktuellen Formulars, hier *fArtikelBew*, kann nun ein Artikelzugang erfasst werden.
- Der Rumpf der ersten IF-Anweisung wird ausgeführt, falls die Abbruchtaste nicht betätigt wird. Der gesamte Rumpf wird als eine Transaktion vereinbart, deren Beginn das Schlüsselwort TRANSACTION und deren Ende das Schlüsselwort ENDTRANSACTION anzeigen.
- SELECT * FROM Artikel WHERE ...: Mit dieser SQL-Anweisung werden aus der Relation *Artikel* die Tupel bzw. Sätze ausgewählt, für die das Attribut *ArtikelNr* den gleichen Wert besitzt, wie die zuvor in das Formular *fArtikelBew* eingegebene Artikelnummer. Während der Ausführung dieser Anweisung wird die globale Zählvariable $SETCOUNT aktualisiert. Sofern die Relation *Artikel* keinen Satz enthält, der die Auswahlbedingung erfüllt, erhält $SETCOUNT den Wert 0.
- Der Rumpf der zweiten bzw. inneren IF-Anweisung wird nur ausgeführt, falls zu dem erfassten Artikelzugang kein Eintrag in der Relation *Artikel* existiert; genau dann ist $SETCOUNT = 0. Der Zweck des Rumpfes besteht in der Erfassung eines Artikelsatzes und im Einfügen des Satzes in die Relation *Artikel*.
- FORM SELECT fArtikel: Es wird das Formular *fArtikel*, das für die Erfassung von Artikeldaten vorgesehen ist, angewählt und auf dem Bildschirm angezeigt.
- FORM DISPLAY INPUT: Ein Artikelsatz kann nun erfasst werden.
- INSERT INTO Artikel VALUES ...: In die Relation *Artikel* wird der erfasste Artikelsatz, Attributwert für Attributwert, eingetragen.
- INSERT INTO ArtikelBew VALUES ...: Erst jetzt darf auch der zuvor erfasste Artikelzugang in die Relation *ArtikelBew* eingetragen werden.

In der Prozedur *$liefere* tritt eine vorgegebene Abfrage auf, nämlich die SQL-Anweisung:

```
SELECT * FROM Artikel WHERE Artikel.ArtikelNr. = fArtikelBew.ArtikelNr
```

parametrisierte Abfrage

Bekanntlich wird mittels dieser Abfrage festgestellt, ob zu einem zuvor erfassten Artikelzugang ein zugehöriger Eintrag in der Relation *Artikel* existiert. Die Erfassung eines Artikelzugangs umschließt insbesondere auch eine Artikelnummer, die als Parameter in die Abfrage eingeht. In der obigen SQL-Anweisung wird der Parameter mittels der Punktqualifizierung angesprochen:

```
fArtikelBew.ArtikelNr
```

ArtikelNr ist der Parameter selbst und die Punktqualifizierung gibt an, dass der Parameter eine Komponente des Formulars *fArtikelBew* ist.

Integritätsbedingung

Übungsaufgabe 4.5

Betrachtet werde die Prozedur *$liefere* in Beispiel 4.1. Mit einigen Anweisungen dieser Prozedur wird eine Integritätsbedingung explizit formuliert, welche die Beziehung zwischen den Relationen *Artikel* und *ArtikelBew* betrifft. Geben Sie an, um welche Integritätsbedingung es sich handelt und erläutern Sie kurz den Zweck dieser Integritätsbedingung im vorliegenden Fall.

d) Beispiel für eine Anwendung mit datenbankgestützter Stapelverarbeitung

Aus Gründen des Umfangs kann hier ebenfalls kein komplettes Anwendungsprogramm angegeben werden. Die nachfolgend vorgestellte Prozedur *$RelReo*, die Bestandteil eines umfangreichen Stapelverarbeitungsprogramms ist, eignet sich jedoch zur Verdeutlichung der wesentlichen, hier interessierenden Zusammenhänge. Der Zweck der Prozedur *$RelReo* ist die Reorganisation bestimmter Relationen einer Datenbank. Diese Datenbank liegt dem in Kap. 3.5.1 behandelten Beispiel 3.6 zugrunde. Auf das Beispiel 3.6 bezieht sich auch das nachfolgend angegebene Beispiel 4.2.

Reorganisation von Relationen

Beispiel 4.2

Betrachtet werde das Beispiel 3.6. In diesem Beispiel treten u.a. folgende, von Zeit zu Zeit zu reorganisierenden Relationen auf:

ArtikelBew(ArtikelNr, ProjektNr, BewDatum, Menge, DispoMenge0),

ZeitBew(ProjektNr, MitarbNr, BewDatum, Zeit, LohnartNr, KoststellNr, ArbWert, Bemerkung),

FertAuftragMat(ProjektNr, StklisteNr, Menge, FertDatum),

FertAuftragZeit(ProjektNr, ZeitlisteNr, ArbWert, FertDatum),

KundenAuftrag(AuftrNr, ProjektNr, KundenNr, Name, AuftrDatum).

Unter Reorganisation ist im gegebenen Fall das Entfernen von Tupeln bzw. Sätzen aus einer Relation zu verstehen, die nicht mehr aktuell sind. Für die genannten Relationen leistet dies die Prozedur *$RelReo*. Die in ZIM/SQL geschriebene Prozedur *$RelReo* lautet:

```
% PROCEDURE $RelReo
%-------------------------------------------------------------------------------------------------------
% Aufrufe:                : $Error
% Forms                   : fRelReo
% Globale Variablen       :
% Lokale Variablen        : Antwort
% Fehlertextnummern       : 0010 0011 0012 0013 0014 0015
%
% Version 1.0             Datum: 01. April 2011
% Letzte Änderung         Datum: 17. November 2011
%-------------------------------------------------------------------------------------------------------
LOCALPROCEDURE ArtikelBew Reorg ( )\
LOCAL (Antwort)
    $Error (3, 0011, Antwort)                % Bildschirmmeldung
    CURSOR 22 1                              % Cursorpositionierung
    SET MEMBERCOUNT ON                       % Anzeige der bearbeiteten Datensätze
    TRANSACTION
    DELETE FROM ArtikelBew WHERE \
        ProjektNr >= fRelReo.VonProjektNr AND \
        ProjektNr <= fRelReo.BisProjektNr
    ENDTRANSACTION
    SET MEMBERCOUNT OFF                      % Abschalten der Datensatzanzeige
ENDPROCEDURE

LOCALPROCEDURE ZeitBewReorg ( )\
LOCAL (Antwort)
    $Error (3, 0012, Antwort)                % Bildschirmmeldung
    CURSOR 22 1                              % Cursorpositionierung
    SET MEMBERCOUNT ON                       % Anzeige der bearbeiteten Datensätze
    TRANSACTION
    DELETE FROM ZeitBew WHERE\
        ProjektNr >= fRelReo.VonProjektNr AND \
        ProjektNr <= fRelReo.BisProjektNr
    ENDTRANSACTION
    SET MEMBERCOUNT OFF                      % Abschalten der Datensatzanzeige
ENDPROCEDURE
```

```
LOCALPROCEDURE FertAuftragMatReorg ( )\
      :
      :
ENDPRODECURE

LOCALPROCEDURE FertAuftragZeitReorg ( )\
      :
      :
ENDPROCEDURE

LOCALPROCEDURE KundenAuftragReorg ( )\
      :
      :
ENDPROCEDURE

PROCEDURE $RelReo ( )\
LOCAL (Antwort)
      FORM SET EXIT Escape
      FORM SET TRANSMIT End
      FORM SELECT fRelReo
      LET fRelReo.Datum = $DATE
      FORM DISPLAY INPUT              % Eingabe der zu löschenden Nummern
      IF $TRANSMITKEY = "End"
            $Error (1, 0010, Antwort)        % Sicherheitsabfrage "Wirklich reorganisieren?"
            IF Antwort = TRUE
                  $Error (3, 0010, Antwort)  % Meldung des Starts der Reorganisation
                  SET MEMBERINTERVAL 1       % Datensatzintervall zur Anzeige
                  ArtikelBewReorg ( )
                  ZeitBewReorg ( )
                  FertAuftragMatReorg ( )
                  FertAuftragZeitReorg ( )
                  KundenAuftragReorg ( )
                        :
                        :
            ENDIF
      ENDIF
ENDPROCEDURE
```

Erläuterung des Beispiels

Der dargestellte Code besteht aus drei Teilen: einem Kommentarblock, fünf lokalen Prozeduren und der Prozedur *$RelReo* selbst. Der Kommentarblock enthält einige dokumentierende Informationen. Die fünf lokalen Prozeduren *ArtikelBewReorg*, *ZeitBewReorg*, *FertAuftragMatReorg*, *FertAuftragZeitReorg* und *KundenAuftragReorg* dienen der Reorganisation der Relationen *ArtikelBew*, *ZeitBew*, *FertAuftragMat*, *FertAuftragZeit* und *KundenAuftrag*. Den gesamten Arbeitsablauf steuert die Prozedur *$RelReo*. Sämtliche lokalen Prozeduren werden von *$RelReo* aufgerufen. Da die lokalen Prozeduren den gleichen Aufbau besitzen, genügt es, im Folgenden eine lokale Prozedur zu erläutern. Zuvor seien jedoch einige Anmerkungen zur Prozedur *$RelReo* gemacht.

Die Anweisungen der Prozedur *$RelReo* dienen folgenden Zwecken:

- FORM SET EXIT Escape: Definition einer Abbruchtaste (siehe Beispiel 4.1).
- FORM SET TRANSMIT End: Definition einer Fortsetzungstaste (siehe Anweisung IF $TRANSMITKEY = "End").

- FORM SELECT fRelReo: Anwählen des Formulars *fRelReo;* das Formular wird auf dem Bildschirm angezeigt.
- LET fRelReo.Datum = $DATE: In das im Formular *fRelReo* vorgesehenen Feld *Datum* wird das aktuelle Datum, das der globalen, vom Betriebssystem verwalteten Variablen $DATE entnommen wird, eingetragen.
- FORM DISPLAY INPUT: Das zuletzt angewählte Formular *fRelReo* wird auf dem Bildschirm angezeigt; nun kann der Benutzer den Arbeitsablauf mittels Eingabe von "Escape" abbrechen oder mittels Eingabe von "End" fortsetzen.
- IF $TRANSMITKEY = "End": Der Rumpf der IF-Anweisung wird ausgeführt, falls "End" eingegeben wurde.
- $Error (1, 0010, Antwort): Es wird die global vereinbarte Fehleranzeigeprozedur *$Error* aufgerufen. Da der erste Parameter auf 1 gesetzt ist, bewirkt die Prozedur die Anzeige des Fehlertextes mit der Nummer 0010; zu dieser Nummer gehört der Text "Wirklich reorganisieren?". Anschließend begibt sich die Prozedur in einen Wartezustand: Gewartet wird auf eine Benutzereingabe. Die Bestätigung der Frage wird durch die Rückgabe des Wertes TRUE oder FALSE in der Variablen *Antwort* angezeigt.
- IF Antwort = TRUE: Der Rumpf der IF-Anweisung wird ausgeführt, falls die Benutzerantwort "ja" lautete.
- $Error (3, 0010, Antwort): Da hier der erste Parameter auf 3 gesetzt ist, wird lediglich die Meldung mit der Textnummer 0010 - "Bitte warten, die Reorganisation läuft." - angezeigt.
- SET MEMBERINTERVAL 1: Das Anzeigeintervall für bearbeitete Datensätze wird auf 1 gesetzt. Damit wird bei eingeschaltetem Datensatzzähler der Zähler am Bildschirm nach jedem bearbeiteten Datensatz um eins erhöht.
- ArtikelBewReorg( ), ZeitBewReorg( ) usw.: Die lokalen Reorganisationsprozeduren werden aufgerufen.

Exemplarisch werde nun die Reorganisationsprozedur *ArtikelBewReorg* erläutert:

- $Error (3, 0011, Antwort): Am Bildschirm wird dem Benutzer der Text mit der Nummer 0011 - "Bitte warten, es wird die Relation *ArtikelBew* reorganisiert" - angezeigt.
- CURSOR 22 1: Der Cursor wird auf die 1. Position der 22. Bildschirmzeile positioniert.
- SET MEMBERCOUNT ON: Die Zähleranzeige MEMBERCOUNT wird angeschaltet; MEMBERCOUNT gibt an, dass der Datensatzzähler der bearbeiteten Datensätze an der aktuellen Cursorposition angezeigt werden soll.
- DELETE FROM ArtikelBew WHERE ProjektNr >= fRelReo.VonProjektNr AND ProjektNr <= fRelReo.BisProjektNr: SQL-Befehl zum Löschen sämtlicher Tupel aus der Relation *ArtikelBew*, welche innerhalb der in dem Formular *fRelReo* vorgegeben Grenzen von Projektnummern liegen.
- SET MEMBERCOUNT OFF: Die Zähleranzeige wird abgeschaltet.

Übungsaufgabe 4.6

Bei der in Beispiel 4.2 behandelten Datenbankanwendung treten Benutzerinteraktionen auf. Wieso ist es dennoch gerechtfertigt, von datenbankgestützter Stapelverarbeitung zu sprechen?

4.3 Relationenalgebra

PL für das Relationenmodell

Wie bereits erwähnt wurde, ist die Relationenalgebra eine Datenmanipulationssprache, die von CODD für das Relationenmodell entwickelt wurde. Das Selektionsvermögen der Relationenalgebra ist umfassend: Sämtliche sinnvollen Abfragen an relationale Datenbanken können formuliert werden. Datenmanipulationssprachen mit dem Selektionsvermögen der Relationenalgebra heißen daher relational vollständig.

relationale Vollständigkeit

Operationen der Relationenalgebra

Neben den klassischen Mengenoperationen Vereinigung, Durchschnitt usw. gehören u.a. die Operationen Projektion, Verbund und Restriktion zum Funktionsumfang der Relationenalgebra. Für die Manipulation von Relationen sind die letztgenannten Operationen unerlässlich. Sie finden sich daher in praktisch allen relationalen Datenmanipulationssprachen. So auch in der im nächsten Kapitel behandelten Sprache SQL.

Nachfolgend werden zunächst einige klassische Mengenoperationen und danach die übrigen genannten Operationen behandelt.

a) Klassische Mengenoperationen

Klassische Mengenoperationen auf zwei vereinigungsverträglichen Relationen R und S sind:

klassische Mengenoperationen

- die Vereinigung $R \cup S$,
- der Durchschnitt $R \cap S$,
- die Differenz R - S und
- die symmetrische Differenz R / S.

Eine klassische Mengenoperation auf zwei Relationen R und S mit disjunkten Attributmengen ist:

- das kartesische Produkt R x S.

Die Eigenschaften "vereinigungsverträglich" und "disjunkt" von Relationen seien kurz erläutert:

vereinigungsverträgliche Relationen

Zwei Relationen gleichen Grades heißen vereinigungsverträglich, wenn jedem Attribut der ersten Relation ein Attribut vom gleichen Datentyp in der zweiten Relation zugeordnet ist.

disjunkte Relationen

Zwei Relationen bzw. Attributkombinationen sind disjunkt, falls sie keine gemeinsamen Attribute enthalten.

Zur Demonstration der klassischen Mengenoperationen diene das folgende Beispiel:

Beispiel 4.3

Gegeben seien zwei Relationen *Projekt* und *Ausschuss*, deren Tupel die Mitarbeiter in einem Projekt und die Mitglieder eines Ausschusses beschreiben:

Projekt

MName	Abteilung	Funktion
Abs	112	Laborant
Beer	116	Chemiker
Hein	112	Konstrukteur

Ausschuss

MName	Abteilung	Funktion
Beer	116	Chemiker
Riedle	114	Laborant

In zwei weiteren Relationen *Personal* und *Gehalt* werden zum einen sämtliche Personen und zum anderen Gehaltsdaten erfasst:

Personal

MName	Abteilung	Funktion
Abs	112	Laborant
Beer	116	Chemiker
Hein	112	Konstrukteur
Riedle	114	Laborant

Gehalt

MName	Grundgehalt	Zulagen
Abs	40000	15000
Beer	60000	50000
Hein	45000	50000
Riedle	40000	10000

Nun zu den einzelnen Mengenoperationen.

Vereinigung

Für die Vereinigung zweier Relationen gilt:

Vereinigung

Seien R und S zwei vereinigungsverträgliche Relationen und bezeichne t einen Tupel, dann besteht die Vereinigung aus der Menge aller Tupel, die in R oder in S oder in R und S enthalten sind:

$R \cup S = \{t \mid t \in R \text{ oder } t \in S \text{ oder } t \in R \text{ und } t \in S\}$.

In Beispiel 4.3 sind *Projekt* und *Ausschuss* vereinigungsverträglich. Die Vereinigung ergibt:

Projekt $\cup$ Ausschuss

MName	Abteilung	Funktion
Abs	112	Laborant
Beer	116	Chemiker
Hein	112	Konstrukteur
Riedle	114	Laborant

Gebildet wird die Gesamtheit der Mitarbeiter bzw. Mitglieder ohne Doppelnennungen: *Personal* = *Projekt* ∪ *Ausschuss*.

Durchschnitt

Es gilt folgende Definition:

Durchschnitt

Der Durchschnitt zweier vereinigungsverträglicher Relationen R und S besteht aus der Menge aller Tupel, die in R und in S enthalten sind:

$$R \cap S = \{t \mid t \in R \text{ und } t \in S\}.$$

Der Durchschnitt von *Projekt* und *Ausschuss* lautet:

Projekt ∩ Ausschuss

MName	Abteilung	Funktion
Beer	116	Chemiker

Das Ergebnis besteht aus dem Mitarbeiter, der beiden Institutionen angehört.

Differenz

Für die Differenz gilt:

Differenz

Die Differenz zweier vereinigungsverträglicher Relationen R und S besteht aus der Menge aller Tupel, die in R aber nicht in S enthalten sind:

$$R - S = \{t \mid t \in R \text{ und } t \notin S\}.$$

Im Falle der Relationen *Projekt* und *Ausschuss* erhält man:

Projekt - Ausschuss

MName	Abteilung	Funktion
Abs	112	Laborant
Hein	112	Konstrukteur

Aus der Relation *Projekt* werden die Mitarbeiter selektiert, die nicht zugleich in *Ausschuss* vertreten sind.

Symmetrische Differenz

Im Gegensatz zur Differenz zweier Relationen bevorzugt die symmetrische Differenz keine der beiden Relationen:

symmetrische Differenz

Die symmetrische Differenz zweier vereinigungsverträglicher Relationen R und S besteht aus der Menge aller Tupel, die in R oder in S aber nicht zugleich in R und S enthalten sind:
$R / S = \{t \mid t \in R$ oder $t \in S$ und $t \notin R \cap S\}$.

Für die Relationen *Projekt* und *Ausschuss* ergibt sich:

Projekt / Ausschuss

MName	Abteilung	Funktion
Abs	112	Laborant
Hein	112	Konstrukteur
Riedle	114	Laborant

Ausgewählt werden die Personen, die nur einer der beiden Institutionen angehören.

Kartesisches Produkt

Von dem kartesischen Produkt wurde bereits in Kap. 3.4.1, bei der Definition des Begriffs *Relation,* Gebrauch gemacht. Für die Definition des kartesischen Produkts wird der Begriff der Verkettung benötigt:

Seien $r = (r_1, r_2, ..., r_m)$ und $s = (s_1, s_2, ..., s_n)$ zwei Tupel, dann bezeichnet die Verkettung (engl. concatenation) $r \sim s$ einen aus r und s zusammengesetzten Tupel:

$r \sim s = (r_1, r_2, ..., r_m, s_1, s_2, ..., s_n)$.

Das kartesische Produkt ist wie folgt definiert:

kartesisches Produkt

Seien R und S zwei Relationen beliebigen Grades, dann besteht das kartesische Produkt R x S dieser Relation aus der Menge aller möglichen Verkettungen von Tupeln aus R und aus S:

R x $S = \{r \sim s \mid r \in R$ und $s \in S\}$.

Gegeben sei eine Relation *Sprache*:

Sprache

Sprache
Deutsch
Englisch

Das kartesische Produkt von *Projekt* und *Sprache* lautet dann:

Projekt x Sprache

MName	Abteilung	Funktion	Sprache
Abs	112	Laborant	Deutsch
Abs	112	Laborant	Englisch
Beer	116	Chemiker	Deutsch
Beer	116	Chemiker	Englisch
Hein	112	Konstrukteur	Deutsch
Hein	112	Konstrukteur	Englisch

Nicht alle Projektmitarbeiter müssen jedoch über Englischkenntnisse verfügen. Dies kommt in der Relation *Projektpersonal* zum Ausdruck (vgl. hierzu auch die Definition der Relation in Kap. 3.4.1):

Projektpersonal

MName	Abteilung	Funktion	Sprache
Abs	112	Laborant	Deutsch
Beer	116	Chemiker	Deutsch
Beer	116	Chemiker	Englisch
Hein	112	Konstrukteur	Deutsch

Offensichtlich gilt: *Projektpersonal* $\subset$ *Projekt* x *Sprache*.

b) Projektion

Die Projektion (engl. projection) gestattet das Abspalten eines Teils einer Relation. Die abzuspaltenden Attribute sind vorzugeben:

Projektion

Bezeichne R eine Relation m-ten Grades und $a = (a_1,...,a_i)$, $i < m$, eine Kombination von in R vertretenen Attributen, dann ist die Projektion der Relation R auf die Attributkombination definiert als:

$R[a] = \{(t[a_1], t[a_2],...,t[a_i]) \mid t \in R\}$.

Hierbei gibt $t[a_i]$ den Wert an, den das Attribut a_i im Tupel t besitzt. $R[a]$ ist selbst wieder eine Relation, die nur die Attribute $a_1,...,\ a_i$ enthält. $R[a]$ entsteht also durch das Streichen der Spalten in Relation R, welche nicht in a enthalten sind. Man beachte, dass in $R[a]$ kein Tupel mehrfach vorkommen darf.

Beispielweise führt die Projektion der Relation *Personal* auf die Attributkombination (*MName*, *Abteilung*) zu folgender neuen Relation:

Personal [MName, Abteilung]

MName	Abteilung
Abs	112
Beer	116
Hein	112
Riedle	114

Und die Projektion von *Personal* auf das Attribut *Abteilung* führt sowohl einer Verkleinerung der Spaltenzahl, als auch der Zeilenzahl:

Personal [Abteilung]

Abteilung
112
114
116

Hingewiesen sei noch darauf, dass das Aufspalten von Relationen im Zuge ihrer Normalisierung auf der Operation der Projektion beruht.

c) Verbund

Zusammenfügen von Relationen

Im Gegensatz zur Projektion ermöglicht der Verbund (engl. join) das Zusammenfügen von Relationen zu neuen Relationen. In eine neue Relation gehen allerdings nur solche Tupelpaare aus zwei zusammenzufügenden Relationen ein, zwischen denen ein vorzugebender Zusammenhang besteht.

Bezeichne "Θ" einen der Vergleichsoperatoren "=", "≠", "<", "≤", ">" und "≥", dann gilt für den Theta-Verbund (engl. Θ-join):

Theta-Verbund

Seien a und b vereinigungsverträgliche Attributkombinationen aus den Relationen R und S, dann ist der Theta-Verbund der Relationen R und S über die Attributkombinationen a und b definiert als:

$R\ [a\ \Theta\ b]\ S = \{r \sim s \mid r \in R, s \in S \text{ und } r[a]\ \Theta\ s[b]\}$.

Hier gilt: Zwei Attributkombinationen gleichen Grades sind vereinigungsverträglich, wenn zu jedem Attribut der einen Attributkombination ein Attribut von gleichen Datentyp in der anderen Attributkombination existiert.

Zusammengeführt und in den Theta-Verbund einbezogen werden nur die Tupel r und s, die hinsichtlich der Attributkombinationen a und b die Qualifikationsbedingung $r[a]\ \Theta\ s[b]$ erfüllen.

natürlicher Verbund

Von praktischer Bedeutung ist vor allem der Sonderfall des Gleich-Verbunds, für den Θ gleich "=" ist. Bei diesem auch als natürlicher Verbund (engl. natural join) bezeichneten Fall dürfen keine Attribute doppelt auftreten. Zur Darstellung des natürlichen Verbunds wird daher die Komplementbildung benötigt:

Komplementbildung

Sei a eine in der Relation R enthaltene Attributkombination und r ein Tupel aus R, dann bezeichnen:

(1) $r[a]$ die Projektion des Tupels r auf die Attributkombination a. $r[a]$ besteht aus den Werten der in a enthaltenen Attribute.

(2) $r[\bar{a}]$ die Projektion des Tupels r auf das Komplement $\bar{a}$ von a. $r[\bar{a}]$ besteht aus den Werten der nicht in a enthaltenen Attribute. Es gilt daher: $r = r[a] \sim r[\bar{a}]$.

Für den Tupel t = (Beer, 116, Chemiker) $\in$ *Personal* und die Attributkombination a = (*MName*, *Funktion*) gilt beispielsweise:

$t[a]$ = (Beer, Chemiker) und $t[\overline{a}]$ = (116).

Nun zur Definition des natürlichen Verbunds:

natürlicher Verbund

Der natürliche Verbund zweier Relationen R und S über die vereinigungsverträglichen Attributkombinationen a und b ist definiert als:

$R[a = b]S = \{r[a] \sim r[\bar{a}] \sim s[\bar{b}] \mid r \in R, s \in S \text{ und } r[a] = s[b]\}$.

Der natürliche Verbund der Relationen *Personal* und *Gehalt* (siehe Beispiel 4.3) über den Attributen *Personal.MName* und *Gehalt.MName* ergibt beispielsweise:

P = Personal [Personal.MName = Gehalt.MName] Gehalt

MName	Abteilung	Funktion	Grundgehalt	Zulagen
Abs	112	Laborant	40000	15000
Beer	116	Chemiker	60000	50000
Hein	112	Konstrukteur	45000	50000
Riedle	114	Laborant	40000	10000

In der neuen Relation P sind die Personalinformationen zusammengefasst, die zuvor auf die Relationen *Personal* und *Gehalt* verteilt waren.

d) Restriktion

Um eine Auswahloperation handelt es sich bei der Restriktion (engl. restriction). Die Operation wählt die Tupel aus einer Relation aus, die einer auf Attributwerten definierten Auswahlbedingung genügen:

Restriktion

Seien a und b vereinigungsverträgliche Attributkombinationen aus der Relation R, dann ist die Restriktion der Relation R bezüglich den Attributkombinationen a und b definiert als:

$R[a \Theta b] = \{t \mid t \in R \text{ und } t[a] \Theta t[b]\}$.

Ausgewählt werden somit diejenigen Tupel, für die der Wert der Attributkombination a gleich oder ungleich oder kleiner usw. ist, als der Wert der Attributkombination b.

Mit Hilfe der Restriktion können beispielsweise die Personen aus der Relation P ausgewählt werden, deren Zulagen das Grundgehalt übersteigen. Diese Auswahl leistet die Restriktion:

$$P[P.Zulagen > P.Grundgehalt]$$

Als Ergebnis erhält man eine neue Relation, die - falls die Auswahlbedingung wirklich restriktiv ist - aus weniger Tupeln als die Ausgangsrelation besteht. Hier gilt:

P[P.Zulagen > P.Grundgehalt]

MName	Abteilung	Funktion	Grundgehalt	Zulagen
Hein	112	Konstrukteur	45000	50000

Kombination von Restriktion und Projektion

Interessieren nun z.B. aber nur der Name und die Abteilung der auszuwählenden Personen, so liefert die obige Restriktion in Verbindung mit einer Projektion auf die genannten Attribute die gewünschte Ergebnisrelation *Q*:

$$Q := P[P.Zulagen > P.Grundgehalt][MName, Abteilung]$$

Die Ergebnisrelation *Q* lautet:

Q

MName	Abteilung
Hein	112

e) Beispiele für Abfragen

Abschließend sei die Datenmanipulation mit Hilfe der Relationenalgebra an einigen Abfragen demonstriert. Die Abfragen beziehen sich auf die in Beispiel 4.3 angegebenen Relationen *Personal* und *Gehalt*.

(1) Wie lauten die Namen der Personen, die der Abteilung 112 angehören?

Offensichtlich erfordert die Abfrage das Manipulieren einer Konstanten, nämlich des Wertes 112. Da die Relationenalgebra nur das Arbeiten mit Relationen gestattet, muss die Konstante in eine Hilfsrelation eingebracht werden. Als Hilfsrelation werde vereinbart:

X

x
112

Hilfsrelation

X ist der Name der Hilfsrelation, *x* ihr einziges Attribut und 112 der Attributwert. Völlig äquivalent zur tabellarischen Darstellung der Hilfsrelation *X* ist folgende Schreibweise:

$$X(x) = 112$$

Bezeichne *R* die Ergebnisrelation in Bezug auf die gegebene Abfrage, dann kann *R* in folgenden Schritten ermittelt werden:

- Einführung einer Hilfsrelation:

  $X(x) = 112$

- Ermitteln eines Zwischenergebnisses *R'*, welches alle zutreffenden Tupel enthält. *R'* ergibt sich aus dem natürlichen Verbund der Relation *Personal* und der Hilfsrelation *X* wie folgt:

  $R' := Personal[Personal.Abteilung = x\ ]\ X$

  Die tabellarische Darstellung von *R'* lautet:

  R'

MName	Abteilung	Funktion
Abs	112	Laborant
Hein	112	Konstrukteur

- Das Ergebnis *R* entsteht durch die Projektion von *R*' auf das Attribut *MName*:

  $R := R'[MName]$

  In tabellarischer Darstellung:

  R

MName
Abs
Hein

Durch die Zusammenfassung der dargestellten Schritte erhält man folgende Abfrage zur Ermittlung von *R*:

1. Abfragebeispiel

$$R := (Personal[Personal.Abteilung = x]X)[MName]$$

(2) Wie lauten die Namen der Personen, die der Abteilung 112 angehören und deren Grundgehalt größer als 40000 ist?

Die Ergebnisrelation *Q* kann in folgenden Schritten ermittelt werden:

- Einführung von zwei Hilfsrelationen:

  $X(x) = 112$ und $Y(y) = 40000$

- Ermitteln der Namen der Personen, die der Abteilung 112 angehören, mit Hilfe der unter (1) formulierten Abfrage:

  $R := (Personal[Personal.Abteilung = x]X)[MName]$

- Ermitteln der Namen der Personen, deren Grundgehalt 40000 übersteigt:

  $Q' := (Gehalt[Gehalt.Grundgehalt > y]Y)[MName]$

  Tabellarische Darstellung von *Q*':

  Q'

MName
Beer
Hein

- Der Durchschnitt von *R* und *Q*' führt zu dem gewünschten Ergebnis:

  $Q := R \cap Q'$

  In Tabellenform:

  Q

MName
Hein

In ausführlicher Schreibweise lautet die Abfrage somit:

2. Abfragebeispiel

$$Q := ((Personal[Personal.Abteilung = x]X)[MName]) \cap ((Gehalt[Gehalt.Grundgehalt > y]Y)[MName])$$

(3) Gesucht sind Namen und Zulagen der Personen, die als Laborant tätig sind.

Die Ergebnisrelation *S* kann in folgenden Schritten gebildet werden (ohne Erläuterungen):
$X(x)$ = Laborant
$S' := Personal[Personal.Funktion = x]\ X$
$S'' := S'[S'.MName = Gehalt.MName]\ Gehalt$
$S := S''[MName, Zulagen]$

In ausführlicher Schreibweise lautet die Abfrage (vereinfachend werden bei der Bezeichnung von Attributen die Relationsnamen weggelassen):

$S := ((Personal[Funktion = x]X)[MName = MName]Gehalt)[MName, Zulagen]$

3. Abfragebeispiel

Aus den angeführten Beispielen wird deutlich, dass die Schreibweise der Relationenalgebra wenig benutzerfreundlich ist. Als Datenmanipulationssprache konnte sich die Relationenalgebra daher in der Praxis nicht durchsetzen. Jedoch ist die Relationenalgebra deshalb von Bedeutung, weil sie ein vollständiges Instrumentarium zur Manipulation relationaler Datenbanken darstellt und weil viele ihrer Operationen, in sprachlich anderer Form, Eingang in die praxisüblichen relationalen Datenmanipulationsprachen gefunden haben.

Bedeutung der Relationenalgebra

Übungsaufgabe 4.7

Betrachtet werde das zuletzt angegebene Beispiel (3) für die Formulierung einer Datenbankabfrage mit Hilfe der Relationenalgebra. Erläutern Sie die Schritte, die zur Ergebnisrelation *S* führen, und geben Sie für jeden Schritt das Zwischenergebnis und außerdem das Endergebnis in Tabellenform an.

Übungsaufgabe 4.8

Gegeben seien die in Beispiel 4.3 spezifizierten Relationen *Personal* und *Gehalt*. Gesucht sind der Name und die Abteilung für die Personen, deren Zulagen das Grundgehalt übersteigen oder die als Chemiker tätig sind. Formulieren Sie diese Abfrage in der Sprache der Relationenalgebra. Bilden Sie die Ergebnisrelation schrittweise und stellen Sie die Zwischenergebnisse sowie das Endergebnis tabellarisch dar.

4.4 Structured Query Language (SQL)

SQL ist eine Datenbanksprache, deren Funktionsumfang sich nicht - wie man aus der Namensgebung SQL für Structured Query Language schließen könnte - auf Abfragen beschränkt. SQL gestattet vielmehr auch die Beschreibung von Daten, die Definition von Indizes sowie das Mutieren von Daten.

SQL/R

In der zweiten Hälfte der siebziger Jahre wurde SQL von der Firma IBM als Datenbanksprache für das relationale System R entwickelt. R ist ein relationales DBVS, das ebenfalls aus dem Hause IBM stammt. Bei SQL/R handelt es sich um ein vollständiges relationales System. Darunter ist ein System zu verstehen, welches das relationale Datenmodell in vollem Umfang unterstützt. In diesem Zusammenhang sei angemerkt, dass auf dem Softwaremarkt neben vollständigen relationalen Systemen auch nicht vollständige Systeme angeboten werden. Letztere unterstützen das Relationenmodell nicht in vollständigem Umfang. Zu den vollständigen Systemen gehören beispielsweise SQL/DS von

IBM und ORACLE von der gleichen Firma. Beispiele für nicht vollständige Systeme sind CONDOR, ein von der Firma Siemens von1973 bis 1981 entwickeltes und gepflegtes DBVS, sowie DBASE von der Firma Ashton Tate.

Einsatz von SQL

Da ein Normungsvorschlag der ANSI einen Sprachstandard für relationale Datenbanken vorsieht, der weitgehend auf SQL basiert, besitzen mittlerweile viele DBVS SQL-Schnittstellen. Interessant ist, dass SQL nicht nur in relationalen Systemen wie ORACLE, UNIFY und ZIM als Datenmanipulationssprache verwendet werden kann, sondern auch in nicht relationalen Systemen wie z.B. dem System UDS von Siemens. Schließlich werden die Anwendungsmöglichkeiten von SQL noch dadurch vergrößert, dass SQL als selbständige und als eingebettete Sprache angeboten wird. Das eingebettete SQL (engl. embedded SQL) ist u.a. für die Wirtssprachen COBOL, PL/I und ZIM verfügbar.

Inhalt des Kapitels

Im vorliegenden Kapitel wird die Sprache SQL in Verbindung mit dem System R behandelt. Eine Übersicht über SQL und das System R gibt das Kap. 4.4.1. Die Datenbeschreibung mit SQL behandelt das Kap. 4.4.2 und die Datenmanipulation mit SQL das Kap. 4.4.3. Die beiden folgenden Kapitel greifen zwei Eigenschaften des System R auf, die aus der Sicht des Anwenders von Interesse sind: In Kap. 4.4.4 wird die externe Ebene des Systems R dargestellt und in Kap. 4.4.5 das im System R vorgesehene Datenlexikon.

4.4.1 SQL und das System R

Bei dem System R handelt es sich um den Prototyp eines relationalen DBVS, der mit SQL als Datenbeschreibungs- und Datenmanipulationssprache im IBM San Jose Research Laboratory entwickelt wurde. Entwicklungsziel war ein "operational vollständiger" Prototyp, mit dem die Praxistauglichkeit des relationalen Ansatzes demonstriert werden sollte. Daher stellt das System R nicht nur grundlegende relationale Datenbankfunktionen bereit, sondern auch zusätzliche Funktionen wie Datenbank-Rekonstruktion, Synchronisation von Datenbankzugriffen, Prüfung der Zugangsberechtigung, Unterstützung der Datenunabhängigkeit usw.

Die Architektur des Systems R orientiert sich an dem ANSI-Architekturmodell. Aus Abb. 4.5 geht hervor, wie sich das System R aus der Sicht eines einzelnen Benutzers darstellt.

Sichten

Laut Abb. 4.5 besteht auf der externen Ebene die Möglichkeit für einen Benutzer, mehrere Sichten (engl. views) zu definieren. Über die Sichten interagiert der Benutzer mit sogenannten Basistabellen. Genau genommen nur mit den Teilen der Basistabellen, welche die Sichten freigeben. Außerdem kann der Benutzer auch ohne Zwischenschaltung von Sichten direkt auf Basistabellen operieren.

Basistabellen

Eine Basistabelle ist ein eigenständiger konzeptioneller Satztyp. Jede Basistabelle kann damit auch so definiert werden, dass sie eine Relation in 3. Normalform darstellt. Die Menge sämtlicher Basistabellen bildet das konzeptionelle Modell. Ob dieses Modell ein relationales Datenmodell oder gar ein global normalisiertes Datenmodell ist, hängt davon ab, welche Schlüsselattribute und sonstigen Attribute in die einzelnen Tabellen einbezogen werden.

Architektur des Systems R

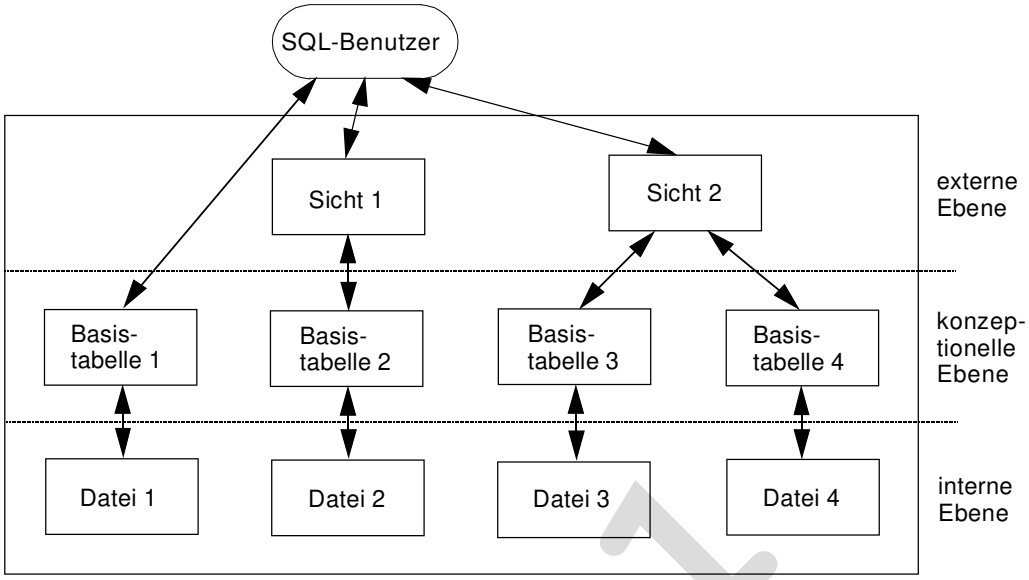


Abb. 4.5. Architektur des Systems R.

Dateien

Jeder Basistabelle entspricht genau eine Datei auf der internen Ebene. Eine Datei enthält sämtliche zu einer Basistabelle gehörigen Sätze bzw. Tupel. Für Zugriffszwecke können pro Datei beliebig viele Indizes definiert werden.

Mit den in SQL verfügbaren Anweisungen können alle drei Ebenen angesprochen werden. Die Anweisungen zur Datenbeschreibung gestatten:

Datenbeschreibungs-ebenen

- die Definition von Sichten auf der externen Ebene,
- die Definition von Basistabellen auf der konzeptionellen Ebene und
- die Definition von Indizes auf der internen Ebene.

Und die Anweisungen zur Datenmanipulation ermöglichen es dem Benutzer, sowohl auf den Sichten der externen Ebene zu operieren, als auch direkt auf den Basistabellen der konzeptionellen Ebene.

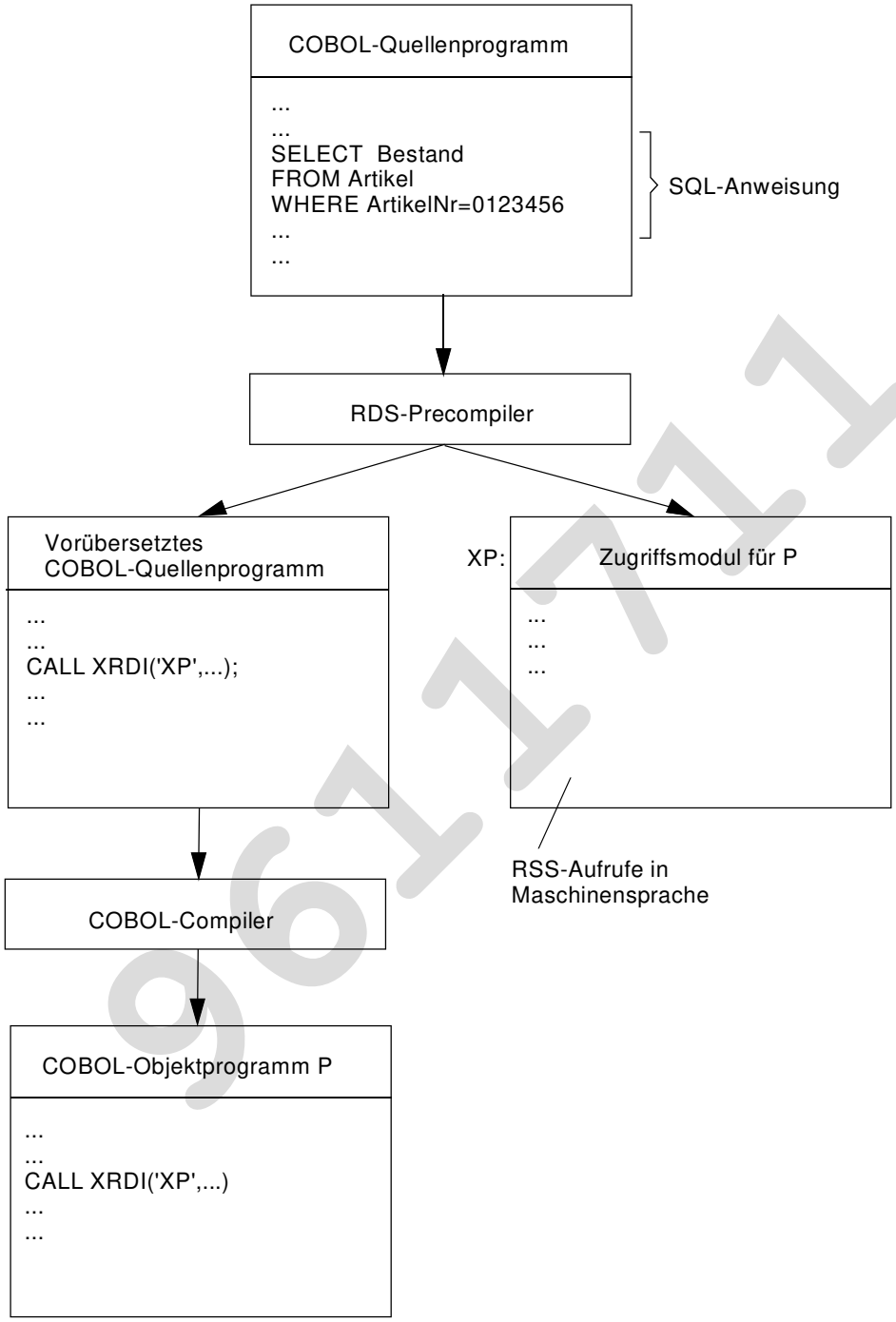
Komponenten des Systems R

Funktional gesehen besteht das System R aus zwei Komponenten: dem Research Storage System (RSS) und dem Relational Data System (RDS). Das RSS ist ein leistungsfähiges Zugriffssystem, welches die von den Benutzern formulierten logischen Datenbankzugriffe - gegebenenfalls unter Verwendung von Indizes - in physische Dateizugriffe umsetzt. Aus zwei Subkomponenten setzt sich seinerseits das RDS zusammen, nämlich

- einem Precompiler zur Vorübersetzung der SQL-Anweisungen in einem Anwendungsprogramm und
- einem Runtime-System, genannt XRDI, zur Steuerung der Ausführung eines übersetzten Anwendungsprogramms.

Abb. 4.6 veranschaulicht das Zusammenwirken der Komponenten des Systems R bei der Übersetzung und Ausführung eines SQL-Anwendungsprogramms. Unterstellt wird hierbei der Fall der Einbettung von SQL-Anweisungen in ein COBOL-Programm.

Wie die Abb. 4.6 zeigt, ersetzt der Precompiler jede in das COBOL-Anwendungsprogramm eingebettete SQL-Anweisung durch eine gewöhnliche COBOL-Anweisung CALL XRDI(...). XRDI ist der Name des Runtime-Systems von R; die



Arbeitsweise des Systems R: Vorübersetzung und Übersetzung

Abb. 4.6. Vorübersetzung und Übersetzung eines COBOL-Anwendungsprogramms mit eingebetteten SQL-Anweisungen.

Anweisung CALL XRDI(...) ruft also das Runtime-System auf. Außerdem erzeugt der Precompiler einen Zugriffsmodul XP, welcher sämtliche Aufrufe des Zugriffssystems RSS für das Anwendungsprogramm umfasst. Im zweiten Übersetzungsschritt wird das vorübersetzte COBOL-Quellenprogramm mit einem gewöhnlichen COBOL-Compiler in ein ausführbares Objektprogramm P übersetzt.

Bei der Ausführung des Objektprogramms P unterliegen sämtliche Datenbankzugriffe der Kontrolle des RDS Runtime-Systems. Die hierbei auftretenden grundlegenden Zusammenhänge gehen aus Abb. 4.7 hervor: Erreicht die Ausführung des Objektprogramms P die Anweisung CALL XRDI(...), so geht die Kontrolle an das RDS Runtime-System über. Das Runtime-System greift auf den Zugriffsmodul XP zu und aktiviert die Sektion aus P, die der ursprünglichen SQL-Anweisung "SELECT Bestand FROM Artikel WHERE ..." entspricht. Diese Sektion löst ihrerseits diejenigen Operationen des RSS Zugriffssystems aus, welche geeignet sind, die von der ursprünglichen SELECT-Anweisung angeforderten Aktionen abzuwickeln.

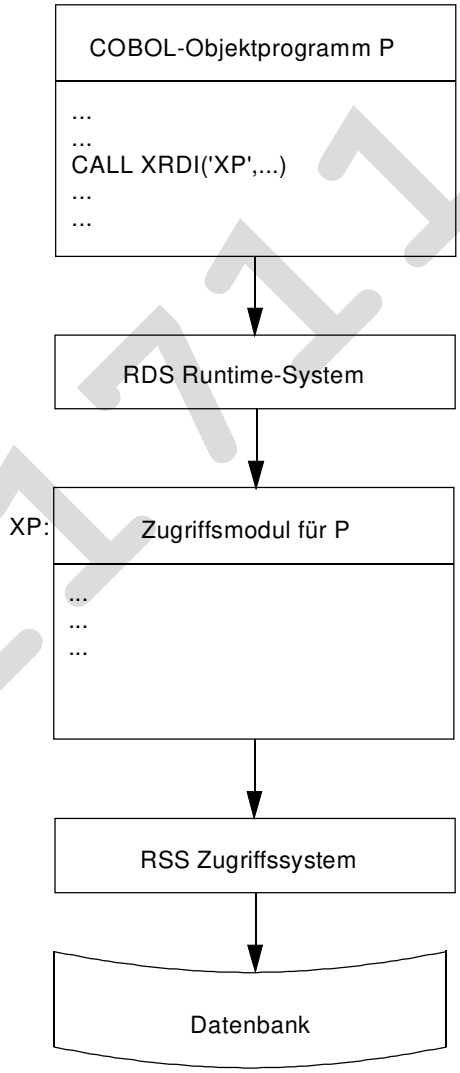


Abb. 4.7. Datenbankzugriffe bei der Ausführung eines Objektprogramms.

Übungsaufgabe 4.9

Das System SQL/R räumt dem Benutzer die Möglichkeit ein, Datenmanipulationen nicht nur auf der externen Ebene, sondern auch auf der konzeptionellen Ebene vorzunehmen. Nennen Sie die daraus resultierenden Vor- und Nachteile.

4.4.2 Datenbeschreibung mit SQL

In diesem Kapitel wird lediglich die Datenbeschreibung auf der konzeptionellen Ebene behandelt. Gegeben ist die konzeptionelle Ebene durch eine Menge von Basistabellen. SQL stellt mehrere Anweisungen bereit, welche die Definition und die Handhabung von Basistabellen sowie von Indizes ermöglichen. Im Einzelnen geht es hierbei um:

- die Erzeugung von Basistabellen mit der CREATE TABLE-Anweisung,
- die Erweiterung von Basistabellen mit der EXPAND TABLE-Anweisung,
- das Löschen von Basistabellen mit der DROP TABLE-Anweisung,
- die Erzeugung von Indizes mit der CREATE INDEX-Anweisung und
- das Löschen von Indizes mit der DROP INDEX-Anweisung.

SQL-Anweisungen zur Datenbeschreibung

Im Folgenden werden diese Anweisungen beschrieben und auf das in Kap. 3.5.1 vorgestellte Beispiel 3.6 angewandt.

a) Erzeugung von Basistabellen (CREATE TABLE-Anweisung)

Mit Hilfe der CREATE TABLE-Anweisung können Basistabellen erzeugt werden. Die CREATE TABLE-Anweisung besitzt folgende Form:

CREATE TABLE-Anweisung

```
CREATE TABLE basistabellename
        (felddefinition [,felddefinition]...)
        [IN SEGMENT segmentname]
```

wobei *felddefinition* für eine Feldvereinbarung in der folgenden Form steht:

```
        feldname (datentyp [,NOT NULL])
```

In dieser, und auch in allen folgenden SQL-Anweisungen, werden Schlüsselwörter groß und Bezeichner klein geschrieben. Außerdem stehen optionale Anweisungsteile zwischen eckigen Klammern.

Zu der CREATE TABLE-Anweisung ist folgendes zu bemerken:

(1) Mit einer CREATE TABLE-Anweisung wird genau eine leere Basistabelle erzeugt, die unmittelbar benutzt werden kann. Es können nun also Daten in die Tabelle eingegeben werden.

(2) Auf den Namen der Basistabelle folgen ein oder mehrere Felddefinitionen. In SQL wird an Stelle des Begriffs "Attribut" der Feldbegriff verwendet. Jede Felddefinition umfasst drei Angaben:

- den Namen des Feldes,
- den Datentyp des Feldes und
- die optionale NOT NULL-Spezifikation.

Während ursprünglich nur die Datentypen

Datentypen im System R

CHAR(n)	-	Zeichenkette fester Länge,
CHAR(n)VAR	-	Zeichenkette variabler Länge,
INTEGER	-	ganze Zahl in einem Vollwort,
SMALLINTEGER	-	ganze Zahl in einem Halbwort,
FLOAT	-	Gleitkommazahl in einem Doppelwort,

zulässig waren, kamen später - vor allem bei kommerziell eingesetzten DBVS - noch Datentypen wie:

DATE	-	Datumstyp der Form JJJJMMTT,
LONGINTEGER	-	ganze Zahl in einem Doppelwort,
ALPHANUMERIC	-	alphanumerischer Datentyp,

hinzu.

NOT NULL-Spezifikation

Mit der NOT NULL-Spezifikation können für Felder, die keine Nullwerte annehmen dürfen, Nullwerte ausgeschlossen werden. Unter einem Nullwert ist ein spezieller Wert zu verstehen, der die Bedeutung "unbekannt" besitzt. Für die Wirkung von Nullwerten gilt:

- Arithmetische Ausdrücke, in denen ein Operand mit einem Nullwert auftritt, nehmen den Wert Null an.
- Vergleiche, in denen ein Operand mit einem Nullwert auftritt, ergeben einen "unbekannten" Wahrheitswert.

Auf Schlüsselfelder ist die NOT NULL-Spezifikation in jedem Falle anzuwenden, da "unbekannte" Schlüsselwerte nicht zulässig sind.

IN SEGMENT-Spezifikation

(3) Den Abschluss einer CREATE TABLE-Anweisung bildet die optionale IN SEGMENT- Spezifikation. Dieser Spezifikation liegt folgende, den Datenbankzugang betreffende Konzeption zugrunde:

Eine Datenbank wird im System R in eine Menge von disjunkten Bereichen, genannt Segmente, unterteilt. Jede Basistabelle gehört genau einem der Segmente an. Ein Segment kann eine oder mehrere Basistabellen - einschließlich der zugehörigen Indizes - enthalten. Wesentlich ist nun die Unterscheidung in zwei Segmentarten:

- Ein öffentliches Segment (engl. public segment) enthält Basistabellen bzw. Daten, die gleichzeitig allen Datenbankbenutzern zugänglich sind.
- Ein privates Segment (engl. private segment) enthält Basistabellen bzw. Daten, die während einer Zeitphase nur einem Benutzer zur Verfügung stehen.

öffentliche und private Segmente

Soll eine Basistabelle in ein öffentliches Segment übernommen werden, so ist die IN SEGMENT-Spezifikation anzuwenden. In der Spezifikation "IN SEGMENT Geteiltes-Segment" beispielsweise ist "Geteiltes-Segment" der Name eines öffentlichen Segments. Falls die IN SEGMENT-Spezifikation fehlt, geht die definierte Basistabelle in ein privates Segment ein, das dem Benutzer gehört, der die Basistabelle definiert hat.

(4) Die Beschreibung der Tabelle wird in das Datenlexikon eingetragen. Die Tabelle ist damit existent.

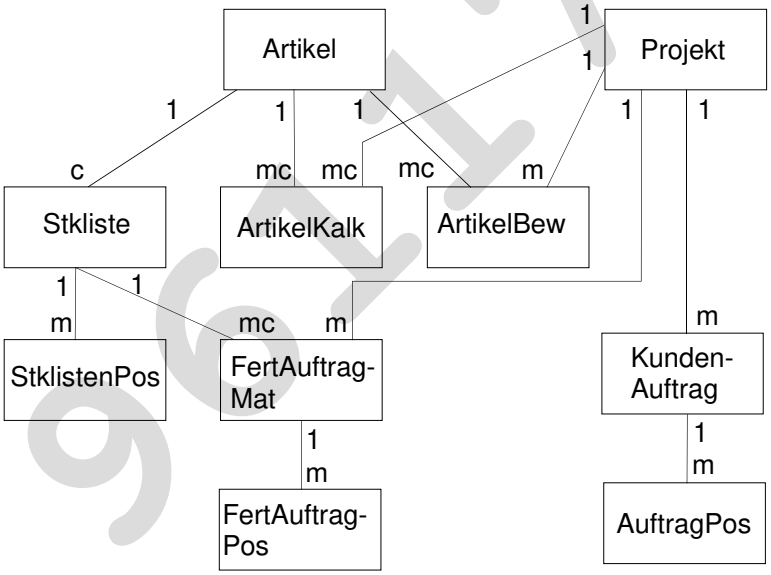
Datenlexikon

An einem Beispiel werde nun die Erzeugung von Basistabellen demonstriert.

Beispiel 4.4

Gegeben sei folgendes relationale Datenmodell, welches einen Auszug aus dem in Kapitel 3.5.1, Beispiel 3.6 beschriebenen Datenmodell darstellt:

Beispiel



Datenmodell

Folgende Relationen beschreiben die angegebenen Entitätsmengen:

Artikel(<u>ArtikelNr</u>, Bezeichnung, TechnDaten, Preis, Bestand, MindBestand, DispoBestand, AuftrBestand)

ArtikelKalk(<u>ArtikelNr</u>, <u>ProjektNr</u>, FertMenge, MatKosten, ZeitKosten, SondKosten, HerstellKosten)

ArtikelBew(<u>ArtikelNr</u>, <u>ProjektNr</u>, <u>BewDatum</u>, Menge, DispoMenge)

Stkliste(<u>StklisteNr</u>, ArtikelNr, StklisteDatum)

StklistenPos(<u>StklisteNr</u>, <u>Position</u>, ArtikelNr, Menge)

FertAuftragMat(<u>ProjektNr</u>, <u>StklisteNr</u>, Menge, FertDatum)

FertAuftragPos(<u>ProjektNr</u>, <u>StklisteNr</u>, <u>Position</u>, ArtikelNr, Menge)

Relationen

Projekt(<u>ProjektNr</u>, FertDatum, EinkaufWoche, LagerWoche, SollZeit, SollMat, SollMenge, IstMenge)

KundenAuftrag(<u>AuftrNr</u>, ProjektNr, KundenNr, Name, AuftrDatum)

AuftragPos(<u>AuftrNr</u>, <u>Position</u>, ArtikelNr, Menge, Preis)

Die SQL-Anweisungen zur Erzeugung von Basistabellen für diese Relationen lauten:

Erzeugung von Basistabellen

```
CREATE TABLE Artikel      (ArtikelNr      (INTEGER, NOT NULL),
                           Bezeichnung    (CHAR (15)),
                           TechnDaten     (CHAR (15)),
                           Preis          (INTEGER),
                           Bestand        (INTEGER),
                           MindBestand    (INTEGER),
                           DispoBestand   (INTEGER),
                           AuftrBestand   (INTEGER))

CREATE TABLE ArtikelKalk  (ArtikelNr      (INTEGER, NOT NULL),
                           ProjektNr      (INTEGER, NOT NULL),
                           FertMenge      (INTEGER),
                           MatKosten      (INTEGER),
                           ZeitKosten     (INTEGER),
                           SondKosten     (INTEGER),
                           HerstellKosten (INTEGER))

CREATE TABLE ArtikelBew   (ArtikelNr      (INTEGER, NOT NULL),
                           ProjektNr      (INTEGER, NOT NULL),
                           BewDatum       (DATE, NOT NULL),
                           Menge          (INTEGER),
                           DispoMenge     (INTEGER))

CREATE TABLE Stkliste     (StklisteNr     (INTEGER, NOT NULL),
                           ArtikelNr      (INTEGER, NOT NULL),
                           StklisteDatum  (DATE))

CREATE TABLE StklistenPos (StklisteNr     (INTEGER, NOT NULL),
                           Position       (INTEGER, NOT NULL),
                           ArtikelNr      (INTEGER, NOT NULL),
                           Menge          (INTEGER))

CREATE TABLE FertAuftrMat(ProjektNr       (INTEGER, NOT NULL),
                           StklisteNr     (INTEGER, NOT NULL),
                           Menge          (INTEGER),
                           FertDatum      (DATE))

CREATE TABLE FertAuftrPos(ProjektNr       (INTEGER, NOT NULL),
                           StklisteNr     (INTEGER, NOT NULL),
                           Position       (INTEGER, NOT NULL),
                           ArtikelNr      (INTEGER, NOT NULL),
                           Menge          (INTEGER))
```

```
CREATE TABLE Projekt        (ProjektNr      (INTEGER, NOT NULL),
                             FertDatum      (DATE),
                             EinkaufWoche   (INTEGER),
                             LagerWoche     (INTEGER),
                             SollZeit       (INTEGER),
                             SollMat        (INTEGER),
                             SollMenge      (INTEGER),
                             IstMenge       (INTEGER))

CREATE TABLE KundenAuftrag(AuftrNr      (INTEGER, NOT NULL),
                             ProjektNr      (INTEGER, NOT NULL),
                             KundenNr       (INTEGER, NOT NULL),
                             Name           (CHAR (15)),
                             AuftrDatum     (DATE))

CREATE TABLE AuftragPos     (AuftrNr       (INTEGER, NOT NULL),
                             Position       (INTEGER, NOT NULL),
                             ArtikelNr      (INTEGER, NOT NULL),
                             Menge          (INTEGER),
                             Preis          (INTEGER))
```

b) Erweiterung von Basistabellen (EXPAND TABLE-Anweisung)

Bereits vorhandene Basistabellen können mit Hilfe von EXPAND TABLE-Anweisungen um zusätzliche Felder erweitert werden. Die EXPAND TABLE-Anweisung lautet:

EXPAND TABLE-Anweisung

```
EXPAND TABLE basistabellenname
        ADD FIELD feldname (datentyp)
```

Mit einer EXPAND TABLE-Anweisung wird eine Basistabelle an der rechten Tabellenseite um genau ein Feld erweitert. Die Erweiterung betrifft alle Sätze bzw. Tupel der Tabelle. Der Wert des neuen Feldes ist für alle Sätze ein Nullwert. Man beachte, dass die NOT NULL-Spezifikation in der EXPAND TABLE-Anweisung nicht zulässig ist.

Im Falle von Beispiel 4.4 wäre z.B. folgende Erweiterung der Basistabelle *KundenAuftrag* denkbar:

```
EXPAND TABLE KundenAuftrag
        ADD FIELD Adresse (CHAR (50))
```

c) Löschen von Basistabellen (DROP TABLE-Anweisung)

Es genügt, in der Löschanweisung den Namen der zu löschenden Tabelle anzugeben:

DROP TABLE-Anweisung

```
DROP TABLE basistabellenname
```

Gelöscht werden nicht nur sämtliche Sätze der benannten Tabelle, sondern auch alle Indizes und Sichten auf diese Tabelle. Außerdem wird die Tabelle selbst eliminiert, d.h. ihre Beschreibung wird aus dem Datenlexikon entfernt, und der von der Tabelle beanspruchte Speicherplatz wird freigegeben.

Beispielsweise könnte die Basistabelle *AuftragPos* aus Beispiel 4.4 mit der Anweisung

```
DROP TABLE AuftragPos
```

gelöscht werden.

d) Erzeugung von Indizes (CREATE INDEX-Anweisung)

Mit einer CREATE INDEX-Anweisung lässt sich genau ein Index für eine Basistabelle erzeugen. Die CREATE INDEX-Anweisung besitzt folgende Form:

CREATE INDEX-Anweisung

```
CREATE [UNIQUE] INDEX indexname ON basistabellenname
        (feldname[order][,feldname[order]]...)
```

Bemerkungen:

(1) Es bezeichnen *indexname* den Namen des Index, der auf der Basistabelle mit dem Namen *basistabellenname* eingerichtet wird. Der Index kann für ein Feld oder für mehrere Felder gemeinsam definiert werden.

Sortierung der Sätze einer Basistabelle

(2) Abhängig vom Wert von *order* werden die Sätze in der zugehörigen Basistabelle wie folgt abgelegt:

- aufsteigend sortiert in Bezug auf das indizierte Feld, falls für *order* der Wert *ASC* angegeben wird und
- absteigend sortiert in Bezug auf das indizierte Feld, falls für *order* der Wert *DESC* angegeben wird.

Sofern die Angabe für *order* fehlt, wird der Voreinstellungswert *ASC* verwendet.

Bei einem für mehrere Felder definierten Index kommt es zu einer mehrstufigen Sortierung. Beispielsweise bewirkt die Anweisung

```
CREATE INDEX Ind ON Tab(A, B, C)
```

folgende Sortierung der Sätze der Tabelle *Tab*:

- Nach aufsteigenden *C*-Werten für einen festen *B*-Wert,
- nach aufsteigenden *B*-Werten für einen festen *A*-Wert und
- nach aufsteigenden *A*-Werten.

UNIQUE-Option

(3) Wird von der Option UNIQUE Gebrauch gemacht, so dürfen in der zugehörigen Basistabelle keine Sätze auftreten, die für das indizierte Feld bzw. die indizierte Kombination von Feldern gleiche Werte aufweisen.

(4) Die Anweisung bewirkt die Eintragung der Indexbeschreibung in das Datenlexikon. Der nun existierende Index wird automatisch von dem Zugriffssystem RSS des Systems R verwaltet.

Eintragung in das Datenlexikon

Macht man bei der Erzeugung eines Index von der UNIQUE-Option Gebrauch, so sollten erst nach der Indexerzeugung Daten in die zugehörige Tabelle eingetragen werden. Andernfalls kann die Tabelle bereits Sätze enthalten, die der UNIQUE-Option nicht genügen.

An dem bekannten Beispiel werde nun die Anwendung der CREATE INDEX-Anweisung demonstriert.

Fortsetzung Beispiel 4.4

Für die bereits definierten Basistabellen *Artikel, ArtikelKalk, ArtikelBew* usw. bieten sich beispielsweise folgende Indexdefinitionen an:

```
CREATE UNIQUE INDEX ArtikelInd ON Artikel (ArtikelNr)

CREATE UNIQUE INDEX ArtikelKalkInd ON ArtikelKalk (ArtikelNr, ProjektNr)

CREATE UNIQUE INDEX ArtikelBewInd ON ArtikelBew (ArtikelNr, ProjektNr,
    BewDatum)

CREATE UNIQUE INDEX StklisteInd ON Stkliste (StklisteNr)

CREATE UNIQUE INDEX StklistenPosInd ON StklistenPos (StklisteNr, Position)

CREATE UNIQUE INDEX FertAuftragMatInd ON FertAuftragMat (ProjektNr,
    StklisteNr)

CREATE UNIQUE INDEX FertAuftragPosInd ON FertAuftragPos (ProjektNr,
    StklisteNr, Position)

CREATE UNIQUE INDEX ProjektInd ON Projekt (ProjektNr)

CREATE UNIQUE INDEX KundenAuftragInd ON KundenAuftrag (AuftrNr)

CREATE UNIQUE INDEX AuftragPosInd ON AuftragPos (AuftrNr, Position)
```

Durchgehend wurde das Feld oder die Feldkombination indiziert, welche in der jeweiligen Basistabelle die Rolle des Schlüssels oder zusammengegsetzten Schlüssels übernimmt. Es obliegt dem Benutzer, Indizes für weitere Felder anzulegen.

Beispiel für Indexdefinitionen

e) Löschen von Indizes (DROP INDEX-Anweisung)

Auch hier genügt es, den Namen des zu löschenden Index anzugeben:

```
DROP INDEX indexname
```

DROP INDEX-Anweisung

Die Anweisung bewirkt die Zerstörung des benannten Index: Sämtliche Indexeinträge werden gelöscht, die Indexbeschreibung wird aus dem Datenlexikon entfernt und der von dem Index belegte Speicherplatz wird freigegeben.

Mit der Anweisung

```
DROP INDEX AuftragPosInd
```

könnte beispielsweise der zuvor definierte Index *AuftragPosInd* (siehe Beispiel 4.4) gelöscht werden.

Übungsaufgabe 4.10

In Beispiel 4.4 wurden lediglich für die Felder oder Feldkombinationen, die Identifikationsschlüssel darstellen, Indizes definiert. Unter welchen Umständen empfiehlt es sich, auch für ein Nichtschlüsselfeld einen Index vorzusehen?

Eigenschaften des Systems R

Die bisherigen Ausführungen lassen folgende Eigenschaften des Systems R erkennen:

(1) Basistabellen stellen nicht zwangsläufig Relationen dar.

Basistabellen und Relationen

Duplikatsätze und Duplikatwerte von Identifikationsschlüsseln sind durchaus zulässig. Eine Basistabelle, in die Duplikatsätze eingetragen wurden, stellt keine Relation dar. Schließt man jedoch Duplikate mittels der UNIQUE-Spezifikation aus und verwendet man zudem ein geeignetes Feld oder eine geeignete Feldkombination als Identifikationsschlüssel, so ist eine Basistabelle zumindest eine Relation in 1. Normalform. Ob sogar eine Relation in 2. oder 3. Normalform vorliegt, hängt davon ab, welche Nichtschlüsselfelder der Benutzer in die Tabelle einbezogen hat.

(2) Die Entitäts-Integrität ist optional.

optionales Verbot von Nullwerten

Die Bedingung der Entitäts-Integrität beinhaltet das Verbot von Nullwerten für Attribute bzw. Felder von Identifikationsschlüsseln. Im System R muss die Bedingung der Entitäts-Integrität explizit formuliert werden, indem die NOT NULL-Spezifikation auf die in die Identifikationsschlüssel einbezogenen Felder angewandt wird.

(3) Die referentielle Integrität wird nicht unterstützt.

referentielle Integrität nicht gewährleistet

Die Bedingung der referentiellen Integrität fordert bekanntlich, dass ein Fremdschlüssel in einer Sohnrelation nur solche Tupel der zugehörigen Vaterrelation referenzieren darf, die tatsächlich existieren. Das System R unterstützt diese Integritätsregel nicht. Der Benutzer erhält folglich keine Hinweise auf eingetretene Verletzungen der referentiellen Integritätsbedingung.

(4) Maßeinheiten von Feldern werden nicht unterstützt.

Maßeinheiten für Felder nicht definierbar

Das System R gestattet es nicht, Maßeinheiten für Felder zu definieren. Unsinnige Auswertungen, wie z.B. der Vergleich einer Gewichtsgröße mit einer Geldgröße, sind daher grundsätzlich nicht ausgeschlossen.

Übungsaufgabe 4.11

Stellen Sie für jede der eben genannten Eigenschaften (1) bis (4) des Systems R beispielhaft dar, welche Defizite sie zur Folge haben können.

4.4.3 Datenmanipulation mit SQL

Auswahlanweisungen und Mutationen

Die Datenmanipulationskomponente von SQL besteht aus der Menge der Anweisungen, welche auf den erzeugten Basistabellen ausgeführt werden können. Zu unterscheiden sind Auswahlanweisungen und Mutationen. Bei den Auswahlanweisungen besteht für den Benutzer die Möglichkeit, auf eingebettete Funktionen zurückzugreifen. Unter die Mutationen fallen Anweisungen zum Ändern, Einfügen und Löschen von Daten in Basistabellen. Insgesamt ergeben sich damit die in Abb. 4.8 gezeigten Gruppen von SQL-Anweisungen.

SQL- Anweisungen zur Datenmanipulation

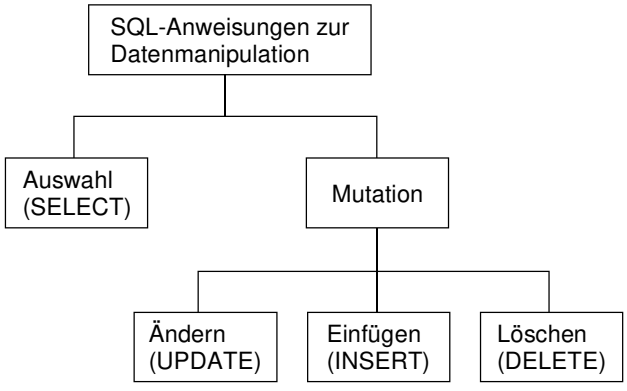


Abb. 4.8. Einteilung der SQL-Anweisungen zur Datenmanipulation.

In Abb. 4.8 wurde für jede Gruppe auch das Schlüsselwort angegeben, mit dem die zu der jeweiligen Gruppe gehörigen SQL-Anweisungen beginnen. Die Behandlung der SQL-Anweisungen zur Datenmanipulation orientiert sich an der vorgestellten Gruppierung. Auf die eingebetteten Funktionen wird gesondert eingegangen.

Da die Wirkungsweise der einzelnen Anweisungen auch an konkreten Daten erläutert werden soll, ist zunächst das Beispiel 4.4 fortzuschreiben.

Beispiel

Fortsetzung Beispiel 4.4

Angenommen sei, dass zu den bereits erzeugten und indizierten Basistabellen *Artikel*, *ArtikelKalk, ArtikelBew* usw. folgende Datensätze eingegeben wurden:

Basistabellen mit Datensätzen

Artikel

Artikel-Nr	Bezeichnung	TechnDaten	Preis	Bestand	Mind-Bestand	Dispo-Bestand	Auftr-Bestand
100100	Gehäuseblech	1 mm	100	50	20	0	0
100200	Schrauben	0,2 * 5 mm	10	150	50	0	0
810010	Netzteil	220 V 15 Watt	3015	10	3	0	0
810020	Netzteil	220 V 50 Watt	4075	2	5	0	5
820050	Empfänger UKW	5 Watt	2050	30	10	0	0
830010	Recorder	Stereo	3500	6	4	0	0
910030	Radio	Weltempfänger	8000	0	0	0	0
920020	Radio	4 Wellen	12000	2	0	0	0

ArtikelKalk

ArtikelNr	ProjektNr	FertMenge	Mat-Kosten	Zeit-Kosten	Sond-Kosten	Herstell-Kosten
910030	140400	10	30000	40000	5000	75000
910030	130320	5	16000	22000	5000	43000

ArtikelBew

ArtikelNr	ProjektNr	BewDatum	Menge	DispoMenge
810010	130400	120504	-5	0
820050	140100	120505	10	0
910030	130300	120505	-2	0
910030	130320	120505	-4	0
910030	140400	120505	10	0

Stkliste

StklisteNr	ArtikelNr	StklisteDatum
80910001	910030	120403
80920001	920020	120225

StklistenPos

StklisteNr	Position	ArtikelNr	Menge
80910001	1	810010	1
80910001	2	100100	3
80910001	3	100200	30

FertAuftragMat

ProjektNr	StklisteNr	Menge	FertDatum
140400	80910001	10	120505

FertAuftragPos

ProjektNr	StklisteNr	Position	ArtikelNr	Menge
140400	80910001	1	810010	10
140400	80910001	2	100100	30
140400	80910001	3	100200	300

Projekt

ProjektNr	Fert-Datum	Einkauf-Woche	Lager-Woche	SollZeit	SollMat	Soll-Menge	Ist-Menge
140400	120504	1225	1227	80	1090	5	0

KundenAuftrag

AuftrNr	ProjektNr	KundenNr	Name	AuftrDatum
21020	140400	1100	Meier	120401
21030	140400	1110	Müller	120405
21040	140400	1080	Abele	120405

AuftragPos

AuftrNr	Position	ArtikelNr	Menge	Preis
21020	1	910030	5	9000
21030	1	910030	5	8900
21040	1	920020	10	13500

Angemerkt sei noch, dass die Artikelnummer in Nummernkreise für einzelne Artikelarten gegliedert ist. Die ersten drei Stellen der Artikelnummer verschlüsseln folgende Artikelgruppen:

Artikelgruppen

100	Mechanik	110	Schrauben
		120	Diverse Kleinteile
200	Kabel	210	Kupferdrähte
300	Elektrik	310	Transistoren
		320	Kondensatoren
400	Bausteine	410	Integrierte Schaltkreise
800	Halbfertigteile	810	Transformatoren
		820	Empfängerplatinen
		830	Kassettenteile
900	Fertigteile	910	Radios
		920	Kassettenradios

Nun zu den SQL-Anweisungen.

a) Auswahl von Daten (SELECT-Anweisung)

Zur Auswahl von Daten steht in SQL die SELECT-Anweisung zur Verfügung. Die Grundform der SELECT-Anweisung lautet:

Grundform der SELECT-Anweisung

```
SELECT    daten
FROM      basistabellen
WHERE     bedingung
```

Nach dem Schlüsselwort SELECT sind die auszugebenden Daten zu spezifizieren und nach FROM die Basistabellen, auf die sich die Abfrage bezieht. Welche Bedingungen die abzufragenden Daten erfüllen müssen, ist nach dem Schlüsselwort WHERE anzugeben. Die Grundform der SELECT-Anweisung kann noch einige Erweiterungen erfahren, beispielsweise durch das Anhängen einer Sortieranforderung oder durch das Einbinden einer weiteren SELECT-Anweisung im Bedingungsteil.

Nachfolgend werden die verschiedenen Formen der SELECT-Anweisung vorgestellt, und zwar gegliedert in elf Gruppen. Jede Anweisung erhält eine Nummer, um die Übersicht zu erleichtern. Außerdem wird jede Anweisung auf die in Beispiel 4.4 beschriebenen Basistabellen angewandt.

(1) Einfache Auswahl

Projektion mit Duplikaten

Bei der einfachsten Form der Auswahl erübrigt sich die Angabe eines Bedingungsteils. Der Teil "WHERE bedingung" fällt daher weg.

```
(1.1) SELECT   ArtikelNr, DispoMenge
      FROM     ArtikelBew
```

Ergebnis:

ArtikelNr	DispoMenge
810010	0
820050	0
910030	0
910030	0
910030	0

Die Anweisung (1.1) ist nichts anderes als eine Projektion der Relation *ArtikelBew* auf die Attributkombination (*ArtikelNr*, *DispoMenge*), wobei das Ergebnis Duplikate enthalten kann.

Projektion ohne Duplikate mittels DISTINCT-Spezifikation

```
(1.2) SELECT DISTINCT  ArtikelNr,
                       DispoMenge
      FROM             ArtikelBew
```

Ergebnis:

ArtikelNr	DispoMenge
810010	0
820050	0
910030	0

Sollen Duplikate im Ergebnis nicht auftreten, so ist dies durch den Zusatz DISTINCT explizit anzugeben. Die Anweisung (1.2) schließt Duplikate aus.

Kopieren einer Basistabelle

```
(1.3) SELECT   *
      FROM     ArtikelBew
```

Ergebnis: Kopie von *ArtikelBew*

Eine Kopie der Tabelle *ArtikelBew* erzeugt die Anweisung (1.3). Das Symbol * (Stern) steht hierbei als Abkürzung für sämtliche Attribute bzw. Attributnamen der Relation *ArtikelBew*.

Übungsaufgabe 4.12

Wie lautet das Ergebnis der folgenden Auswahlanweisung?

```
SELECT    AuftrNr, Name, AuftrDatum
FROM      KundenAuftrag
```

(2) Qualifizierte Auswahl

Bei der qualifizierten Auswahl müssen die Ergebnisdaten einer Bedingung genügen, die im Teil "WHERE bedingung" zu spezifizieren ist. Betrachtet werde zunächst eine einfache Bedingung.

Auswahl mit Einzelbedingung

```
(2.1) SELECT   ArtikelNr
      FROM     ArtikelBew
      WHERE    BewDatum = 120505
      AND      ProjektNr = 140100
```

Ergebnis:

ArtikelNr
820050

Ermittelt werden die Nummern der Artikel, die am 5.5.12 eine Bewegung erfuhren und dem Projekt 140100 zuzuordnen sind.

Im Falle der Anweisung (2.1) besteht der Qualifikationsteil aus zwei Einzelbedingungen, die mit dem logischen Operator AND verknüpft werden. Außerdem richten sich beide Einzelbedingungen an die gleiche Tabelle, nämlich *ArtikelBew*.

Die Auswertung der Anweisung vollzieht sich wie folgt:

Für jeden Tupel der Relation *ArtikelBew* wird geprüft, ob

- der Wert von *BewDatum* gleich 120505 ist **und** ob
- der Wert von *ProjektNr* gleich 140100 ist.

Sind beide Bedingungen für einen Tupel erfüllt, so liegt ein zutreffender Tupel vor. Das auszugebende Ergebnis wird nun aus den Attributwerten der zutreffenden Tupel gebildet.

Eine Anweisung zur qualifizierten Auswahl kann sich wesentlich komplexer darstellen. Einerseits kann sie mehrere Tabellen betreffen und andererseits kann sie aus mehr als zwei verknüpften Einzelbedingungen bestehen. Für die Verknüpfung von Einzelbedingungen im Qualifikationsteil kommen die logischen Operatoren

Verknüpfung von Einzelbedingungen mit logischen Operatoren

AND, OR und NOT

in Frage. In den Einzelbedingungen selbst sind die Vergleichsoperatoren

und Vergleichsoperatoren

=, <>, <, <=, > und >=

zulässig. Zudem kann durch das Setzen von Klammern im Qualifikationsteil die Auswertungsfolge vorgegeben werden.

Übungsaufgabe 4.13

Wie lautet das Ergebnis der folgenden Auswahlanweisung?

```
SELECT   ArtikelNr
FROM     Artikel
WHERE    ArtikelNr > 900000
AND      Bestand < MindBestand
```

Übungsaufgabe 4.14

Wie lautet das Ergebnis der folgenden Auswahlanweisung?

```
SELECT   AuftrNr, Name, AuftrDatum
FROM     KundenAuftrag
WHERE    Name = 'Meier'
OR       ProjektNr = 140400
```

Setzen von Klammern

```
(2.2) SELECT   ArtikelNr, ProjektNr
      FROM     ArtikelBew
      WHERE    BewDatum = 120505
      AND      (ProjektNr = 140100
               OR Menge < 0)
```

Ergebnis:

ArtikelNr	ProjektNr
910030	130300
910030	130320
820050	140100

Es werden die Artikel- und die Projektnummer der Artikel ermittelt, die am 5.5.12 eine Bewegung erfahren haben und die entweder dem Projekt mit der *ProjektNr* 140100 zuzuordnen sind oder für die die Bewegung eine Entnahme darstellt. Entnahmen werden also durch negative Mengen und Zugänge durch positive Mengen dargestellt.

Das Setzen der Klammer bewirkt, dass zuerst der Ausdruck in der Klammer ausgewertet wird und dann die gesamte nach WHERE angegebene Bedingung. Dies geschieht in folgenden Schritten:

1. Schritt: Auswerten der Klammer

Für jeden Tupel von *ArtikelBew* wird geprüft, ob

- die *ProjektNr* gleich dem Wert 140100 ist **oder** ob
- der Wert von *Menge* kleiner Null ist.

Auswertung einer Klammer

Ist eine dieser beiden Bedingungen erfüllt oder sind beide Bedingungen erfüllt, so ist auch der gesamte Ausdruck in der Klammer erfüllt; dieser stellt seinerseits eine Bedingung dar, die aus zwei verknüpften Einzelbedingungen besteht.

2. Schritt: Auswerten der gesamten nach WHERE angegebenen Bedingung
Für jeden Tupel von *ArtikelBew* wird geprüft, ob
- der Wert von *BewDatum* gleich 120505 ist **und** ob
- die in der Klammer angegebene Bedingung erfüllt ist.

Sind beide Bedingungen für einen Tupel erfüllt, so liegt ein zutreffender Tupel vor.

Aus der Menge der zutreffenden Tupel wird nun das auszugebende Ergebnis gebildet.

Übungsaufgabe 4.15

Gesucht ist der Bestand der Fertigteile, deren Preis über 10000 liegt und deren Bestand größer als Null ist. Geben Sie die zugehörige Auswahlanweisung und das mit der Anweisung ermittelte Ergebnis an.

(3) Auswahl mit Sortierung

Hier geht es um die Sortierung des Ergebnisses. Eine Ergebnissortierung kann mittels der zusätzlichen Spezifikation

```
feldname [ORDER][,feldname[ORDER]]...
```

Ergebnissortierung mittels ORDER-Spezifikation

in analoger Weise wie in einer CREATE INDEX-Anweisung angefordert werden. Die Sortierung lässt sich auf mehrere Felder bzw. Attribute ausdehnen.

absteigende Sortierung

(3.1)
```
SELECT   ArtikelNr, ProjektNr
FROM     ArtikelBew
WHERE    BewDatum = 120505
ORDER    BY ProjektNr DESC
```

Ergebnis:

ArtikelNr	ProjektNr
910030	140400
820050	140100
910030	130320
910030	130300

Ermittelt werden die Artikel- und die Projektnummern der Artikel, die am 5.5.12 bewegt wurden; das Ergebnis wird absteigend nach der Projektnummer ausgegeben.

aufsteigende Sortierung

(3.2)
```
SELECT   ArtikelNr, Bestand
FROM     Artikel
WHERE    MindBestand >= 10
ORDER    BY Bestand
```

Ergebnis:

ArtikelNr	Bestand
820050	30
100100	50
100200	150

Ermittelt werden die Artikelnummer und der Bestand der Artikel, deren Mindestbestand 10 oder mehr Stück beträgt; das Ergebnis wird aufsteigend sortiert nach dem Bestand ausgegeben.

(4) Auswahl mit Verbund

Falls sich die Auswahl an mehrere Tabellen richtet, müssen im Qualifikationsteil zwei Maßnahmen ergriffen werden:

- Der herzustellende Verbund zwischen zwei oder mehr Tabellen ist explizit zu formulieren.
- Die Zugehörigkeit jedes angesprochenen Attributes zu einer der Tabellen ist mittels Punktqualifizierung in der Form

Punktqualifizierung

```
tabellenname.feldname
```

anzugeben; "feldname" bezeichnet hierbei ein angesprochenes Attribut bzw. Datenfeld.

Auswahl mit natürlichem Verbund

```
(4.1) SELECT DISTINCT Artikel.ArtikelNr,
                      Artikel.Bestand
      FROM    Artikel, ArtikelBew
      WHERE   ArtikelBew.ArtikelNr =
              Artikel.ArtikelNr
      AND     Artikel.MindBestand = 0
      AND     ArtikelBew.Bewdatum = 120505
```

Ergebnis:

ArtikelNr	Bestand
910030	0

Ermittelt werden die Artikelnummer und der Bestand der Artikel, die am 5.5.12 bewegt wurden und deren Mindestbestand gleich Null ist.

Das angegebene Ergebnis wird etwa in folgender Weise gebildet:

Da sich die Abfrage an zwei Tabellen richtet, müssen die zusammengehörigen Tupel bzw. Sätze miteinander verbunden werden. Hier gehören die Sätze zusammen, welche die gleiche Artikelnummer aufweisen. Über die Artikelnummer ist also der Verbund herzustellen. Dies leistet die Einzelbedingung:

ArtikelBew.ArtikelNr = Artikel.ArtikelNr

Folge der Auswertungsschritte

Das Ergebnis des Verbunds über die Artikelnummer lautet hier (aus Platzgründen werden einige Attribute nicht angegeben):

Artikel

ArtikelNr	...	Preis	Bestand	MindBestand	...
810010		3015	10	3	
820050		2050	30	10	
910030		8000	0	0	
910030		8000	0	0	
910030		8000	0	0	

ArtikelBew

ArtikelNr	ProjektNr	BewDatum	...
810010	130400	120504	
820050	140100	120505	
910030	130300	120505	
910030	130320	120505	
910030	140400	120505	

Aus dieser Tabelle sind die Tupel zu eliminieren, die nicht am 5.5.12 bewegt wurden und die keinen Mindestbestand von Null aufweisen:

ArtikelNr	...	Preis	Bestand	MindBestand	...	ArtikelNr	ProjektNr	BewDatum	...
910030		8000	0	0		910030	130300	120505	
910030		8000	0	0		910030	130320	120505	
910030		8000	0	0		910030	140400	120505	

Aus der erhaltenen Tabelle sind nun sämtliche Spalten bis auf die interessierenden Spalten *ArtikelNr* und *Bestand* zu entfernen:

ArtikelNr	Bestand
910030	0
910030	0
910030	0

In einem letzten Auswertungsschritt werden Duplikate eliminiert. Es entsteht damit das Ergebnis:

ArtikelNr	Bestand
910030	0

Die Auswertungsschritte bei einer Auswahl mit einem Verbund über zwei Tabellen lassen sich somit wie folgt zusammenfassen:

Auswertungsschritte bei einem Verbund über zwei Tabellen

- Verbinden der zusammengehörigen Tupel beider Tabellen.
- Eliminieren der Tupel, welche die Qualifikationsbedingung nicht erfüllen.
- Projizieren des Zwischenergebnisses auf die auszugebenden Attribute.
- Eliminieren von Duplikaten.

Bei einem Verbund über mehr als zwei Tabellen können einige der genannten Schritte mehrfach auftreten.

Verbund über mehr als zwei Tabellen

```
(4.2) SELECT DISTINCT AuftrNr, ArtikelNr,
                      Bestand
      FROM    KundenAuftrag, AuftragPos,
              Stkliste, StklistenPos,
              Artikel
      WHERE   KundenAuftrag.Name =
              'Müller'
      AND     KundenAuftrag.AuftrNr =
              AuftragPos.AuftrNr
      AND     AuftragPos.ArtikelNr =
              Stkliste.ArtikelNr
      AND     Stkliste.StklisteNr =
              StklistenPos.StklisteNr
      AND     StklistenPos.ArtikelNr =
              Artikel.ArtikelNr
      AND     StklistenPos.ArtikelNr <
              800000
```

Ergebnis:

AuftrNr	ArtikelNr	Bestand
21030	100100	50
21030	100200	150

Ermittelt werden die Auftragsnummer, die Artikelnummer und der Bestand der Artikel, die als Einzelteile in die von dem Kunden Müller georderten Aufträge eingehen.

Übungsaufgabe 4.16

Betrachtet werde die Auswahlanweisung (4.2). Erläutern Sie die wesentlichen Auswertungsschritte, die zu dem angegebenen Ergebnis führen. Gehen Sie hierbei insbesondere auf die Bildung von Verbunden ein.

Übungsaufgabe 4.17

Wie lautet das Ergebnis der folgenden Auswahlanweisung?

```
SELECT   DISTINCT Name
FROM     KundenAuftrag, AuftragPos
WHERE    KundenAuftrag.AuftrNr = AuftragPos.AuftrNr
AND      AuftragPos.ArtikelNr = 910030
```

Übungsaufgabe 4.18

Zu ermitteln sind die Nummer und das Fertigungsdatum für jedes Projekt, welches den Artikel mit der Nummer "910030" einschließt. Geben Sie die zugehörige Auswahlanweisung und das Ergebnis der Anweisung an.

Eine Tabelle kann auch mit sich selbst verbunden werden. In einer Auswahlanweisung tritt die gleiche Tabelle dann zweifach in Erscheinung. Zur Unterscheidung zwischen den beiden Fällen des Auftretens führt man zwei beliebige Namen, beispielsweise *First* und *Second*, ein und verwendet diese Namen als Qualifizierer in der SELECT- und in der WHERE-Klausel.

Verbund einer Tabelle mit sich selbst

(4.3)
```
SELECT   First.ProjektNr,
         Second.ProjektNr
FROM     ArtikelBew First,
         ArtikelBew Second
WHERE    First.BewDatum =
         Second.BewDatum
AND      First.ProjektNr <
         Second.ProjektNr
```

Ergebnis:

ProjektNr	ProjektNr
130300	130320
130300	140100
130300	140400
130320	140100
130320	140400
140100	140400

Ermittelt werden alle Paare von Projektnummern für Projektpaare, die an gleichen Tagen Artikelbewegungen aufweisen.

Für die Forderung, dass die erste Projektnummer kleiner als die zweite Projektnummer ist, lassen sich zwei Gründe nennen:

- Es werden Paare mit identischen Projektnummern ausgeschlossen, also Paare der Form (x,x).
- Es werden Paare ausgeschlossen, die aus anderen Paaren durch das Vertauschen der Projektnummern hervorgehen; von den beiden Paaren (x,y) und (y,x) ist nur ein Paar von Interesse.

(5) Auswahl mittels ANY-Klausel

In einer Auswahlanweisung kann der Qualifikationsteil aus einer ANY-Klausel bestehen. Die ANY-Klausel hat die Form:

ANY-Klausel

```
feldname = ANY(SELECT ... FROM ... WHERE ...)
```

Eine ANY-Klausel enthält folglich eine zwischen zwei Klammern zu setzende Subabfrage. Das Ergebnis der Subabfrage ist stets eine Menge von Werten, die mit dem Feld namens *feldname* in eine "ANY-Beziehung" gesetzt werden. Was darunter zu verstehen ist, sei an einem Beispiel gezeigt.

Auswahl mit ANY-Klausel

```
(5.1) SELECT   Name
      FROM     KundenAuftrag
      WHERE    AuftrNr = ANY (SELECT AuftrNr
                              FROM   AuftragPos
                              WHERE ArtikelNr =
                                    910030)
```

Ergebnis:

Name
Meier
Müller

Ermittelt werden die Namen der Kunden, welche den Artikel mit der Artikelnummer 910030 geordert haben.

Zur Bedeutung der ANY-Klausel:

Auswertung der ANY-Klausel

Der Ausdruck "feldname = ANY(...)" ist genau dann wahr, wenn irgendein Element bzw. Wert des Ergebnisses der Subabfrage (...) gleich dem Wert von *feldname* ist. Im gegebenen Fall liefert die Subabfrage folgende Ergebnismenge:

{21020, 21030}

Damit lautet die ANY-Klausel:

AuftrNr = ANY({21020, 21030})

In der Tabelle KundenAuftrag wird nun nacheinander jeder Tupel daraufhin geprüft, ob der Wert von *AuftrNr* gleich einem der Werte 21020 oder 21030 ist. Jeder Tupel, für den dies zutrifft, wird ausgewählt. Hier also die Tupel:

AuftrNr	ProjektNr	KundenNr	Name	AuftrDatum
21020	140400	1100	Meier	120401
21030	140400	1110	Müller	120405

Die Projektion dieses Zwischenergebnisses auf das Attribut Name liefert das bekannte Endergebnis:

Name
Meier
Müller

Neben der angegebenen ANY-Klausel sind noch weitere Versionen dieser Klausel zulässig:

```
feldname  <   ANY(SELECT...FROM...WHERE...)
feldname  ≤   ANY(SELECT...FROM...WHERE...)
feldname  >   ANY(SELECT...FROM...WHERE...)
feldname  ≥   ANY(SELECT...FROM...WHERE...)
feldname  <>  ANY(SELECT...FROM...WHERE...)
```

Formen der ANY-Klausel

Beispielsweise ist der Ausdruck "feldname < ANY(...)" genau dann wahr, wenn irgendein Element bzw. Wert des Ergebnisses der Subabfrage (...) größer als der Wert von *feldname* ist. Hierzu ein Abfragespiel:

```
(5.2) SELECT  ArtikelNr
      FROM    FertAuftragPos
      WHERE   Menge < ANY (SELECT  Menge
                           FROM    FertAuftragPos)
```

Ergebnis:

ArtikelNr
100100
810010

Auswahl mit < ANY(...)

Ermittelt werden die Nummern der Artikel, deren Fertigungsmenge kleiner ist als die maximale Fertigungsmenge.

Der Ausdruck "Menge < ANY ({10, 30, 300})" ist für die Tupel wahr, für die der Wert von *Menge* kleiner ist als der aktuell maximale Wert der Menge {10, 30, 300}. Leider führt die wörtliche Übersetzung von "ANY" in diesem Fall zu einer Missinterpretation und kann daher nicht in die Begründung des Ergebnisses einfließen!

ANY-Klauseln sind durchaus vermeidbar. So lässt sich die Auswahlanweisung (5.1) auch ohne ANY-Klausel formulieren:

Umgehung der ANY-Klausel

```
(5.3) SELECT  DISTINCT Name
      FROM    KundenAuftrag, AuftragPos
      WHERE   KundenAuftrag.AuftrNr =
              AuftragPos.AuftrNr
      AND     AuftragPos.ArtikelNr =
              910030
```

Ergebnis:

Name
Meier
Müller

Ermittelt werden die Namen der Kunden, welche den Artikel mit der Artikelnummer 910030 geordert haben. (Äquivalent ist die Abfrage (5.1)). Eine dritte Formulierungsmöglichkeit eröffnet die IN-Klausel, denn "= ANY" ist gleichbedeutend mit "IN".

(6) Auswahl mittels IN-Klausel

Auf die Bedeutung der IN-Klausel wurde eben hingewiesen. Die IN-Klausel kann im Qualifikationsteil einer Auswahlanweisung auftreten. Sie besitzt die Form:

```
feldname IN (SELECT ... FROM ... WHERE ...)
```

IN-Klausel

Der Ausdruck "feldname IN (...)" ist genau dann wahr, wenn der Wert von *feldname* in der Ergebnismenge der Subabfrage (...) vertreten ist. IN kann folglich als Mitgliedschaftsoperator ∈ verstanden werden.

Auswahl mit IN-Klausel

```
(6.1) SELECT   Name
      FROM     KundenAuftrag
      WHERE    AuftrNr IN
               (SELECT  AuftrNr
                FROM    AuftragPos
                WHERE   ArtikelNr = 910030)
```

Ergebnis:

Name
Meier
Müller

Ermittelt werden die Namen der Kunden, welche den Artikel mit der Artikelnummer 910030 geordert haben. (Äquivalent sind die Abfragen (5.1) und (5.3)).

Im gegebenen Fall lautet das Ergebnis der Subabfrage bekanntlich {21020, 21030}. In der Tabelle *KundenAuftrag* treten zwei Tupel mit einem dieser Werte auf. Die Namen in diesen Tupeln, nämlich Meier und Müller, bilden das Ergebnis der Auswahlanweisung (6.1).

Verschachtelung von IN-Klauseln

IN-Klauseln dürfen beliebig tief ineinander verschachtelt werden. Das folgende Beispiel enthält zwei ineinander verschachtelte IN-Klauseln.

Auswahl mit verschachtelten IN-Klauseln

```
(6.2) SELECT   ProjektNr, FertDatum
      FROM     Projekt
      WHERE    ProjektNr IN
               (SELECT ProjektNr
                FROM KundenAuftrag
                WHERE AuftrNr IN
                          (SELECT  AuftrNr
                           FROM    AuftragPos
                           WHERE   ArtikelNr =
                                   910030))
```

Ergebnis:

ProjektNr	FertDatum
140400	120504

Ermittelt werden die Nummer und das Fertigungsdatum für jedes Projekt, welches den Artikel mit der Nummer 910030 einschließt.

Jede Auswahlanweisung, welche IN-Klauseln bzw. = ANY-Klauseln enthält, lässt sich auch ohne Verwendung dieser Klauseln formulieren. Man vergleiche hierzu die Auswahlanweisungen (5.1), (5.3) und (6.1).

Übungsaufgabe 4.19

Geben Sie für die Auswahlanweisung (6.2) eine alternative Formulierung an, die ohne IN-Klausel und ohne = ANY-Klausel auskommt.

In einer Auswahlanweisung mit IN-Klausel, kann sich die Subabfrage auf die gleiche Tabelle beziehen. Das folgende Beispiel verdeutlicht dies.

```
(6.3) SELECT   Name
      FROM     KundenAuftrag
      WHERE    ProjektNr IN
               (SELECT ProjektNr
                FROM     KundenAuftrag
                WHERE    Name = 'Meier')
```

Ergebnis:

Name
Abele
Meier
Müller

IN-Klausel mit Subabfrage auf gleicher Tabelle

Ermittelt werden die Namen der Kunden, die in den Projekten vertreten sind, in denen auch der Kunde Meier vertreten ist.

Eine IN-Klausel darf auch in negierter Form verwendet werden. Sie hat dann die Gestalt: NOT IN (...). Hierzu ein Beispiel.

```
(6.4) SELECT   Name
      FROM     KundenAuftrag
      WHERE    AuftrNr NOT IN
               (SELECT  AuftrNr
                FROM     AuftragPos
                WHERE    ArtikelNr = 910030)
```

Ergebnis:

Name
Abele

Auswahl mit NOT IN-Klausel

Ermittelt werden die Namen der Kunden, welche den Artikel mit der Nummer 910030 nicht geordert haben.

(7) Auswahl mittels ALL-Klausel

Im Qualifikationsteil einer Auswahlanweisung kann auch die ALL-Klausel verwendet werden. Sie tritt in folgenden Formen auf:

```
feldname  =   ALL (SELECT ... FROM ... WHERE ...)
feldname  <>  ALL (SELECT ... FROM ... WHERE ...)
feldname  <   ALL (SELECT ... FROM ... WHERE ...)
feldname  ≤   ALL (SELECT ... FROM ... WHERE ...)
feldname  >   ALL (SELECT ... FROM ... WHERE ...)
feldname  ≥   ALL (SELECT ... FROM ... WHERE ...)
```

Formen der All-Klausel

Beispielsweise ist der Ausdruck "feldname <> ALL (...)" genau dann wahr, wenn alle Werte des Ergebnisses der Subabfrage (...) von dem Wert von feldname verschieden sind. Folglich ist "<> ALL (...)" identisch mit "NOT IN (...)".

Auswahl mit <> All-Klausel

```
(7.1) SELECT  Name
      FROM    KundenAuftrag
      WHERE   910030 <> ALL
              (SELECT  ArtikelNr
               FROM    AuftragPos
               WHERE   AuftragPos.AuftrNr =
                       KundenAuftrag.AuftrNr)
```

Ergebnis:

Name
Abele

Ermittelt werden die Namen der Kunden, welche den Artikel mit der Nummer 910030 nicht geordert haben. (Äquivalent ist die Abfrage (6.4)).

(8) Auswahl mittels EXISTS-Klausel

EXISTS ist ein Existenzquantor mit der Bedeutung "Es existiert ein ...". Der Ausdruck "EXISTS (SELECT...FROM...WHERE...)" ist folglich genau dann wahr, wenn das Ergebnis der Subabfrage (...) nicht leer ist.

Auswahl mit EXISTS-Klausel

```
(8.1) SELECT  Name
      FROM    KundenAuftrag
      WHERE   EXISTS
              (SELECT *
               FROM    AuftragPos
               WHERE   AuftrNr =
                       KundenAuftrag.AuftrNr
               AND     ArtikelNr = 910030)
```

Ergebnis:

Name
Meier
Müller

Ermittelt werden die Namen der Kunden, welche den Artikel mit der Nummer 910030 geordert haben. (Äquivalent sind die Abfragen (5.1), (5.3) und (6.1)).

Auch hier kann man von der Negationsform NOT EXISTS Gebrauch machen. Ein Beispiel möge dies verdeutlichen.

Auswahl mit NOT EXISTS-Klausel

```
(8.2) SELECT  Name
      FROM    KundenAuftrag
      WHERE   NOT EXISTS
              (SELECT *
               FROM    AuftragPos
               WHERE   AuftrNr =
                       KundenAuftrag.AuftrNr
               AND     ArtikelNr = 910030)
```

Ergebnis:

Name
Abele

Ermittelt werden die Namen der Kunden, welche den Artikel mit der Nummer 910030 nicht geordert haben. (Äquivalent sind die Abfragen (6.4) und (7.1)).

(9) Auswahl mit UNION

UNION ist nichts anderes als der Vereinigungsoperator "∪" der klassischen Mengenlehre. Mit dem Operator UNION können zwei Auswahlanweisungen miteinander verknüpft

werden. Das Ergebnis der so entstehenden Gesamtanweisung besteht aus der Zusammenfassung der Ergebnisse der beiden Einzelanweisungen. Duplikate werden bei der Bildung des Gesamtergebnisses eliminiert.

```
(9.1) SELECT   ArtikelNr
      FROM     Artikel
      WHERE    Bestand = MindBestand

      UNION

      SELECT   ArtikelNr
      FROM     ArtikelBew
      WHERE    BewDatum = 120505
```

Ergebnis:

ArtikelNr
820050
910030

Verknüpfung von Auswahlanweisungen mit UNION

Ermittelt werden die Nummern der Artikel, deren Bestand gleich dem Mindestbestand ist oder die am 5.5.12 bewegt wurden.

Im Beispiel (9.1) liefert die obere SELECT-Anweisung die Ergebnismenge {910030} und die untere SELECT-Anweisung die Ergebnismenge {820050, 910030}. Die Vereinigung beider Ergebnismengen ergibt das Gesamtergebnis {820050, 910030}.

(10) Auswahl mit berechneten Werten

Sowohl die SELECT-Klausel als auch die WHERE-Klausel dürfen arithmetische Ausdrücke einschließen, in denen Felder bzw. Feldnamen auftreten. Zulässige arithmetische Operatoren sind "/" (Division), "*" (Multiplikation), "+" (Addition) und "-" (Subtraktion).

arithmetische Operatoren

```
(10.1) SELECT   ArtikelNr, Preis / 100
       FROM     Artikel
       WHERE    ArtikelNr >= 900000
```

Ergebnis:

ArtikelNr	Preis
910030	80
920020	120

Auswahl unter Einschluss von Rechenoperationen

Ermittelt werden die Preise der Endprodukte in EUR. (Es wird davon ausgegangen, dass die in der Tabelle Artikel angegebenen Preise auf Cent lauten).

(11) Auswahl unter Einschluss von Nullwerten

Hier sei unterstellt, dass die Tabelle *AuftragPos* einen Nullwert enthält und folgendes Aussehen besitzt:

AuftragPos

AuftrNr	Position	ArtikelNr	Menge	Preis
21020	1	910030	5	9000
21030	1	910030	5	8900
21040	1	920020	?	13500

Basistabelle mit Nullwert

Der Satz mit dem Identifikationsschlüsselwert (21040,1) enthält also einen Nullwert. Betrachtet werde nun die Abfrage:

```
(11.1) SELECT  AuftrNr
       FROM    AuftragPos
       WHERE   Menge > 4
```

Ergebnis:

AuftrNr
21020
21030

Ermittelt werden die Nummern der Aufträge, für die die georderte Menge pro Auftragsposition größer als 4 ist.

Im Ergebnis tritt die Auftragsnummer des Tupels, welcher den Nullwert enthält, nicht auf. Und zwar deshalb, weil das Ergebnis des Vergleichs eines Nullwerts mit einem anderen Wert unabhängig vom Vergleichsoperator niemals wahr ist. So ergibt keiner der folgenden Vergleiche den Wert "wahr":

Vergleiche mit einem Nullwert

```
null < 4        null <> 4
null > 4        null  =  null
null = 4        null <> null
```

Ein spezielles Prädikat der Form:

Abfrage von Nullwerten

```
feldname IS [NOT] NULL
```

ermöglicht es in SQL, eine Tabelle daraufhin zu prüfen, ob für ein Feld Nullwerte vorliegen.

Auswahl unter Einschluss einer Nullwert-Abfrage

```
(11.2) SELECT  AuftrNr
       FROM    AuftragPos
       WHERE   Menge > 4
       OR      Menge IS NULL
```

Ergebnis:

AuftrNr
21020
21030
21040

Ermittelt werden die Nummern der Aufträge, für die die georderte Menge pro Auftragsposition größer als 4 oder unbekannt ist.

Vergleicht man die Ergebnisse der Abfragen (11.1) und (11.2), so kann man den Schluss ziehen, dass das Feld *Menge* für die Auftragsnummer 21040 einen Nullwert aufweist.

b) Auswahl mit eingebetteten Funktionen

Eine wesentliche Steigerung des Leistungsumfangs von SQL resultiert aus der Möglichkeit der Verwendung eingebetteter Funktionen. In der SELECT-Klausel einer Abfrage oder Subabfrage kann der Benutzer Funktionen angeben, die jeweils auf die Werte einer Tabellenspalte angewandt werden. Diese Funktionen sind:

eingebettete Funktionen

```
COUNT  -  Ermitteln der Anzahl von Werten einer Tabellenspalte
SUM    -  Bilden der Summe der Werte einer Tabellenspalte
AVG    -  Bilden des Durchschnitts der Werte einer Tabellenspalte
MAX    -  Ermitteln des größten Wertes einer Tabellenspalte
MIN    -  Ermitteln des kleinsten Wertes einer Tabellenspalte
```

Die Funktionen SUM und AVG sind nur auf numerische Werte anwendbar. Bei allen fünf Funktionen kann dem Funktionsargument das Schlüsselwort DISTINCT vorangestellt werden, um redundante Duplikatwerte auszuschließen. Einzig im Sonderfall von COUNT (*) darf DISTINCT nicht verwendet werden.

Ausschluss von Duplikatwerten mit DISTINCT

(12) Funktionen in der SELECT-Klausel

Betrachtet werde zunächst der Sonderfall COUNT (*). Bei dieser Funktion geht es um die Ermittlung aller Zeilen einer Tabelle.

```
(12.1) SELECT  COUNT (*)
       FROM    Artikel
```

Ergebnis:

COUNT (*)
8

Funktion COUNT(*) in der SELECT-Klausel

Ermittelt wird die Anzahl der Tupel bzw. Zeilen der Tabelle *Artikel*.

Im Gegensatz zu allen anderen Funktionen berücksichtigt COUNT(*) auch Nullwerte. So werden auch Zeilen mitgezählt, die ausschließlich Nullwerte enthalten. Bei allen anderen Funktionen werden die in einer Tabellenspalte eventuell vorhandenen Nullwerte erst eliminiert, bevor die Auswertung erfolgt.

Auswahl mit COUNT DISTINCT

```
(12.2) SELECT  COUNT (DISTINCT ArtikelNr)
       FROM    ArtikelBew
```

Ergebnis:

COUNT (DISTINCT ArtikelNr)
3

Ermittelt wird die Anzahl der unterschiedlichen Artikel, die in der Tabelle *ArtikelBew* vertreten sind.

(13) Funktionen in der SELECT-Klausel mit Prädikat

Mittels eines Prädikats kann die Anzahl der Elemente, auf die eine Funktion anzuwenden ist, eingeschränkt werden. Der Begriff "Prädikat" bezeichnet hier eine in der WHERE-Klausel anzugebende Bedingung.

-

```
(13.1) SELECT  COUNT (*)
       FROM    ArtikelBew
       WHERE   BewDatum = 120505
```

Ergebnis:

COUNT (*)
4

Auswahl mit COUNT (*) und Bedingung

Ermittelt wird die Anzahl der Artikelbewegungen, welche am 5.5.12 aufgetreten sind.

```
(13.2) SELECT  SUM (Menge)
       FROM    AuftragPos
       WHERE   ArtikelNr = 910030
```

Ergebnis:

SUM (Menge)
10

Auswahl mit SUM (...) und Bedingung

Ermittelt wird die Gesamtmenge der georderten Artikel mit der Nummer 910030.

Übungsaufgabe 4.20

Gesucht ist die Gesamtanzahl der Mechanikteile, die in der Stückliste des Enderzeugnisses mit der Artikelnummer 910030 auftreten. Geben Sie die zugehörige Auswahlanweisung und das Ergebnis an.

(14) Funktionen in Subabfragen

Wie bereits erwähnt wurde, sind Funktionen auch in Subabfragen zulässig. Hierzu ein Beispiel:

Auswahl mit MAX (...) in Subabfrage

```
(14.1) SELECT  AuftrNr
       FROM    AuftragPos
       WHERE   Menge <
               (SELECT  MAX (Menge)
                FROM    AuftragPos)
```

Ergebnis:

AuftrNr
21020
21030

Ermittelt werden die Nummern der Aufträge, für deren Auftragspositionen die bestellte Menge kleiner ist als die aktuell größte Menge einer Auftragsposition.

(15) Anwendung von GROUP BY und HAVING

GROUP BY und HAVING sind Funktionen, die der Bildung von Ergebnisdaten aus ausgewählten Daten dienen. Ebenso wie die ORDER-Spezifikation sind GROUP BY und HAVING im Anschluss an die WHERE-Klausel - falls eine solche vorhanden ist - anzugeben. Die Spezifikationsform lautet:

GROUP BY HAVING-Spezifikation

```
GROUP BY feldname [,feldname,...] [HAVING arithmetischer Ausdruck]
```

Gruppierung von Zeilen einer Tabelle

Die Funktion GROUP BY bildet aus den Zeilen der angesprochenen Tabelle Gruppen von Zeilen. Alle Zeilen einer Gruppe weisen für das GROUP BY-Feld oder die GROUP BY- Feldkombination den gleichen Wert auf. Auf jede der gebildeten Gruppen wird die SELECT-Klausel gesondert angewandt. Das Ergebnis der Anwendung der SELECT-Klausel auf eine Gruppe muss für jeden arithmetischen Ausdruck in der SELECT-Klausel je zu einem einzelnen Wert führen.

Auswahl mit GROUP BY-Spezifikation

```
(15.1) SELECT  ArtikelNr, SUM (Menge)
       FROM    AuftragPos
       GROUP   BY ArtikelNr
```

Ergebnis:

ArtikelNr	SUM (Menge)
910030	10
920020	10

Ermittelt werden die Nummer und die georderte Gesamtmenge für jeden Artikel.

Bei der Abfrage (15.1) werden zwei Gruppen gebildet, eine für die Artikelnummer 910030 und eine für die Artikelnummer 920020. Die erste Gruppe umfasst zwei Tupel bzw. Zeilen; über die dazugehörigen Mengen 5 und 5 wird die im Ergebnis angegebene Summe 10 gebildet. Die zweite Gruppe besteht aus nur einer Zeile mit der Menge 10.

HAVING kann nur in Verbindung mit der GROUP BY- Spezifikation verwendet werden. HAVING besitzt die gleiche Wirkung wie die WHERE-Klausel. Während jedoch die WHERE-Klausel alle Zeilen einer Tabelle betrifft, wird der nach HAVING anzugebende arithmetische Ausdruck für jede Gruppe gesondert ausgewertet. Pro Gruppe muss der arithmetische Ausdruck genau einen Wert liefern.

Auswahl mit GROUP BY HAVING-Spezifikation

```
(15.2) SELECT  ArtikelNr
       FROM    AuftragPos
       GROUP   BY ArtikelNr
       HAVING COUNT (*) > 1
```

Ergebnis:

ArtikelNr
910030

Ermittelt werden die Nummern der Artikel, die öfter als einmal geordert wurden.

Im Falle der Abfrage (15.2) werden die gleichen Gruppen gebildet wie bei der Abfrage (15.1). Die HAVING-Spezifikation fordert nun, dass die SELECT-Klausel nur auf solche Gruppen angewandt wird, für die die Bedingung "COUNT(*) > 1" gilt. Diese Bedingung erfüllen die Gruppen, welche aus mehr als einer Zeile bestehen. Hier ist dies lediglich die zur Artikelnummer 910030 gehörige Gruppe.

Abschließend werde ein Beispiel für eine Abfrage betrachtet, in der mehrere der behandelten Spezifikationen auftreten.

Beispiel 4.5

Gegeben sei eine Tabelle mit Auftragspositionen, die aus der in Beispiel 4.4 angegebenen Tabelle *AuftragPos* durch Hinzufügen weiterer Tupel bzw. Zeilen hervorgegangen ist:

AuftragPos

AuftrNr	Position	ArtikelNr	Menge	Preis
21020	1	910030	5	9000
21020	2	920020	8	13500
21020	3	920060	10	17000
21030	1	910030	5	8900
21030	2	920020	12	13000
21040	1	920020	20	13500
21040	2	910030	15	8500
21040	3	920060	20	16000
21040	4	930040	40	21000
21060	1	920020	5	13500

Gegenüber Beispiel 4.4 treten hier weitere Enderzeugnisse auf, die jedoch keiner weiteren Erläuterung bedürfen.

Betrachtet werde nun folgende Abfrage (vgl. DATE 1990, S. 156):

Auswahl mit mehreren Spezifikationen

```
SELECT   ArtikelNr, MAX (Menge)
FROM     AuftragPos
WHERE    Menge > 5
GROUP    BY ArtikelNr
HAVING   SUM (Menge) > 20
ORDER    BY 2, ArtikelNr DESC
```

Ergebnis:

ArtikelNr	MAX (Menge)
920060	20
920020	20
930040	40

Ermittelt werden die Artikel, für die die gesamte georderte Menge - jedoch ohne Berücksichtigung von Auftragspositionen mit Mengen von 5 oder darunter - größer als 20 ist; für die ermittelten Artikel werden die Artikelnummer und die maximale in einer Auftragsposition auftretende Menge ausgegeben und zwar absteigend sortiert nach der Maximalmenge und aufsteigend sortiert nach der Artikelnummer innerhalb der Maximalmenge.

Die Auswertung der Abfrage erfolgt in sechs Schritten:

Auswertungsschritte

(1) FROM AuftragPos:
Es wird eine Kopie der Tabelle *AuftragPos* angelegt. Die folgenden Manipulationen werden auf der Kopie ausgeführt.

(2) WHERE Menge > 5:
Sämtliche Zeilen mit einer Menge von 5 oder weniger werden eliminiert.

(3) GROUP BY ArtikelNr.:
Die verbleibenden Zeilen werden zu Gruppen mit gleicher Artikelnummer zusammengefasst.

(4) HAVING SUM (Menge) > 20:
Gruppen, für die die Summe der Mengen 20 oder weniger beträgt, werden eliminiert.

(5) SELECT ArtikelNr, MAX (Menge):
Aus den verbliebenen Gruppen wird je die größte in einer Auftragsposition bzw. Zeile auftretende Menge gewählt und zusammen mit der Artikelnummer in einer Zwischenergebnistabelle abgelegt.

(6) ORDER BY 2, ArtikelNr DESC:
Das Ergebnis ist nach MAX *(Menge)* aufsteigend zu sortieren und innerhalb von MAX *(Menge)* absteigend nach *ArtikelNr*. Da das zweite Feld keinen Namen besitzt -MAX *(Menge)*ist ein arithmetischer Ausdruck und hat keinen Namen -, wird seine ordinale Position in der ORDER BY-Spezifikation verwendet. Die ordinale Position beträgt hier 2.

Übungsaufgabe 4.21

Gegeben sei die in Beispiel 4.5 spezifizierte Tabelle *AuftragPos*. Wie lautet die Auswahlanweisung für folgende Abfrage:

Gesucht sind die Artikel, für die mehrere Auftragspositionen vorliegen; für jeden dieser Artikel sind die Artikelnummer und der im Durchschnitt pro Auftragsposition berechnete Preis auszugeben und zwar absteigend sortiert nach dem Durchschnittspreis.

Geben Sie neben der Abfrage auch das Ergebnis der Abfrage an.

c) Ändern von Daten (UPDATE-Anweisung)

UPDATE-Anweisung

Zu unterscheiden ist hier zwischen dem Ändern von Daten in nur einer Tabelle oder in mehreren Tabellen. In beiden Fällen muss die Integrität der Daten gewährleistet werden.

(16) Update in einer Tabelle

Das Ändern von Daten in einer Tabelle kann einen Satz oder mehrere Sätze betreffen. Möglich ist es beispielsweise auch, das Ändern von Daten mit einer Subabfrage zu verbinden. Zu jedem dieser drei Fälle wird nachfolgend ein Beispiel angegeben.

Update eines Satzes:

Update eines Satzes

```
(16.1)  UPDATE   Artikel
        SET      Bezeichnung = 'Gehäuse',
                 Preis = Preis + 10,
                 MindBestand = 25
        WHERE    ArtikelNr = 100100
```

Die Bezeichnung, der Preis und der Mindestbestand des Artikels mit der Nummer 100100 werden aufdatiert.

Update mehrerer Sätze:

Update mehrerer Sätze

```
(16.2)  UPDATE   Artikel
        SET      MindBestand = 2 * MindBestand
        WHERE    ArtikelNr >= 300000
        AND      ArtikelNr < 400000
```

Für Einzelteile aus dem Bereich der Elektrik wird der Mindestbestand verdoppelt.

Update mit Subabfrage:

Update mit Subabfrage

```
(16.3)  UPDATE   AuftragPos
        SET      Menge = 0
        WHERE    'Meier'=
                 (SELECT   Name
                  FROM     KundenAuftrag
                  WHERE    AuftrNr = AuftragPos.AuftrNr)
```

Alle Auftragspositionen des Kunden Meier werden storniert.

(17) Update in mehreren Tabellen

Grundsätzlich können mit einer UPDATE-Anweisung nur Daten in einer Tabelle geändert werden. Abhängig davon, welche Daten mit einer UPDATE-Anweisung geändert werden, sind zwei Fälle zu unterscheiden:

Konsistenz der Datenbank

- Die UPDATE-Anweisung belässt die Datenbank in einem konsistenten Zustand.
- Die UPDATE-Anweisung führt zu einer Verletzung der Konsistenz der Datenbank.

Im hier interessierenden zweiten Fall ist die Konsistenz der Datenbank durch eine oder mehrere weitere UPDATE-Anweisungen wieder herzustellen. An einem Beispiel sei dies verdeutlicht.

die Konsistenz verletzende UPDATE-Anweisung

```
(17.1)  UPDATE   KundenAuftrag
        SET      AuftrNr = 21010
        WHERE    AuftrNr = 21020
```

```
(17.2)  UPDATE   AuftragPos
        SET      AuftrNr = 21010
        WHERE    AuftrNr = 21020
```

die Konsistenz wiederherstellende UPDATE-Anweisung

Die Auftragsnummer 21020 wird auf den Wert 21010 abgeändert.

Nach der Durchführung der Anweisung (17.1) ist die Konsistenz der Datenbank (temporär) verletzt. Die Anweisung (17.2) stellt die Konsistenz wieder her.

An diesem Beispiel wird deutlich, dass bei der Änderung der Werte globaler Attribute Inkonsistenzen eintreten können, die durch weitere Änderungsanweisungen zu beheben sind.

d) Einfügen von Daten (INSERT-Anweisung)

INSERT-Anweisung

Das Einfügen von Daten ist im System SQL/R gleichbedeutend mit dem Einfügen von Sätzen in leere oder nichtleere Tabellen. Bei einem einzufügenden Satz müssen nicht notwendigerweise für alle Felder Werte angegeben werden. Zu unterscheiden ist zwischen dem Einfügen eines Satzes oder mehrerer Sätze in eine Tabelle.

(18) Einfügen eines Satzes

Zuerst sei ein Beispiel für das Einfügen eines vollständigen Satzes angegeben. Darunter ist ein Satz zu verstehen, der keine Nullwerte enthält.

```
(18.1)  INSERT INTO KundenAuftrag VALUES
                    (21060, 140100, 1170, 'Riedel', 920406)
```

Einfügen eines vollständigen Satzes in eine Tabelle

In die Tabelle *KundenAuftrag* wird ein Auftrag mit den angegebenen Daten eingefügt.

Da im System SQL/R die Anordnung der Felder einer Tabelle signifikant ist, müssen die Feldwerte in der richtigen Reihenfolge angeordnet werden.
Nun zum Einfügen eines unvollständigen Satzes.

```
(18.2)  INSERT INTO KundenAuftrag (AuftrNr, KundenNr, Name)  VALUES
                    (21070, 1190, 'Seiler')
```

Einfügen eines unvollständigen Satzes in eine Tabelle

In die Tabelle *KundenAuftrag* wird ein Satz eingefügt, der für die Felder *AuftrNr*, *KundenNr* und *Name* die angegebenen Werte und für die Felder *ProjektNr* und *AuftrDatum* Nullwerte enthält.

(19) Einfügen mehrerer Sätze

Angegeben werde ein Beispiel für das Einfügen mehrerer Sätze in eine leere Tabelle. Natürlich ist ein Einfügen mehrerer Sätze in eine nichtleere Tabelle auch möglich.

Einfügen mehrerer Sätze in eine leere Tabelle

```
(19.1)  INSERT  INTO Halbfertigteile
                SELECT  ArtikelNr
                FROM    Artikel
                WHERE   ArtikelNr >= 800000
                AND     ArtikelNr <  900000
```

In die einspaltige Tabelle *Halbfertigteile* werden die Nummern sämtlicher Halbfertigteile eingefügt.

e) Löschen von Daten (DELETE-Anweisung)

DELETE-Anweisung

Wie die UPDATE-Anweisung und die INSERT-Anweisung bezieht sich die DELETE-Anweisung stets auf eine Tabelle. Zu unterscheiden ist zwischen dem Löschen eines Satzes und dem Löschen mehrerer Sätze. "Mehrere" kann auch "alle" bedeuten.

(20) Löschen eines Satzes

Mit einer in der WHERE-Klausel anzugebenden Bedingung wird der zu löschende Satz identifiziert. Das folgende Beispiel verdeutlicht dies.

Löschen eines Satzes

```
(20.1)  DELETE  FROM Artikel
        WHERE   ArtikelNr = 920020
```

In der Tabelle *Artikel* wird der Satz mit der Artikelnummer 920020 gelöscht.

(21) Löschen mehrerer Sätze

Das Löschen von Daten ist auch in Verbindung mit einer Subabfrage möglich. Falls auf die Subabfrage mehrere Sätze zutreffen, werden diese gelöscht.

Löschen mehrerer Sätze

```
(21.1)  DELETE  FROM AuftragPos
        WHERE   120405 =
                (SELECT  AuftrDatum
                 FROM    KundenAuftrag
                 WHERE   AuftrNr = AuftragPos.AuftrNr)
```

Für die am 5.4.12 eingegangenen Kundenaufträge werden sämtliche Auftragspositionen gelöscht.

Recht einfach nimmt sich das Löschen aller Sätze einer Tabelle aus.

Löschen aller Sätze

```
(21.2)  DELETE FROM AuftragPos
```

In der Tabelle *AuftragPos* werden alle Sätze gelöscht.

Man beachte, dass im Falle der Anweisung (21.2) zwar alle Sätze gelöscht werden, nicht jedoch die Tabelle selbst. *AuftragPos* ist nun als leere Tabelle vorhanden und steht für weitere Manipulationen, beispielsweise für das Einfügen neuer Sätze, zur Verfügung.

Übungsaufgabe 4.22

Betrachtet werde die Anweisung (20.1). Diese Anweisung ist nicht unproblematisch. Erläutern Sie, welches Problem mit der Anweisung verbunden ist und wie man dieses Problem beheben kann.

Übungsaufgabe 4.23

Gegeben seien die in Beispiel 4.4 spezifizierten Tabellen. Wie lauten die SQL-Anweisungen zur Beantwortung der folgenden Abfragen? (Geben Sie jeweils auch das Abfrageergebnis an.)

a) Gesucht sind die Artikelnummern und die zu fertigenden Mengen der Halbfertigerzeugnisse, die im Rahmen des Projekts hergestellt werden, dem der Auftrag des Kunden Meier vom 1.4.12 zugeordnet wurde.

b) Gesucht sind die Artikelnummer, die Bezeichnung und der Lagerbestand derjenigen Mechanikteile, die in der am 3. April 2012 gültigen Stückliste des Enderzeugnisses mit der Artikelnummer 910030 auftreten und deren Bestand größer oder gleich dem jeweiligen Mindestbestand ist. Das Ergebnis ist absteigend sortiert nach dem Lagerbestand auszugeben.

4.4.4 Externe Ebene des Systems R

externe Schemata

und

Sichten

Zu Beginn des Kap. 4.4.1 wurde bereits dargelegt, dass ein Benutzer des Systems SQL/R unmittelbar auf Basistabellen operieren kann oder aber über zwischengeschaltete Sichten. Demnach ist die externe Ebene des Systems R gegeben durch eine Menge von Basistabellen und eine Menge von Sichten. Und entsprechend bestehen externe Schemata aus Definitionen von Basistabellen und aus Definitionen von Sichten.

Im System R ist eine Sicht eine virtuelle Tabelle, die aus einer Basistabelle oder aus mehreren Basistabellen abgeleitet ist. Die Definition einer Sicht wird im Datenlexikon des Systems gespeichert. Mit der Ablage der Definition einer Sicht im Datenlexikon ist die Sicht zur Benutzung verfügbar.

Nachfolgend wird dargelegt, wie Sichten im System R erzeugt, gelöscht und benutzt werden können.

a) Erzeugung von Sichten (DEFINE VIEW-Anweisung)

Eine Sicht lässt sich im System R mittels der DEFINE VIEW-Anweisung definieren. Die DEFINE VIEW-Anweisung besitzt folgende Form:

DEFINE VIEW-Anweisung

```
DEFINE  VIEW sichtname
            [(feldname[,feldname]...)]
            AS SELECT-Anweisung
```

Auf die Schlüsselwörter DEFINE VIEW folgen der Name der Sicht und die Namen der in die Sicht einbezogenen Felder. Falls die Angabe von Feldnamen fehlt, werden die Feldnamen der zugrundeliegenden Basistabellen verwendet. Den Abschluss bildet eine AS SELECT-Anweisung. Sie spezifiziert die Basistabellen sowie die Spalten und Zeilen der Basistabellen, die in die Sicht einbezogen werden sollen.

Teilmengen von Tabellenzeilen und -spalten in einer Sicht

Man beachte, dass im System R in eine Sicht nicht nur Teilmengen von Spalten von Basistabellen eingehen können, sondern auch Teilmengen von Zeilen.

Nun ein einfaches Beispiel für die Definition einer Sicht, die sich nur auf eine Basistabelle bezieht.

Definition einer Sicht auf eine Tabelle

```
DEFINE  VIEW EndErzStamm
            AS    SELECT   ArtikelNr, Bezeichnung, TechnDaten, Preis
                  FROM     Artikel
                  WHERE    ArtikelNr >= 900 000
```

In die Sicht *EndErzStamm* werden lediglich Felder mit Stammdatencharakter - also keine Bestands- oder Bewegungsdaten - für die Teilmenge der Enderzeugnisse einbezogen. Die Namen der Felder der Sicht werden aus der Basistabelle *Artikel* übernommen.

Bei dem folgenden Beispiel erstreckt sich die definierte Sicht auf drei Basistabellen. Es wird also ein Verbund gebildet.

```
DEFINE VIEW Erzeugnispreise (ErzNr, Grundpreis, Kunde, AuftrMenge,Endpreis)
        AS  SELECT  Artikel.ArtikelNr, Artikel.Preis, KundenAuftrag.Name,
                    AuftragPos.Menge, AuftragPos.Preis
            FROM    Artikel, KundenAuftrag, AuftragPos
            WHERE   Artikel.ArtikelNr = AuftragPos.ArtikelNr
            AND     AuftragPos.AuftrNr = KundenAuftrag.AuftrNr
```

Definition einer Sicht auf mehrere Tabellen

In der Sicht *Erzeugnispreise* entsprechen die Felder *ErzNr* und *Grundpreis* den Feldern *ArtikelNr* und *Preis* der Tabelle *Artikel*, das Feld *Kunde* dem Feld *Name* der Tabelle *KundenAuftrag* und die Felder *AuftrMenge* und *Endpreis* den Feldern *Menge* und *Preis* der Tabelle *AuftragPos*. Um Mehrdeutigkeiten anzuschließen, müssen die Felder der Sicht *Erzeugnispreise* neue Namen erhalten. Die Sicht enthält Felder aus den drei angegebenen Basistabellen. Sie erstreckt sich allerdings nicht auf alle Tupel der Tabelle *Artikel*. Vielmehr bezieht sich der in der WHERE-Klausel spezifizierte Verbund nur auf Enderzeugnisse.

Übungsaufgabe 4.24

Betrachtet werde die Sicht *Erzeugnispreise* auf die in Beispiel 4.4 angegebene Datenbank. Bekanntlich stellt eine Sicht eine virtuelle Tabelle dar, d.h. eine nur scheinbar existierende Tabelle. Stellen Sie die virtuelle Tabelle *Erzeugnisspreise*, wie sie sich dem Benutzer präsentiert, dar. Geben sie also die Spalten und die Zeilen der Tabelle an.

b) Löschen von Sichten (DROP VIEW-Anweisung)

Eine existierende Sicht kann jederzeit mit Hilfe einer DROP VIEW-Anweisung gelöscht werden. Die DROP VIEW-Anweisung besitzt folgende Form:

```
DROP VIEW sichtname
```

DROP VIEW-Anweisung

Mit der Anweisung

```
DROP VIEW Erzeugnispreise
```

Löschen einer Sicht

beispielsweise wird die Sicht *Erzeugnispreise* eliminiert. Die Definition der Sicht wird aus dem Datenlexikon entfernt und die Sicht ist damit nicht mehr existent. Die der Sicht zugrundeliegenden Basistabellen werden durch das Löschen der Sicht in keiner Weise berührt. Im umgekehrten Fall ist dies jedoch anders: Wird eine Basistabelle gelöscht, so hat dies automatisch das Löschen der Sichten zur Folge, die diese Basistabelle einbeziehen.

c) Datenmanipulation auf Sichten

Sobald eine Sicht definiert ist, kann sie - mit noch darzustellenden Einschränkungen - wie eine Basistabelle benutzt werden. Als Beispiel sei eine Auswahlanweisung betrachtet:

Auswahlanweisung auf einer Sicht

```
SELECT   ErzNr, Endpreis
FROM     Erzeugnispreise
WHERE    Kunde = 'Meier'
ORDER    BY ErzNr
```

Ermittelt werden die Nummer und der Endpreis der Artikel, die der Kunde Meier geordert hat; das Ergebnis wird aufsteigend sortiert nach der Erzeugnisnummer ausgegeben.

Da eine Sicht eine Art Fenster auf reale Daten und keine eigenständige Datenkopie ist, gilt für die Datenmanipulation auf Sichten im System R:

- Auswahloperationen sind uneingeschränkt möglich,
- Mutationen sind nur unter teils starken Einschränkungen möglich.

Beispielsweise gelten für UPDATE-Operationen auf einer Sicht folgende Einschränkungen:

Einschränkungen für UPDATE-Operationen

(1) Die Sicht muss sich auf genau eine Basistabelle beziehen.

(2) Jede Zeile der Sicht muss einer Zeile der Basistabelle entsprechen; die Sicht kann jedoch weniger Zeilen als die Basistabelle enthalten.

(3) Jede Spalte der Sicht muss einer Spalte der Basistabelle entsprechen; die Sicht kann jedoch weniger Spalten als die Basistabelle enthalten.

Wegen diesen Einschränkungen kann eine Sicht

Restriktionen für Sichten

- nicht durch eine Projektion mit Elimination von Duplikaten,
- nicht durch einen Verbund und
- nicht durch eine Vereinigung

erzeugt werden. Zudem darf eine Sicht keine GROUP BY-Spezifikation und keine berechneten Felder enthalten. Sichten, welche die genannten Einschränkungen erfüllen, heißen "updatable" und andernfalls "nicht updatable".

4.4.5 Datenlexikon des Systems R

Auf die Rolle des Datenlexikons im System R wurde bereits mehrfach hingewiesen. Im vorliegenden Kapitel werden diese Hinweise um einige grundsätzlicheren Bemerkungen ergänzt.

Datenlexikon enthält Daten über Daten

Unter Datenlexikon wird im System R eine Systemkomponente verstanden, welche "Daten über Daten" enthält. Genauer gesagt, welche Beschreibungen der Objekte verwaltet,

aus denen sich R-Datenbanken zusammensetzen. Bei einem relationalen System wie R sind solche Objekte Relationen (Basistabellen), Sichten, Attribute (Felder), Indizes usw. Wesentlich erleichtert wird die Verwaltung, wenn man das Datenlexikon seinerseits als eine Menge von Relationen gestaltet und auf das Datenlexikon die gleiche Sprache anwendet wie auf die Daten selbst. Im System R besteht das Datenlexikon aus einer Menge von Tabellen, die mit Hilfe der Sprache SQL wie Basistabellen manipuliert werden können. Exemplarisch seien zwei Relationen des R-Datenlexikons betrachtet, nämlich "Tables" und "Columns".

Datenlexikon besteht aus Relationen

Die Tabelle *Tables* enthält pro angelegter Basistabelle eine Zeile bzw. einen Satz mit u.a. folgenden Angaben:

Relation "Tables"

TName	-	Name der Basistabelle und zugleich primärer Schlüssel,
Creator	-	Instanz bzw. Person, welche die Tabelle erzeugt hat,
NCols	-	Anzahl der Spalten bzw. Felder der Basistabelle.

Die Tabelle Columns enthält pro Spalte bzw. Feld einer Basistabelle eine Zeile bzw. einen Satz mit u.a. folgenden Angaben:

Relation "Columns"

TName, CName	-	Namen der Tabelle und der Spalte, beide Namen bilden den zusammengesetzten primären Schlüssel,
ColType	-	Datentyp der Spalte bzw. des Feldes,
Length	-	Länge der Spalte bzw. des Feldes in Bytes.

Bereits aus diesen wenigen Angaben wird deutlich, dass man sich mit Hilfe des Datenlexikons ein genaues Bild über den Aufbau einer R-Datenbank machen kann. In diesem Zusammenhang sei nochmals betont, dass eine R-Datenbank nur diejenigen Basistabellen, Indizes, Sichten usw. enthält, die in dem Datenlexikon verzeichnet sind. Mit an das Datenlexikon gerichteten Abfragen kann man sich daher jederzeit über die aktuelle Struktur einer Datenbank informieren. Zwei Beispiele für solche Abfragen mögen dies verdeutlichen.

```
SELECT   TName
FROM     Columns
WHERE    CName = 'ArtikelNr'
```

Ermittelt werden die Namen sämtlicher Basistabellen, welche eine Spalte namens "ArtikelNr" enthalten.

an das Datenlexikon gerichtete Abfragen

```
SELECT   CName
FROM     Columns
WHERE    TName = 'Artikel'
```

Ermittelt werden die Namen sämtlicher Spalten bzw. Felder der Basistabelle *Artikel*.

Da bestimmte Manipulationen auf einem Datenlexikon den Inhalt des Lexikons und damit eventuell auch die Datenbankstruktur verändern können, sollte ein differenzierter Zugang zum Datenlexikon vorgesehen werden. So sollten dem gewöhnlichen Datenbankbenutzer zwar Auswahloperationen auf dem Datenlexikon zugestanden werden, nicht aber UPDATE-, INSERT- und DELETE-Operationen.

differenzierter Zugang zum Datenlexikon

Eigenschaften der Sprache SQL

Nachdem nun ein Überblick über das System SQL/R gegeben wurde, seien noch einige Anmerkungen zu den Eigenschaften der Sprache SQL angefügt. DATE (1990) stellt u.a. folgende Merkmale von SQL heraus:

(1) Relationale Vollständigkeit

Das Selektionsvermögen der Sprache SQL entspricht dem der Relationenalgebra. Es können folglich sämtliche sinnvollen Abfragen an relationale Datenbanken formuliert werden.

(2) Nichtprozeduralität

Bei SQL handelt es sich nicht um eine prozedurale Sprache, sondern um eine spezielle deskriptive Sprache. Denn es wird jeweils beschrieben was gesucht wird, und nicht wie die Suche durchzuführen ist.

(3) Datenunabhängigkeit

Sämtliche SQL-Anweisungen zur Datenmanipulation enthalten keine Referenz auf Zugriffspfade. SQL unterstützt damit die physische Datenunabhängigkeit uneingeschränkt.

(4) Einfache Erweiterbarkeit

Mit Hilfe von eingebetteten Funktionen kann das Auswahlvermögen von SQL sehr wirksam und auf einfache Weise erweitert werden.

Obwohl das System R nur ein Prototyp eines relationalen Systems ist, macht es Sinn, sich mit den Eigenschaften von R auseinanderzusetzen. Denn viele positive Eigenschaften von R flossen in heute marktgängige Systeme ein. Und andererseits umfasst das Marktangebot auch Systeme, welche den mit SQL/R gesetzten Standard noch nicht einmal erreichen. Eine sorgfältige Systemauswahl bleibt dem Benutzer daher leider nicht erspart.

Lösungen zu den Übungsaufgaben

Übungsaufgabe 4.1

Der Benutzer sollte eine deskriptive Datenmanipulatiossprache verwenden, da er dann lediglich die gewünschten Ergebnisse zu spezifizieren hat. Auch bei stark variierenden Abfragen bleibt in diesem Fall der mit der Abfrageformulierung verbundene Aufwand im Rahmen. Andere Verhältnisse liegen bei der Verwendung einer prozeduralen Datenmanipulationssprache vor. Dieser Sprachtyp erfordert die detaillierte, prozedurale Ausformulierung von Abfragen. Stark variierende Abfragen verursachen daher einen beträchtlichen Formulierungsaufwand.

Übungsaufgabe 4.2

Ein gelegentlicher Benutzer ist beispielsweise ein Manager, welcher von Zeit zu Zeit Abfragen zur Entwicklung und zur Aufgliederung des Unternehmensumsatzes an eine Datenbank richtet.

Ein routinemäßiger Benutzer einer Datenbank ist zum Beispiel ein Sachbearbeiter, der mit Hilfe eines interaktiven Datenbank-Anwendungsprogramms täglich Kundenangebote bearbeitet.

Ein professioneller Datenbankbenutzer ist zum Beispiel ein Programmierer, der für die Behebung von Fehlern in bestimmen Datenbank-Anwendungsprogrammen sowie für den Test dieser Programme zuständig ist.

Übungsaufgabe 4.3

Der freien Abfrageform könnte sich z.B. ein regionaler Vertriebsleiter bedienen, um Informationen wie Datum der letzten Bestellung, Bestellumfang, offene Rechnungen usw. über einen bestimmten Kunden im Zusammenhang mit der Frage der Weiterführung der Geschäftsbeziehungen mit diesem Kunden einzuholen.

Als Beispiel für eine vorgegebene Abfrage im Rahmen eines interaktiven Verarbeitungsprozesses sei die Anforderung sämtlicher Stammdaten eines Kunden, dessen Adresse und Bankverbindung sich geändert haben, genannt. In diesem Fall der Dateipflege ist als Parameter z.B. die Kundennummer einzugeben.

Auf die Stapelverarbeitung greift man häufig bei der Reorganisation von Datenbanken zurück. Ein Reorganisationszweck kann z.B. im Löschen der Sätze einer Auftragsdatei bestehen, deren Auftragsdatum vor dem Datum eines vorgegebenen Stichtags liegt.

Übungsaufgabe 4.4

Die Ausführung von vorgegebenen Abfragen und von datenbankgestützten Stapelverarbeitungsprogrammen kann die Änderung von Daten der Datenbank oder die Neueingabe von Daten in die Datenbank einschließen. In beiden Fällen werden Daten in der Datenbank abgespeichert. Dies sollen die mit "Datenspeicherung" beschrifteten Pfeile in den Abbildungen 4.3 und 4.4 ausdrücken.

Übungsaufgabe 4.5

Es handelt sich hierbei um die Bedingung der referentiellen Integrität. In dem gegebenen Beispiel fordert diese Bedingung, dass für das globale Attribut *ArtikelNr* in der Vaterrelation *Artikel* ein statischer Wertebereich und in der Sohnrelation *ArtikelBew* ein dynamischer Wertebereich zu verwenden ist. Das Attribut *ArtikelNr* darf in der Relation *ArtikelBew* also nur solche Werte annehmen, welche für dieses Attribut in der Relation *Artikel* auch tatsächlich vorhanden sind.

Übungsaufgabe 4.6

Bei der in Beispiel 4.2 behandelten Datenbankanwendung finden sämtliche Benutzerinteraktionen vor dem Beginn der eigentlichen Stapelverarbeitung statt. Die Interaktionen dienen dazu, einen Stapelverarbeitungslauf vorzubereiten. So werden z.B. die zu reorganisierenden Daten eingegrenzt und eine Sicherheitsabfrage ("Wirklich reorganisieren?") abgewickelt. Erst danach werden die Transaktionen auf der Datenbank ausgeführt. Da dem Benutzer hierbei keine Eingriffsmöglichkeiten eingeräumt werden, ist es gerechtfertigt, von datenbankgestützter Stapelverarbeitung zu sprechen.

Übungsaufgabe 4.7

Die Ergebnisrelation *S* wird wie folgt ermittelt:

- Zuerst wird eine Hilfsrelation *X* gebildet, um mit der Konstanten "Laborant" arbeiten zu können:

  $X(x)$ = Laborant

  X

x
Laborant

- Nun wird mittels eines natürlichen Verbunds der Relation *Personal* und der Hilfsrelation *X* eine Relation *S'* gebildet, welche nur Tupel enthält, für die das Attribut *Funktion* den Wert "Laborant" aufweist:

  $S' := Personal[Personal.Funktion = x]X$

S´

MName	Abteilung	Funktion
Abs	112	Laborant
Riedle	114	Laborant

- Durch einen weiteren natürlichen Verbund der Relationen *S'* und *Gehalt* über das Attribut *MName* wird anschließend eine Relation *S''* erzeugt, welche die bereits in S' vorhandenen Tupel um die Attribute *Grundgehalt* und *Zulagen* erweitert:

  S'': = *S* ' [*S* '.*MName* = *Gehalt.MName*]*Gehalt*

  S"

MName	Abteilung	Funktion	Grundgehalt	Zulagen
Abs	112	Laborant	40000	15000
Riedle	114	Laborant	40000	10000

- Im letzten Schritt wird die Ergebnisrelation *S* durch die Projektion der Relation *S''* auf die Attributkombination (*MName, Zulagen*) gebildet:

  S: = *S''*[*MName*, *Zulagen*]

  S

MName	Zulagen
Abs	15000
Riedle	10000

Übungsaufgabe 4.8

Ein möglicher Lösungsweg könnte wie folgt aussehen:

- Zunächst wird eine Hilfsrelation *X* für die Konstante "Chemiker" gebildet:

  X(*x*) = Chemiker

  X

x
Chemiker

- Mit Hilfe eines natürlichen Verbundes können nun sämtliche in der Relation *Personal* verzeichneten Chemiker in eine neue Relation *P'* übernommen werden:

  P': = *Personal*[*Personal.Funktion* = *x*]*X*

  P'

MName	Abteilung	Funktion
Beer	116	Chemiker

- Die Projektion der Relation *P*' auf die interessierenden Attribute *MName* und *Abteilung* überführt die Relation *P'* in die gewünschte Form:

  P": = *P*'[*MName*,*Abteilung*]

P''

MName	Abteilung
Beer	116

- Mit Hilfe einer Projektion können nun die in der Relation *Gehalt* verzeichneten Mitarbeiter, deren Zulagen das Grundgehalt übersteigen, in eine neue Relation *G'* einbezogen werden:

  G': = *Gehalt*[*Gehalt.Zulagen* > *Gehalt.Grundgehalt*]

  G'

MName	Grundgehalt	Zulagen
Hein	45000	50000

- Durch einen natürlichen Verbund der Relationen *G'* und *Personal* kann die Relation *G'* um die Attribute von *Personal* erweitert werden:

  G'': = *G'*[*G'*. *MName* = *Personal*. *MName*]*Personal*

  G''

MName	Grundgehalt	Zulagen	Abteilung	Funktion
Hein	45000	50000	112	Konstrukteur

- Die Projektion der Relation *G''* auf die interessierenden Attribute *MName* und *Abteilung* führt zu der erwünschten Reduktion von *G''*:

  G''' = *G''*[*MName*,*Abteilung*]

  G'''

MName	Abteilung
Hein	112

- Durch die Vereinigung der beiden vereinigungsverträglichen Relationen *P''* und *G'''* erhält man die Ergebnisrelation S:

  S: = *P''* ∪ *G'''*

  S

MName	Abteilung
Beer	116
Hein	112

Übungsaufgabe 4.9

Datenmanipulationen auf der konzeptionellen Ebene implizieren, dass die Sicht der Benutzer auf die angelegten Basistabellen nicht eingeschränkt wird. Dies ist lediglich dann von Vorteil, wenn keine zentrale Instanz existiert, welche die Datenbank pflegt. In einem solchen Fall pflegen die Benutzer gemeinschaftlich die Datenbank. Dies setzt voraus, dass jeder Benutzer Zugang zu "seinen" Basistabellen hat.

Der unbeschränkte Zugang der Benutzer zur konzeptionellen Ebene bringt erhebliche Nachteile mit sich. So werden die Vorteile, welche aus der Differenzierung in die externe und in die konzeptionelle Ebene des 3-Schichten-Modells resultieren, stark beschnitten. Insbesondere kann die logische Datenunabhängigkeit nicht mehr gewährleistet werden,

da Datenmanipulationen auf der konzeptionellen Ebene auch die Veränderung von Datenstrukturen, wie z.B. das Eliminieren von Attributen, einschließen. Problematisch sind solche Eingriffe dann, wenn sie von verschiedenen Benutzern ohne gegenseitige Abstimmung vorgenommen werden.

Übungsaufgabe 4.10

Für Felder oder Feldkombinationen, die nicht Identifikationsschlüssel sind, sollte dann ein Index vorgesehen werden, wenn diese Felder oder Feldkombinationen immer wieder als Kriterien für den Datenzugriff im Rahmen von Abfragen oder von Datenmanipulationen dienen. Unterstellt man beispielsweise, dass ein nicht näher zu charakterisierender Benutzer der in Beispiel 4.4 beschriebenen Datenbasis häufig alle Lagerbewegungen eines Tages erfragt, so erscheint es sinnvoll, einen Index für das Feld *BewDatum* in der Basistabelle *ArtikelBew* vorzusehen.

Übungsaufgabe 4.11

(1) Basistabellen stellen nicht zwangsläufig Relationen dar: Es können bei ungeeigneter Definition eines Feldes *ArtikelNr* in einer Tabelle *Artikel* zwei Artikel mit der gleichen Artikelnummer erfasst werden.

(2) Die Entitäts-Integrität ist optional: Arithmetische Ausdrücke wie Menge ∗ Preis können undefinierte Werte ergeben, wenn für die entsprechenden Felder Nullwerte erlaubt sind.

(3) Die referentielle Integrität wird nicht unterstützt: In einer Vaterrelation *Artikel* können z.B. Artikel gelöscht werden, obwohl in einer Sohnrelation *ArtikelBew* noch Bewegungen zu den Artikeln vorhanden sind.

(4) Maßeinheiten von Feldern werden nicht unterstützt: Unsinnige Auswertungen wie beispielsweise Artikelnummer ∗ Preis werden ausgeführt.

Übungsaufgabe 4.12

Das Ergebnis lautet:

AuftrNr	Name	AuftrDatum
21020	Meier	120401
21030	Müller	120405
21040	Abele	120405

Übungsaufgabe 4.13

Das Ergebnis ist die leere Menge. Es existiert also kein Fertigteil (ArtikelNr > 900000) für das der Bestand kleiner ist als der Mindestbestand (Bestand < MindBestand).

Übungsaufgabe 4.14

Das Ergebnis lautet:

AuftrNr	Name	AuftrDatum
21020	Meier	120401
21030	Müller	120405
21040	Abele	120405

Ermittelt werden Angaben für die Kundenaufträge, die von dem Kunden "Meier" erteilt wurden **oder** die dem Projekt mit der Nummer "140400" zugeordnet wurden **oder** für die beide Bedingungen erfüllt sind.

Übungsaufgabe 4.15

Auswahlanweisung:

```
SELECT   ArtikelNr, Bestand
FROM     Artikel
WHERE    ArtikelNr  > 900000
AND      Preis      > 10000
AND      Bestand    > 0
```

Ergebnis:

ArtikelNr	Bestand
120020	2

Eine Klammersetzung erübrigt sich hier.

Übungsaufgabe 4.16

Die Auswertung der gegebenen Auswahlanweisung vollzieht sich etwa in folgenden Schritten:

(1) Ermittlung der Aufträge des Kunden "Müller" durch Zugriff zur Tabelle *Kunden-Auftrag* (vereinfachend werden nur die interessierenden Attribute angegeben):

Kundenauftrag

AuftrNr	...	Name	...
21030	...	Müller	...

(2) Ermittlung der von dem Kunden "Müller" georderten Artikel mittels eines Verbundes von *KundenAuftrag* und *AuftragPos* über das Attribut *AuftrNr*:

Kundenauftrag			AuftragPos		
AuftrNr ...	Name ...		AuftrNr ...	ArtikelNr ...	
21030 ...	Müller ...		21030 ...	910030 ...	

(3) Ermittlung der Stückliste, welche zu dem in Schritt (2) gefundenen Endprodukt gehört, mit Hilfe eines Verbunds von *AuftragPos* und *Stkliste* über das Attribut *ArtikelNr*:

AuftragPos			Stkliste		
AuftrNr ...	ArtikelNr ...		StklisteNr ...	ArtikelNr ...	
21030 ...	910030 ...		80910001 ...	910030 ...	

(4) Ermittlung der Teile, welche in die in Schritt (3) ermittelte Stückliste eingehen, mit Hilfe eines Verbundes von *Stkliste* und *StklistenPos* über das Attribut *StklisteNr*:

Stkliste			StklistenPos		
StklisteNr ...	ArtikelNr ...		StklisteNr ...	ArtikelNr ...	
80910001 ...	910030 ...		80910001 ...	810010 ...	
80910001 ...	910030 ...		80910001 ...	100100 ...	
80910001 ...	910030 ...		80910001 ...	100200 ...	

(5) Erweiterung der Sätze der Tabelle *StklistenPos*, welche im Schritt (4) gefunden wurden, um zusätzliche Attribute mit Hilfe eines Verbundes von *StklistenPos* und *Artikel* über das Attribut *ArtikelNr*:

StklistenPos			Artikel		
StklisteNr ...	ArtikelNr ...		ArtikelNr ...	Bestand ...	
80910001 ...	810010 ...		810010 ...	10 ...	
80910001 ...	100100 ...		100100 ...	50 ...	
80910001 ...	100200 ...		100200 ...	150 ...	

(6) Selektion der Teile aus dem vorgenommenen vierfachen Verbund, welche weder Endprodukte noch Halbfertigprodukte darstellen:

Kundenauftrag	AuftragPos	Stkliste		StklistenPos	
AuftrNr ...	AuftrNr ...	StklisteNr ...	ArtikelNr ...	StklisteNr ...	ArtikelNr ...
21030 ...	21030 ...	80910001 ...	910030 ...	80910001 ...	100100 ...
21030 ...	21030 ...	80910001 ...	910030 ...	80910001 ...	100200 ...

Artikel	
ArtikelNr ...	Bestand ...
100100 ...	50 ...
100200 ...	150 ...

(7) Projektion des nach dem Schritt (6) vorliegenden Zwischenergebnisses auf die interessierenden Attribute *AuftrNr*, *ArtikelNr* und *Bestand*:

AuftrNr	...	ArtikelNr...	Bestand
21030	...	100100	... 50
21030	...	100200	... 150

Anmerkung:

In den Schritten (3), (4) und (5) wurden lediglich die vorzunehmenden aktuellen Verbunde dargestellt, nicht aber die dann bereits vorliegenden Zwischenergebnisse, die aus zwei, drei und vier nacheinander vorgenommenen Verbunden resultieren.

Übungsaufgabe 4.17

Ergebnis:

Name
Meier
Müller

Ermittelt werden die Namen der Kunden, welchen den Artikel mit der Artikelnummer "910030" geordert haben.

Übungsaufgabe 4.18

Auswahlanweisung:

```
SELECT DISTINCT ProjektNr, FertDatum
FROM    AuftragPos, KundenAuftrag, Projekt
WHERE   Projekt.ProjektNr =
        KundenAuftrag.ProjektNr
AND     KundenAuftrag.AuftrNr =
        AuftragPos.AuftrNr
AND     AuftragPos.ArtikelNr = 910030
```

Ergebnis:

ProjektNr	FertDatum
140400	120504

Übungsaufgabe 4.19

```
SELECT DISTINCT ProjektNr, FertDatum
FROM    AuftragPos, KundenAuftrag, Projekt
WHERE   Projekt.ProjektNr =
        KundenAuftrag.ProjektNr
AND     KundenAuftrag.AuftrNr =
        AuftragPos.AuftrNr
AND     AuftragPos.ArtikelNr = 910030
```

Ergebnis:

ProjektNr	FertDatum
140400	120504

Übungsaufgabe 4.20

```
SELECT   SUM(Menge)
FROM     Stkliste, StklistenPos, Artikel
WHERE    Stkliste.ArtikelNr=910030
AND      Stkliste.StklisteNr=StklistenPos.StklisteNr
AND      StklistenPos.ArtikelNr=Artikel.ArtikelNr
AND      Artikel.ArtikelNr>=100000
AND      Artikel.ArtikelNr<200000
```

Ergebnis:

SUM (Menge)
33

Übungsaufgabe 4.21

```
SELECT     ArtikelNr, AVG (Preis)
FROM       AuftragPos
GROUP BY ArtikelNr
HAVING COUNT (*) > 1
ORDER BY 2 DESC
```

Ergebnis:

ArtikelNr	AVG (Preis)
920060	16500
920020	13375
910030	8800

Erläuterungen:

(1) FROM AuftragPos:
Es wird eine Kopie der Tabelle *AuftragPos* angelegt. Die Operationen werden auf dieser Kopie ausgeführt.

(2) GROUP BY ArtikelNr:
In der Kopie der Tabelle werden Gruppen mit gleichen Artikelnummern gebildet.

(3) HAVING COUNT (*) > 1
Es werden die Gruppen der Artikel eliminiert, für die nur eine Auftragsposition vorliegt.

(4) SELECT ArtikelNr, AVG (Preis)
Für jede Gruppe von Artikeln wird der Durchschnittspreis berechnet. Die Durchschnittspreise pro Artikel werden mit der Projektion der kopierten Tabelle auf das Attribut *ArtikelNr* zu einer neuen Tabelle mit den Attributen *ArtikelNr* und *AVG(Preis)* zusammengefasst.

(5) ORDER BY 2 DESC
Das Resultat des Schrittes (4) wird absteigend nach dem zweiten Feld, dem Durchschnittspreis je Gruppe, sortiert.

Übungsaufgabe 4.22

Das Attribut *ArtikelNr* in der Tabelle *Artikel* stellt ein globales Attribut mit statischem Wertebereich dar. In den restlichen Tabellen, in denen das Attribut *ArtikelNr* auftritt, ist der Wertebereich dynamisch. Das heißt beispielsweise, dass für das Attribut *ArtikelNr* der Tabelle *Stkliste,* nur solche Werte vorkommen dürfen, die auch das Attribut *ArtikelNr* in der Tabelle *Artikel* aufweist.

Der zulässige Wertebereich des Attributs *ArtikelNr* ist durch die Vaterrelation, in diesem Fall also durch die Tabelle *Artikel,* festgelegt.

Wird nun ein Wert für das Feld *ArtikelNr* gelöscht, so ist darauf zu achten, dass dieser Wert in keiner anderen Tabelle mehr vorkommt. Diese auch als referentielle Integrität bezeichnete Konsistenzbedingung verhindert beispielsweise, dass Stücklisten ohne Artikel in der Datenbank enthalten sind. Um die referentielle Integrität sicherzustellen, überprüft man vor dem Löschen einer *ArtikelNr* die Wertebereiche sämtlicher Sohnrelationen und löscht die *ArtikelNr* nur dann, falls sie in keiner der Sohnrelationen auftritt.

Übungsaufgabe 4.23

a)

```
SELECT ArtikelNr, SUM(FertAuftragPos.Menge)
FROM   KundenAuftrag, FeritAuftragPos
WHERE  KundenAuftrag.Name = "Meier"
AND    KundenAuftrag.AuftrDatum = 120401
AND    KundenAuftrag.ProjektNr = FertAuftragPos.ProjektNr
AND    FertAuftragPos.ArtikelNr >= 800000
AND    FertAuftragPos.ArtikelNr < 900000
GROUP BY ArtikelNr
```

Ergebnis:

ArtikelNr	Summe(FertAuftragPos.Menge)
810010	10

b)

```
SELECT ArtikelNr, Bezeichnung, Bestand
FROM   Artikel, StklisteNr, StrklistenPos
WHERE  Stkliste.ArtikelNr = 910030
AND    Stkliste.StklisteDatum = 120403
AND    Stkliste.StklisteNr = StklistenPos.StklisteNr
AND    StklistenPos.ArtikelNr >= 100000
AND    StklistenPos.ArtikelNr < 200000
AND    StklistenPos.ArtikelNr = Artikel.ArtikelNr
AND    Artikel.Bestand >= Artikel.MindBestand
ORDER BY  Artikel.Bestand DESC
```

Ergebnis:

ArtikelNr	Bezeichnung	Bestand
100200	Schrauben	150
100100	Gehäuseblech	50

Übungsaufgabe 4.24

Erzeugnispreise

ErzNr	Grundpreis	Kunde	AuftrMenge	Endpreis
910030	8000	Meier	5	9000
910030	8000	Müller	5	8900
920020	12000	Abele	10	13500