

# Datenbanken

<http://vfhdb.oncampus.de>

Stand 16.03.2017 07:52



## Inhalt

Datenbanken .....	5
1 Einführung .....	8
1.1 Geschichtliche Entwicklung .....	8
1.2 Grundbegriffe .....	11
1.3 Freitextaufgaben zu Einführung .....	12
2 DB-Systeme .....	13
2.1 Aufgaben eines DBMS .....	13
2.2 Die Drei-Ebenen-Schemaarchitektur .....	15
2.3 Komponenten eines DBMS .....	17
2.4 Freitextaufgaben zu DB-Systeme .....	19
3 DB-Entwurf .....	21
3.1 Entwurfsaufgabe .....	21
3.2 Phasenmodell .....	23
3.3 Freitextaufgaben zu DB-Entwurf .....	27
4 Datenmodelle .....	29
4.1 Hierarchisches und Netzwerkmodell .....	29
4.2 Relationenmodell .....	32
4.2.1 Operationen auf Relationen .....	34
4.3 ER-Modell .....	42
4.4 EER-Modell .....	47
4.5 CrowFoot-Notation .....	53
4.6 Weitere Modelle .....	54
4.7 Freitextaufgaben zu Datenmodelle .....	57
5 Relationale Datenbanken .....	58
5.1 Abbildung vom (E)ER- auf das Relationenmodell .....	58
5.2 Normalisierung .....	67
5.3 Relationenalgebra .....	76
5.4 Freitextaufgaben zu Relationale Datenbanken .....	84
6 Structured Query Language (SQL) .....	86
6.1 Datenbankdefinition .....	87
6.1.1 Erzeugen einer Datenbank .....	87
6.1.2 Relationen erstellen .....	88
6.1.3 Datentypen und Domänen .....	91
6.1.4 Erweiterte Form der CREATE TABLE-Anweisung .....	97
6.1.5 Primärschlüsseldefinition .....	97
6.1.6 Fremdschlüsseldefinition .....	100
6.1.7 Eindeutigkeitspezifikation .....	102

---

6.1.8 CHECK-Spezifikation .....	103
6.1.9 Indizes .....	104
6.1.10 Relationen ändern .....	106
6.1.11 Relationen löschen .....	108
6.1.12 COMMIT WORK/ROLLBACK WORK .....	110
6.2 Datenbankabfragen .....	110
6.2.1 Die SELECT-Anweisung .....	110
6.2.2 Die ORDER BY-Klausel .....	117
6.2.3 Die WHERE-Klausel .....	120
6.2.4 Die GROUP BY-Klausel .....	126
6.2.5 Die HAVING-Klausel .....	131
6.2.6 Joins .....	134
6.2.7 Unterabfragen .....	142
6.2.8 Kombination von unabhängigen Unterabfragen .....	152
6.3 Datenbankänderungen .....	154
6.3.1 Dateneingabe .....	154
6.3.2 Daten ändern .....	158
6.3.3 Daten löschen .....	160
6.4 Weiterführende Informationen .....	161
6.4.1 SQL-Anweisungsgruppen .....	162
6.4.2 SQL-Operatoren .....	163
6.4.3 WHERE-Bedingung .....	164
6.4.4 Outer Joins .....	166
6.5 Freitextaufgaben zu Structured Query Language .....	167
7 Sichten, Rechtevergabe, Integrität .....	169
7.1 Sichten .....	169
7.2 Rechtevergabe .....	174
7.3 Integrität .....	178
7.4 Syntax-Diagramme .....	185
7.4.1 Syntax der GRANT-Anweisung .....	185
7.4.2 CREATE ASSERTION-Syntax .....	186
7.5 Freitextaufgaben zu Sichten, Rechtevergabe, Integrität .....	187
8 Anwendungen mit Datenbanken .....	190
8.1 Grenzen von SQL .....	192
8.2 Call-Schnittstelle .....	194
8.3 Embedded SQL .....	196
8.3.1 Dynamic SQL .....	201
8.4 ODBC .....	202

---

---

8.5 JDBC .....	207
8.6 PHP-Anwendungen .....	213
8.7 PL/SQL .....	215
8.8 Sicherheit bei datenbankgestützten Webapplikationen - Die SQL-Injection .....	222
8.9 4GL-Systeme .....	229
8.10 Ergänzende Informationen .....	233
8.10.1 Beispiel zur Call-Schnittstelle .....	234
8.10.2 Relation für das Beispiel zu Embedded SQL .....	236
8.10.3 Syntax von EXEC SQL WHENEVER .....	237
8.10.4 Syntax von EXEC SQL Cursor .....	237
8.10.5 Beispiel: Umfangberechnung eines Polygons .....	237
8.10.6 Beispiel zu Dynamic SQL .....	240
8.10.7 Client-Erweiterungen .....	241
8.10.8 JDBC-Beispielprogramm .....	243
8.11 Freitextaufgaben zu Anwendungen mit Datenbanken .....	244
9 Transaktionsverwaltung und Wiederherstellung .....	248
9.1 Grundbegriffe der Transaktionsverwaltung .....	248
9.2 Sperrtechniken .....	253
9.2.1 Das 2PL-Protokoll .....	260
9.3 Transaktionsunterstützung in SQL .....	262
9.4 Wiederherstellung .....	263
9.5 Wiederherstellungsverfahren .....	266
9.6 ARIES-Algorithmus .....	270
9.7 Freitextaufgaben zu Transaktionsverwaltung und Wiederherstellung .....	272
10 Miniwelt Hochschule .....	275
10.1 Textuelle Beschreibung .....	275
10.2 Eine EER-Modellierung .....	276
10.3 Das Relationenmodell .....	277
10.4 Die CREATE TABLE-Anweisungen .....	283
10.5 Datensätze .....	287

## Anhang

I Literaturverzeichnis .....	294
II Abbildungsverzeichnis .....	295
III Tabellenverzeichnis .....	301
IV Medienverzeichnis .....	305
V Aufgabenverzeichnis .....	306
VI Index .....	308

---

## Datenbanken



Prof. Dr. habil. J. S. Lie, Ostfalia Hochschule für angewandte Wissenschaften Prof. Dr. H. Faasch, Leuphana Universität Lüneburg unter Mitwirkung von Dipl.-Inform. C. Tabara, Dipl.-Ing. (FH) D. Schierschlicht, S. Scholz, Dipl.W.-Inf. (FH) I. Lünig

### Rechtliche Hinweise

Die Inhalte dieses Studienmoduls (Text, Bild, Ton, Software usw.) sind urheberrechtlich geschützt und nur zum privaten, studienspezifischen Gebrauch durch den Nutzer/ die Nutzerin bestimmt, der/die eine Berechtigung zur Online-Nutzung dieses Moduls im Rahmen des Hochschulverbundes Virtuelle Fachhochschule und seiner Verbundhochschulen erhalten hat.

Die Nutzung darf gleichzeitig nur auf einem Personalcomputer erfolgen. Kopien/ Vervielfältigungen in jeglicher Form dürfen nur vorgenommen werden, falls und soweit dies ausdrücklich erlaubt worden bzw. ausnahmsweise gesetzlich zulässig ist. Die Bestandteile des Programms und der Inhalte dürfen nicht verändert werden. Weiterverbreitung und Nutzung zur öffentlichen Wiedergabe in jeglicher Form, auch in Teilen, ist unzulässig. Öffentliche Vorführungen sind nicht gestattet.

Das Herunterladen oder der sonstige Erhalt von Inhalten einschließlich Software über das Lernraumsystem erfolgt auf eigenes Risiko. Es wird keine Haftung übernommen für Schäden an dem Computersystem des Nutzers/der Nutzerin oder sonstigen zur Nutzung verwendeten technischen Geräten, für den Verlust von Daten oder für sonstige Schäden aufgrund des Herunterladens oder sonstiger Aktivitäten im Zusammenhang mit dem Lernraumsystem, es sei denn die Haftung beruhe auf einer vorsätzlichen oder grob fahrlässigen Pflichtverletzung.

Es wird keine Verantwortung für Inhalte von externen Links übernommen, die in Studienmodule aufgenommen worden sind.

In dem Studienmodul werden Marken, geschäftliche Bezeichnungen und dgl. verwendet. Auch wenn diese nicht als solche gekennzeichnet sind, unterliegen diese den entsprechenden gesetzlichen Schutzbestimmungen.

*Für Dateien, die heruntergeladen werden können, wie z.B. PDF-Dateien, offline-Version des Studienmoduls usw., gilt **zusätzlich**:*

Die Inhalte der Download-Datei (Text, Bild, Ton, Software usw.) sind urheberrechtlich geschützt und nur zum privaten, studienspezifischen Gebrauch durch den Nutzer/die Nutzerin bestimmt, der/die eine Berechtigung zur Online-Nutzung dieses Studienmoduls im Rahmen des Hochschulverbundes Virtuelle Fachhochschule und seiner Verbundhochschulen erhalten hat. Die Nutzung darf gleichzeitig nur auf einem Personalcomputer erfolgen. Kopien/Vervielfältigungen in jeglicher Form dürfen nur gemacht werden, falls und soweit dies ausdrücklich erlaubt worden bzw. ausnahmsweise gesetzlich zulässig ist. Die Bestandteile des Programms und der Inhalte dürfen nicht verändert werden. Weiterverbreitung und Nutzung zur öffentlichen Wiedergabe in jeglicher Form, auch in Teilen, ist unzulässig. Öffentliche Vorführungen sind nicht gestattet.

© J. S. Lie, H. Faasch, 2011



#### Gliederung

### Datenbanken

#### 1 Einführung

#### 2 DB-Systeme

#### 3 DB-Entwurf

#### 4 Datenmodelle

#### 5 Relationale Datenbanken

#### 6 Structured Query Language (SQL)

7 Sichten, Rechtevergabe, Integrität

8 Anwendungen mit Datenbanken

9 Transaktionsverwaltung und Wiederherstellung

10 Miniwelt Hochschule

## 1 Einführung



### Zeitumfang

Die voraussichtliche Bearbeitungsdauer dieser Lerneinheit (ohne Übungsaufgaben!) beträgt ca. 2 Stunden.



### Lernziele

Diese Lerneinheit gibt einen Überblick über die geschichtliche Entwicklung im Fachgebiet Datenbanken und die Definition der Grundbegriffe, die in diesem Studienmodul verwendet werden.



### Gliederung

#### 1 Einführung

##### 1.1 Geschichtliche Entwicklung

##### 1.2 Grundbegriffe

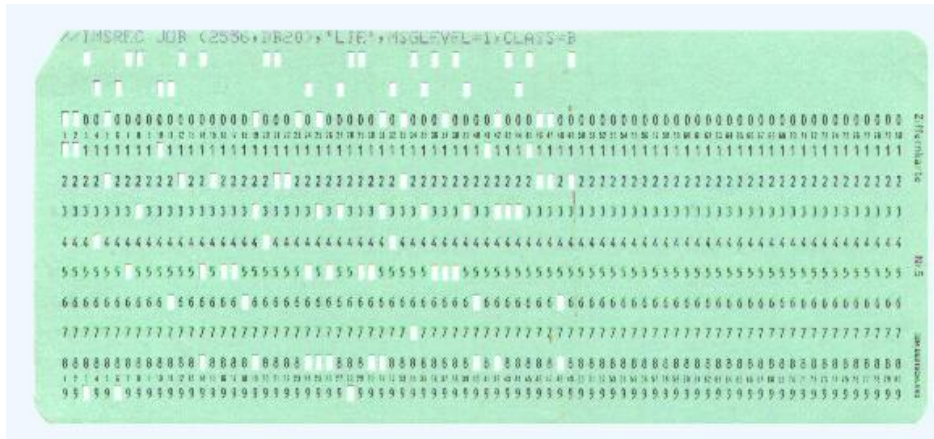
##### 1.3 Freitextaufgaben zu Einführung

## 1.1 Geschichtliche Entwicklung

Datenbanksysteme sind heute ein Hilfsmittel zur effizienten, rechnergestützten Organisation, Erzeugung, Manipulation und Verwaltung großer Datensammlungen. Ihre Entwicklungsgeschichte lässt sich in fünf Generationen gliedern, von denen die ersten beiden die Vorläufer von Datenbanksystemen umfassen.

Die Hauptaufgabe der EDV-Systeme in den fünfziger Jahren war die programmgesteuerte Bearbeitung von Daten. Das Speichermedium bestand aus einem Lochkartendeck oder aus Magnetbändern, welche nur eine sequenzielle Verarbeitungsform zuließen. Insbesondere erlaubten die ersten Dateisysteme ausschließlich **sequenziellen Zugriff** auf die Datensätze einer einzelnen Datei.





Lochkarte

Ein Jahrzehnt danach verfügten die Rechner über Magnetplatten als (zu der Zeit schnelle) Sekundärspeicher. Das wichtigste Merkmal der neu eingeführten Dateisysteme war die Möglichkeit des wahlfreien Zugriffs. Es wurde ermöglicht, mit Hilfe einer speziellen **Index-Datei** oder einer **Hash-Funktion**, auf einen bestimmten Datensatz direkt über seine Adresse zuzugreifen.



Festplatte der neueren Generation

Die Dateisysteme werden also als Vorläufer eines Datenbanksystems angesehen. Bei einem Dateisystem werden die Daten nur statisch zu den Programmen, von denen sie verarbeitet werden, zugeordnet. Soll ein Programm auf eine externe Datei zugreifen, so muss der Dateiaufbau dem Programm bekannt gemacht werden. Jedes Programm enthält damit eine eigene Beschreibung der Daten, die ausschließlich und direkt auf die Struktur einer durch das Programm bearbeitende Datei abgestimmt

ist. Unter Umständen muss jedem einzelnen Programm eine Kopie der benötigten Dateien für die entsprechende Bearbeitungsdauer zur Verfügung gestellt werden. Somit können einzelne Dateien redundant vorkommen. Diese **Redundanz** entsteht durch die Mehrfachspeicherung und kann nicht zentral kontrolliert werden. Dadurch, dass einzelne Programme die von ihnen benutzten Dateien verändern können, ohne dass andere Programme, die mit dem gleichen Datenbestand arbeiten, die Änderungen mitbekommen, entsteht auch eine **Inkonsistenz** der Daten. Die Datenbestände sind **inflexibel** gegenüber Veränderungen in der Anwendung, zum Beispiel falls sich im Laufe der Zeit neue Tätigkeiten oder Arbeitsvorgänge ergeben. Und zuletzt sind auch die **Datenschutzprobleme** nicht zu vernachlässigen, die dadurch entstehen können, dass der Zugriff auf die Dateien eines umfangreichen Dateisystems nicht angemessen kontrolliert werden kann.

In den siebziger Jahren bildeten die prärelationalen Systeme. Diese waren durch die Einführung einer ersten Unterscheidung zwischen logischen und physischen Informationen sowie durch eine zunehmende Verwaltung (im Gegensatz zur reinen Verarbeitung) der Daten gekennzeichnet. Zum ersten Mal wurden **Datenmodelle** zur logischen Beschreibung physikalischer Strukturen benutzt. Das hierarchische Datenmodell und das Netzwerk-Modell entstanden und wurden für die Implementierung der ersten "echten" Datenbanksystemen verwendet.

Seit Anfang der achtziger Jahre sind die **Relationale Datenbanksysteme** kommerziell verfügbar. Beispiele sind Adabas, Ingres, DB2 und Oracle in den früheren Versionen. Ihre Organisation der Daten erlaubt eine wesentlich deutlichere Unterscheidung zwischen einem physischen und einem logischen Datenmodell. Die relationalen Systeme besitzen einen hohen Grad an physischer **Datenunabhängigkeit**, basieren auf einem einfachen konzeptionellen Modell und stellen dem Benutzer **mächtige Sprachen** zur Verfügung.

Mittlerweile ging die Entwicklung weiter. Dabei werden Entwicklungen im Bereich der Programmiersprachen berücksichtigt, insbesondere die **Objektorientierung**. Als Zwischenschritt entstanden objektrelationale Datenbanksysteme (wie z.B. Postgres, Oracle ab Version 8), doch es gibt auch zunehmend mehr objektorientierte Datenbanksysteme (O2, ObjectStore, db4o).



**Welche Datenbanksysteme werden zur Zeit verbreitet eingesetzt?**

Die relationalen Datenbanksysteme haben eine sehr große Verbreitung gefunden, aber zunehmend werden auch immer mehr objektrelationale Datenbanksysteme eingesetzt. Der Trend geht, wie auch bei den Programmiersprachen, zu objektorientierten Datenbanksystemen, von denen es jedoch nur einige gibt.

Neben den relationalen Datenbanken hat sich eine Reihe von anders strukturierten Datenbanken zur Speicherung großer Datenmengen entwickelt, die als schemafreie Datenbanken oder auch NoSQL (Not Only SQL)-Datenbanken bezeichnet werden.

## 1.2 Grundbegriffe

Eine **Datenbank** (DB) ist ein integrierter, einheitlich strukturierter Datenbestand, während ein **Datenbank-Managementsystem** (DBMS), auch Datenbank-Verwaltungssystem (DBVS) genannt, ein Software-System zur Verwaltung der Datenbank ist.

Ein **Datenbanksystem** (DBS) besteht aus einer oder mehreren Datenbanken und einem Datenbank-Managementsystem.

Es gibt verschiedene Arten von Benutzungsschnittstellen. Dabei können die Endbenutzer im Dialog Anfragen und Änderungen formulieren, den Anwendungsprogrammierern wird der Zugriff von Programmen gestattet, und der Datenbank-Administrator verfügt über einen exklusiven Zugriff.

Ein **Datenmodell** ist ein System von Konzepten zur Strukturierung und Modellierung von Datenbeständen. Beispiele von Datenmodellen sind das hierarchische Datenmodell, das Netzwerk-Datenmodell und das relationale Datenmodell.

Eine **Datenbanksprache** ist ein Darstellungsmittel zur Beschreibung von Datenbeständen (DDL=Data Description Language), Datenmanipulationen (DML=Data Manipulation Language), Datenanfragen (DQL= Data Query Language) und Datenintegrität und Rechtevergabe (DCL=Data Control Language).

Ein **Datenbankschema** ist die Beschreibung der Strukturen eines zeitveränderlichen Datenbestandes mit Hilfe eines Modellierungsmittels (z.B. ER-Modell oder UML) und charakterisiert mögliche Zustände sowie mögliche Anfragen und Änderungen.

Die **Datenbankausprägung** bezeichnet den momentan gültigen Zustand des Datenbestandes. Dieser Zustand richtet sich nach der im Datenbankschema festgelegten Strukturbeschreibung.

Der **Datenbankadministrator**/Die **Datenbankadministratorin** hat als Hauptaufgaben die Benutzerverwaltung, die Organisation der Datensicherung, das Laden der Daten in die Datenbank, die Überwachung der Systemleistung und gegebenenfalls die Optimierung der Zugriffspfade.



Welche Begriffe soll ich mir nun merken?

Alle hier vorgestellten Begriffe sind für das spätere Verständnis der Inhalte relevant.

## 1.3 Freitextaufgaben zu Einführung

1. Was kennzeichnet die wesentlichen Verbesserungen durch die relationalen Datenbanksysteme?

### Lösung zeigen

Bei der zweiten Generation handelte es sich noch um Dateisysteme, während zur vierten Generation die relationalen Datenbanksysteme zählen. Die relationalen Systeme basieren auf einem einfachen konzeptionellen Modell und stellen mächtige Sprachen zur Verfügung.

2. Was ist ein Datenbankschema?

### Lösung zeigen

Ein Datenbankschema ist die Beschreibung der Strukturen eines zeitveränderlichen Datenbestandes mit Hilfe eines Modellierungsmittels (z.B. ER-Modell) und charakterisiert mögliche Zustände sowie mögliche Anfragen und Änderungen.

3. Welcher der vorgestellten Grundbegriffe ist eher der Software zuzuordnen?

### Lösung zeigen

Datenbank-Managementsystem, Datenbanksystem und Datenbanksprache.

## 2 DB-Systeme



### Zeitumfang

Die voraussichtliche Bearbeitungsdauer dieser Lerneinheit (ohne Übungsaufgaben!) beträgt ca. 2 Stunden.



### Lernziele

In dieser Lerneinheit erfahren Sie die wesentlichen Aufgaben eines Datenbankmanagementsystems. Die Drei-Ebenen-Schemaarchitektur gibt eine schichtenorientierte Sichtweise für ein Datenbanksystem, damit es plattform- und anwendungsunabhängig gestaltet werden kann. Anschließend werden die Komponenten eines Datenbankmanagementsystems (DBMS) behandelt.



### Gliederung

- 2 DB-Systeme
- 2.1 Aufgaben eines DBMS
- 2.2 Die Drei-Ebenen-Schemaarchitektur
- 2.3 Komponenten eines DBMS
- 2.4 Freitextaufgaben zu DB-Systeme

### 2.1 Aufgaben eines DBMS

Die folgenden neun Funktionen werden meistens als die typischen Charakteristika eines Datenbanksystems angesehen.

1. **Integration:** Alle Daten werden einheitlich verwaltet. Nur eine einheitliche logische Sichtweise aller Daten innerhalb der Datenbanken wird angeboten. Von einem relationalen Datenbanksystem müssen daher sämtliche Daten dem Benutzer in Tabellen- oder Relationenform dargestellt werden. Dies betrifft auch die sogenannten Katalogdaten (auch Metadaten oder Datenbeschreibungsdaten genannt).
2. **Operationen:** Die Operationen oder Datenbankanweisungen zur Datenspeicherung, -suche oder -änderung werden in einer einheitlichen Datenbanksprache angeboten. Die Operationen sind möglichst einfach in der Datenbanksprache formulierbar.

3. **Katalog:** Es wird eine integrierte Beschreibung der Daten angeboten. Diese Metadaten können mit den Mitteln der Datenbanksprache genauso gelesen werden wie die normalen Nutzdaten.
4. **Benutzersichten:** Es ist möglich, mit Datenbankanweisungen für verschiedene Benutzer oder Benutzerklassen die Daten verschiedenen Sichten zuzuordnen. Dadurch können die Benutzer die Daten in unterschiedlichen Detaillierungsstufen sehen. Die Definition von Sichten ist mit den Möglichkeiten des Datenschutzes stark verknüpft.
5. **Konsistenz-Erhaltung:** Die Datenbanksprache bietet Konstrukte an, mit denen korrekte Datenbankzustände beschrieben werden können. Das Datenbanksystem garantiert dann zur Laufzeit die Einhaltung der Korrektheit der Datenbank.
6. **Datenschutz:** Für verschiedene Benutzer oder Benutzerklassen kann festgelegt werden, auf welchen Originaldaten oder Sichten sie welche Operationen ausführen dürfen. Wenn Sichten, Benutzerklassen und Operationen mit der Datenbanksprache sehr detailliert festgelegt werden können, lassen sich unberechtigte Zugriffe auf die Daten gut ausschließen.
7. **Transaktionen:** Folgen von Anweisungen zur Änderung von Daten lassen sich zu atomaren Einheiten zusammenfassen. Das Datenbanksystem garantiert dann entweder die vollständige Ausführung dieser Anweisungsfolgen oder die Rücksetzung der Datenbank in den Zustand vor Beginn der Ausführung.
8. **Synchronisation:** Transaktionen von Benutzern, die gleichzeitig mit dem Datenbanksystem arbeiten, werden synchronisiert. Für jeden einzelnen Benutzer ergibt sich das logische Bild, als arbeite er allein mit dem Datenbanksystem.
9. **Datensicherung:** Die Daten werden nach Systemfehlern, etwa Plattenfehlern, wieder in einen konsistenten Zustand überführt. Dieses Rücksetzen der Datenbank sollte weitgehend automatisch geschehen.

Weitere wünschenswerte Eigenschaften eines Datenbanksystems, die allerdings schlecht objektivierbar sind, sind ein intuitiv zu bedienendes System mit einer klaren Strukturierung, klar aufgebaute und gut lesbare Handbücher, eine komfortable Programmiersprachen-Anbindung und eine möglichst flexible Datenimport- und Datenexport-Unterstützung.



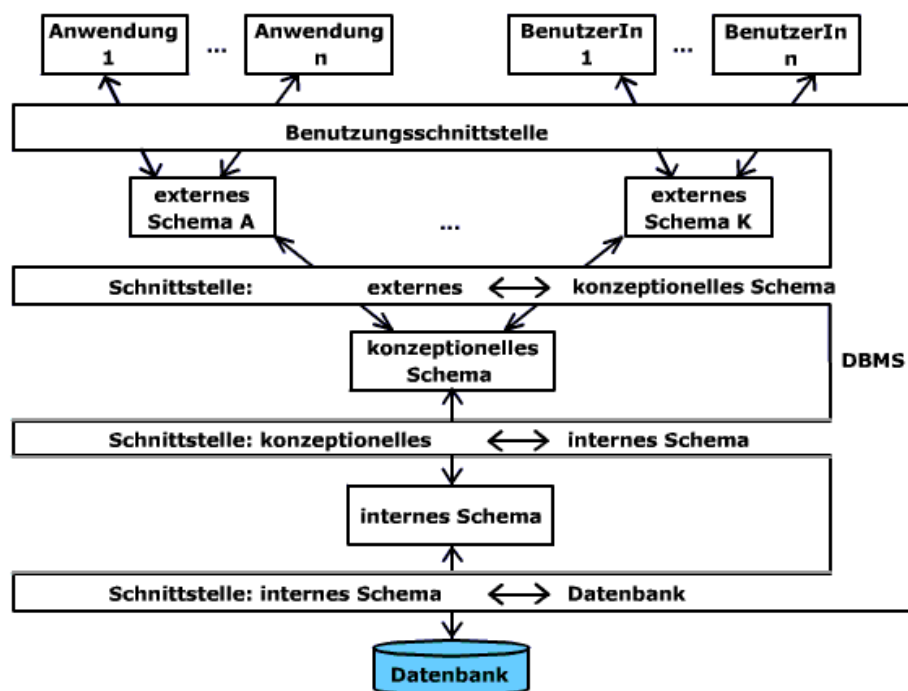
Lohnt es sich, ein Datenbankmanagementsystem selbst zu implementieren?

Nein, da es zu aufwändig ist. Es gibt ausreichend frei verfügbare Datenbankmanagementsysteme, falls man die Kosten für ein kommerzielles System scheut.

## 2.2 Die Drei-Ebenen-Schemaarchitektur

In jedem Datenmodell ist es wichtig, zwischen der Beschreibung einer Datenbank (**Metadaten**) und der Datenbank selbst (**Daten**) zu unterscheiden. Die Beschreibung wird im Datenbankschema während der Entwurfsphase festgelegt und wird als weitgehend zeitinvariant betrachtet, da nur selten Änderungen vorgenommen werden. Dagegen sind Änderungen am Inhalt einer Datenbank, also an den Daten selbst, im allgemeinen häufig.

Ein **Datenbank-Schema** ist die Beschreibung der möglichen Zustände und Übergänge. Die Bestandteile eines Datenbank-Schemas sind die Strukturbeschreibung (beinhaltet die Grundstruktur von Datenbankzuständen), die Integritätsbedingungen (Regeln für zulässige Datenbankzustände und -Änderungen), die Datenbank-Operationen (Basiszugriffe auf die Datenbankzustände) und die Autorisierungsregeln (Regeln für erlaubte Zugriffe).



Drei-Ebenen-Schemaarchitektur



In der Online-Version befindet sich an dieser Stelle eine Simulation.

#### Animierte Darstellung der Drei-Ebenen-Schemaarchitektur

Die Architektur basiert auf der Identifikation von drei unterschiedlichen Abstraktionsebenen und den ihnen zugeordneten Schemata und hat die logische und physische Datenunabhängigkeit als Ziel. Auf jeder der drei Ebenen können jeweils verschiedene Typen von Daten und Schemata identifiziert werden. Speziell wird unterschieden zwischen

- der physischen Datenorganisation
- der logischen Gesamtsicht der Daten und
- den einzelnen Benutzersichten, die sich im allgemeinen voneinander unterscheiden.

Auf der **konzeptionellen Ebene** wird die logische Gesamtsicht aller Daten in der Datenbank und ihrer Beziehungen untereinander im konzeptionellen Schema dargestellt. Zu diesem Zweck wird ein Datenmodell (z.B. das Entity-Relationship-Modell oder das Relationenmodell) eingesetzt. Damit die physische Datenunabhängigkeit gewahrt bleibt, muss das konzeptionelle Schema frei von Datenstruktur- oder Zugriffsaspekten sein.

Die **externe Ebene** umfasst alle individuellen Sichten der einzelnen Benutzer oder Benutzergruppen auf die Datenbank. Die Sichten werden jeweils einzeln in einem eigenen externen Schema beschrieben. Darin ist genau der Ausschnitt der konzeptionellen Gesamtsicht enthalten, den der Benutzer sehen möchte oder darf.

Die **interne Ebene** liegt dem physikalischen Speicher am nächsten, ist aber nicht mit diesem zu verwechseln. Denn die physisch gespeicherten Daten werden nicht als Seiten oder Blöcke sondern als Datensätze betrachtet. Im internen Schema sind Informationen über die Art und den Aufbau der verwendeten Datenstrukturen und spezieller Zugriffsmechanismen darauf oder über die Anordnung der Sätze im (logischen) Adressraum enthalten. Ferner wird die Lokalisierung der Daten auf den zur Verfügung stehenden Sekundärspeicher-Medien geregelt.



Kann ich bei einer Datenbankimplementierung auf irgendeine Ebene verzichten?



Wenn die Datenbank gut strukturiert ist, kann man auf die externe Ebene verzichten, da man in dem Fall für den Zugriff auf die Daten auch ohne Sichten auskommen kann.

## 2.3 Komponenten eines DBMS

Die besonders relevanten Komponenten eines Datenbankmanagementsystems sind der Eingabe-/Ausgabe-Prozessor, der Parser, der Vorübersetzer, die Autorisierungskontrolle, die Integritätsprüfung, der Update-Prozessor, der Query-Prozessor, der Optimierer, das Zugriffs- bzw. Ausführungsprogramm, der Transaktions-Manager, der Recovery-Manager, das Logbuch und die Speicherverwaltung.

Der **Ein-/Ausgabe-Prozessor** ist der Benutzerin/dem Benutzer unmittelbar zugeordnet und nimmt Kommandos entgegen, um entweder Erfolgs- oder Fehlermeldungen zurückzugeben. Der Parser führt eine syntaktische Analyse des an die Datenbank gerichteten Auftrags durch. Dabei ist z.B. die Korrektheit der Schlüsselwörter der verwendeten Sprache (z.B. SELECT aber nicht SLECT) oder der korrekte Aufbau des Kommandos zu überprüfen.

Bei eingebetteten Kommandos kann der Aufruf eines **Vorübersetzers** notwendig sein. Dieser ist programmiersprachenabhängig (z.B. für C) und ist für die Analyse des Quellprogramms sowie die Ersetzung aller Datenbank-Kommandos durch Unterprogrammaufrufe zuständig.

Bei der **Autorisierungskontrolle** wird festgestellt, ob eine Person mit den angesprochenen Daten arbeiten darf.

Nach diesen Schritten liegt der ursprüngliche Auftrag in einer internen Form vor (bei relationalen Anfragen z.B. ein Baum, mit den Operanden als Blätter und den auszuführenden Operationen als innere Knoten des Baums). Für die weitere Verarbeitung der Zwischenform wird davon ausgegangen, dass das DBMS sich unterschiedlich verhält, je nachdem, ob es sich bei dem Auftrag um eine Anfrage oder ein Datenbank-Update handelt.

Die **Integritätsprüfung** erfolgt mit Hilfe von Integritätsbedingungen. Diese überwachen im Falle eines Updates die semantische Korrektheit (Konsistenz) der Datenbank, wie z.B. "Die Matrikelnummer eines Studierenden muss eindeutig sein." Der **Update-Prozessor** erweitert dabei die interne Zwischenform des Auftrags.

Bei Anfragen kann die Integritätsprüfung entfallen. Dafür ist es bei Anfragen über einem externen Schema nötig, dass der **Query-Prozessor** diese in das konzeptionelle Schema übersetzt bzw. Abkürzungen in den externen Schemata durch ihre Definition ersetzt.

Wurde die Anfrage zu kompliziert formuliert, sind manche Systeme in der Lage, dies in einem gewissen Umfang zu erkennen. In diesem Fall kann die Anfrage an einen **Optimierer** übergeben werden, der die Zwischenform so verändert, dass sie einer effizienter ausführbaren Formulierung entspricht, ohne das Ergebnis zu verändern.

Als nächste Aufgabe muss das DBMS feststellen, welche Zugriffsstrukturen auf die Daten verfügbar sind (z.B. Indizes), einen möglichst effizienten Zugriffsplan auswählen und ein **Zugriffs- bzw. Ausführungsprogramm** generieren. Das heißt, dass für den Auftrag Code generiert werden muss. Dabei werden auch Optimierungsschritte in Abhängigkeit von den internen Datenstrukturen und deren Zugriffspfade durchgeführt.

Eine **Transaktion** ist das Programm (bzw. die Folge von Lese-/Schreibbefehlen), welches aus einem einzelnen Auftrag gemäß dem oben beschriebenen Ablauf erzeugt wird. Im Allgemeinen steht eine Datenbank mehreren Personen zur Verfügung. In diesem Fall entsteht das Problem, dass mehrere Transaktionen (quasi-) parallel ablaufen und deswegen synchronisiert werden müssen.

Der **Transaktions-Manager**, der im Datenbankmanagementsystem enthalten ist, lässt die Datenbank nach außen für jeden einzelnen Benutzer wie ein exklusiv verfügbares Betriebsmittel erscheinen, die einzelnen Transaktionen werden intern jedoch zeitlich überlappt abgearbeitet. Für den Benutzer arbeitet der Transaktions-Manager nach dem "Alles oder nichts"-Prinzip, d.h. eine Transaktion wird immer entweder vollständig oder gar nicht ausgeführt. Stellt der Transaktions-Manager während der Ausführung einer Transaktion fest, dass diese nicht erfolgreich zu Ende gebracht werden kann, übergibt er sie dem sogenannten Recovery-Manager.

Der **Recovery-Manager** hat als Aufgabe, die Datenbank in den Zustand zurückzusetzen, in dem sie sich vor Beginn der Transaktion befand. Insbesondere müssen auch alle Änderungen, die von der Transaktion in der Datenbank bereits vorgenommen wurden, rückgängig gemacht werden. Dafür wird das Logbuch der Datenbank verwendet. Der Recovery-Manager wird auch dann aktiv, wenn das System einen Hard- oder Software-Fehler macht, z.B. bei Schreib-/Lese Fehlern auf der Festplatte oder beim Absturz des Betriebssystems.

Im **Logbuch** wird eine Historie aller Änderungen in der Datenbank und der Statusänderungen jeder Transaktion geschrieben, um ein Wiederherstellen eines konsistenten Datenbankzustandes zu ermöglichen. Das Logbuch sollte sich außerhalb des Datenbankmanagementsystems befinden und entsprechend gesichert werden. Die Protokollierung kann entweder physisch oder logisch stattfinden. Bei der logischen Protokollierung werden jeweils die Operationen angegeben, während bei der physischen Protokollierung Abbilder der Datenobjekte gespeichert werden.

Die **Speicherverwaltung** umfasst den Puffer-Manager sowie den Geräte- und Sekundärspeicher-Manager (der auch als Data-Manager bezeichnet wird). Der Puffer-Manager dient der Verwaltung der Hardware-Betriebsmittel, die dem Datenbankmanagementsystem zur Verfügung stehen. Der Data-Manager führt unter der Kontrolle des Transaktions-Managers alle physischen Zugriffe auf die Datenbank aus.



Welche sind die Kernkomponenten eines Datenbanksystems?

Die Kernkomponenten eines Datenbanksystems sind: der Eingabe-/Ausgabe-Prozessor, der Parser, der Vorübersetzer, die Autorisierungskontrolle, die Integritätsprüfung, der Update-Prozessor, der Query-Prozessor, der Optimierer, das Zugriffs- bzw. Ausführungsprogramm, der Transaktions-Manager, der Recovery-Manager, das Logbuch und die Speicherverwaltung.

## 2.4 Freitextaufgaben zu DB-Systeme

1. Welchen Sinn hat die Drei-Ebenen-Schemaarchitektur?

### Lösung zeigen

Sie ermöglicht die Trennung der logischen und physischen Daten. Durch die Abstraktionsebenen ist Datenunabhängigkeit möglich.

2. Warum sind Transaktionen wesentlich beim Einsatz von Datenbanksystemen?

### Lösung zeigen

Transaktionen stellen sicher, dass die Inhalte der Datenbank sich in einem konsistenten Zustand befinden oder in einen solchen überführt werden können.

3. Was ist der wesentliche Unterschied zwischen Datenschutz und Datensicherung?

### Lösung zeigen

Der Datenschutz schließt unberechtigte Zugriffe auf den Datenbestand aus, die Datensicherung stellt sicher, dass im Falle eines Systemfehlers die Datenbank wieder in einen konsistenten Zustand überführt werden kann.

4. Wann ist es sinnvoll, ein Datenbanksystem einzusetzen?

**Lösung zeigen**

Der Einsatz eines Datenbanksystems sollte erwogen werden, falls bereits umfangreiche Datenmengen von mehreren Anwendungsprogrammen bearbeitet werden, eine neue umfangreiche Datensammlung geplant wird, abzusehen ist, dass zu einer kleineren Datensammlung immer weitere Dateien dazukommen, auf ein neues Betriebssystem gewechselt wird und man sich sowieso in eine neue Umgebung einarbeiten muss, und falls eine Fachkraft vorhanden ist, die das Datenbanksystem betreuen wird.

## 3 DB-Entwurf



### Zeitumfang

Die voraussichtliche Bearbeitungsdauer dieser Lerneinheit (ohne Übungsaufgaben!) beträgt ca. 2 Stunden.



### Lernziele

Eine Hauptaufgabe einer Informatikerin oder eines Informatikers kann sein, die benötigten Informationen in einer Datenbank vollständig und umfassend zu analysieren und ein zukunftsorientiertes Datenbankschema zu entwerfen. In dieser Lerneinheit werden die Schritte der Vorgehensweise der Entwurfsaufgabe beschrieben und die erwarteten Zwischenergebnisse dieser Vorgehensweise dargelegt. Das in der Softwareentwicklung häufig eingesetzte Phasenmodell findet hier seine Anwendung.



### Gliederung

- 3 DB-Entwurf
- 3.1 Entwurfsaufgabe
- 3.2 Phasenmodell
- 3.3 Freitextaufgaben zu DB-Entwurf

## 3.1 Entwurfsaufgabe

Ein **Informationssystem** beinhaltet die Anwendungssoftware, das Datenbanksystem (bestehend aus dem Datenbankmanagementsystem und den Daten), Hardware, Personal usw. Damit ist das Datenbanksystem der zentrale Kern eines Informationssystems.



Wenn ein Datenbankschema für ein Fachproblem entworfen werden soll, müssen die Daten strukturiert werden und die möglichen Anwendungen beschrieben werden.

Dabei gibt es ein paar Randbedingungen, die beachtet werden müssen:

- Der Informationsbedarf, der die aktuell beabsichtigten, aber auch die künftig zu erwartenden Benutzungen (Geschäftsprozesse) umfasst, muss geklärt werden. Es sollen nur tatsächlich benötigte Daten gespeichert werden und die erforderlichen Funktionen implementiert werden. Ebenfalls die zu erwartende Antwortzeit, die Anzahl der Benutzer sowie die Aspekte des Datenschutzes und der Datensicherheit müssen berücksichtigt werden. Also handelt es sich hier um die anwendungsbezogenen Aspekte.
- Das eventuell verfügbare Datenbankmanagementsystem muss auf seine Einsatzmöglichkeit hin überprüft werden bzw., falls noch kein Datenbankmanagementsystem vorhanden ist, müssen die Anforderungen an ein solches System aufgestellt werden, unter Berücksichtigung der Hardwarekonfiguration, des Betriebssystems und der Programmiersprachen, die zum Einsatz kommen sollen. Hier geht es um die soft- bzw. hardwarebezogenen Seiten.

Die Aufgabe des Datenbankentwurfs verfolgt mehrere Ziele:

- Der Informationsbedarf muss erfüllt werden.
- Die Betriebssicherheit des Datenbanksystems muss gewährleistet sein. Das heißt, es muss auf die Integrität der Daten geachtet werden, der Datenschutz muss durch entsprechende Maßnahmen sichergestellt werden, bei einem Mehrbenutzersystem muss die Synchronisation erfolgen, und für den Betrieb muss eine Datensicherung erfolgen, um im Falle eines Ausfalls des Systems oder eines Verlustes von Daten die Wiederherstellung des Datenbestandes zu ermöglichen.
- Das Datenbanksystem soll effizient betrieben werden. Das schließt zum Beispiel ein: geringe Antwortzeiten (durch geeigneten Entwurf), minimalen Speicherbedarf (durch Vermeidung von redundanter Datenhaltung), hohe Verfügbarkeit (durch geeignete Auswahl der Soft- und Hardware) usw. Die Anwendungsdaten jeder Anwendung sollen aus den in der Datenbank gespeicherten Daten abgeleitet werden können. Anwendungsdaten stellen den Teil der Gesamtinformationen dar, die für eine bestimmte Anwendung benötigt werden. Sie sollen, soweit möglich, nicht-redundant dargestellt werden, um Speicherplatz zu sparen und Anomalien zu verhindern.
- Das Datenbanksystem soll flexibel und bei neuen Anwendungsfunktionen leicht anpassbar sein.

Der **Entwurfsprozess** wird als eine Folge von Entwurfsdokumenten beschrieben, die von einer abstrakten, anwendungsnahen Beschreibungsebene hin zur tatsächlichen Realisierung der Datenbank führen. Konkrete Entwurfsschritte bilden ein Entwurfsdokument auf ein anderes ab, wobei der Beschreibungsformalismus

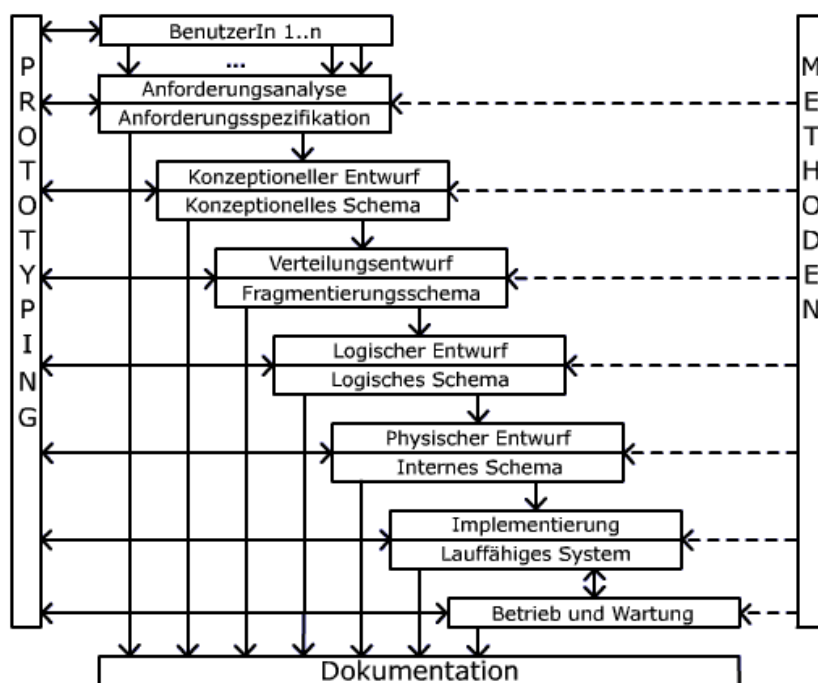
beibehalten werden oder auch wechseln kann (zum Beispiel vom ER-Modell zum Relationenmodell). Die Entwurfsschritte können sowohl manuell als auch automatisiert ablaufen. Primär sollen die Entwurfsschritte von der formalen Seite des Entwurfsprozesses her über zwei Eigenschaften verfügen:

- Informationserhalt heißt, dass der transformierten Datenbankbeschreibung folgend alle Informationen gespeichert werden können, die bei der ursprünglichen Modellierung möglich waren.
- Konsistenzerhaltung bedeutet, dass Regeln und Einschränkungen, die im Eingabedokument gewährleistet wurden, auch in der neuen Modellierung respektiert werden.

Eine mögliche Vorgehensweise für den Datenbankentwurf wird mit dem Phasenmodell beschrieben.

### 3.2 Phasenmodell

Der Realisierung einer Datenbankanwendung geht eine Kosten-/Nutzen-Analyse voraus, in der geklärt werden muss, ob die Realisierung des Systems einen betriebswirtschaftlichen oder sonstigen Nutzen hat, der den Aufwand der Implementierung rechtfertigen kann. Der Datenbank-Lebenszyklus hat mehrere Phasen:



Phasenmodell (Datenbank-Lebenszyklus)



In der Online-Version befindet sich an dieser Stelle eine Simulation.

#### **Animation des Phasenmodells (Datenbank-Lebenszyklus)**

Die Anforderungsanalyse, auch die Ist-Analyse genannt, bildet zusammen mit dem konzeptionellen, Verteilungs-, logischen und physischen Entwurf den Datenbankentwurf.

In der ersten Phase, **Anforderungsanalyse und -spezifikation**, geht es darum, dass alle Anforderungen der potentiellen DatenbankbenutzerIn an die einzurichtende Datenbank erhoben werden.

*Vorgehensweise:* Der Informationsbedarf wird durch Befragung der BenutzerInnen ermittelt. Vor der Befragung sollen sämtliche Dokumentationen gesichtet werden, um sich in das Anwendungsgebiet einzuarbeiten. Außerdem muss festgestellt werden, worüber Daten gespeichert werden sollen, welche Daten das sind und was der Zweck der Speicherung dieser Daten ist. Zu klären ist auch, wie die Daten verarbeitet werden sollen. Dabei müssen die zu erwartenden Anfragen- und Änderungshäufigkeiten abgeschätzt werden, die Zugriffsrechte, d.h. Lese- und Schreibrechte auf Daten, müssen Personen bzw. Personengruppen zugeordnet werden, Datenschutzaspekte müssen berücksichtigt werden. Zu guter Letzt muss die Konsistenz und die Integrität der Datenbank gewährleistet werden, unter anderem durch verschiedene Bedingungen, wie:

- Wertebereiche

*Beispiel:* Das Attribut Alter darf keine negative Zahl enthalten und muss kleiner als 150 sein.

- Plausibilitäten

*Beispiel:* Das Geburtsjahr bei der Immatrikulation darf nicht größer sein als "das aktuelle Jahr"-18 (Studierende müssen volljährig sein).

- Abhängigkeiten

*Beispiel:* Die Steuerklasse hängt vom Familienstand ab, d.h. eine verheiratete Person darf nicht Steuerklasse I haben.

*Ergebnis:* Eine informale Beschreibung der Problemstellung in Form von Texten, tabellarischen Aufstellungen, Diagrammen, Formblättern usw.



Die zweite Phase, **konzeptioneller Entwurf**, stellt eine erste formale Beschreibung der Problemstellung dar. Als Sprachmittel wird ein semantisches Datenmodell, wie zum Beispiel das erweiterte Entity-Relationship-Modell (EER-Modell), verwendet.

*Vorgehensweise:* Die Anfrageanalyse wird zunächst auf hohem implementierungsunabhängigen Niveau formuliert. Es werden verschiedene Sichten modelliert, wie zum Beispiel spezielle Sichten für verschiedene Benutzergruppen (Immatrikulationsamt, Bibliothek, Prüfungsamt usw.). Die modellierten Sichten werden anschließend in Bezug auf mögliche Konflikte analysiert. Arten von Konflikten sind:

- **Namenskonflikte** treten auf, wenn verschiedene Begriffe für dasselbe Konzept (Synonyme) oder wenn derselbe Begriff für mehrere Konzepte (Homonyme) der modellierten Anwendung benutzt werden. Synonyme wären zum Beispiel HochschullehrerIn und ProfessorIn, während Homonyme unter anderem durch Fachsprachen entstehen können. Zum Beispiel hat der Begriff "Schlüssel" im Datenbankbereich und in der Umgangssprache verschiedene Bedeutungen.
- **Typkonflikte** gibt es, wenn verschiedene Strukturen für dasselbe Element modelliert werden. Die Bibliothek benötigt beispielsweise andere Attribute der Studierenden als das Prüfungsamt.
- **Wertebereichskonflikte** entstehen, wenn gleiche Attribute in verschiedenen Sichten unterschiedliche Wertebereiche haben. Telefonnummern können zum Beispiel als Zahlen oder Zeichenketten abgespeichert werden.
- **Bedingungskonflikte** treten auf, wenn in verschiedenen Sichten unterschiedliche Integritätsbedingungen angegeben werden, wie beispielsweise verschiedene Schlüssel für ein Element (Identifikation der Studierenden über Matrikelnr. *oder* Name und Geburtsdatum).
- **Strukturkonflikte** entstehen dadurch, dass der gleiche Sachverhalt durch unterschiedliche Datenmodellkonstrukte ausgedrückt wird. So kann zum Beispiel die Partitionierung von *Person* in *Mann* und *Frau* auch durch das Attribut *Geschlecht* vom Aufzählungsdatentyp {m,w} dargestellt werden.

Zum Schluss müssen die Sichten in ein Gesamtschema integriert werden, wobei die im vorigen Schritt erkannten Konflikte aufgelöst werden müssen, um ein konsistentes Gesamtschema zu erhalten.

*Ergebnis:* Ein konzeptionelles Gesamtschema, wie zum Beispiel ein EER-Diagramm.

Die dritte Phase, der **Verteilungsentwurf**, hat als Ziel die Verteilung von Daten und Programmen, die auf die Daten zugreifen. Dies wird nur dann benötigt, wenn die Daten auf mehreren Rechnern verteilt vorliegen sollen.

Eine mögliche Methode ist, verschiedene Objekttypen des konzeptionellen Schemas auf unterschiedliche Knoten zu verteilen, wobei das Problem auftritt, wie mit Beziehungen zwischen Objekten auf verschiedenen Knoten zu verfahren ist. Eine andere Möglichkeit ist, einzelne Objekttypen verteilt zu speichern, wobei diese Verteilung wiederum auf zwei Arten erfolgen kann. Entweder werden die zugehörigen Objekte eines Objekttyps verteilt oder die einzelnen Objekte werden selbst verteilt realisiert. Im Relationenmodell werden diese beiden Aufteilungen als *horizontale* bzw. *vertikale Fragmentierung* bezeichnet. Die **horizontale Verteilung** ist die Aufteilung der Daten einer Relation auf mehrere Relationen, wobei die Tupelstruktur erhalten bleibt. Sie ist dann sinnvoll, wenn der Datenbestand in Gruppen zusammengefasst werden kann, z.B. eine Aufteilung nach Anfangsbuchstaben der Nachnamen der Studierenden, wenn die Sachbearbeiter im Immatrikulationsamt nur bestimmte Anfangsbuchstaben bearbeiten. Die **vertikale Verteilung** beschreibt die "senkrechte" Aufteilung einer Relation in mehrere Einzelrelationen. Sie kommt dann in Frage, wenn z.B. Teile einer Relation viel Speicherplatz benötigen, aber eher selten abgefragt werden, oder wenn an einzelnen Standorten bevorzugt jeweils bestimmte Attribute der globalen Relation benötigt werden. Die Personalabteilung benötigt alle Daten der Lehrenden, während z.B. für die Bibliothek das Fach der Lehrenden irrelevant ist. Der Verteilungsentwurf kommt beim Entwurf verteilter Datenbanken zum Einsatz.

Die vierte Phase, der **logische Entwurf**, benutzt als *Sprachmittel* Datenmodelle des ausgewählten Datenbankmanagementsystems, wie zum Beispiel das Relationenmodell.

*Vorgehensweise:* Im ersten Schritt wird das konzeptionelle Schema in das Modell der Zieldatenbank transformiert, etwa vom Entity-Relationship-Modell (ER-Modell) oder EER-Modell in das relationale Modell. Im zweiten Schritt soll das Relationenschema anhand von Gütekriterien verbessert werden, zum Beispiel durch Minimierung redundanter Speicherung. Im Relationenmodell ist dieser Schritt als Normalisierung bekannt.

*Ergebnis:* Das logische Schema, wie zum Beispiel eine Sammlung von Relationen.

In der fünften Phase, dem **physischen Entwurf**, wird die interne Ebene definiert. Es wird festgelegt, welche Attribute in welchen Relationen mit besonderen Zugriffsmechanismen ausgestattet werden sollen. Abhängig vom eingesetzten Datenbanksystem kann eventuell auch angegeben werden, welche Datenstrukturen für die Zugriffspfade benutzt werden sollen, wie zum Beispiel B-Bäume oder Hash-Verfahren. Als *Sprachmittel* werden die Data Definition Language (DDL), die Data Manipulation Language (DML), die Data Query Language (DQL) und die Data Control Language (DCL) eines Datenbankmanagementsystems eingesetzt.

*Vorgehensweise:* Im Relationenmodell werden die Relationen und mögliche Sichten definiert, wobei Attribute, auf die vermutlich oft zugegriffen wird, beispielsweise mit Indizes versehen werden.

*Ergebnis:* Das sogenannte physische (oder auch interne) Schema.

In der sechsten Phase, **Implementierung**, wird die Datenbank eingerichtet, die Sichten werden implementiert und Zugriffsrechte werden erteilt. Außerdem werden die Indizes, die in der Phase des physischen Entwurfs festgelegt wurden, in dieser Phase implementiert. Die Stammdaten werden hierbei erfasst.

*Ergebnis:* Das lauffähige System.

Die siebte und letzte Phase, **Betrieb und Wartung**, besteht aus dem Produktionsbetrieb und der Wartung. Falls es sich im Produktionsbetrieb als erforderlich herausstellen sollte, müssen in dieser Phase gegebenenfalls auch Anpassungen des Systems vorgenommen werden.

Begleitend in allen Entwurfphasen sollten Validierungsmethoden eingesetzt werden, um den aktuellen Entwurfschritt überprüfen zu können. Hierfür kann auf die bewährten Methoden des Software Engineerings zur Qualitätssicherung im Entwurfsprozess zurückgegriffen werden:

- **Verifikation:** Der formale Beweis (etwa von Schemaeigenschaften) kann in vielen Entwurfsschritten eingesetzt werden, da die verwendeten Datenmodelle auf einer klaren mathematischen Semantik aufbauen.
- **Prototyping:** In frühen Phasen ermöglicht das Prototyping beispielhaftes Arbeiten mit der Datenbank vor der endgültigen Implementierung.
- **Validation mit Testdaten:** Im Rahmen des Prototyping kann eine Überprüfung der Richtigkeit des Entwurfs anhand von realen oder künstlichen Testdaten erfolgen. In frühen Phasen können Testdaten per Hand erstellt werden, während mit fortschreitendem Entwurf dafür Werkzeuge eingesetzt werden können.



Muss der Entwurfsprozess immer so durchgeführt werden?

Ja, da auf diese Weise ein vollständiger und umfassender Datenbankentwurf möglich ist.

### 3.3 Freitextaufgaben zu DB-Entwurf

1. In welcher Phase des Datenbankentwurfs wird das EER-Modell eingesetzt?

**Lösung zeigen**

Das EER-Modell wird beim konzeptionellen Entwurf eingesetzt.

2. Was für Konfliktarten können beim konzeptionellen Entwurf auftreten?

**Lösung zeigen**

Namenskonflikte, Typkonflikte, Wertebereichskonflikte, Bedingungskonflikte, Strukturkonflikte

3. Welche Verteilungsarten gibt es?

**Lösung zeigen**

Horizontale Verteilung und vertikale Verteilung.

## 4 Datenmodelle



### Zeitumfang

Die voraussichtliche Bearbeitungsdauer dieser Lerneinheit (ohne Übungsaufgaben!) beträgt ca. 6 Stunden.



### Lernziele

In der Entwicklung des Fachgebietes Datenbanken wurden mehrere Modelle eingesetzt. Als klassische Modelle gelten das hierarchische, das Netzwerkmodell und das Relationenmodell. Das Relationenmodell ist bis heute noch sehr weit verbreitet. Deshalb wird es ausführlich behandelt und bildet den Kern dieses Studienmoduls. Das Entity-Relationship-Modell (ER-Modell) bietet eine Möglichkeit, den zu modellierenden Weltausschnitt (auch Miniwelt genannt) zu abstrahieren. Mit Hilfe der Erweiterungen des ER-Modells (EER-Modells) kann die Modellierungsarbeit erleichtert werden, um die Vielfalt unserer Datenwelt abbilden zu können. Weitere Modelle werden kurz vorgestellt. Das objektorientierte Modell wird im zweiten Datenbanken-Studienmodul ausführlich behandelt.



### Gliederung

- 4 Datenmodelle
  - 4.1 Hierarchisches und Netzwerkmodell
  - 4.2 Relationenmodell
  - 4.3 ER-Modell
  - 4.4 EER-Modell
  - 4.5 CrowFoot-Notation
  - 4.6 Weitere Modelle
  - 4.7 Freitextaufgaben zu Datenmodelle

### 4.1 Hierarchisches und Netzwerkmodell

Die Konstruktionsgrundlage für das **hierarchische Modell** ist die Baumstruktur. Das hierarchische Modell baut auf Datensätzen (Record-Typen) auf, die zu Satzklassen zusammengefasst werden. Die Satzexemplare einer Klasse haben gleich benannte Felder mit jeweils gleichen Datentypen. Zwischen den Satzklassen können hierarchische

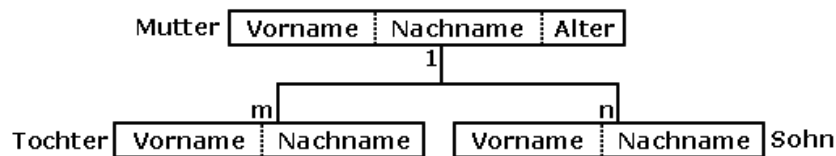
Beziehungen bestehen. Ein Record-Typ entspricht einem Knoten, während die 1:n-Beziehungen zwischen den Record-Typen den Kanten entsprechen.

Die übergeordnete Klasse ist die Elternklasse ("Mutter" im nachfolgenden Beispiel), während die untergeordnete Klasse Kinderklasse ("Tochter" bzw. "Sohn" im nachfolgenden Beispiel) genannt wird. Ein Satz aus der Elternklasse kann mit mehreren Sätzen aus der Kinderklasse in Beziehung stehen, wobei diese Beziehung nicht benannt werden kann.

Das Datenbankschema besteht daher aus einer oder mehreren Hierarchien von Satzklassen. Die Hierarchien werden immer von oben nach unten dargestellt, somit sind die Eltern- und die Kinderrolle von Satzklassen direkt erkennbar.



Beispiel



Beispiel zum hierarchischen Modell

Dabei entstehen beim hierarchischen Modell allerdings zwei Probleme:

1. *Es gibt keine Möglichkeit, n:m-Beziehungen darzustellen.*

Beispiel: Eine Mutter kann beliebig viele Kinder haben. Jedes Kind kann wiederum eigene Kinder haben. Dass ein Kind auch einen Vater hat, lässt sich jedoch in der gleichen Hierarchie nicht darstellen.

*Ausweg:* virtuelle Knoten. Dabei werden Record-Typen virtuell dupliziert, indem von den virtuellen Knoten Verweise auf tatsächlich vorhandene Sätze realisiert werden. Durch diese Art der Modellierung entsteht jedoch ein Netzwerkmodell, da außer Hierarchien auch Netze darstellbar sind.

2. *Zugriffe sind nur gemäß der Hierarchie möglich.*

Die wichtigste Operation im hierarchischen Modell ist somit der Durchlauf durch einen Baum, und zwar entweder von oben nach unten (von Eltern zu Kinder) oder von links nach rechts (bei den Kinderklassen).

Das **Netzwerkmodell** wurde 1971 von dem Normungsausschuss CODASYL/DBTG (Conference on Data Systems Languages & Data Base Task Group) festgelegt und ist eine Weiterentwicklung des hierarchischen Modells.

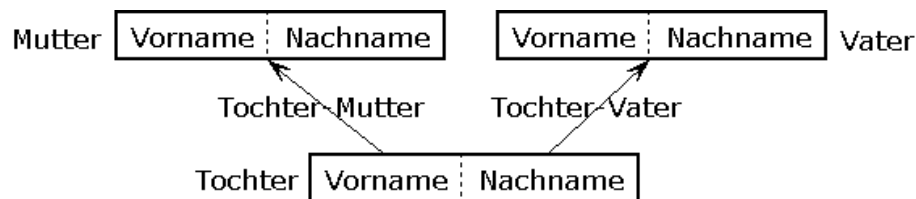
Als Objekte werden Sätze angeboten, wie im hierarchischen Modell auch. Zwischen zwei Satzklassen kann es mehrere 1:n-Beziehungen geben. Da die Beziehungstypen (auch "Set" genannt) im Netzwerkmodell benannt werden, ist es möglich, zwischen zwei Satzklassen auch mehrere Beziehungen anzugeben.

Bezüglich der betrachteten Beziehung wird die Ausgangs-Objektklasse "Owner" und die Ziel-Objektklasse "Member" genannt. In der Regel führt die Pfeilrichtung von Setklassen von der Owner- zur Memberklasse, manchmal wird jedoch die umgekehrte Richtung benutzt.

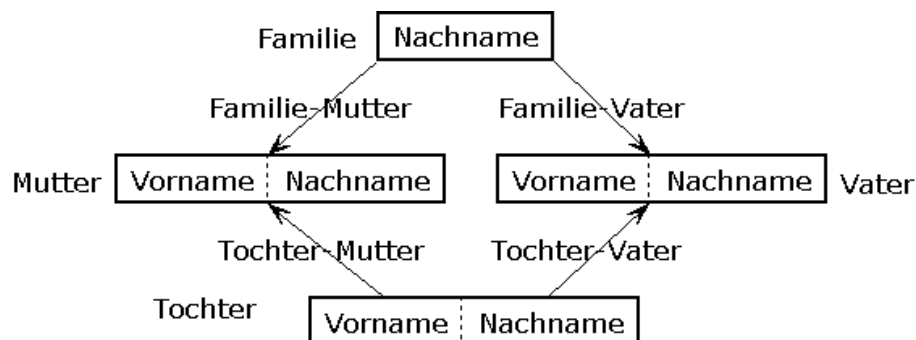
Die Ausprägungen von Netzwerkschemata kann man sich als Satzexemplare vorstellen, die gemäß der Setklassen ringförmig verkettet sind. Dabei führt die Verkettung vom jeweiligen Ownersatz über seine Membersätze zurück zum Ownersatz.



Beispiel



Beispiel für ein einfaches Netz



Beispiel für ein Netz mit Zyklus



Findet man heute noch hierarchische bzw. Netzwerkmodelle vor?

Ja, insbesondere in Großrechnerumgebungen. Das bekannteste hierarchische Datenbanksystem, welches noch eingesetzt wird, ist IMS von IBM.

## 4.2 Relationenmodell

Das Relationenmodell wurde von Codd im Jahre 1970 eingeführt und ist mittlerweile das am weitesten verbreitete Datenmodell. Der Sprung von der Theorie in die Praxis wurde durch System/R (von IBM) und Ingres um 1975 vollzogen.

Kommerziell verfügbare Systeme (ab 1980) sind unter anderem:

- SQL/DS und DB2 von IBM
- Oracle von Oracle Inc.
- Ingres von Relational Technology Inc., jetzt Computer Associates
- MS SQL Server bzw. MS Access von Microsoft
- Open Source Systeme: Postgres SQL, Interbase/Firebird, MySQL, MaxDB

Die mathematische Grundlage des Relationenmodells liefert die Relationenalgebra, die der Mengenalgebra ähnlich ist. Das Relationenmodell bietet die wenigsten Modellierungskonstrukte an. Es gibt kein explizites Konstrukt, um Beziehungen zwischen Relationen zu modellieren, diese werden über die Werte der Attribute dargestellt.

Eine Relation kann man sich anschaulich als Tabelle vorstellen. Jede Relation hat einen **Namen** und enthält mehrere Attribute, wobei ein **Attribut** einer Spalte in der Tabelle entspricht. Die Datensätze der Relation heißen **Tupel** und entsprechen den Zeilen einer Tabelle. Die Relationen werden durch das **Relationenschema** beschrieben.

Jedem Attribut wird ein Wertebereich zugeordnet. Wertebereiche können verschiedene Datentypen sein. In der folgenden Tabelle wird eine Übersicht dargestellt, die einige Datentypen, ANSI-SQL-Datentypen, die ihnen entsprechen, und die dazugehörigen Datentypen des Datenbanksystems Oracle beinhaltet.

Datentyp	ANSI-SQL	Oracle
----------	----------	--------



ganze Zahlen	INTEGER	NUMBER(38)
	INT	
	SMALLINT	
Dezimalzahlen	NUMERIC(p,s)	NUMBER(p,s)
	DECIMAL(p,s)	
	FLOAT(b)	NUMBER
	DOUBLE PRECISION	
	REAL	
Zeichenkette	CHARACTER(n)	CHAR(n)
	CHAR(n)	
	CHARACTER VARYING(n)	VARCHAR(n)
	CHAR VARYING(n)	
	VARCHAR(n)	
Boolesche Werte	BOOLEAN	BOOLEAN
Datum	DATE	DATE
Zeit	TIME	DATE
Zeitstempel	TIMESTAMP	TIMESTAMP



#### Verschiedene Datentypen

Mit den Attributen können alle auf den Wertebereichen definierten Operationen ausgeführt werden. So können z.B. Zeichenketten verglichen werden, zwei Daten voneinander abgezogen werden oder einfach Zahlen addiert, subtrahiert, multipliziert, geteilt usw. werden.

Um ein Objekt bzw. ein Tupel eindeutig identifizieren zu können, wird ein *Schlüsselattribut* definiert. Dies kann entweder aus einem einzigen oder aber auch aus mehreren Attributen desselben Objekts bestehen.

Ein **Primärschlüssel** ist ein vom Datenbankadministrator ausgezeichnete Schlüssel. Des weiteren gibt es auch **Fremdschlüssel**, auf die im Kapitel Structured Query Language, Unterkapitel Datenbankdefinition näher eingegangen wird.



## Gliederung

4.2 Relationenmodell4.2.1 Operationen auf Relationen

## 4.2.1 Operationen auf Relationen



## Beispiel

Am Beispiel eines Ausschnittes der Miniwelt Hochschule sollen mögliche Operationen auf Relationen erläutert werden. Eine mögliche Ausprägung der Datenbank ist zum Beispiel:

MatrNr	Name	Vorname	GebDat	Semester	UrlaubsSem	Ort
9812964	Meier	Hans	05.12.1985	1	0	Braunschweig
9652425	Müller	Karla	12.10.1984	5	0	Wolfsburg
9365461	Schmitt	Marc	27.08.1981	11	2	Lüneburg



**Tabelle Studierende**  
aus der Miniwelt Hochschule

PersNr	Name	Vorname	Fach
5234260	Zimmer	Monika	Mathematik
9652425	Irrgang	Rolf	Mediendesign



**Tabelle Lehrende**  
aus der Miniwelt Hochschule

Auf Relationen können verschiedene Operationen ausgeführt werden. An dieser Stelle werden zwar bereits SQL-Anweisungen angegeben, um die Bedeutung der Relationenalgebra für die Praxis zu verdeutlichen, SQL wird jedoch später ausführlich behandelt.

1. Die **Selektion** ist die Auswahl bestimmter Tupel einer Relation.

Sie kann als eine Art Filter angesehen werden, denn es werden aus einer gegebenen Relation alle Tupel herausgesucht, welche einer vorgegebenen Bedingung genügen.



Beispiel

Gib alle Studierenden aus, die Praktische Informatik studieren.

Die SQL-Anweisung dazu lautet:



```
SELECT *  
FROM Studierende  
WHERE Studiengang='PI';
```

In der Bedingung sind sowohl Vergleichsoperatoren (=, <>, <=, >=, <, >) als auch logische Verknüpfungen ("UND": AND, "ODER": OR, "NICHT": NOT) möglich.

2. Die **Projektion** ist die Auswahl bestimmter Attribute einer Relation.



Beispiel

Gib die Namen und die Semesterzahl aller Studierenden aus.



```
SELECT Name, Semester  
FROM Studierende;
```

Duplikate sind zusammenzufassen und in SQL durch DISTINCT auszudrücken.



Beispiel



```
SELECT DISTINCT Name  
FROM Studierende;
```

### 3. Mengenoperationen

Die Voraussetzung für Mengenoperationen ist, dass die beiden beteiligten Relationen R und S "verträglich" sind, d.h. die Relationen müssen die gleiche Anzahl Attribute haben und die Attribute müssen in beiden Relationen paarweise die gleichen Wertebereiche besitzen. In der Praxis werden im Normalfall keine

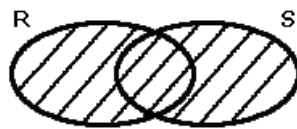
Mengenoperationen mit Relationen sondern mit Projektionen auf Relationen ausgeführt. In diesem Fall müssen die Projektionen die gleiche Anzahl Attribute mit paarweise gleichen Wertebereichen besitzen.

- a. Die **Vereinigung** zweier Relationen R und S hat als Ergebnisrelation die Mengenvereinigung der beiden Relationen:



Formel

$$E = R \cup S$$



Vereinigung  
zweier  
Relationen  
(Relationenmodell)

Die allgemeine SQL-Anweisung dazu lautet:



```
SELECT DISTINCT *
FROM R
UNION SELECT DISTINCT *
FROM S;
```



Beispiel



```
SELECT DISTINCT Name, Vorname
FROM Studierende
UNION SELECT DISTINCT Name, Vorname
```

```
FROM Lehrende;
```

b. Die **Differenz** zweier Relationen R und S:



Formel

$$E = R - S$$



Differenz  
zweier  
Relationen  
(Relationenmodell)

Die allgemeine SQL-Anweisung dazu lautet:



```
SELECT DISTINCT *
FROM R
EXCEPT SELECT DISTINCT *
FROM S;
```

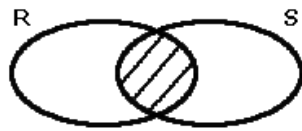
Bei ORACLE lautet das Schlüsselwort für die Differenz nicht EXCEPT sondern MINUS.

c. Der **Durchschnitt** zweier Relationen R und S (kann auch als Differenz dargestellt werden):



Formel

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



**Durchschnitt  
zweiter  
Relationen  
(Relationenmodell)**

Die allgemeine SQL-Anweisung dazu lautet:



```
SELECT DISTINCT *
FROM R
INTERSECT SELECT DISTINCT *
FROM S;
```



**Beispiel**

Gib alle Studierenden der Hochschule aus, die den gleichen Vor- und Nachnamen wie Lehrende haben (z.B. Hans Müller).



```
SELECT DISTINCT Name, Vorname
FROM Studierende
INTERSECT SELECT DISTINCT Name, Vorname
FROM Lehrende;
```

4. Das kartesische Produkt zweier Relationen R und S enthält alle Attribute von R und S.



**Übung**

Überlegen Sie, wie das Ergebnis aussehen müsste, und überprüfen Sie Ihre Überlegung durch Anklicken der einzelnen Schaltflächen in der Tabelle der Ergebnisrelation.

R	A	B
	1	2

	2	2
--	---	---



Tabelle der Relation R

S	C	D	E
	1	2	3
	4	5	6
	7	8	9



Tabelle der Relation S

R x S	A	B	C	D	E
	?	?	?	?	?
	1	2	1	2	3
	?	?	?	?	?
	1	2	4	5	6
	?	?	?	?	?
	1	2	7	8	9
	?	?	?	?	?
	2	2	1	2	3
	?	?	?	?	?
	2	2	4	5	6
	?	?	?	?	?
	2	2	7	8	9


**Kartesisches Produkt der Relationen R und S (interaktiv)**

Die allgemeine SQL-Anweisung dazu lautet:



```
SELECT *
FROM R
CROSS JOIN S;
```

5. Der **Gleichverbund** verknüpft zwei Relationen über alle gemeinsamen Attribute, indem er jeweils zwei Tupel verschmilzt, falls sie dort gleiche Werte aufweisen.


**Übung**

Überlegen Sie, wie das Ergebnis aussehen müsste, und überprüfen Sie Ihre Überlegung durch Anklicken der einzelnen Schaltflächen in der Tabelle der Ergebnisrelation.

R	A	B	C
	1	1	0
	1	0	1
	1	1	2


**Tabelle der Relation R**

S	C	D	E	F
	0	1	2	3
	2	2	1	4
	0	1	1	1


**Tabelle der Relation S**

$R \bowtie S$	A	B	C	D	E	F
---------------	---	---	---	---	---	---



	?	?	?	?	?	?
	1	1	0	1	2	3
	?	?	?	?	?	?
	1	1	0	1	1	1
	?	?	?	?	?	?
	1	1	2	2	1	4



Natürlicher Verbund der Relationen R und S (interaktiv)

Die allgemeine SQL-Anweisung dazu lautet:



```
SELECT *
FROM R
EQUI JOIN S
ON R.C=S.C;
```

Eine Verallgemeinerung des Verbund-Begriffes ergibt sich zum einen, wenn die Verknüpfung auch über nicht gleichnamige Attribute erlaubt ist, und zum anderen, wenn die Tupel bezüglich anderer Vergleichsbeziehungen verknüpft werden (Vergleichsoperator aus  $\{=, <>, <=, >=, <, >\}$ ). In diesem Fall wird von "Theta-Verbund" gesprochen.

6. Die **Umbenennung** wird benötigt, um zwei Relationenschemata etwa für eine Vereinigung kompatibel zu machen oder um die zu verbindenden Attribute vor einem natürlichen Verbund zu verändern.

In SQL ist eine Umbenennung durch AS gegeben:



```
SELECT MatrNr AS ID
FROM Studierende;
```

Bei Informix kann die Umbenennung wie folgt durchgeführt werden:



```
RENAME COLUMN Studierende.MatrNr  
TO ID;
```

Von den drei klassischen Datenmodellen ist das Relationenmodell am einfachsten zu handhaben, da Anfragen deskriptiv formuliert werden und dadurch auch sehr schnell formuliert werden können.



```
SELECT MatrNr AS ID  
FROM Studierende;
```

## 4.3 ER-Modell

Das **Entity-Relationship-Modell (ER-Modell)** wurde von P. P. Chen im Jahre 1976 vorgestellt. Es ist eines der bekanntesten semantischen Datenmodelle, und obwohl es bislang in keinem Datenbanksystem verwirklicht wurde, wird es als de-facto-Standardmodell für frühe Entwurfsphasen der Datenbankentwicklung eingesetzt.

Die grundlegenden Modellierungskonzepte sind:

1. Entitäten/Objekte (entities)

Ein **Entity** ist ein Objekt der "realen Welt" (konkret oder abstrakt), wie zum Beispiel:

- eine bestimmte Lehrveranstaltung
- eine bestimmte Lehrende/ein bestimmter Lehrender

2. Beziehungen (relationships)

Eine **Beziehung** beschreibt ein Verhältnis zwischen Entities, wie zum Beispiel:

- die Teilnahme einer/eines bestimmten Studierenden an einer bestimmten Lehrveranstaltung
- das Anbieten einer bestimmten Lehrveranstaltung durch eine bestimmte Lehrende/einen bestimmten Lehrenden

3. Attribute (attributes)

Ein **Attribut** ist eine Eigenschaft eines Entity oder einer Beziehung, z.B.:

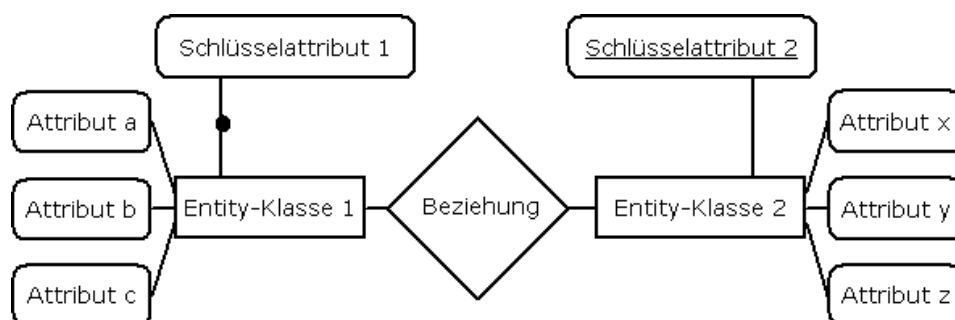
- die Matrikelnummer einer/eines Studierenden
- die Bezeichnung einer Lehrveranstaltung

**Schlüsselattribute** sind Attribute, die erlauben, ein Entity eindeutig zu identifizieren, wie z.B. die Matrikelnummer, weil eine Matrikelnummer innerhalb einer Hochschule jeweils ein einziges Mal vergeben wird. Gäbe es keine Matrikelnummer, könnten der Name, der Vorname und das Geburtsdatum (als Tripel) einer Studierenden/eines Studierenden als Schlüsselattribut verwendet werden.

**Optionale Attribute** sind solche, die nicht zwingend einen Wert enthalten müssen.

Im ER-Modell werden die Entity-Klassen durch Rechtecke, die Attribute durch Ovale oder abgerundete Rechtecke und die Beziehungen durch Rauten, die mit den betreffenden Entities durch Linien verbunden sind, dargestellt. Die Schlüsselattribute werden entweder durch Unterstreichung oder durch einen ausgefüllten Kreis auf der Verbindungslinie zwischen Entity und Attribut gekennzeichnet. Die optionalen Attribute werden durch einen Kreis auf der Verbindungslinie zwischen Entity und Attribut gekennzeichnet.

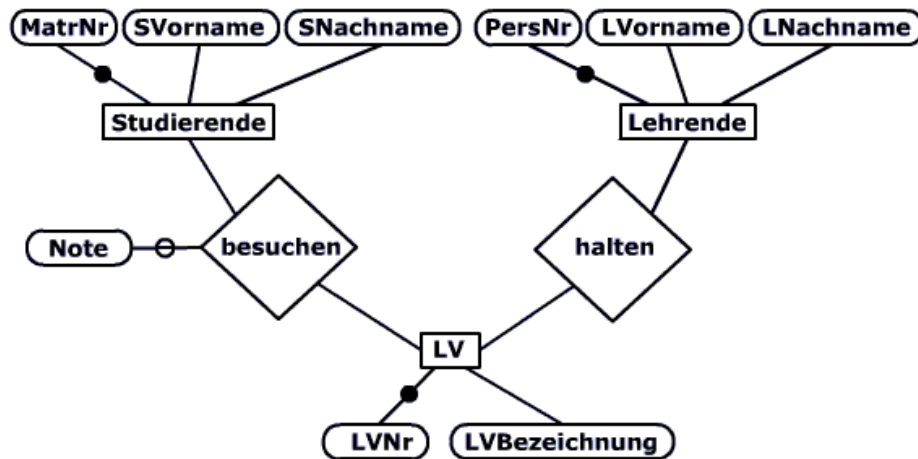
Die folgende Abbildung enthält die grundlegenden Modellierungskonzepte, wobei auch beide Darstellungsarten für Schlüsselattribute berücksichtigt wurden. Dies dient lediglich der Veranschaulichung, es soll jedoch nur die eine *oder* die andere Darstellungsart innerhalb eines ER-Modells verwendet werden.



Grundlegende Modellierungskonzepte des ER-Modells



Beispiel



 **Ausschnitt aus der VFH-Welt (ER-Modell)**

In der Online-Version befindet sich an dieser Stelle eine Simulation.

Das angegebene Beispiel besteht aus den Entities

- *Studierende* mit den Attributen ,SVorname und SNachname
- *Lehrende* mit den Attributen ,LVorname und SNachname

sowie

- LV (Lehrveranstaltung) mit den Attributen `LVBezeichnung`

und aus den Beziehungen *besuchen* (mit dem optionalen Attribut *Note*) und *halten*.

Studierende, die eine Lehrveranstaltung besuchen, können zwar auch eine Note dafür erhalten, müssen aber nicht. In der Relation "besuchen" kann also bei einigen Studierenden das Attribut "Note" leer sein.

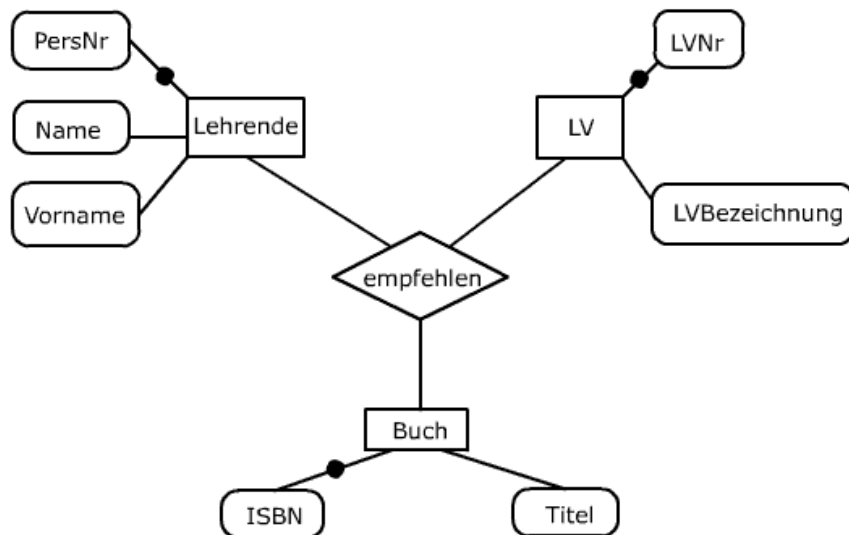
Im ER-Modell sind mehrstellige Beziehungen erlaubt.

Im folgenden Beispiel wird anhand einer dreistelligen Beziehung gezeigt, wie eine Umsetzung in drei zweistellige Beziehungen erfolgen kann.



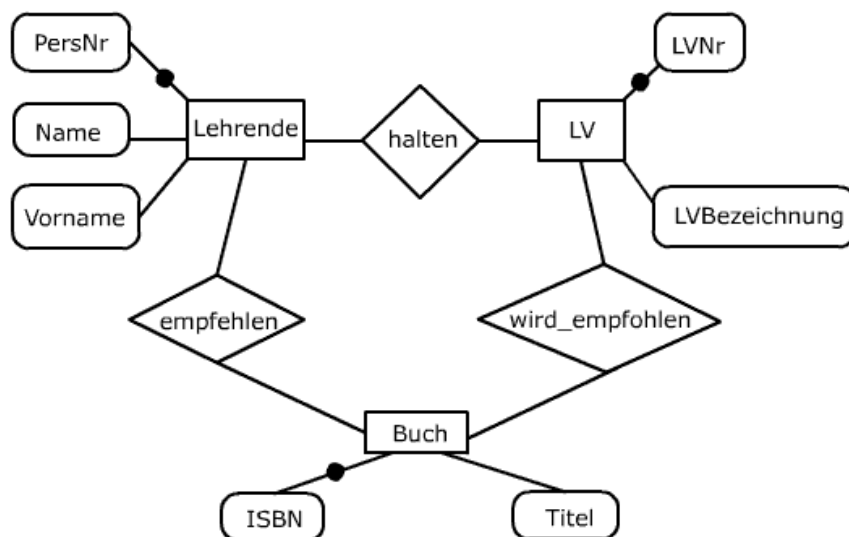
### Beispiel

Beispiel für die Umsetzung einer dreistelligen Beziehung in drei zweistellige Beziehungen: die dreistellige Beziehung *empfehlen*



Dreistellige Beziehung (ER-Modell)

wird in die drei zweistellige Beziehungen halten, empfehlen und wird\_empfohlen umgewandelt:



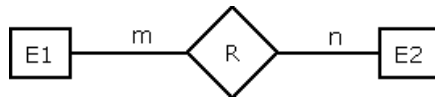
Zweistellige Beziehungen (ER-Modell)



In der Online-Version befindet sich an dieser Stelle eine Simulation.

**Animation der Umsetzung einer dreistelligen Beziehung in drei zweistellige Beziehungen**

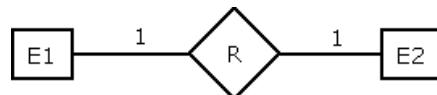
Die Beziehungen im ER-Modell sind m:n-Beziehungen:

**m:n-Beziehung**

Das heißt, dass bei zwei beteiligten Entity-Klassen E1 und E2 jeweils ein Entity aus der Klasse E1 mit beliebig vielen Entities aus der Klasse E2 in Beziehung steht und ein Entity aus der Klasse E2 mit beliebig vielen Entities aus der Klasse E1 in Beziehung steht. Ein Beispiel wäre, dass Studierende mehrere ( $n \geq 0$ ) Lehrveranstaltungen belegen können und eine Lehrveranstaltung von mehreren ( $m \geq 0$ ) Studierenden belegt werden kann.

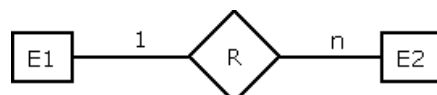
Es gibt zwei Sonderfälle der m:n-Beziehung, und zwar:

1. 1:1-Beziehung



Jedes Entity aus der Klasse E1 steht mit höchstens einem Entity aus der Klasse E2 in Beziehung. Ein Beispiel dafür wäre der Beziehungstyp "verheiratet" zwischen Personen.

2. 1:n-Beziehung



Jedes Entity aus E1 steht mit  $n \geq 0$  (keinem, einem oder mehreren) Entities aus E2 in Beziehung. Beispiel: Eine Lehrende kann mehrere Abschlussarbeiten betreuen.

Um aber z.B. darzustellen, dass ein Buchexemplar zu genau einem Buch gehört und ein Buch mehrere Exemplare haben kann, wäre eine n:1-Beziehung notwendig. Diese und andere Einschränkungen des ER-Modells haben durch die Erweiterung um verschiedene andere Modellierungskonstrukte zum Erweiterten Entity-Relationship-Modell (EER-Modell) geführt.



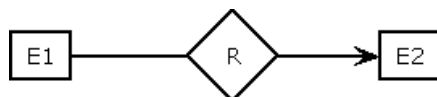
Das Relationenmodell kommt mir recht anschaulich vor. Warum brauchen wir das ER-Modell?

Das ER-Modell ist mächtiger als das Relationenmodell, wenn es darum geht, den Realitätsausschnitt zu modellieren, da das ER-Modell auf semantischer Modellierung beruht. Durch die Möglichkeit, Objekte und ihre Beziehungen durch Attribute zu beschreiben, wird der Datenbankentwurfsprozess sehr gut unterstützt.

## 4.4 EER-Modell

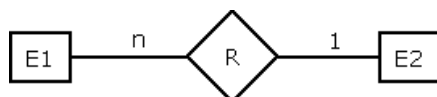
Da das Entity-Relationship-Modell über relativ wenige Modellierungskonzepte verfügt, um die vielfältigen Strukturen der "realen Welt" darzustellen, gab es in der Vergangenheit unterschiedliche Versuche, es zu erweitern, um z.B. die Modellierung von Abstraktionen zu ermöglichen. Wir werden das erweiterte Entity-Relationship-Modell (EER-Modell) vorstellen, welches an der Abteilung Datenbanken der TU Braunschweig in den Jahren 1987-1991 entwickelt wurde.

Eine Einschränkung des ER-Modells ist die Tatsache, dass sich nur m:n-Beziehungen mit deren beiden Sonderfällen 1:1- und 1:n-Beziehung modellieren lassen. Daher ist eine naheliegende Erweiterung die **funktionale Beziehung**, die auch als n:1-Beziehung bezeichnet werden kann. Diese stellt eine eindeutige Zuordnung eines Entity zu einem anderen Entity dar. Jedem Entity der Klasse E1 wird maximal ein Entity der Klasse E2 zugeordnet. Die funktionale Beziehung wird graphisch wie die normale Beziehung im ER-Modell dargestellt, nur dass die Verbindungslinie zwischen der Beziehung und der Entity-Klasse E2 mit einem Pfeil auf der Seite der Entity-Klasse endet:



**Funktionale Beziehung (EER-Modell)**

Eine alternative Darstellung ist die als n:1-Beziehung:





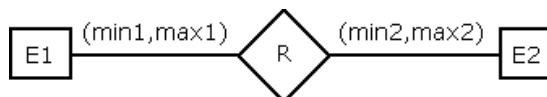
#### n:1-Beziehung (EER-Modell)

Die funktionale Beziehung kann auch als sogenannte einfache Komponente (auch objektwertiges Attribut genannt) modelliert werden. Die grafische Darstellung sieht folgendermaßen aus:



#### Einfache Komponente (EER-Modell)

Eine weitere Erweiterung des ER-Modells ist die Angabe von Kardinalitäten bei Beziehungsklassen. **Kardinalitäten** sind eine Möglichkeit, implizite Integritätsbedingungen darzustellen. Dies erfolgt graphisch durch die Angabe zweier Zahlenpaare (min1, max1) und (min2, max2):



#### Kardinalität (EER-Modell)

Dabei ist min1 die minimale Anzahl von Entities aus der Klasse E2, mit denen ein Entity aus der Klasse E1 in Beziehung stehen kann, während max1 das Maximum angibt. Analog dazu ist min2 die minimale Anzahl von Entities aus der Klasse E1, mit denen ein Entity aus der Klasse E2 in Beziehung stehen kann, während max2 das Maximum angibt.

Gilt  $\min1 = \max1 = \min2 = \max2 = 1$ , so wird eine 1:1-Beziehung modelliert, die nur dann realisiert werden kann, wenn E1 und E2 die gleiche Anzahl von Entities enthalten.

Im ER-Modell steht eine beschränkte Anzahl an Datentypen zur Verfügung, wie BOOL, INT, REAL, STRING. Wünschenswert wären jedoch auch **Nichtstandard-Datentypen** wie POINT (für die Angabe der x-, y- und z-Koordinaten eines Punktes), LINE (für die Angabe des Anfang- und Endpunktes sowie der Länge einer Strecke) oder POLYGON (für die Angabe der Anzahl der Punkte und der Fläche eines Polygons).

Die **Erweiterung von Attributen** um strukturierte bzw. zusammengesetzte Attributwerte ist eine weitere Möglichkeit, die Modellierungskonzepte des ER-Modells zu erweitern. Dies kann geschehen durch:

- **Tupelbildung** (zusammengesetzte Attribute), z.B.

Adresse=PROD(Straße, Nr, PLZ, Ort)



Graphisch werden die zusammengesetzten Attribute wie auch die normalen Attribute dargestellt, nur dass anstelle des Datentyps die Angabe des Tupels erfolgt.



Beispiel

Beispiel: Studierende haben (genau) eine Anschrift.

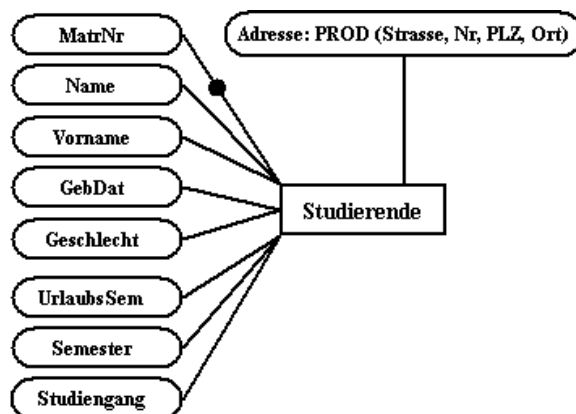


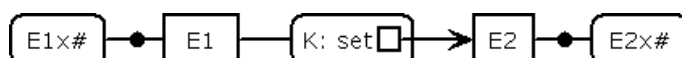
Abbildung einer mehrwertigen Komponente auf das Relationenmodell (Beispiel)

Diese Modellierung wurde nur beispielhaft angegeben, um die Tupelbildung zu illustrieren. Gerade bei Studierenden ist es oft so, dass sie eine Heimatanschrift und eine Semesteranschrift haben, wofür sich eine Modellierung empfiehlt, wie sie in der Miniwelt Hochschule realisiert wurde.

- **Mengenbildung** (mengenwertige/mehrwertige Attribute) wie
  - *set* (Mengen), z.B. Telefon=SET(Privat, Dienstlich),
  - *bag* (Multimengen=ungeordnete Mengen mit Duplikaten), z.B. Geburtstag=BAG(Datum) und
  - *list* (Listen=geordnete Mengen mit Duplikaten), z.B. Autor=LIST(Autor1, Autor2, Autor3).

Seltener benutzt werden objektwertige Attribute, um darzustellen, dass ein oder mehrere Entities Komponente/n eines anderen Entity ist/sind.

Die graphische Darstellung der mengenwertigen Attribute ist wie folgt:



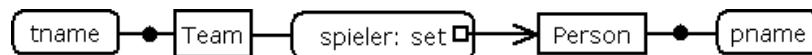
Mengenwertiges Attribut (EER-Modell)

Gleiche Darstellung wird auch für *bag* und *list* benutzt, nur dass statt *set*, je nach Art des mengenwertigen Attributs, *bag* oder *list* angegeben werden müssen.



Beispiel

Ein Team hat einen Namen und besteht aus mehreren Spielern (Personen), die ebenfalls durch ihre Namen identifiziert werden.

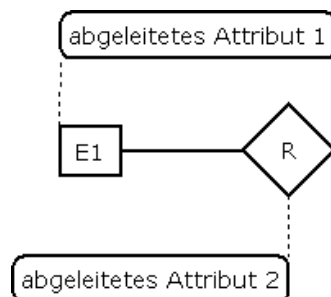


Beispiel für mengenwertiges Attribut (EER-Modell)

- **abgeleitete (berechnete) Attribute,**

z.B. Kindergeld=AnzahlKinder\*Kindergeld\_pro\_Kind

Die abgeleiteten Attributen unterscheiden sich in der graphischen Darstellung nur geringfügig von den normalen Attributen, und zwar ist die Verbindungslinie zwischen Attribut und Entity bzw. Attribut und Beziehung *gestrichelt*:

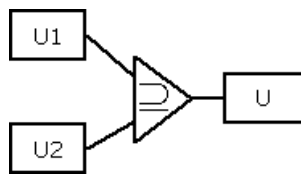


Abgeleitetes Attribut  
(EER-Modell)

Zu guter Letzt müssen die Generalisierung, Spezialisierung und Partitionierung erwähnt werden.

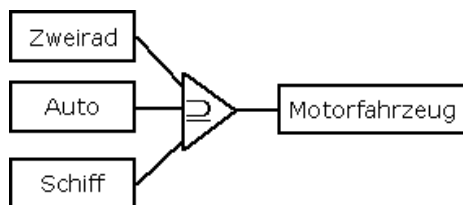
Bei der **Generalisierung** werden zwei oder mehr Entity-Klassen zu einer übergeordneten Klasse generalisiert. Die **Spezialisierung** ist die Umkehrung der Generalisierung. Bei der Generalisierung erbt die übergeordnete Klasse alle gemeinsamen Attribute der untergeordneten Klassen, während bei der Spezialisierung die Klassen sich von oben nach unten spezialisieren. Graphisch werden die vorgestellten Konstrukte folgendermaßen dargestellt:

Generalisierung:

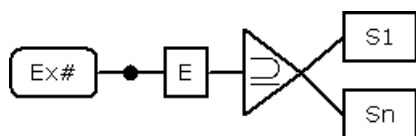
Generalisierung  
(EER-Modell)

Beispiel

*Zweirad*, *Auto* und *Schiff* werden zu *Motorfahrzeug* generalisiert. Es gibt jedoch Zweiräder (z.B. Fahrräder) und Schiffe (z.B. Segelschiffe), die keine Motorfahrzeuge sind.

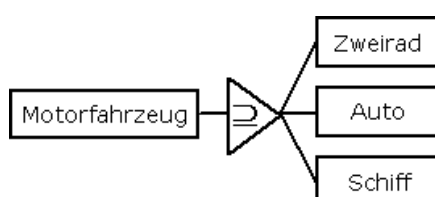
Beispiel für Generalisierung  
(EER-Modell)

Spezialisierung:

Spezialisierung  
(EER-Modell)

Beispiel

*Motorfahrzeug* wird zu *Zweirad*, *Auto* und *Schiff* spezialisiert. D.h. es gibt weitere Motorfahrzeuge (z.B. Flugzeuge), die weder Zweiräder, noch Autos, noch Schiffe sind.

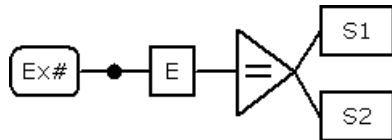




### Beispiel für Spezialisierung (EER-Modell)

Die **Partitionierung** ist ein Spezialfall der Spezialisierung, wobei die Exemplare der Klasse disjunkt sind.

Partitionierung:

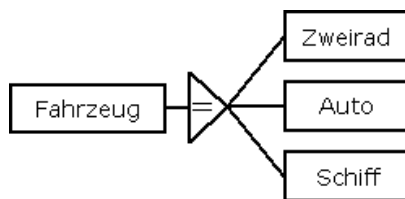


### Partitionierung (EER-Modell)



### Beispiel

*Fahrzeuge können nur Zweiräder, Autos oder Schiffe sein.*



### Beispiel für Partitionierung (EER-Modell)

Mit den vorgestellten Konstrukten verfügen wir über eine Erweiterung des ER-Modells, welche sich sehr gut zur Modellierung von Anwendungswelten einsetzen lässt.



Welche Konstrukte des EER-Modells werden in der Praxis eingesetzt?

Funktionale Beziehung, Kardinalitätsangabe, Mengenbildung, Generalisierung, Partitionierung, Spezialisierung sind die Konstrukte, die am häufigsten zum Einsatz kommen.

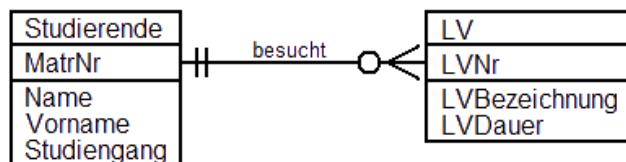
Ein Beispiel für eine vollständige EER-Modellierung befindet sich unter [Miniwelt Hochschule](#).

## 4.5 CrowFoot-Notation

Die CrowFoot-Notation, auch bekannt als Martin-Notation, ist von James Martin entwickelt worden. Es ist eine alternative Darstellungsform von Kardinalitäten zwischen zwei Entities in einem Entity-Relationship-Modell. In vielen Modellierungstools, wie Visio oder PowerDesigner, wird diese Notation zur Darstellung der Kardinalitäten verwendet. In der UML (Unified Modelling Language) findet die CrowFoot-Notation ebenfalls Anwendung.



Beispiel





CrowFoot Beispiel

Das obige Beispiel zeigt die CrowFoot-Notation einer 1:n-Beziehung zwischen zwei Entities. Die Beziehung wird durch eine Linie zwischen den Entities repräsentiert. An den beiden Enden einer Beziehung stehen die Kardinalitäten, die durch Symbolpaare visualisiert werden. Im Gegensatz zum ER-Modell wird hier auf die Raute als Darstellung einer Beziehung verzichtet.

Folgende Symbole werden für die Visualisierung der Kardinalitäten verwendet:

Symbol	Verwendung
 	"0"
 	"1"

 	mehrere
--	---------

Aus diesen Symbolen lassen sich vier Symbolpaare bilden. Diese Symbolpaare werden an beiden Enden der Beziehung als Kardinalität verwendet.

Bedeutung der Symbolpaare:

Symbolpaar	Beschreibung
 	Ein Entity nimmt an genau einer Beziehung teil.
 	Ein Entity nimmt höchstens an einer Beziehung teil.
 	Ein Entity nimmt mindestens an einer Beziehung teil.
 	Ein Entity nimmt an beliebig vielen Beziehungen teil.

## 4.6 Weitere Modelle

Als weitere Datenmodelle werden kurz das NF2-Modell, objektorientierte Datenmodelle und deduktive Datenbanken vorgestellt.

Das **NF2-Modell** (Non First Normal Form) stellt eine Erweiterung des relationalen Modells um komplexe Attributwerte dar, d.h. Attribute können selbst wieder Relationen sein.



Beispiel

PersNr	Vorname	Nachname	Telefon
			TelNr
1523345	Hans	Schulze	54224
			74625
2314856	Franziska	Maier	52456
3495067	August	Müller	65252
			75362
4526748	Michaela	Sommer	76472

Die obige Relation enthält Personaldaten der Fachhochschulangehörigen in Tabellenform. Die Personaldaten umfassen eine Unterrelation mit Telefonnummern. Im Folgenden wird die Relation um Informationen über die Fachbereiche erweitert. Die bisherige Relation wird damit zur Unterrelation. Die Personaldaten werden zum Attribut *Belegschaft* zusammengefasst und es kommt noch ein Attribut *Fachbereich* hinzu.

Fachbereich	Belegschaft			
	PersNr	Vorname	Nachname	Telefon
				TelNr
Informatik	1523345	Hans	Schulze	54224
				74625
	2314856	Franziska	Maier	52456
	3495067	August	Müller	65252
				75362
Mathematik	4526748	Michaela	Sommer	76472

Beispielprototypen für das NF2-Modell sind DASDBS (Darmstädter Datenbank-Kern) und AIM-P (Advanced Information Management Prototype) von IBM.

**Objektorientierte Datenmodelle** sind eine Erweiterung der klassischen Datenmodelle um

1. mehr Konzepte zur besseren Darstellung der Struktur von Anwendungsobjekten, wie z.B.

- *komplexe* Werte können mit Typkonstrukturen (set of, tuple of, list of) beschrieben werden,
  - *Objektidentität* kann gespeicherte Objekte von Werten, die sie besitzen kann, unterscheiden,
  - *Vererbung von Attributen und Methoden* zwischen Objekt-Typen, die in einer IST-Beziehung zueinander stehen.
2. mehr Konzepte zur Darstellung objektspezifischer Operationen, wie z.B. Methoden. Durch Methoden werden sowohl die Struktur der Beschreibung der Anwendungsdaten als auch die erlaubten Operationen festgelegt, mit denen die Anwendungsdaten manipuliert werden dürfen.



Beispiel

Im Relationenmodell gäbe es die Relation *Studierende* mit *Einfügen*, *Ändern* und *Löschen* von Tupeln als möglichen Operationen. Diese Operationen können aber genauso gut z.B. auf die Relation *Buch* angewandt werden.

Beim objektorientierten Modell können dagegen Operationen speziell für jeden Objekttyp definiert werden. Für den Objekttyp *Studierende* könnte es die Operationen *Immatrikulation*, *Exmatrikulation* und *Prüfung\_ablegen* geben, während vom Objekttyp *Person* die Methode *Adresse\_ändern* geerbt werden könnte. Objektorientierte Datenbankmanagementsysteme sind zum Beispiel db4o (database for objects) und ObjectDB

**Deduktive Datenbanken** benutzen als Ansatz eine Beschreibung von Datenbanken als Sammlungen von Fakten und Regeln, auf denen Deduktionen (Herleitungen) auf der Basis eines Logik-Kalküls durchgeführt werden können. Die Konzepte der deduktiven Datenbanken sind Fakten und Regeln. Die *Fakten* sind atomare Formeln ohne Variablen und sind vergleichbar mit den Tupeln einer relationalen Datenbank, und die Regeln (Klauseln) werden dafür benutzt, um Informationen aus vorliegenden Grunddaten abzuleiten. Dies ist vergleichbar mit einer Sichtdefinition auf relationale Datenbanken. Die deduktiven Datenbanken haben eine mächtigere Anfragesprache, wodurch rekursive Anfragen ermöglicht werden.



Beispiel

*Fakt:* Angestellte (pid\_meier, 'Meier', 'Bettina', 'Braunschweig', 2000, '29.04.1981')  
*Regel:* Angestellte (PID, Name, Vorname, Wohnort, Gehalt, Geburtsdatum) /\ Gehalt >= 15000 Schwerverdiener (PID, Name, Vorname)



## 4.7 Freitextaufgaben zu Datenmodelle

1. Welche Vorteile hat das EER-Modell gegenüber dem ER-Modell?

**Lösung zeigen**

Die Vorteile des EER-Modell bestehen aus den Erweiterungen wie Nichtstandard-Datentypen, Kardinalitäten, funktionale Beziehungen, Generalisierung, Spezialisierung.

2. Vergleichen Sie die drei "klassischen" Datenmodelle (hierarchisches, Netzwerk- und Relationenmodell) nach folgenden Kriterien: Einfachheit der Benutzung, Sprachebene und Effizienz.

**Lösung zeigen**

	Hierarchisches Modell	Netzwerkmodell	Relationenmodell
Einfachheit der Benutzung	umständlich	umständlich	einfach
Sprachebene	sehr niedrig	mittel	hoch
Effizienz	sehr gut	gut	gut

3. Welche Vorteile hat das ER-Modell gegenüber den drei klassischen Modellen?

**Lösung zeigen**

Die Vorteile sind die Vermeidung der Datenredundanz und die Übersichtlichkeit des gesamten Modells.

## 5 Relationale Datenbanken



### Zeitumfang

Die voraussichtliche Bearbeitungsdauer dieser Lerneinheit (ohne Übungsaufgaben!) beträgt ca. 2 Stunden.



### Lernziele

Bisher haben Sie den konzeptionellen Teil dieses Fachgebietes kennengelernt. Nun werden Sie erfahren, wie Ihr Konzept auf ein Datenbanksystem umgesetzt werden soll. Zunächst wird das EER-Modell in das Relationenmodell transformiert. Das Ergebnis der Transformation wird immer in bereits normalisierten Relationen münden. Alternativ kann das ER Modell in Crow Foot Notation dargestellt werden. Warum die Normalisierung im Relationenmodell trotzdem wesentlich ist, erfahren Sie im darauffolgenden Abschnitt.



### Gliederung

#### 5 Relationale Datenbanken

##### 5.1 Abbildung vom (E)ER- auf das Relationenmodell

##### 5.2 Normalisierung

##### 5.3 Relationenalgebra

##### 5.4 Freitextaufgaben zu Relationale Datenbanken

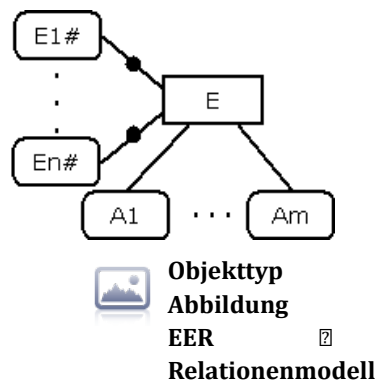
### 5.1 Abbildung vom (E)ER- auf das Relationenmodell

Der logische Entwurf beinhaltet die Transformation des konzeptionellen Datenbankschemas in das für die Implementierung vorgesehene Datenmodell. Nachfolgend wird ein Verfahren angegeben, mit dessen Hilfe die Transformation durchgeführt werden kann.

#### 1. Basisobjekttypen und Beziehungstypen

Die Objekt- und Beziehungstypen werden jeweils auf Relationenschemata abgebildet. Die Attribute werden zu Attributen des Relationenschemas, die Schlüsselattribute werden als Schlüsselattribute übernommen. Die nachfolgenden Bilder sollen das veranschaulichen.

- Aus dem **Objekttyp**  $E$  mit den Schlüsselattributen  $E1\#, \dots, En\#$  und den weiteren Attributen  $A1, \dots, Am$ :



wird die Relation

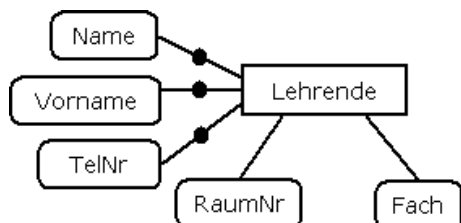
$E(\underline{E1\#}, \dots, \underline{En\#}, A1, \dots, Am)$

mit den Schlüsselattributen  $E1\#, \dots, En\#$  und den weiteren Attributen  $A1, \dots, Am$ .



Beispiel

Aus dem Objekttyp *Lehrende* mit den Schlüsselattributen *Name*, *Vorname*, *TelNr* und den weiteren Attributen *RaumNr* und *Fach*:



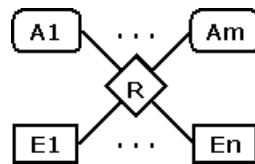
Beispiel für die Abbildung  
eines Objekttyps auf das  
Relationenmodell

wird die Relation

*Lehrende* (*Name*, *Vorname*, *TelNr*, *RaumNr*, *Fach*)

mit den Schlüsselattributen *Name*, *Vorname*, *TelNr* und den weiteren Attributen *RaumNr*, *Fach*.

- Aus dem **Beziehungstyp**  $R$  mit den Attributen  $A1, \dots, Am$  und den beteiligten Objekttypen  $E1$  (mit Schlüsselattribut/Schlüsselattributen  $E1x\#$ ),  $\dots$ ,  $En$  (mit Schlüsselattribut/Schlüsselattributen  $Enx\#$ )



**Beziehungstyp  
Abbildung  
EER  
Relationenmodell**

wird die Relation

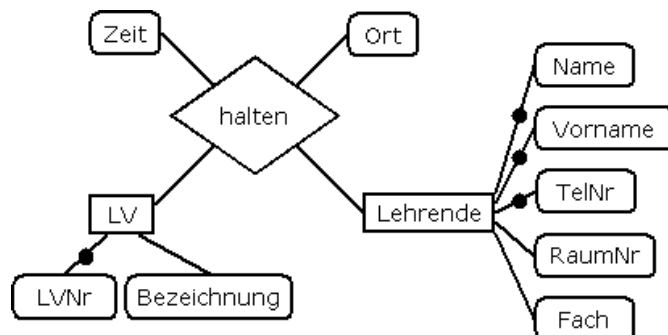
$$R (\underline{E1x\#} \rightarrow E1.E1x\#, \dots, Enx\# \rightarrow \underline{En.Enx\#}, A1, \dots, Am)$$

mit den Schlüsselattributen  $E1x\#, \dots, Enx\#$ , die gleichzeitig auch Fremdschlüssel sind, und den weiteren Attributen  $A1, \dots, Am$ .



**Beispiel**

Aus dem Beziehungstyp *halten* (mit den Attributen *Zeit*, *Ort*) und den beteiligten Objekttypen *LV* (=Lehrveranstaltung, mit dem Schlüsselattribut *LVNr*) und *Lehrende* (mit den Schlüsselattributen *Name*, *Vorname*, *TelNr*)



**Beispiel für die Abbildung eines Beziehungstyps  
auf das Relationenmodell**

wird die Relation

$$\text{halten} (\underline{LVNr} \rightarrow LV.LVNr, \underline{Name} \rightarrow \text{Lehrende.Name}, \underline{Vorname} \rightarrow \text{Lehrende.Vorname}, \underline{TelNr} \rightarrow \text{Lehrende.TelNr}, \text{Zeit}, \text{Ort})$$

mit den Schlüsselattributen *LVNr*, *Name*, *Vorname*, *TelNr*, die gleichzeitig auch Fremdschlüssel sind, und den weiteren Attributen *Zeit*, *Ort*.

- Die **funktionale Beziehung**



**Funktionale Beziehung**  
Abbildung EER  
Relationenmodell

ist ein Sonderfall und wird anders als die normalen Beziehungstypen umgesetzt. Sie bildet keine neue Relation, sondern es wird lediglich das Schema des Objekttyps E1 um Fremdschlüsselattribute

$$E2x\# \rightarrow E2x.E2x\#$$

ergänzt (diese sind gleichzeitig auch Primärschlüssel, zusammen mit E1x#) und der Objekttyp E2 wie bei einer gewöhnlichen Beziehung transformiert. Dabei steht E2x# für das Schlüsselattribut/die Schlüsselattribute des Objekttyps E2. Die Relationen E1 und E2 sehen also folgendermaßen aus:

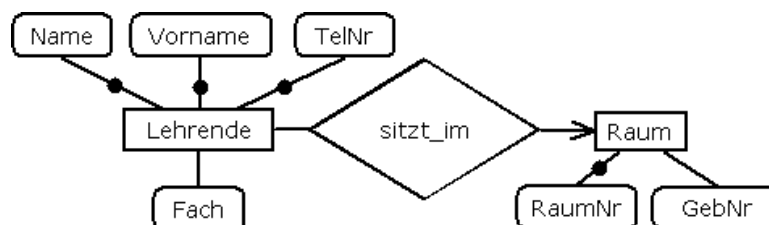
E1 (E11#, ..., E1n#, A11, ..., A1m, E2x# → E2x.E2x#)

E2 (E21#, ..., E2n#, A21, ..., A2m)



Beispiel

Die funktionale Beziehung



**Beispiel für die Abbildung einer funktionalen Beziehung auf das Relationenmodell**

wird in folgende Relationen transformiert:

Lehrende (Name, Vorname, TelNr, Fach, RaumNr → Raum.RaumNr)

Raum (RaumNr, GebNr)

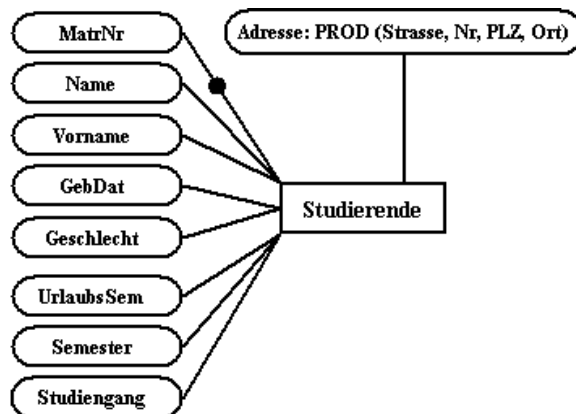
## 2. Tupelbildung

Die einzelnen Attribute des Tupels werden zu normalen Attributen

Adresse=PROD(Straße, Nr, PLZ, Ort)



Beispiel



Beispiel für die Abbildung einer einfachen  
Komponenten auf das Relationenmodell

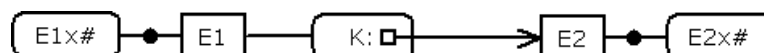
wird auf die Relation

Studierende (MatrNr, Name, Vorname, GebDat, Geschlecht, UrlaubsSem, Semester, Studiengang, Strasse, Nr, PLZ, Ort)

abgebildet.

### 3. Komponenten

- einfache Komponenten werden wie funktionale Beziehungen behandelt:



Einfache Komponente  
Abbildung EER → Relationenmodell

wird auf die Relationen

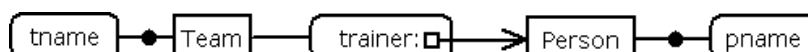
$E1 (\underline{E1x\#}, \dots, K \rightarrow E2.E2x\#)$

$E2 (\underline{E2x\#}, \dots)$

abgebildet. Dabei stehen  $E1x\#$  und  $E2x\#$  für die Schlüsselattribute der Objekttypen  $E1$  und  $E2$  und die Auslassungspunkte für eventuelle weitere Attribute der beiden Objekttypen.



Beispiel





### Beispiel für die Abbildung einer einfachen Komponenten auf das Relationenmodell

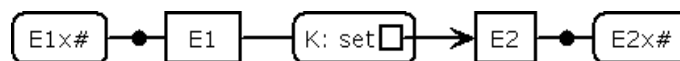
wird auf die Relation

Team (tname, trainer → Person.pname)

Person (pname, ...)

abgebildet.

- bei **mehrwertigen Komponenten** entsteht, wie bei einer einfachen Beziehung auch, eine zusätzliche Relation:



### Mehrwertige Komponente Abbildung EER → Relationenmodell

wird auf die Relationen

E1 (E1x#, ...)

E2 (E2x#, ...)

K (E1x# → E1.E1x#, E2x# → E2.E2x#)

abgebildet. Dabei stehen  $E1x\#$  und  $E2x\#$  für die Schlüsselattribute der Objekttypen  $E1$  und  $E2$  und die Auslassungspunkte für eventuelle weitere Attribute der beiden Objekttypen.

Bei *list* statt *set* kommt ein neues Attribut *position* vom Datentyp INTEGER in der Relation  $K$  hinzu, wobei dann  $E1x\#$  und *position* Schlüsselattribute sind.

Bei *bag* statt *set* kommt ein neues Attribut *anzahl* vom Datentyp INTEGER in der Relation  $K$  hinzu, wobei dann  $E1x\#$  und  $E2x\#$  Schlüsselattribute bleiben.



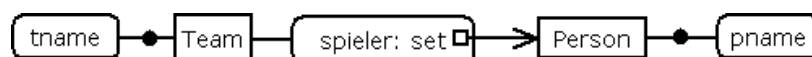
Beispiel



### Beispiel für die Abbildung einer mehrwertigen Komponenten auf das Relationenmodell

wird auf die Relationen

Team (tname, ...)



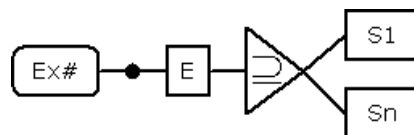
Person (pname, ...)

spieler (tname → Team.tname, pname → Person.pname)

abgebildet.

#### 4. Typkonstruktionen

- **Spezialisierung:**



**Spezialisierung**  
**Abbildung EER**   
**Relationenmodell**

wird auf die Relationen

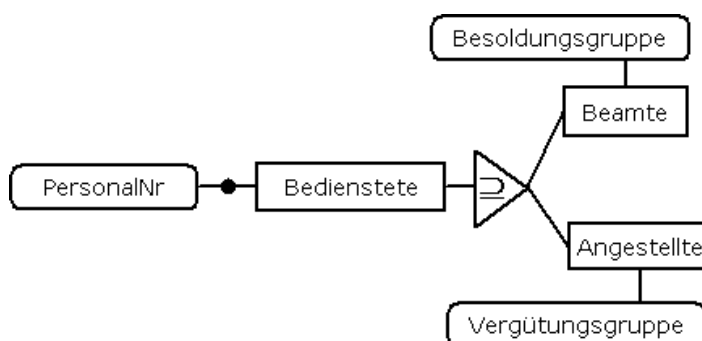
$S1 (\underline{Ex\#} \rightarrow E.Ex\#, \dots)$

$S_n (\underline{Ex\#} \rightarrow E.Ex\#, \dots)$

abgebildet. Dabei stehen  $Ex\#$  für das Schlüsselattribut/die Schlüsselattribute des Objekttyps  $E$  und die Auslassungspunkte für zusätzliche spezielle Attribute der Objekttypen  $S1, \dots, S_n$ .



**Beispiel**



**Beispiel für die Abbildung der Spezialisierung auf das  
Relationenmodell**

wird auf die Relationen

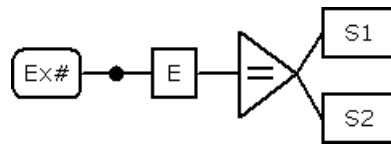
Beamte (PersonalNr → Bedienstete.PersonalNr, Besoldungsgruppe)

Angestellte (PersonalNr → Bedienstete.PersonalNr, Vergütungsgruppe)



abgebildet.

◦ **Partitionierung:**



**Partitionierung  
Abbildung EER**

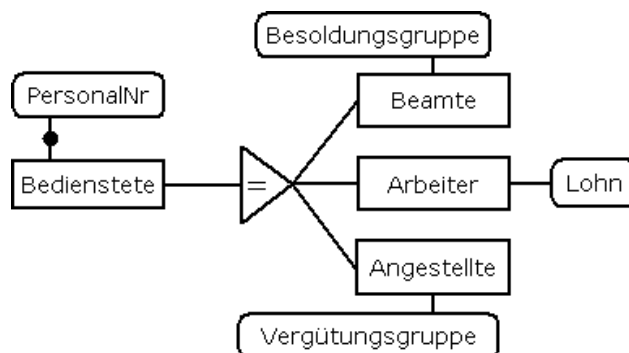


**Relationenmodell**

Die Partitionierung wird genauso wie die Spezialisierung transformiert. Es ist jedoch eine zusätzliche Integritätsbedingung erforderlich, und zwar dass jeder Schlüsselwert in genau einer der Spezialrelationen auftritt.



**Beispiel**



**Beispiel für die Abbildung der Partitionierung  
auf das Relationenmodell**

wird auf die Relationen

Beamte (PersonalNr → Bedienstete.PersonalNr, Besoldungsgruppe)

Angestellte (PersonalNr → Bedienstete.PersonalNr, Vergütungsgruppe)

Arbeiter (PersonalNr → Bedienstete.PersonalNr, Lohn)

abgebildet. Zusätzlich gilt die Integritätsbedingung, dass dieselbe PersonalNr nicht *gleichzeitig* in *zwei* der Relationen "Beamte", "Angestellte" und "Arbeiter" auftreten darf. Dies ist bei der Implementierung durch entsprechende Maßnahmen (z.B. Trigger) sicherzustellen.

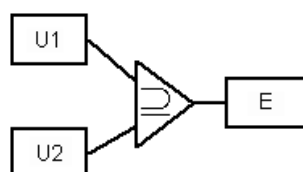
- **Generalisierung:**

Für die Transformation der Generalisierung gibt es mehrere Möglichkeiten. Bei der hier angegebenen Transformationsmöglichkeit wird die Oberklasse einer Generalisierung durch eine Relation dargestellt. In dieser Relation sind die Attribute der Unterklassen enthalten, einschließlich der gemeinsamen Attribute und der Schlüssel. Zusätzlich werden boolesche Attribute benötigt, und zwar eines für jede Unterklasse. Diese Attribute geben an, zu welcher Unterklasse ein Objekt jeweils gehört. Dies führt dazu, dass einige Integritätsbedingungen eingeführt werden müssen:

- Die booleschen Attribute dürfen in jedem Tupel nur in einem Fall auf *true* gesetzt sein, in allen anderen Fällen müssen sie *false* sein.
- In Abhängigkeit von diesen Attributen dürfen die nicht gemeinsamen Attribute mit einem Wert belegt oder müssen *null* sein.

Der Vorteil dieser Art von Transformation ist, dass die Möglichkeit gegeben ist, auf eine Relation zuzugreifen, die alle Exemplare der Generalisierungs-Oberklasse enthält. Einen Nachteil gibt es allerdings auch, und zwar dann, wenn es mehrere/zu viele Unterklassen gibt. Denn die Anzahl der booleschen Attribute erhöht sich immer und dadurch wird (unnötigerweise) Speicherplatz verbraucht.

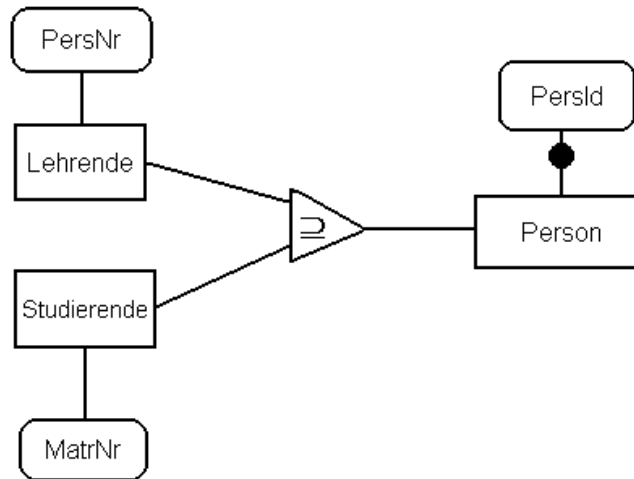
Alternative kann die Verwendung eines weiteren Datentyps, der eine Markierung enthält, die angibt, zu welcher Untermenge dieses gehört. Damit ist es auch möglich, diese Markierung zusammen mit dem ursprüngliche Primärschlüssel zu dem neuen zusammen-gesetzten Primärschlüssel zu machen.



**Abbildung der  
Generalisierung  
auf das  
Relationsmodell**



**Beispiel**



**Abbildung der Generalisierung auf das Relationenmodell als Beispiel**

wird auf die Relationen

Person (PersNr → Lehrende.PersNr, MatrNr → Studierende.MatrNr, PersId)

abgebildet.

Nach Anwendung der vorgestellten Transformationsschritte entsteht eine Sammlung von Relationenschemata, die in Verbindung mit den formulierten Integritätsbedingungen den gleichen Weltausschnitt modelliert wie das erweiterte ER-Diagramm. Diese können anschließend meist direkt in eine relationale Datenbank umgesetzt werden, es sei denn, sie müssen normalisiert werden, um Anomalien zu verhindern.



Darf ich die vorgestellten Transformationen als Patentrezept einsetzen?

Ja, das ER- und EER-Modell können nach dem angegebenen Schema in das Relationenmodell transformiert werden.

## 5.2 Normalisierung

Schlecht entworfene Relationenschemata können zu sogenannten Anomalien führen. Anhand eines Beispiels wird gezeigt, welche Anomalien auftreten können.

Die Beispielrelation ist: Lehrende\_LV (PersNr, Name, Raum, LV\_Nr, LV\_Bez, SWS)

wobei die Attribute *PersNr*, *Name* und *Raum* die Personalnummer, den Namen bzw. das Büro der Lehrenden und *LV\_Nr*, *LV\_Bez* und *SWS* die Nummer, die Bezeichnung und die Anzahl Semesterwochenstunden der Lehrveranstaltung angeben.

Im Datenbankbereich finden wir drei Arten von Anomalien:

1. **Änderungsanomalien:** Durch Änderung einer Information werden andere Informationen redundant in der Datenbank vorkommen.

In der Beispielrelation soll eine Änderung vorgenommen werden: Eine Lehrende bekommt einen weiteren Raum zugeordnet, weil ihr eine zusätzliche Funktion innerhalb ihrer Hochschule übertragen wird (z.B. als neue Prüfungsausschussvorsitzende).

Zwei Nachteile können dabei auftreten:

- Erhöhter Speicherbedarf, da für die einzustellenden Daten ein weiteres Tupel angelegt werden muss, in dem die Attribute *Name*, *LV\_Nr*, *LV\_Bez*, *SWS* redundant enthalten sind.
- Leistungseinbußen bei Änderungen, da mehrere Einträge geändert werden müssen.

2. **Einfügeanomalien:** Das Einfügen einer Information kann nicht erfolgen, weil eine andere Information noch nicht verfügbar ist.

In die Beispielrelation soll eine Lehrveranstaltung eingetragen werden, für die jedoch noch keine vortragende Person bestimmt wurde. *PersNr* ist aber das Schlüsselattribut der Relation.

3. **Löschanomalien:** Durch Löschen einer Information werden andere Informationen unerwünschterweise aus der Datenbank entfernt.

Beim Löschen einer Lehrveranstaltung in der obengenannten Beispielrelation, die von einer einzigen Person abgehalten wird, geht der Inhalt der gesamten Tupel verloren, somit auch die gesamten Informationen über die betreffende Person. Wenn mehrere Lehrveranstaltungen von einer Person abgehalten werden, wird es kein Problem geben.

**Normalisierung** bedeutet, Relationenschema aufzuspalten mit den Zielen, Redundanzen zu verringern und die obengenannten Anomalien zu verhindern.

### Datenabhängigkeiten

Die Datenabhängigkeiten erlauben Aussagen darüber, welche Relationen aus einer Menge aller Möglichkeiten als "sinnvoll" anzusehen sind und welche nicht. Sie können in zwei Klassen unterteilt werden, und zwar intrarelationale und interrelationale Datenabhängigkeiten.

a. **Intrarelationale Datenabhängigkeiten** beziehen sich ausschließlich auf eine Attributmenge. Dazu zählen:

1. funktionale Abhängigkeiten

Eine **funktionale Abhängigkeit** gilt dann innerhalb einer Relation zwischen Attributmengen  $E1$  und  $E2$ , wenn in jedem Tupel der Relation aus der Gleichheit der Werte in den Attributen der Menge  $E1$  auf die Gleichheit der Werte in den Attributen der Menge  $E2$  geschlossen werden kann. Die Notation ist  $E1 \rightarrow E2$  ( $E2$  ist von  $E1$  funktional abhängig).



Beispiel

$ISBN \rightarrow \{\text{Titel}, \text{Verlag}\}$ , d.h. zwei Bücher-Tupel, die in der ISBN übereinstimmen, müssen auch im Titel und Verlag übereinstimmen

$\{\text{Ort}, \text{Strasse}, \text{Nr}\} \rightarrow \text{PLZ}$ , d.h. zwei Tupel, die im Ort, in der Strasse und in der Nr übereinstimmen, müssen auch die gleiche PLZ haben.

2. mehrwertige Abhängigkeiten

Die **mehrwertige Abhängigkeit** beschreibt die Zuordnung einer Menge von Attributwerten zu einem anderen Attributwert. Die Notation ist  $E1 \twoheadrightarrow E2$ .



Beispiel

Lehrende können auch mehrere Fächer unterrichten, d.h. es gilt die mehrwertige Abhängigkeit:  $\text{Lehrende} \twoheadrightarrow \text{Fach}$

Mehrwertige Abhängigkeiten verursachen Redundanzen!

b. **Interrelationale Datenabhängigkeiten** erlauben Aussagen darüber, wann eine Datenbank, als Ganzes betrachtet, als "sinnvoll" anzusehen ist, und sind auch als **Fremdschlüssel** bekannt.



Beispiel

Wenn wir die Relation besuchen aus unserer Miniwelt Hochschule betrachten, haben wir zwei Fremdschlüssel, nämlich *MatrNr* und *LVNr*, denn sie dürfen in der Relation besuchen nur dann auftreten, wenn *MatrNr* in der Relation *Studierende* und *LVNr* in der Relation *LV* bereits vorhanden sind.

### Normalformen



Beispiel

Gegeben sei die Relation

Vertrieb (PersNr, Name, Vorname, PLZ, Einsatzort, ArtNr, ArtName, Typ, Umsatz)

mit der folgenden Belegung:

PersNr	Name	Vorname	PLZ	Einsatzort	ArtNr	Art Name	Typ	Umsatz
1508	Benz	Jens	38100	Braunschweig	174, 219	Fernseher	JVC AV-32H20, Sony KV-29 LS35E	1800,900
1147	Hase	Paul	28159	Bremen	538	Herd	Siemens HN17023	350

"PersNr" ist die Personalnummer, "ArtNr" ist die Artikelnummer, "ArtName" ist die Artikelbezeichnung und "Typ" ist das jeweilige Modell. Eine Person wird nur an einem Ort eingesetzt.

Anhand dieser (nicht normalisierten) Relation sollen die ersten drei Normalformen erläutert werden.

#### 1. *Erste Normalform* (1NF)

Die 1NF erlaubt nur *atomare Attribute* in den Relationenschemata, d.h. als Attributwerte sind Elemente von Standard-Datentypen wie *INTEGER* oder *STRING* erlaubt, aber kein *ARRAY* oder *SET*. Dies bedeutet, dass mehrfache Redundanzen auftreten können.

Die Attribute "ArtNr", "Typ" und "Umsatz" in der Beispiel-Relation enthalten mehrere Werte, was nicht der ersten Normalform entspricht. Damit die Relation der ersten Normalform entspricht, müssen die Attributwerte auf mehrere Tupel verteilt werden. Das sieht dann so aus:

<u>PersNr</u>	Name	Vorname	PLZ	Einsatzort	<u>ArtNr</u>	Art Name	Typ	Umsatz
1508	Benz	Jens	38100	Braun-schweig	174	Fern-seher	JVC AV-32H20	1800
1508	Benz	Jens	38100	Braun-schweig	219	Fern-seher	Sony KV-29 LS35E	900
1147	Hase	Paul	28159	Bremen	538	Herd	Siemens HN17023	350

Dabei sind "PersNr" und "ArtNr" Schlüsselattribute, und die Relation befindet sich nun in der ersten Normalform.

Die obige Relation ist problematisch in der Pflege, denn es können z.B. nur dann Personen aufgenommen werden, wenn es auch dazugehörige Artikelnummern gibt, und das Löschen eines Artikels kann bewirken, dass die gesamte Information über eine Person verloren geht.

## 2. Zweite Normalform (2NF)

Eine Relation ist in der zweiten Normalform, wenn sie in der ersten Normalform ist und darüber hinaus alle Nichtschlüsselattribute *voll funktional* von den Schlüsselattributen abhängen.

**Voll funktional** heißt: ein Attribut benötigt den gesamten Schlüssel zur Identifikation und kann nicht bereits durch einen Teil des Schlüssels identifiziert werden.

Für den Fall, dass es bereits durch einen Teil der Schlüsselattribute identifiziert werden kann, liegt eine **partielle Abhängigkeit** vor.

Falls ein Attribut durch ein Nichtschlüsselattribut identifiziert wird und das Nichtschlüsselattribut wiederum durch ein Schlüsselattribut identifiziert wird, spricht man von **transitiver Abhängigkeit**.



## Beispiel

In der obigen Relation (erste Normalform) haben wir folgende funktionalen Abhängigkeiten:

$\text{PersNr} \rightarrow \text{Name}$ ,  
 $\text{PersNr} \rightarrow \text{Vorname}$ ,  
 $\text{PersNr} \rightarrow \text{PLZ}$ ,  
 $\text{PersNr} \rightarrow \text{Einsatzort}$ ,  
 $\text{ArtNr} \rightarrow \text{ArtName}$ ,  
 $\text{ArtNr} \rightarrow \text{Typ}$ ,  
 $\{\text{PersNr}, \text{ArtNr}\} \rightarrow \text{Umsatz}$ .

Alternativ können die funktionalen Abhängigkeiten auch folgendermaßen angegeben werden:

$\text{PersNr} \rightarrow \{\text{Name}, \text{Vorname}, \text{PLZ}, \text{Einsatzort}\}$ 
 $\text{ArtNr} \rightarrow \{\text{ArtName}, \text{Typ}\}$ ,
  $\{\text{PersNr}, \text{ArtNr}\} \rightarrow \text{Umsatz}$ .

D.h., die Attribute "Name", "Vorname", "PLZ", "Einsatzort", "ArtNr", "ArtName", "Typ" und "Umsatz" hängen jeweils nur von einem Teil des Schlüssels ab, da der Schlüssel aus den beiden Attributen "PersNr" und "ArtNr" besteht. Durch Zerlegen der Relation in Teilrelationen wird erreicht, dass alle Attribute voll funktional von allen Schlüsselattributen abhängen. Das Ergebnis sind die folgenden drei Relationen, "Personal", "Produkte" und "Umsatz":

<u>PersNr</u>	Name	Vorname	PLZ	Einsatzort
1508	Benz	Jens	38100	Braunschweig
1147	Hase	Paul	28159	Bremen

<u>ArtNr</u>	ArtName	Typ
174	Fernseher	JVC AV-32H20
219	Fernseher	Sony KV-29 LS35E
538	Herd	Siemens HN17023

<u>PersNr</u>	ArtNr	Umsatz
1508	174	1800
1508	219	900
1147	538	280



### 3. Dritte Normalform (3NF)

Eine Relation ist in der dritten Normalform, wenn sie in der zweiten Normalform ist und keine transitiv abhängigen Nichtschlüsselattribute existieren.



Beispiel

Im obigen Beispiel (zweite Normalform) gibt es die transitive Abhängigkeit PersNr → Einsatzort, denn "Einsatzort" wird nicht nur direkt durch "PersNr" bestimmt, sondern auch indirekt über "PLZ". Diese transitive Abhängigkeit wird durch Aufspaltung der Relation "Personal" beseitigt. Die neuen Relationen sind also:

<u>PersNr</u>	Name	Vorname	PLZ
1508	Benz	Jens	38100
1147	Hase	Paul	28159

<u>PLZ</u>	Einsatzort
38100	Braunschweig
28159	Bremen

<u>ArtNr</u>	ArtName	Typ
174	Fernseher	JVC AV-32H20
219	Fernseher	Sony KV-29 LS35E
538	Herd	Siemens HN17023

<u>PersNr</u>	<u>ArtNr</u>	Umsatz
1508	174	1800
1508	219	900
1147	538	280

### 4. Boyce-Codd'sche Normalform (BCNF)

Eine Relation ist in der BCNF, wenn jede funktionale Abhängigkeit von Attributen A → B zur Folge hat, dass A den Schlüssel enthält.



Beispiel

Wettkampfgeschehen (Spieler, Mannschaftsnamen, Kapitän)

Wettkampzugehörigkeit (Spieler, Mannschaftsnamen)      Wettkampfkapitän (Mannschaftsnamen, Kapitän)



Boyce-Codd'sche Normalform (BCNF)

Löschanomalie: Wenn der letzte Spieler eine Mannschaft verlässt, gehen auch die Informationen über den Kapitän der Mannschaft verloren.

### 5. Vierte Normalform (4NF)

Die vierte Normalform stellt eine Verschärfung der BCNF dar. Eine Relation ist in der vierten Normalform, wenn sie höchstens eine mehrwertigen Abhängigkeit beinhaltet. Eine mehrwertige Abhängigkeit beschreibt die Zuordnung einer Menge von Attributwerten zu genau einem anderen Attributwert.



Beispiel

Angenommen, es gibt die Relation

Privates (PersNr, Hobbies, Fahrzeuge),

in der zu jeder Person die Hobbies und die Art der Fahrzeuge (Auto, Motorrad, Fahrrad) gespeichert wird. Die Relation hat die mehrwertigen Abhängigkeiten

$\text{PersNr} \twoheadrightarrow \text{Hobbies}$  und

$\text{PersNr} \twoheadrightarrow \text{Fahrzeuge}$ .

Wenn nun die Relation in die beiden Relationen

Hobbies (PersNr, Hobbies) und

Fahrzeuge (<PersNr, Fahrzeuge)

aufgespalten wird, dann befindet sie sich in der vierten Normalform

Des weiteren gibt es noch die fünfte Normalform. Für die meisten Entwurfsverfahren kommt man jedoch mit der dritten Normalform aus.



## Übung

Zur nachfolgenden Relation sollen die zweite und dritte NF angegeben werden:

<u>PersNr</u>	Name	Titel	Gehalt	<u>ProjNr</u>	ProjBez	Stunden	FB Kürzel	Fachbereich
001	Maier	Dipl.- Inform.	4000	4711	Proj. 11	20	Fb I	Informatik
001	Maier	Dipl.- Inform.	4000	4712	Proj. 12	30	Fb I	Informatik
001	Maier	Dipl.- Inform.	4000	4718	Proj. 18	50	Fb I	Informatik
002	Schulz	Dr.rer.nat.	5800	4712	Proj. 12	60	Fb I	Informatik
002	Schulz	Dr.rer.nat.	5800	4718	Proj. 18	40	Fb I	Informatik
003	Müller	Dr.-Ing.	5800	4711	Proj. 11	50	Fb E	Elektrotechnik
003	Müller	Dr.-Ing.	5800	4732	Proj. 32	50	Fb E	Elektrotechnik

1NF → 2NF

Die Relation

R: (PersNr, Name, Titel, Gehalt, ProjNr, ProjBez, Stunden, FB-Kürzel, Fachbereich)

ist in der ersten Normalform und sie wird in die drei Relationen

R1: (PersNr, Name, Titel, Gehalt, FB-Kürzel, Fachbereich)

R2: (PersNr, ProjNr, Stunden)

R3: (ProjNr, ProjBez)

aufgespalten, um der zweiten Normalform zu entsprechen.

2NF → 3NF

Die Relationen R2 und R3 entsprechen bereits der dritten Normalform, die Relation R1 muss allerdings erneut aufgespalten werden, und zwar wie folgt:

R1\_1: (PersNr, Name, Titel, Gehalt, FB-Kürzel)

R1\_2: (FB-Kürzel, Fachbereich)

Es bestehen folgende funktionalen Abhängigkeiten:

FA1: PersNr → {Name, Gehalt, Titel, FB-Kürzel, Fachbereich}

FA2: ProjNr → ProjBez

FA3: {PersNr, ProjNr}  $\rightarrow$  Stunden

FA4: FB-Kürzel  $\rightarrow$  Fachbereich



Wenn ich das EER-Modell in meiner Vorgehensweise beim Datenbankentwurf einsetze, in welcher Normalform befinden sich die Relationen nach der Transformation?

In der Regel befinden sich die Relationen in der dritten Normalform.

## 5.3 Relationenalgebra

Eine algebraische Struktur (Algebra) ist eine Menge  $M$  zusammen mit einer oder mehreren Operationen auf  $M$ . Dabei bedeutet ( $n$ -stellige) Operation auf  $M$  eine Abbildung von  $M^n$  in  $M$ .

Bei der Relationenalgebra, die von Codd eingeführt wurde, sind die Mengen Relationen mit den folgenden Operationen:

1. **Selektion**, in Zeichen  $\sigma$

Die Syntax der Selektion ist:



Formel

$\sigma[\text{Bedingung}](\text{Relation})$

Bei der Selektion werden die Zeilen der Relation ausgesucht, die der angegebenen Bedingung genügen. Die Bedingung kann eine Konstanten-Selektion, eine Attribut-Selektion oder eine logische Verknüpfung mehrerer Konstanten- oder Attribut-Selektionen mit  $\vee$ ,  $\wedge$ ,  $\neg$  sein.

- Die Konstanten-Selektion hat die Form



Formel

Attribut  $\theta$  Konstante

Dies bedeutet, dass für jedes Tupel der Relation der Wert eines Attributwertes mit der vorgegebenen Konstante verglichen wird. Mögliche Vergleichssymbole sind: =, <>, <=, <, >=, >.



Beispiel

Die Konstanten-Selektion "alle Studierende, die Praktische Informatik studieren", in Zeichen

$\sigma[\text{Studiengang}='PI'](\text{Studierende})$

hat als Ergebnisrelation:

MatrNr	Name	Vorname	GebDat	Ge- schlecht	Ur- laubs- Sem	Semester	Stu- dien- gang
2358712	Meier	Siegfried	07.10.1985	m	1	5	PI
6432753	König	Mathilde	07.10.1985	w	2	12	PI
5236478	Hesse	Sarah	07.10.1985	w	0	2	PI
9812964	Meier	Hans	05.12.1985	m	0	1	PI
9365461	Schmitt	Marc	27.08.1981	m	2	11	PI
3108642	Meier	Martina	18.11.1983	w	2	12	PI

- Die Attribut-Selektion hat die Form



Formel

Attribut1  $\theta$  Attribut2

Dies bedeutet, dass für jedes Tupel zwei Attributwerte verglichen werden, die gleiche oder kompatible Wertebereiche besitzen müssen.



Beispiel

Die Attribut-Selektion "alle Studierende, deren Anzahl der Urlaubssemester größer ist als die Anzahl der Semester", in Zeichen

$$\sigma[\text{Urlaubssemester} > \text{Semester}](\text{Studierende})$$

2. **Projektion**, in Zeichen  $\pi$

Die Syntax der Projektion ist:



Formel

$$\pi[\text{Attributmenge}](\text{Relation})$$

Bei der Projektion werden Spalten der Relation ausgeblendet, bzw. es werden nur die Attribute eingeblendet, die in der Attributmenge angegeben sind. Dabei werden Tupel, die mehrfach vorkommen, nur einmal ausgegeben.



Beispiel

Die Projektion "Name und Semesterzahl aller Studierenden", in Zeichen

$$\pi[\text{Name, Semester}](\text{Studierende})$$

hat als Ergebnisrelation:

Name	Semester
Meier	5
Schulze	10
König	12
Baum	4
Dreier	8
Hesse	2
Meier	1
Müller	5
Schmitt	11
Müller	6
Meier	12
Thiess	6

3. **Verbund**, in Zeichen  $\bowtie$ 

Die Syntax des Verbundes ist:



Formel

Relation1  $\bowtie$  Relation2

Dies bedeutet, dass Tupel zweier Relationen über gleichbenannte Attribute bei gleichen Werten aneinandergehängt werden. Tupel, die keinen Partner finden, werden nicht berücksichtigt.



Beispiel

Der Verbund "Bezeichnungen der Lehrveranstaltungen, von denen auch die Lehrenden bekannt sind, die sie halten", in Zeichen

$\pi$  [LVBezeichnung] (Lehrveranstaltung  $\bowtie$  halten)

hat als Ergebnisrelation:

LVBezeichnung
Mathematik I
Mathematik II
Physik
BWL
Informatik
Mediendesign
Labor Informatik
Labor Datenbanken
Datenbanken
Datenbanksysteme

Der Verbund ist kommutativ, d.h.

Relation1  $\bowtie$  Relation2 = Relation2  $\bowtie$  Relation1

und assoziativ, d.h.

(Relation1  $\bowtie$  Relation2)  $\bowtie$  Relation3 =

$\text{Relation1} \bowtie (\text{Relation2} \bowtie \text{Relation3})$ .

4. Mengenoperationen. Dazu gehören:

- **Vereinigung**, in Zeichen  $\cup$

Die Syntax der Vereinigung ist:



Formel

$\text{Relation1} \cup \text{Relation2}$

Bei der Vereinigung werden alle Tupel aus den beiden beteiligten Relationen in die Ergebnisrelation übernommen, wobei doppelte Tupel eliminiert werden.



Beispiel

Die Vereinigung "alle Studierende und Lehrende der Hochschule", in Zeichen  $\pi[\text{Name}, \text{Vorname}](\text{Studierende}) \cup \pi[\text{Name}, \text{Vorname}](\text{Lehrende})$  hat als Ergebnisrelation:

Name	Vorname
Meier	Siegfried
Schulze	Heiner
König	Mathilde
Baum	Meta
Dreier	Magnus
Hesse	Sarah
Meier	Hans
Müller	Karla
Schmitt	Marc
Müller	Hans
Müller	Udo
Meier	Martina
Thiess	Hugo
Zimmer	Monika



Irrgang	Rolf
Meier	Martina
Thein	Siegfried
Schmidt	Manfred
Schulze	Hans
Maier	Franziska
Müller	August
Sommer	Michaela

- **Differenz**, in Zeichen -

Die Syntax der Differenz ist:



Formel

Relation1 - Relation2

Bei der Differenz werden in die Ergebnisrelation solche Tupel übernommen, die nur in der ersten, nicht aber in der zweiten Relation vorkommen.



Beispiel

Die Differenz "alle Studierende der Hochschule, die nicht den gleichen Vor- und Nachnamen wie Lehrende haben", in Zeichen  $\pi[\text{Name, Vorname}](\text{Studierende}) - \pi[\text{Name, Vorname}](\text{Lehrende})$  hat als Ergebnisrelation:

Name	Vorname
Meier	Siegfried
Schulze	Heiner
König	Mathilde
Baum	Meta
Dreier	Magnus
Hesse	Sarah

Meier	Hans
Müller	Karla
Schmitt	Marc
Thiess	Hugo

- **Durchschnitt**, in Zeichen  $\cap$

Die Syntax des Durchschnitts ist:



Formel

$\text{Relation1} \cap \text{Relation2}$

Beim Durchschnitt werden in die Ergebnisrelation solche Tupel übernommen, die in beiden Relationen vorkommen.



Beispiel

Der Durchschnitt "alle Studierende und Lehrende der Hochschule, die den gleichen Vor- und Nachnamen haben (z.B. Hans Schmidt)", in Zeichen  $\pi[\text{Name, Vorname}](\text{Studierende}) \cap \pi[\text{Name, Vorname}](\text{Lehrende})$  hat als Ergebnisrelation:

Name	Vorname
Müller	Hans
Müller	Udo
Meier	Martina

5. **Umbenennung**, in Zeichen  $\beta$

Die Syntax der Umbenennung ist:



Formel

$\beta[\text{neu} \leftarrow \text{alt}](\text{Relation})$

Bei der Umbenennung werden Attribute umbenannt, um die Relationen für Operationen wie Verbund, Vereinigung, Differenz zueinander kompatibel zu machen.



Beispiel

*MatrNr* und *PersNr* werden in *ID* umbenannt, um die Relationen Studierende und Lehrende über das entsprechende Attribut vereinigen zu können.  $\beta[ID \leftarrow MatrNr](Studierende) \cup \beta[ID \leftarrow PersNr](Lehrende)$  hat als Ergebnisrelation:

ID
2358712
1562367
6432753
7564258
2356984
5236478
9812964
9652425
9365461
7654321
4297531
3108642
1230789
5234260
9652425
1234567
1357924
2468013
1133556
9870321

1523345

2314856

3495067

4526748

## 5.4 Freitextaufgaben zu Relationale Datenbanken

1. Mit welchen Normalformen kommt man in der Praxis aus?

**Lösung zeigen**

In der Regel benötigt man nur die 3. Normalform.

2. Warum werden Relationenschemata normalisiert?

**Lösung zeigen**

Relationenschemata werden normalisiert, um Redundanzen zu verringern und Änderungs-, Einfüge- und Löschanomalien zu verhindern.

3. Gegeben sei eine Relation für eine Lieferungsdatenbank mit den Attributen Lieferant (Schlüsselattribut), Teil (Schlüsselattribut), Anzahl, Ort und Entfernung. In der 1. Normalform sieht die Relation wie folgt aus:

<u>Lieferant</u>	<u>Teil</u>	Anzahl	Ort	Entfernung
L1	T1	300	London	600
L1	T2	200	London	600
L1	T3	400	London	600
L1	T4	200	London	600
L1	T5	100	London	600
L1	T6	100	London	600
L2	T1	300	Paris	450
L2	T2	400	Paris	450
L3	T2	200	Paris	450

L4	T2	200	Brüssel	150
L4	T4	300	Brüssel	150
L4	T5	400	Brüssel	150

Erstellen Sie hierfür die zweite und die dritte Normalform.

**Lösung zeigen**

Es bestehen die funktionalen Abhängigkeiten (FAs): FA1: {Lieferant, Teil} → Anzahl

FA2: Lieferant → Ort

FA3: Ort → Entfernung

zweite NF:	Lieferung (Lieferant, Teil, Anzahl)
	Lieferstelle (Lieferant, Ort, Entfernung)
dritte NF:	Lieferung (Lieferant, Teil, Anzahl)
	Lieferstelle (Lieferant, Ort)
	Distanz (Ort, Entfernung)

## 6 Structured Query Language (SQL)



### Zeitumfang

Die voraussichtliche Bearbeitungsdauer dieser Lerneinheit (ohne Übungsaufgaben!) beträgt ca. 20 Stunden.



### Lernziele

SQL-"Structured Query Language" ("strukturierte Abfragesprache") ist die Standardabfragesprache für relationale Datenbanken. Mit ihr werden Relationen (Tabellen) erzeugt oder geändert, Daten abgefragt, hinzugefügt, geändert oder gelöscht.

Mitte der siebziger Jahre entwickelte IBM das erste relationale Datenbanksystem (System R) und die Datenbanksprache SEQUEL als Vorläufer von SQL. Im Verlauf der weiteren Entwicklungen kam der Bedarf nach einer einheitlichen Sprache auf. 1986/87 wurde SQL als einheitlicher Standard vom ANSI und der ISO verabschiedet. Der gebräuchlichste Standard, SQL2 bzw. SQL92 wurde 1992 festgelegt. 1999 wurde der Standard SQL3 verabschiedet, der Objektorientiertheit berücksichtigt. SQL:2003 beinhaltet u.a. SQL/XML und SQL/MED. Der aktuelle Standard ist SQL:2006 aus dem Jahre 2006 und legt die Benutzung von SQL in Verbindung mit XML fest.



Lässt sich SQL auf jedem beliebigen Datenbankmanagementsystem anwenden?

SQL wird zwar von nahezu allen Herstellern relationaler Datenbankmanagementsysteme (RDBMS) wie Oracle, Sybase, Microsoft SQL Server, Access, MySQL oder Ingres angeboten, die Implementierungen enthalten jedoch herstellerspezifische Änderungen bzw. Erweiterungen. Aus diesem Grund ist es unbedingt notwendig, diese herstellerspezifischen Abweichungen der SQL-Syntax im jeweiligen RDBMS-Handbuch nachzulesen.

[Liste aller SQL-Anweisungsgruppen](#)

**Gliederung**

- 6 Structured Query Language (SQL)
- 6.1 Datenbankdefinition
- 6.2 Datenbankanfragen
- 6.3 Datenbankänderungen
- 6.4 Weiterführende Informationen
- 6.5 Freitextaufgaben zu Structured Query Language

## 6.1 Datenbankdefinition

Im Folgenden wird erläutert, wie eine Datenbank eingerichtet und eine Relation (Tabelle) erzeugt wird. Zu einer vollständigen Relation gehören neben den Attributen (Spalten) auch die Festlegung des Wertebereichs der einzelnen Attribute, Primär- und Fremdschlüssel und eventuell Indizes.

**Gliederung**

- 6.1 Datenbankdefinition
- 6.1.1 Erzeugen einer Datenbank
- 6.1.2 Relationen erstellen
- 6.1.3 Datentypen und Domänen
- 6.1.4 Erweiterte Form der CREATE TABLE-Anweisung
- 6.1.5 Primärschlüsseldefinition
- 6.1.6 Fremdschlüsseldefinition
- 6.1.7 Eindeutigkeitspezifikation
- 6.1.8 CHECK-Spezifikation
- 6.1.9 Indizes
- 6.1.10 Relationen ändern
- 6.1.11 Relationen löschen
- 6.1.12 COMMIT WORK/ROLLBACK WORK

### 6.1.1 Erzeugen einer Datenbank

Einige DB-Systeme verwalten mehrere Datenbanken, deshalb besteht der erste Schritt im Anlegen einer Datenbank. Hierzu wird in SQL der Befehl CREATE SCHEMA benutzt. Viele RDBMS benutzen jedoch den Befehl CREATE DATABASE.

**Beispiel**

Es soll eine Datenbank mit dem Namen "Bibliothek" angelegt werden.

SQL-Anweisung:



```
CREATE DATABASE Bibliothek;
```



Jede SQL-Anweisung endet mit einem Semikolon. Die SQL-Anweisungen werden in dieser Lerneinheit lediglich wegen der besseren Lesbarkeit groß geschrieben. Generell wird in SQL aber nicht zwischen Groß- und Kleinschreibung unterschieden. Umlaute sollten vermieden werden.

## 6.1.2 Relationen erstellen

### (Einfache Form der CREATE TABLE - Anweisung)

Nach dem Anlegen der Datenbank werden die Relationen des Datenbankentwurfs durch entsprechende Tabellen in der Datenbank angelegt. Zum Erzeugen einer solchen Relation oder Tabelle mit SQL wird die Anweisung CREATE TABLE verwendet. Dieser Anweisung folgt der Name der Relation, der innerhalb der Datenbank nur einmal vorkommen darf. Dem Namen der Relation folgen ein oder mehrere Attributdefinitionen (Bestimmung der Spalten).

Eine Attributdefinition beginnt mit dem Namen des Attributs (der Spalte), der innerhalb der Relation nur einmal existieren darf. Zu den einzelnen Attributen wird der Datentyp (Wertebereich) festgelegt und eventuell bereits ein voreingestellter Wert zugewiesen.

**SYNTAX**

Syntax einer CREATE TABLE - Anweisung:





```
CREATE TABLE [Schema-Name.]Relationenname
  (Attributdefinition ,
  Attributdefinition,...);
```



Hinweise zur Notation der Syntax:

rot:	Schlüsselwort
schwarz:	wählbarer Name
[ ]:	optional



### Schema-Name

Beim Anlegen einer Relation wird ein Schema-Name zur Identifizierung des Erstellers im Systemkatalog hinterlegt. In der Regel entspricht der Schema-Name der Benutzerkennung. Wenn ein Datenbankadministrator für einen anderen Teilnehmer eine Relation definieren will, aber auch wenn ein Benutzer auf Relationen anderer Benutzer zugreifen möchte, muss er dem Relationennamen den Schema-Namen mit einem Punkt voransetzen. Auch bei nachfolgenden SQL-Syntax-Angaben gilt, dass der Schema-Name nur dann angegeben werden muss, wenn es sich bei den Relationen um "fremde" (also nicht eigene) Relationen handelt.

### Attributdefinition

Die Attribute (Spalten) der Relation (Tabelle) werden folgendermaßen definiert:



SYNTAX

Die Attribute (Spalten) der Relation (Tabelle) werden folgendermaßen definiert:



```
Attributname Datentyp [Default-Wert] [NOT NULL]
```

Der Datentyp beschreibt, welche Werte das Attribut annehmen kann, z.B. INTEGER oder VARCHAR. Der Default-Wert ist der voreingestellte Wert eines Attributs, z.B. "30". Standardmäßig ist der Wert NULL voreingestellt. Soll sichergestellt sein, dass alle Attribute auch Werte haben, kann NOT NULL angegeben werden.

Der Wert NULL bedeutet, dass an dieser Stelle der Wert für das Attribut fehlt. NULL ist **nicht** identisch mit dem numerischen Wert 0 oder dem Leerzeichen. Der numerische Wert 0 oder ein Leerzeichen werden in Form eines Binärwertes in der Zelle gespeichert. Bei NULL wird nichts gespeichert.

**Beispiel**

Es soll eine Relation "Buch" mit den Attributen "ISBN", "Titel", "Autor", "Exemplare", "Seitenanzahl" und "Leihfrist" angelegt werden. Die Attribute "ISBN", "Exemplare", "Seitenanzahl" und "Leihfrist" dürfen nicht leer sein. Für das Attribut "Leihfrist" soll standardmäßig der Wert 30 voreingestellt sein.

SQL-Anweisung:



```
CREATE TABLE Buch
(
  ISBN VARCHAR(20) NOT NULL,
  Titel VARCHAR(100),
  Exemplare NUMBER(3,0) NOT NULL,
  Seitenanzahl NUMBER(4,0) NOT NULL,
  Leihfrist NUMBER(5,0) DEFAULT 30);
```

**Übung****Übung sql.1**

Sie möchten Ihre CD-Sammlung katalogisieren. Erstellen Sie zu diesem Zweck eine Relation "CD\_Bestand\_IhreMatrikelnummer" (z.B. CD\_Bestand\_45873) mit den Attributen "CD\_Nr", "CD\_Titel", "CD\_Laenge" und "Erscheinungsjahr". Die Relation kann durch den Befehl



```
select * from CD_Bestand_IhreMatrikelnummer;
```

angezeigt werden.

**SQL Interpreter**[SQL Interpreter](#) **Lösung zeigen**

SQL-Anweisung:



```
CREATE TABLE CD_Bestand_45873
  (CD_Nr VARCHAR2(20) NOT NULL,
   CD_Titel VARCHAR2(100),
   CD_Laenge NUMBER(3,0) NOT NULL,
   Erscheinungsjahr NUMBER (2,0) NOT NULL);
```

**6.1.3 Datentypen und Domänen**

Datentyp	Erläuterung
INTEGER	Ganze Zahlen mit Vorzeichen; Umfang abhängig von der Implementierung
DECIMAL	Dezimalzahlen; es kann angegeben werden wie viele Vor- und Nachkommastellen gewünscht sind; Umfang abhängig von der Implementierung
FLOAT	Fließpunktzahlen in Exponentenschreibweise; Umfang abhängig von der Implementierung
CHAR(n)	Zeichenkette in der festen Länge n; der Maximalwert für n ist oft 255
VARCHAR(n)	Zeichenkette mit maximal n Zeichen; der Maximalwert für n ist oft 255
VARCHAR2(n)	Oracle-Implementierung, gleichbedeutend mit VARCHAR(n)

DATE	Datumsangaben mit Jahr, Monat, Tag (JJJJ-MM-TT)
TIME	Zeitangaben mit Stunden, Minuten, Sekunden
NUMBER(n,m)	Zahlen mit maximaler Länge n und m Nachkommastellen



Bei Oracle gibt es keine zwei unterschiedliche Datentypen DATE und TIME, sondern lediglich den Datentyp DATE, der die Uhrzeit gleichzeitig beinhaltet. Die Funktion SYSDATE gibt das aktuelle Datum und die aktuelle Uhrzeit zurück. Wenn mit SYSDATE gerechnet wird (z.B. Addition, Subtraktion) wird das Ergebnis in Tagen zurückgegeben.

Die Funktion TO\_DATE wandelt eine übergebene Zeichenkette in den Datentyp DATE um.



#### Beispiel

Zeige alle Studierende an, die vor 1985 geboren sind. Name, Vorname und MatrNr sind anzuzeigen.

Name	Vorname	MatrNr
Schulze	Heiner	1562367
Baum	Meta	7564258
Dreier	Magnus	2356984
Müller	Karla	9252425
Schmitt	Marc	9365461
Meier	Martina	3108642
Thiess	Hugo	1230789
Zander	Wolfgang	1286385



**Tabelle Studierende**  
aus der Miniwelt Hochschule

SQL-Anweisung:



```
SELECT Name, Vorname, MatrNr
FROM Studierende
WHERE GebDat < TO_DATE('01.01.1985', 'DD.MM.YYYY');
```

Die Funktion MONTHS\_BETWEEN in Oracle gibt die Anzahl der Monate zurück, die zwischen zwei angegebenen Daten liegen. ROUND wird benutzt, um das Ergebnis der Division auf die nächstgrößere/-kleinere ganze Zahl auf-/abzurunden.



Beispiel

Geben Sie das Alter der Studierenden in Jahren aus. Name, Vorname und MatrNr sind mit anzuzeigen. Eine Spaltenbezeichnung "Alter" müsste in Anführungszeichen gesetzt werden, weil das Wort ALTER ein Schlüsselwort in SQL ist.

Alter/Jahr	Name	Vorname	MatrNr
23	Meier	Siegfried	2358712
28	Schulze	Heiner	1562367
23	König	Mathilde	6432753
26	Baum	Meta	7564258
24	Dreier	Magnus	2356984
23	Hesse	Sarah	5236478
22	Meier	Hans	9812964
24	Müller	Karla	9252425
27	Schmitt	Marc	9365461
22	Müller	Hans	7654321
20	Müller	Udo	4297531
25	Meier	Martina	3108642
24	Thiess	Hugo	1230789
29	Zander	Wolfgang	1286385



**Tabelle Studierende**  
aus der Miniwelt Hochschule

SQL\_Anweisung:



```
SELECT ROUND(MONTHS_BETWEEN(SYSDATE,GebDat)/12)
"Alter/Jahr", Name, Vorname, MatrNr
FROM Studierende;
```

In den unterschiedlichen RDBMS sind unterschiedliche Datentypen vorgegeben (s. Handbuch des RDBMS). Die in der Übersichtstabelle genannten sind die gebräuchlichsten Datentypen.

Um den zulässigen Wertebereich eines Datentyps einzuschränken, wird eine sogenannte Domäne festgelegt. Die Domäne setzt sich zusammen aus dem Datentyp und dem definierten Wertebereich. Dies geschieht mit der Anweisung CREATE DOMAIN.



SYNTAX

Syntax der CREATE DOMAIN-Anweisung:



```
CREATE DOMAIN Domänenname [AS] datentyp
[DEFAULT (Literal|CURRENT|USER|SYSTEM USER|NULL)]
[CONSTRAINT Regelname CHECK (Bedingung)];
```

DEFAULT gibt an, welcher Standardwert in die leeren Felder der entsprechenden Spalte eingesetzt wird. Das kann z. B. ein Zeichen (Literal) sein, das aktuelle Datum und die Uhrzeit (CURRENT), die Benutzerkennung des Anwenders (USER, SYSTEM USER) oder ein NULL-Wert. CHECK ermöglicht die Einschränkung des Wertebereichs der definierten Domäne durch eine Bedingung, z.B. "<100". Für die CHECK-Bedingung gelten dieselben Syntaxregeln wie für die WHERE-Bedingung. Der Zusatz CONSTRAINT dient zur Definition eines Namens für die definierte Regel, den das RDBMS intern benutzt. Dieser sogenannte Regelname ist ein eindeutiger Name. Wenn die CONSTRAINT-Klausel benutzt wurde, erhält man später bei eventueller Verletzung einer CHECK-Klausel eine genaue Fehlermeldung, welche CHECK-Klausel verletzt wurde.

**Beispiel**

Der Wertebereich NUMBER für das Attribut "Leihfrist" soll weiter eingeschränkt werden. Dazu wird eine Domäne mit dem Namen "Fristen" definiert, die nur die Werte 15, 30 und 60 annehmen kann. Der Wert 30 ist voreingestellt.

SQL-Anweisung:



```
CREATE DOMAIN Fristen AS NUMBER(2,0)
DEFAULT 30
CHECK (VALUE IN (15, 30, 60));
```

Nach CREATE DOMAIN wird der Name der Domäne angegeben. Nach AS wird der gewünschte Datentyp angegeben. Mit DEFAULT wird der voreinzustellende Wert definiert. Die CHECK-Bedingung steht immer in Klammern. Die Bedingung "VALUE IN (15, 30, 60)" beschränkt die Domäne "Fristen" auf die Werte 15, 30 und 60.

Auf diese Domäne kann jetzt bei der Erstellung der Relationen zurückgegriffen werden. Die Definition für die Relation "Buch" sieht dann wie folgt aus:

SQL-Anweisung:



```
CREATE TABLE Buch
( ISBN VARCHAR(20) NOT NULL,
  Titel VARCHAR(100),
  Exemplare NUMBER(3,0) NOT NULL,
  Seitenanzahl NUMBER(4,0) NOT NULL,
  Leihfrist Fristen);
```



Oracle hat die Anweisung CREATE DOMAIN nicht implementiert. Dies kann umgangen werden, indem bei der Erstellung der Relationen die CHECK-Klausel verwendet wird.

**Übung****Übung sql.2**

Schränken Sie den Wertebereich NUMBER für das Attribut "CD\_Laenge" weiter ein. Definieren Sie dazu einen Wertebereich "Laenge", die nur die Werte 45, 60 und 90 annehmen kann. Der Wert 60 soll voreingestellt sein. Definieren Sie anschließend die Relation "CD\_Bestand\_IhreMatrikelnummer" neu und verwenden Sie dabei den Wertebereich "Laenge".

(Diese Übung kann nicht mit dem SQL-Interpreter getestet werden, da Oracle kein domain implementiert hat. Es kann aber die angesprochene Alternativlösung mit Hilfe der CHECK-Klausel benutzt werden.)

### SQL Interpreter

[SQL Interpreter](#) 

### Lösung zeigen



```
CREATE DOMAIN Laenge AS NUMBER (2,0)
DEFAULT 60
CHECK (VALUE IN (45, 60, 90));
```

**SQL-Anweisung 1:**



```
CREATE TABLE CD_Bestand_45873
(CD_Nr VARCHAR(20) NOT NULL,
CD_Titel VARCHAR(100),
CD_Laenge Laenge,
Erscheinungsjahr NUMBER (2,0) NOT NULL);
```

**SQL-Anweisung 2:**



```
CREATE TABLE CD_Bestand_45873
(CD_Nr VARCHAR(20) NOT NULL,
CD_Titel VARCHAR(100),
CD_Laenge NUMBER (2,0) DEFAULT 60
CHECK (CD_Laenge IN (45, 60, 90)),
Erscheinungsjahr NUMBER (2,0) NOT NULL);
```



## Alternativlösung für ORACLE:

### 6.1.4 Erweiterte Form der CREATE TABLE-Anweisung

Die einfachste Form der CREATE TABLE-Anweisung beinhaltet lediglich den Relationennamen und die Attributdefinitionen bestehend aus Attributname, Datentyp/Domäne und eventuell einer Wertzuweisung (s. [Relationen erstellen](#)).



SYNTAX



```
CREATE TABLE [Schema-Name.]Relationenname  
(Attributdefinition,  
  Attributdefinition,...);
```

Soll die Relation noch genauer bestimmt werden, können den Attributdefinitionen eine Primär- und Fremdschlüsseldefinition, eine Eindeutigkeits- und eine CHECK-Spezifikation folgen.



SYNTAX



```
CREATE TABLE [Schema-Name.]Relationenname  
(Attributdefinition,  
  Attributdefinition,...  
  [, Primärschlüsseldefinition]  
  [, Fremdschlüsseldefinition]  
  [, Eindeutigkeitspezifikation]  
  [, CONSTRAINT Regelname CHECK-Spezifikation]);
```

Innerhalb der CREATE TABLE-Anweisung müssen der Relationenname und mindestens eine Attributdefinition angegeben werden. Alle anderen Klauseln sind optional. Einige RDBMS unterstützen nur die einfache Form der CREATE TABLE-Anweisung.

### 6.1.5 Primärschlüsseldefinition

Das Attribut, das als Primärschlüssel definiert wird, muss eindeutig sein. Es darf keine gleichen Werte und keine leeren Zellen aufweisen (**NOT NULL**). Ein Attribut, das sich für einen Primärschlüssel eignet, ist z.B. das Attribut "ISBN". Mit ihm kann jedes Tupel (jede Zeile) der Relation eindeutig identifiziert werden.

ISBN	Titel	Seitenanzahl	Exemplare	Leihfrist
3802551230	Pustebblume	22	2	15
3499225085	Die Pest	136	6	30
3100101065	Die Pest	362	3	30
3451280000	Die Bibel	1863	5	60



**Relation Buch**  
aus der Miniwelt Hochschule



#### Beispiel

Die Relation Buch soll das Attribut "ISBN" als Primärschlüssel erhalten.

SQL-Anweisung:



```
CREATE TABLE Buch
( ISBN VARCHAR(20) NOT NULL,
  Titel VARCHAR(100),
  Exemplare NUMBER(3,0) NOT NULL,
  Leihfrist Fristen,
  PRIMARY KEY (ISBN));
```

Ein Primärschlüssel kann auch mehrere Attribute umfassen. In der folgenden Relation "leihen\_aus" ist die eindeutige Identifizierung eines Tupels nur durch die Kombination der Attribute "MatrNr", "ExemplarNr" und "ISBN" zu erreichen. Es muss daher auch nicht mehr das einzelne Attribut, sondern die Kombination der Primärschlüssel eindeutig sein.



#### Beispiel

MatrNr	ExemplarNr	ISBN	Ausleihdatum	Rueckgabedatum
--------	------------	------	--------------	----------------

1562367	1	3423101776	01.01.2008	30.01.2008
6432753	1	3453186834	02.01.2008	31.01.2008
7564258	1	3423202777	02.01.2008	31.01.2008
2358712	1	3453140982	01.01.2008	30.01.2008
2358712	2	3453186834	01.01.2008	30.01.2008



**Relation leihen\_aus**  
aus der Miniwelt Hochschule

SQL-Anweisung:



```
CREATE TABLE leihen_aus
(MatrNr CHAR(7) NOT NULL,
ExemplarNr VARCHAR(2) NOT NULL,
ISBN VARCHAR(20) NOT NULL,
Ausleihdatum DATE NOT NULL,
Rueckgabedatum DATE NOT NULL,
PRIMARY KEY (MatrNr, ExemplarNr, ISBN));
```



### Übung

#### Übung sql.3

Entfernen Sie die alte Relation "CD\_Bestand\_IhreMatrikelnummer" aus der Datenbank mit folgendem Befehl:



```
DROP TABLE CD_Bestand_IhreMatrikelnummer;
```

Legen Sie die Relation "CD\_Bestand\_IhreMatrikelnummer" erneut mit den Attributen "CD\_Nr", "CD\_Titel", "CD\_Laenge" und "Erscheinungsjahr" an. Legen Sie jetzt auch ein geeignetes Attribut als Primärschlüssel fest.

**Lösung zeigen**

## SQL-Anweisung 1



```
DROP TABLE CD_Bestand_45873;
```

## SQL-Anweisung 2



```
CREATE TABLE CD_Bestand_45873  
(CD_Nr VARCHAR(20) NOT NULL,  
CD_Titel VARCHAR(100),  
CD_Laenge NUMBER (3,0) NOT NULL,  
Erscheinungsjahr NUMBER (2,0) NOT NULL,  
PRIMARY KEY (CD_Nr));
```

### 6.1.6 Fremdschlüsseldefinition

Verknüpfungen zwischen Relationen erfolgen über Werte. Der Fremdschlüssel (FOREIGN KEY) beschreibt eine derartige Beziehung zwischen zwei Relationen. Ein Fremdschlüssel ist ein Attribut, das auf den Primärschlüssel in einer anderen Relation verweist. So sind z.B. "MatrNr" und "ExemplarNr" in der Relation "leihen\_aus" Fremdschlüssel. "MatrNr" verweist auf den Primärschlüssel "MatrNr" in der Relation "Studierende" und "ExemplarNr" verweist auf den Primärschlüssel "ExemplarNr" in der Relation "Exemplar". Ein Fremdschlüssel muss nicht eindeutig sein. So hat z.B. der Fremdschlüssel "MatrNr" in der Relation "leihen\_aus" mehrere identische Werte, denn ein Student kann mehrere Bücher ausleihen.



Ein Fremdschlüssel ist demnach kein wirklicher Schlüssel, sondern stellt eine Beziehung zu einem Schlüssel in einer anderen Relation dar.

Nach der Definition eines Fremdschlüssels muss noch angegeben werden, auf welche Relation sich der Fremdschlüssel bezieht, d.h. welche Relation referenziert wird. Ein Fremdschlüssel wird definiert durch die Schlüsselworte FOREIGN KEY gefolgt von der

Attributliste in Klammern, danach vom Schlüsselwort REFERENCES und zum Schluss vom Namen der Relation, auf die verwiesen wird.



Beispiel

Die Relation "leihen\_aus" soll mit den Attributen "MatrNr", "ExemplarNr", "ISBN", "Ausleihdatum" und "Rueckgabedatum" erstellt werden. Dabei ist die Kombination von "MatrNr", "ExemplarNr" und "ISBN" der Primärschlüssel. Fremdschlüssel sind jeweils "MatrNr", "ExemplarNr" und "ISBN".

SQL-Anweisung:



```
CREATE TABLE leihen_aus
(MatrNr CHAR(7) NOT NULL,
ExemplarNr VARCHAR(2) NOT NULL,
ISBN VARCHAR(20) NOT NULL,
Ausleihdatum DATE NOT NULL,
Rueckgabedatum DATE NOT NULL,
PRIMARY KEY (MatrNr, ExemplarNr, ISBN),
FOREIGN KEY (MatrNr) REFERENCES Studierende,
FOREIGN KEY (ExemplarNr, ISBN)
REFERENCES Exemplar(ExemplarNr, ISBN));
```



Jetzt stellt sich noch die Frage, was geschieht, wenn der von einem Fremdschlüssel referenzierte Schlüssel gelöscht wird?



Beispiel

Die Bücher Pustebblume von Peter Lustig werden aus dem Bestand der Bibliothek entfernt. Das entsprechende Tupel der Relation "Buch" wird gelöscht. Was geschieht in diesem Fall mit dem entsprechenden Attributwert "ExemplarNr" in der Relation "Exemplar"?

Da die Bücher nicht mehr vorhanden sind, können auch alle Einträge über sie entfernt werden. Der Fremdschlüssel wird dem referenzierten Schlüssel angepasst. Dies geschieht mit dem Ausdruck `ON DELETE CASCADE`, der dem Fremdschlüssel und seiner Referenz folgt.

SQL-Anweisung:



```
CREATE TABLE leihen_aus
(MatrNr CHAR(7) NOT NULL,
ExemplarNr VARCHAR(2) NOT NULL,
ISBN VARCHAR(20) NOT NULL,
Ausleihdatum DATE NOT NULL,
Rueckgabedatum DATE NOT NULL,
PRIMARY KEY (MatrNr, ExemplarNr),
FOREIGN KEY (MatrNr) REFERENCES Studierende,
FOREIGN KEY (ExemplarNr, ISBN)
REFERENCES Exemplar(ExemplarNr, ISBN)
ON DELETE CASCADE);
```

In vielen Fällen ist es jedoch sinnvoller, dass der referenzierte Schlüssel nicht verändert werden darf. Dafür wird mit dem Ausdruck `ON DELETE RESTRICT` gesorgt. Dies wird auch automatisch dann angenommen, wenn keine Angaben zum Vorgehen bei Änderungen gemacht werden. Als weitere Möglichkeit kann der Fremdschlüssel mit dem Ausdruck `ON DELETE SET NULL` auf NULL gesetzt werden, wenn der referenzierte Schlüssel gelöscht wird. Dies geht allerdings nur, wenn dort leere Zellen zugelassen sind.



Syntax für einen Fremdschlüssel:



```
FOREIGN KEY (Attributenliste)
REFERENCES Relationenname[(Attributenliste)]
```

## 6.1.7 Eindeutigkeitspezifikation

Mit der Eindeutigkeitspezifikation können Attribute oder Attributkombinationen, die eindeutig sind, neben dem Primärschlüssel als weitere Schlüssel bestimmt werden. In der Relation "leihen\_aus" soll sichergestellt werden, dass keine zwei Studierende zur selben Zeit dasselbe Buchexemplar ausleihen können. Dies geschieht mit der UNIQUE-Klausel gefolgt vom Namen des zu spezifizierenden Attributs (oder der Attribute) in Klammern. In diesem Fall sind es die Attribute "ExemplarNr" und "ISBN".

SQL-Anweisung:



```
CREATE TABLE leihen_aus
(MatrNr CHAR(7) REFERENCES Studierende,
ExemplarNr VARCHAR(2),
ISBN VARCHAR(20),
Ausleihdatum DATE NOT NULL,
Rueckgabedatum DATE NOT NULL,
PRIMARY KEY(MatrNr,ExemplarNr),
FOREIGN      KEY      (ExemplarNr,ISBN)      REFERENCES
    Exemplar(ExemplarNr,ISBN),
CONSTRAINT                                Exemplar_bereits_ausgeliehen
    UNIQUE(ExemplarNr,ISBN));
```



SYNTAX

Syntax der Eindeutigkeitspezifikation:



```
[CONSTRAINT Bezeichnung] UNIQUE (Attributname)
```

Mit Hilfe des Schlüsselworts CONSTRAINT kann eine Bezeichnung für die UNIQUE-Klausel angegeben werden. Die Fehlermeldung, die vom Datenbanksystem zurückgegeben wird, beinhaltet auch die angegebene Bezeichnung, und so kann leichter festgestellt werden, wodurch die Fehlermeldung verursacht wurde.

## 6.1.8 CHECK-Spezifikation

Die CHECK-Klausel kann außer innerhalb einer Domänendefinition auch innerhalb einer Relationendefinition verwendet werden. Unabhängig von einer Domänendefinition ist dadurch die Festlegung erlaubter Attributwerte möglich. Außerdem kann die Bedingung der CHECK-Klausel in diesem Fall mehrere Attribute betreffen.

**Beispiel**

Es soll eine Relation "Studierende" mit den Attributen "MatrNr", "Name", "Vorname", "UrlaubsSem" und "Semester" angelegt werden. Es soll sichergestellt sein, dass kein Eintrag mit einer größeren Anzahl "UrlaubsSem" als "Semester" getätigt werden kann.

SQL-Anweisung:



```
CREATE TABLE Studierende
  (MatrNr CHAR(7) NOT NULL,
   Name VARCHAR(30) NOT NULL,
   Vorname VARCHAR(20) NOT NULL,
   UrlaubsSem NUMBER(1,0),
   Semester NUMBER(2,0)
   CHECK (UrlaubsSem <= Semester));
```

**SYNTAX**

Syntax der CHECK-Spezifikation:



```
CHECK (Bedingung)
```

Für die Bedingung der CHECK-Klausel gelten die gleichen Regeln wie für die WHERE-Bedingung.



Eine CHECK-Spezifikation wird bei jeder Änderung des Datensatzes überprüft, was sehr aufwändig sein kann.



## 6.1.9 Indizes

### (CREATE INDEX)

Mit einem Index kann man die Eindeutigkeit eines Attributtyps (einer Spalte) sicherstellen und Anfragen beschleunigen. Ein Index lässt sich für ein oder mehrere Attribute einer Relation definieren.

In einer Relation mit vielen Einträgen kann die Antwortzeit einer Anfrage wie



```
SELECT * FROM Buch WHERE ISBN = '3453186834';
```

erheblich verkürzt werden, wenn ein Index auf dem Attribut "ISBN" eingerichtet wird. Ein Index lohnt sich vor allem für Attribute mit vielen unterschiedlichen Einträgen. Da die CREATE INDEX-Anweisung nicht im SQL-Standard enthalten ist, wird hier die Syntax von ORACLE, Sybase und DB2 vorgestellt:



Syntax CREATE INDEX-Anweisung von ORACLE, Sybase und DB2:



```
CREATE [UNIQUE] INDEX Indexname  
ON Relationenname (Attributliste)
```

Der Zusatz UNIQUE stellt sicher, dass für alle Tupel der Relation die Kombination der Werte der Indexattribute verschieden ist.

Die Anfragen werden zwar im Allgemeinen (nicht immer!) durch Indizes beschleunigt, aber Einfüge- und Änderungsoperationen werden dadurch zumeist langsamer.



### Übung sql.4

1. Definieren Sie eine Relation "Liedsammlung\_IhreMatrikelnummer" mit den Attributen "LiedNr", "Liedtitel", "Interpret", "Spieldauer" und "CD\_Nr".

2. Stellen Sie über einen Fremdschlüssel eine sinnvolle Beziehung zur Relation "CD\_Bestand\_IhreMatrikelnummer" her.
3. Die Kombination "Liedtitel" und "Interpret" soll eindeutig sein. Jedes Lied soll länger als eine Minute sein.

**Lösung zeigen**

SQL-Anweisung:



```
CREATE TABLE Liedsammlung_45873
(LiedNr NUMBER(3,0) NOT NULL,
Liedtitel VARCHAR(100),
Interpret VARCHAR(100),
Spieldauer NUMBER(3,0) NOT NULL,
CD_Nr VARCHAR(20) NOT NULL,
FOREIGN KEY (CD_Nr) REFERENCES CD_Bestand_45873,
UNIQUE (Liedtitel, Interpret),
CHECK (Spieldauer > '1'));
```

**SQL Interpreter**[|SQL Interpreter](#)

## 6.1.10 Relationen ändern

Um in bereits existierende Relationen neue Attribute einzufügen oder vorhandene zu löschen, wird der Befehl ALTER TABLE benutzt.

Nach den Schlüsselworten ALTER TABLE ist der Relationenname anzugeben. Sollen neue Attribute eingefügt bzw. geändert werden, muss anschließend das Schlüsselwort ADD bzw. MODIFY angegeben werden und danach in Klammern die einzufügenden Attribute und deren Datentypen. Sollen hingegen vorhandene Attribute gelöscht werden, muss nach ALTER TABLE das Schlüsselwort DROP COLUMN angegeben werden und danach ohne Klammern die zu löschenden Attribute.

**Beispiel**

Aus der Relation "Buch" mit den Attributen "ISBN", "Titel" und "Seitenanzahl" soll das Attribut "Seitenanzahl" gelöscht werden.

SQL-Anweisung:



```
ALTER TABLE Buch  
DROP COLUMN Seitenanzahl;
```

**Beispiel**

Das Attribut "Seitenanzahl" soll wieder in die Relation "Buch" eingefügt bzw. geändert werden.

SQL-Anweisung:



```
ALTER TABLE Buch  
ADD (Seitenanzahl NUMBER(5,0));
```

bzw.



```
ALTER TABLE Buch  
MODIFY (Seitenanzahl NUMBER(4,0));
```

Mit der ALTER TABLE-Anweisung lassen sich auch Eindeutigkeitspezifikationen, Primär- und Fremdschlüssel und CHECK-Spezifikationen ändern. Die Syntax für diese Änderungen ist allerdings ziemlich umfangreich und auch stark abhängig vom benutzten Datenbanksystem. Es wird daher empfohlen, für weiterführende Informationen das Handbuch des verwendeten Systems zu konsultieren.

**SYNTAX**

Vereinfachte Syntax der ALTER TABLE-Anweisung:



```
ALTER TABLE [Schema-Name.]Relationenname  
ADD Attributdefinition | MODIFY Attributdefinition  
| DROP COLUMN Attributname [RESTRICT | CASCADE];
```

Wird beim Löschen eines Attributs "RESTRICT" angegeben, wird das Attribut nur dann aus der Relation entfernt, wenn es nirgendwo sonst (z.B. in einer Sichtdefinition, SQL-Routine, Triggerdefinition) verwendet wird.

Wird beim Löschen eines Attributs hingegen "CASCADE" angegeben, wird sowohl das Attribut als auch die davon abhängigen Datenbankobjekte (wie Sichten, Trigger etc.) gelöscht.



Ein angefügtes Attribut einer Relation, die bereits Daten enthält, bekommt in allen Zellen den Wert NULL. Die Zellen der neu angefügten Attribute von existierenden Tupeln können mit der UPDATE-Anweisung gefüllt werden.

### 6.1.11 Relationen löschen

Mit der DELETE-Anweisung können einzelne Tupel oder eine ganze Relation gelöscht werden. Zum vollständigen Löschen einer Relation wird die DROP TABLE-Anweisung benutzt.

Mit der Anweisung DROP lassen sich Relationen, Views, Indizes und alle anderen Objekte löschen, die mit CREATE definiert wurden.



Beispiel

Die Relation "Buch" soll vollständig gelöscht werden.

SQL-Anweisung:



```
DROP TABLE Buch;
```



SYNTAX

Syntax der DROP TABLE-Anweisung:



```
DROP TABLE [Schema-Name.]Relationenname | Index | View;
```



Übung

**Übung sql.5**

1. Fügen Sie das Attribut "Bewertung" in die Relation "Liedsammlung\_IhreMatrikelnummer" ein.
2. Entfernen Sie das Attribut "Erscheinungsjahr" aus der Relation "CD\_Bestand\_IhreMatrikelnummer".
3. Löschen Sie die Relationen "Liedsammlung\_IhreMatrikelnummer" und "CD\_Bestand\_IhreMatrikelnummer".

**Lösung zeigen**

SQL-Anweisung 1:



```
ALTER TABLE Liedsammlung_45873  
ADD (Bewertung NUMBER(2,0));
```

SQL-Anweisung 2:



```
ALTER TABLE CD_Bestand_45873 DROP Erscheinungsjahr;
```

SQL-Anweisung 3:



```
DROP TABLE Liedsammlung_45873;
```

SQL-Anweisung 4:



```
DROP TABLE CD_Bestand_45873;
```

**SQL Interpreter**[SQL Interpreter](#) 

## 6.1.12 COMMIT WORK/ROLLBACK WORK

Mit der Anweisung COMMIT WORK lassen sich alle Dateneingaben und Datenänderungen bestätigen. Erst nach dieser Bestätigung werden die Änderungen in die Datenbank übernommen.

Mit der Anweisung ROLLBACK WORK lassen sich alle Änderungen wieder rückgängig machen, der alte Zustand der Datenbank wird wieder hergestellt.

Um die Konsistenz einer Datenbank zu bewahren, liegen nach Beginn einer Transaktion (INSERT, DELETE, UPDATE) die Daten zunächst in doppelter Form (alt und neu) vor. Nach Eingabe der COMMIT-Anweisung wird der neue Zustand festgeschrieben, nach Eingabe der ROLLBACK-Anweisung bleibt der alte Zustand erhalten.

## 6.2 Datenbankanfragen

**Gliederung**

### 6.2 Datenbankanfragen

#### 6.2.1 Die SELECT-Anweisung

#### 6.2.2 Die ORDER BY-Klausel

#### 6.2.3 Die WHERE-Klausel

#### 6.2.4 Die GROUP BY-Klausel

#### 6.2.5 Die HAVING-Klausel

#### 6.2.6 Joins

#### 6.2.7 Unteranfragen

#### 6.2.8 Kombination von unabhängigen Unteranfragen

### 6.2.1 Die SELECT-Anweisung

Eine SQL-Anweisung setzt sich zusammen aus dem Befehlsschlüsselwort (z. B. SELECT) und weiteren Schlüsselwörtern, Klauseln genannt, die oft optional sind (z.B. WHERE, GROUP BY,...). Im Folgenden wird die SELECT-Anweisung mit ihren Klauseln (ORDER BY, WHERE, GROUP BY und HAVING) vorgestellt. Es wird gezeigt, wie mit Hilfe der SELECT-Anweisung auf eine Relation der Datenbank zugegriffen wird, um die gesuchten

Informationen herauszufiltern und eventuell noch zu ordnen, zu gruppieren und zu verrechnen (Datenbankabfragen über mehrere Relationen und Unterabfragen werden im Anschluss vorgestellt). Das Ergebnis einer SELECT-Anweisung wird zwar in Form einer Tabelle auf dem Bildschirm dargestellt, es wird aber nicht automatisch eine neue Tabelle oder Relation in der Datenbank angelegt.

### Projektion (Ausgabe von Attributen)

Bei der Projektion wird nicht die komplette Relation (Tabelle) ausgegeben, sondern die Auswahl eines Attributes oder mehrerer Attribute (Spalten).



#### Beispiel

Es soll das Attribut "Titel" der Relation "Buch" ausgegeben werden.

Zunächst wird nach dem Schlüsselwort SELECT der gewünschte Attributname angegeben. Anschließend wird nach dem Schlüsselwort FROM der Name der Relation angegeben, in der sich das Attribut befindet.

SQL-Anweisung:



```
SELECT Titel  
FROM Buch;
```

Ausgabe:

Titel
Pustoblume
Die Pest
...



Sollen mehrere Attribute ausgegeben werden, müssen die Namen der gewünschten Attribute durch Kommata getrennt nach dem Schlüsselwort SELECT aufgeführt werden.

SQL-Anweisung:



```
SELECT ISBN, Titel  
FROM Buch;
```

Ausgabe:

ISBN	Titel
3802551230	Pusteblume
...	...
3423101776	Solaris



Die Reihenfolge der Spalten wird durch die Reihenfolge der Attributnamen in der Anweisung bestimmt.

### Ausgabe einer ganzen Relation



Beispiel

Die Relation "Buch" soll vollständig ausgegeben werden.

SQL-Anweisung:



```
SELECT *  
FROM Buch;
```

Damit eine Relation vollständig ausgegeben wird, muss nach dem SELECT ein Stern angegeben werden.

### Ausgabe eines Attributes ohne identische Tupel

SQL selektiert standardmäßig alle Tupel einer Relation oder eines Attributs. Wird in der Anfrage der Ausdruck DISTINCT vor den Attributnamen gesetzt, werden identische Tupel zusammengefasst. Zusätzlich werden die Tupel aufsteigend geordnet ausgegeben (von A nach Z oder von 1 aufwärts zählend). Anstelle von DISTINCT ist auch die (standardmäßig voreingestellte) Eingabe von ALL möglich.



Beispiel

Im Attribut Titel gibt es zwei Einträge "Die Pest". Mit folgender Anweisung wird jedoch nur ein Eintrag "Die Pest" ausgegeben:

SQL-Anweisung:





```
SELECT DISTINCT Titel  
FROM Buch;
```

### Umbenennung

Für die Ausgabe einer Abfrage kann der Name eines Attributes geändert werden, um z.B. für verschiedene Benutzergruppen unterschiedliche, angepasste Projektionen zu erzeugen.



#### Beispiel

Die Relation "Buch" soll mit den Attributen "Titel", "Seitenanzahl" und "ISBN" ausgegeben werden. Die Spalte "ISBN" soll jedoch nicht mit "ISBN" sondern mit "Buchnummer" betitelt sein.

SQL-Anweisung:



```
SELECT ISBN Buchnummer, Titel, Seitenanzahl  
FROM Buch;
```

Ausgabe:

Buchnummer	Titel	Seitenanzahl
3802551230	Pustebblume	22
...	...	...
3423101776	Solaris	236



Um eine Spalte in der Ausgabe anders zu betiteln, wird in der Auflistung der Attribute nach ISBN und einem Leerzeichen Buchnummer als umbenannter Name angegeben.

### Virtuelle Spalte

Eine sogenannte virtuelle Spalte ist ein Attribut (mit einem konstanten Wert), das nicht in der angesprochenen Datenbank existiert, sondern lediglich zum besseren Verständnis oder zur besseren Lesbarkeit in der Projektion erscheint.

**Beispiel**

Die Relation "Buch" soll mit den Attributen "Titel" und "Seitenanzahl" ausgegeben werden. Zusätzlich soll in einer neuen Spalte (virtuelle Spalte) in jeder Zeile das Wort "Seiten" enthalten sein.

SQL-Anweisung:



```
SELECT Titel, Seitenanzahl, 'Seiten'  
FROM Buch;
```

Ausgabe:

Titel	Seitenanzahl	'Seiten'
Lustig, Peter	22	Seiten
...		
Lem, Stanislaw	236	Seiten



Mit der Ausgabe einer in der Basisrelation nicht existierenden Spalte lässt sich z.B. ein realer Wert kommentieren. Der Name der virtuellen Spalte wird in Hochkommata eingeschlossen in die Liste der Attributnamen eingefügt.

**Aggregierungsfunktionen**

Aggregierungsfunktionen sind Kardinalität (COUNT), Summe (SUM), Maximum (MAX), Minimum (MIN) und Mittelwert (AVG).

**Beispiel**

Es soll die Anzahl aller Datensätze (Zeilen) der Relation "Buch" bestimmt werden.

SQL-Anweisung:



```
SELECT COUNT ( * )  
FROM Buch;
```

**Wie viele Bücher besitzt die Bibliothek insgesamt?**

SQL-Anweisung:



```
SELECT SUM (Exemplare)
FROM Buch;
```

**Arithmetische Operationen**

Arithmetische Operationen sind z.B. Addition, Subtraktion, Multiplikation, Division, Sinus, Wurzel, Modulo oder Vorzeichenumkehr.

**Beispiel**

Es soll ermittelt werden, wie viele Semester jeder Student insgesamt studiert hat. Dazu müssen "Semester" und "UrlaubsSem" addiert und in der neuen Spalte "GesamtSem" ausgegeben werden.

SQL-Anweisung:



```
SELECT Semester, UrlaubsSem, Semester + UrlaubsSem
GesamtSem
FROM Studierende;
```

Nach der Addition von "Semester" und "UrlaubsSem" wird durch ein Leerzeichen getrennt der Name der Ausgabespalte "GesamtSem" angegeben.

**Relationen anderer BenutzerInnen**

Für die Ausgabe oder Änderung einer Relation, die einer anderen Person gehört, wird der Schema-Name der Relation (in der Regel entspricht dieser der Benutzerkennung) gefolgt von einem Punkt vor dem Relationennamen, angegeben.



```
SELECT *
FROM Meier.Buch;
```



### Syntax der SELECT-Anweisung (Grundform):



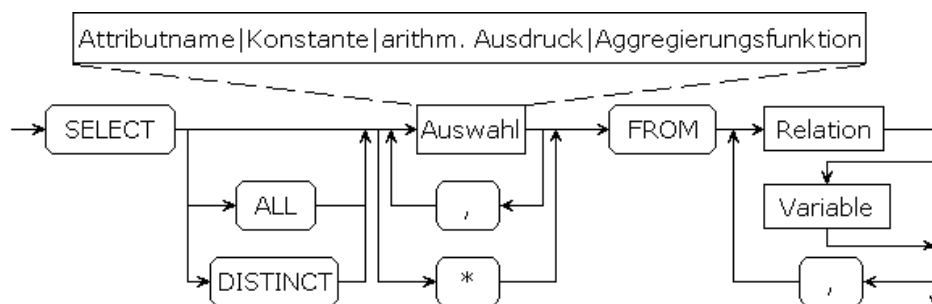
```
SELECT [ALL | DISTINCT] * | {Attributname |
Konstante | arithmetischer Ausdruck |
Aggregierungsfunktion}
[, {Attributname | Konstante | arithmetischer Ausdruck |
Aggregierungsfunktion}] *
FROM [Schema-Name.]Relationname;
```



### Syntax der SELECT-Anweisung (Grundform)

(Der Inhalt der eckigen Klammern ist optional. Der senkrechte Strich "|" bedeutet "oder". Der hochgestellte Asteriskus bedeutet, dass der Inhalt der Klammern beliebig oft wiederholt werden kann. Eine Konstante kann z.B. eine virtuelle Spalte oder eine Zahl sein.)

### Flussdiagramm zur SELECT-Anweisung (Grundform)



### Syntax der SELECT-Anweisung (Grundform)



In der Online-Version befindet sich an dieser Stelle eine Simulation.

### Animierte Syntax der SELECT-Anweisung (Grundform)



## Übung

**Übung sql.6**

Wie viele Seiten hat ein Buch durchschnittlich?

**SQL Interpreter**

[SQL Interpreter](#) 

**Lösung zeigen**

SQL-Anweisung:



```
SELECT AVG ( Seitenanzahl ) FROM Buch;
```

**6.2.2 Die ORDER BY-Klausel**

Mit der ORDER BY-Klausel wird die Ausgabe nach bestimmten Kriterien sortiert.



## Beispiel

Gesucht ist eine Liste der Relation "Buch" mit "Titel" und "ISBN", die alphabetisch nach Titeln geordnet ist.

SQL-Anweisung:



```
SELECT ISBN, Titel  
FROM Buch  
ORDER BY Titel;
```

Ausgabe:

ISBN	Titel
3897212188	CGI kurz und gut
3453140982	Das Vogelmädchen
3499225085	Der Ekel

...

...

Im Anschluss an die ORDER BY-Klausel ist der Zusatz ASC bzw. DESC möglich. ASC (englisch: ascending) ist gleichbedeutend mit keiner Angabe und bewirkt eine aufsteigende Sortierung. DESC (englisch: descending) bewirkt eine absteigende Sortierung.

**Beispiel**

Gesucht ist eine Liste der Relation "Buch" mit "ISBN" und "Titel", die absteigend nach "ISBN" geordnet ist.

SQL-Anweisung:



```
SELECT ISBN, Titel
FROM Buch
ORDER BY ISBN DESC;
```

Ausgabe:

ISBN	Titel
3897212188	CGI kurz und gut
3897212013	Perl 5 kurz und gut
3802551230	Pusteblume
...	...

Es ist auch eine Mehrfachsortierung möglich. Dazu werden nach der ORDER BY-Klausel mehrere Spaltennamen angegeben. Bei Gleichheit im ersten Ordnungskriterium wird nach dem zweiten Kriterium sortiert usw.

**SYNTAX**

Syntax der SELECT-Anweisung mit ORDER BY:



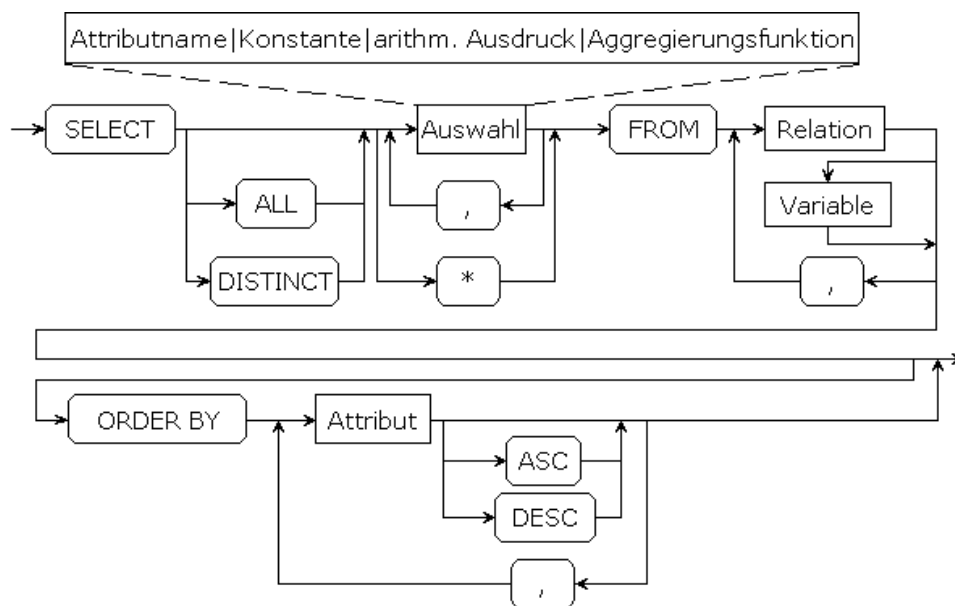
```

SELECT [ALL | DISTINCT] * | {Attributname | Konstante
    | arithmetischer Ausdruck | Aggregierungsfunktion}[
{Attributname |
Konstante      |      arithmetischer      Ausdruck
    Aggregierungsfunktion}] *
FROM [Schema-Name.]Relationname
[ORDER BY Attributliste | Referenznummer [ASC|DESC]];

```

(Referenznummer ist die Stellung der Spalte innerhalb der Anordnung in der Ausgabetabelle. So hat z.B. die dritte Spalte die Referenznummer 3. Die Reihenfolge der einzelnen Klauseln ist obligatorisch. Wenn einige nicht verwendet werden, muss für die übrigen die Reihenfolge eingehalten werden.)

### Flussdiagramm zur SELECT-Anweisung mit ORDER BY



Flußdiagramm zur SELECT-Anweisung mit ORDER BY



In der Online-Version befindet sich an dieser Stelle eine Simulation.

Animierte Syntax der SELECT-Anweisung mit ORDER BY



## Übung

**Übung sql.7**

Gesucht ist eine Relation "Studierende" mit "MatrNr", "Name" und "Vorname". Die Liste soll alphabetisch nach "Name" im ersten Kriterium und bei Gleichheit nach "Vorname" im zweiten Kriterium geordnet sein.

**SQL Interpreter**

[SQL Interpreter](#) 

**Lösung zeigen**

SQL-Anweisung:



```
SELECT MatrNr , Name , Vorname  
FROM Studierende  
ORDER BY Name, Vorname;
```

**6.2.3 Die WHERE-Klausel**

In den meisten Fällen sollen gar nicht alle Tupel einer Relation angezeigt werden sondern lediglich die Tupel, auf die eine ganz bestimmte Bedingung zutrifft. Diese Selektion geschieht mit der sogenannten WHERE-Klausel. Die WHERE-Klausel schränkt die durch ein SELECT gefundene Menge ein. Dabei kann die Suchbedingung ein einfacher Vergleich oder eine zusammengesetzte Bedingung sein. Es ist darauf zu achten, dass zwei Ausdrücke, die miteinander verglichen werden, vom gleichen Datentyp sein müssen.

**Einfacher Vergleich**

Gesucht sind alle Bücher der Relation "Buch" mit "Titel" und "ISBN", von denen es drei Exemplare gibt.

SQL-Anweisung:





```
SELECT Titel, ISBN
FROM Buch
WHERE Exemplare=3;
```

Ausgabe:

Titel	ISBN
CGI kurz und gut	3897212188
Das Vogelmädchen	3453140982
...	...
Perl 5 kurz und gut	3897212013

Für diese Abfrage wird in der Bedingung der WHERE-Klausel ein Vergleich angestellt. Es sollen nur die Tupel ausgegeben werden, für die gilt: Exemplare = 3.



Beispiel

Gesucht sind alle Bücher der Relation "Buch" mit "Titel", deren "Seitenanzahl" auch nach Abzug von 10 % größer als 200 ist.

SQL-Anweisung:



```
SELECT Titel
FROM Buch
WHERE Seitenanzahl - (Seitenanzahl * 10/100) > 200;
```

**SQL-Anweisung**

Ausgabe:

Titel
Das Vogelmädchen
Der kleine Hobbit
...
Solaris

### Zusammengesetzte Bedingung

Die WHERE-Klausel in Verbindung mit Operatoren ist eine zusammengesetzte Bedingung.



Gesucht sind alle "Titel" der Relation "Buch" mit mehr als 20 Seiten. Der Titel "Die Pest" soll nicht ausgegeben werden.

SQL-Anweisung:



```
SELECT Titel
FROM Buch
WHERE Seitenanzahl > 20 AND NOT Titel = 'Die Pest';
```

Bei Verwendung des Operators AND müssen beide Bedingungen erfüllt sein, damit das Tupel selektiert wird. Wird ein Attribut mit einer Zeichenkette verglichen, muss diese in einfachen Hochkommata eingeschlossen werden.



Beispiel

Es soll eine Liste mit Büchern (ISBN und Titel), die zwischen 100 und 300 Seiten haben, erstellt werden.

SQL-Anweisung:



```
SELECT ISBN, Titel
FROM Buch
WHERE Seitenanzahl BETWEEN 100 AND 300;
```

Das AND ist in diesem Fall kein logischer Operator, sondern Bestandteil des Operators BETWEEN.



Beispiel

Gesucht sind alle "Titel" der Relation "Buch" mit den ISBN 3897212188 und 3453140982.

SQL-Anweisung:



```
SELECT Titel
FROM Buch
WHERE ISBN IN ('3897212188', '3453140982');
```

Folgende SQL-Anweisung könnte eventuell alternativ eingesetzt werden:



```
SELECT Titel
FROM Buch
WHERE ISBN LIKE '3897212188' OR ISBN LIKE '3453140982';
```

Mit "OR" ist allerdings nicht das natürlichsprachliche "oder" im Sinne von "entweder... oder...", sondern das mathematische "oder" gemeint, d.h. es kann eine der Bedingungen oder aber es können beide Bedingungen gleichzeitig wahr sein.



Oracle liefert also beide Buchtitel, falls auch tatsächlich beide ISBN vorhanden sind.



Beispiel

Gesucht ist eine Liste aller Titel, die mit "P" beginnen, wobei der eventuell vorangestellte Artikel nicht berücksichtigt werden soll (z.B. "Die" von "Die Pest").

SQL-Anfrage:



```
SELECT Titel
FROM Buch
WHERE Titel LIKE 'P%' OR Titel LIKE 'D_ _ _P%';
```

Ausgabe:

Titel
Pusteblume
Die Pest



Hinweis

Da nur ein Teil des Namens der gesuchten Titel bekannt ist, wird auch nur dieser Teil in der Abfrage benutzt. Der Rest wird durch Stellvertreterzeichen ersetzt:

% (Prozent)	ist der Platzhalter für Null oder beliebig viele Zeichen
_ (Unterstrich)	ist der Platzhalter für genau ein Zeichen. Zwei Unterstriche stehen für genau zwei Zeichen usw.

(DOS und UNIX: "%" entspricht "\*" und "\_" entspricht "?")

Es sollen also alle Bücher gelistet werden, die entweder mit "P%" oder mit "D\_ \_P%" übereinstimmen. Eine derartige Abfrage wird durch den Operator OR erreicht, bei dem nur eine der beiden Bedingungen wahr sein muss, damit das Tupel selektiert wird.

Mit ROWNUM kann die Ausgabe der Tupel aus der Ergebnismenge eingeschränkt werden. Z. B. ROWNUM <= 3 bedeutet, dass nur die ersten drei Tupel ausgegeben werden.



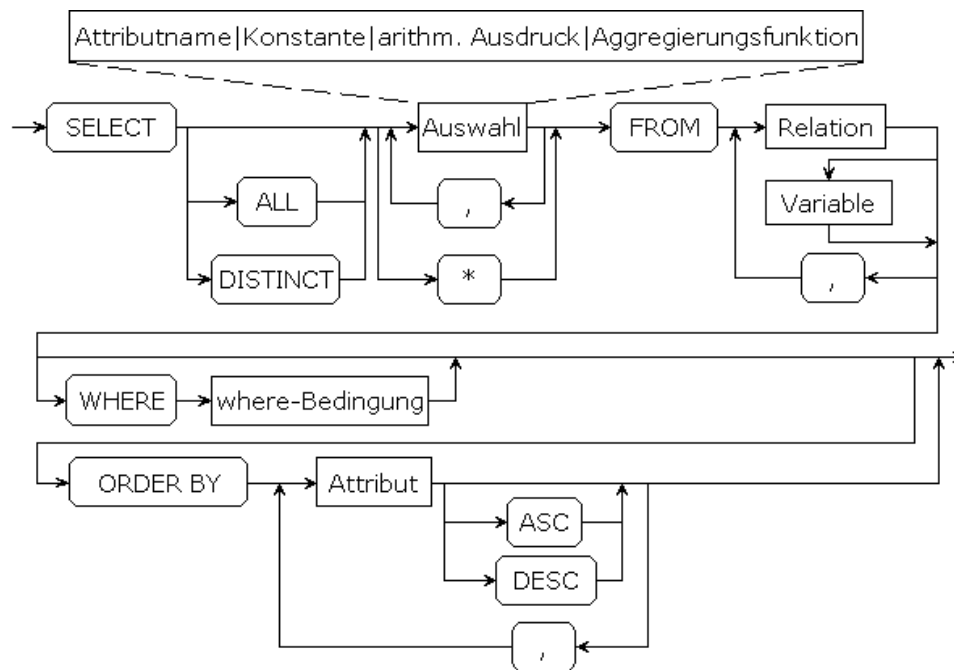
SYNTAX

### Syntax der SELECT-Anweisung mit WHERE-Bedingung



```
SELECT [ALL | DISTINCT] * | {Attributname |
Konstante | arithmetischer Ausdruck |
Aggregierungsfunktion} [, {Attributname |
Konstante | arithmetischer Ausdruck |
Aggregierungsfunktion}] *
FROM [Schema-Name.]Relationname
[WHERE -Bedingung]
[ORDER BY Attributliste | Referenznummer [ASC | DESC]];
```

### Flussdiagramm zur SELECT-Anweisung mit WHERE-Bedingung



Flußdiagramm zur SELECT-Anweisung mit WHERE-Bedingung



In der Online-Version befindet sich an dieser Stelle eine Simulation.

Animierte Syntax der SELECT-Anweisung mit WHERE-Bedingung



### Übung

#### Übung sql.8

Gesucht ist eine Liste aller weiblichen Studierenden mit Name und Vorname, die am 07.10.1985 Geburtstag haben.

#### Lösung zeigen

SQL-Anweisung:



```

SELECT Name, Vorname
FROM Studierende
WHERE Geschlecht = 'w'
AND GebDat = '07.10.1985';
  
```

#### SQL Interpreter

[SQLInterpreter](#) 

## Übung

## Übung sql.9

Gesucht ist eine Liste aller männlichen Studierenden mit Name und Vorname, die jünger als 33 sind.

## Lösung zeigen


SQL-Anweisung:



```
SELECT Name, Vorname
FROM Studierende
WHERE Geschlecht = 'm'
AND SYSDATE - GebDat < (33*365);
```

Anmerkung: Da das Ergebnis in Tagen zurückgeliefert wird, muss auch das Alter mit der Anzahl Tage eines Jahres multipliziert werden, um korrekt vergleichen zu können. Der Einfachheit halber wird angenommen, dass ein Jahr 365 Tage hat, Schaltjahre werden also nicht berücksichtigt.

## SQL Interpreter

[SQLInterpreter](#) 

## 6.2.4 Die GROUP BY-Klausel

Mit der GROUP BY-Klausel lassen sich Informationen aus einer Tabelle gezielt zusammenfassen.

<u>Container</u>	Volumen	Ware	Herkunft
1	70	Baumwolle	USA
2	100	Bananen	Brasilien
3	90	Melonen	Spanien
4	120	Bananen	Südafrika
5	80	Baumwolle	USA

6	80	Tomaten	Spanien
---	----	---------	---------



Beispiel

Das Gesamtvolumen der Einzellieferungen jeder Ware soll zusammengefasst dargestellt werden.

SQL-Anweisung:



```
SELECT Ware, SUM (Volumen)
FROM Container
GROUP BY Ware;
```

Ausgabe:

Ware	SUM (Volumen)
Bananen	220
Baumwolle	150
Melonen	90
Tomaten	80



Beispiel

Gesucht ist eine Liste mit "Ware", "Herkunft" und "Gesamtvolumen". Das "Gesamtvolumen" soll nach "Herkunft" aufgeschlüsselt sein, d.h. es soll das Gesamtvolumen pro Ware pro Herkunft gezeigt werden.

SQL-Anweisung:



```
SELECT Ware, Herkunft, SUM (Volumen) Gesamtvolumen
FROM Container
GROUP BY Ware, Herkunft;
```

Ausgabe:

Ware	Herkunft	Gesamtvolumen
Bananen	Brasilien	100
Bananen	Südafrika	120

Baumwolle	USA	150
Melonen	Spanien	90
Tomaten	Spanien	80

Bei Verwendung der GROUP BY-Klausel müssen alle Attribute, die ausgegeben werden sollen, in irgendeiner Weise zusammengefasst werden. So ist z.B. folgende SQL-Anweisung NICHT möglich:



```
SELECT Ware, Herkunft, SUM (Volumen)
FROM Container
GROUP BY Ware;
```

In der Ausgabe würde es in einer Zelle der Spalte "Herkunft" zwei Einträge geben:

Ware	Herkunft	SUM (Volumen)
Bananen	Brasilien, Südafrika???	220
Baumwolle	USA	150
...	...	...

Aus diesem Grund müssen alle im Projektionsteil (nach der SELECT-Klausel) aufgeführten Attribute zusammengefasst werden, indem sie entweder in der GROUP BY-Klausel angegeben werden oder im Projektionsteil durch eine der fünf Aggregierungsfunktionen (COUNT, SUM, MAX, MIN und AVG) zusammengefasst werden.



#### Beispiel

Gesucht ist eine Liste aller Warentypen mit der Anzahl der Lieferungen pro Warentyp.

SQL-Anweisung:



```
SELECT Ware, COUNT (*)
FROM Container
GROUP BY Ware;
```

Ausgabe:

Ware	COUNT (*)
------	-----------



Bananen	2
Baumwolle	2
Melonen	1
Tomaten	1

Das RDBMS muss einen Datenbestand erst sortieren, bevor es ihn gruppieren kann. Durch Setzen eines Index auf den Attributen, die zur Gruppierung herangezogen werden, kann die Ausführungszeit reduziert werden, da die Attribute dann bereits sortiert vorliegen.



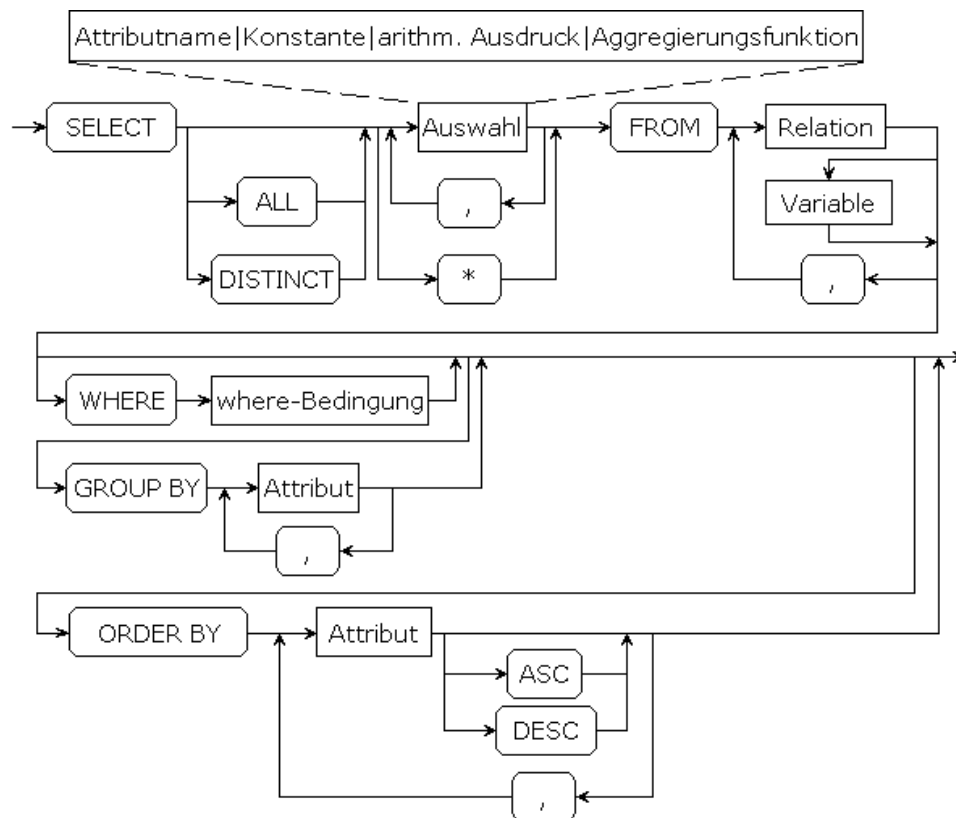
SYNTAX

#### Syntax der SELECT-Anweisung mit GROUP BY-Klausel



```
SELECT [ALL | DISTINCT] * | {Attributname |  
Konstante | arithmetischer Ausdruck |  
Aggregierungsfunktion} [, {Attributname |  
Konstante | arithmetischer Ausdruck |  
Aggregierungsfunktion}] *  
FROM [Schema-Name.]Relationname  
[WHERE -Bedingung]  
[GROUP BY Attributliste]  
[ORDER BY Attributliste | Referenznummer [ASC | DESC]];
```

#### Flussdiagramm zur SELECT-Anweisung mit GROUP BY



Flußdiagramm zur SELECT-Anweisung mit GROUP BY



In der Online-Version befindet sich an dieser Stelle eine Simulation.

Animierte Syntax der SELECT-Anweisung mit GROUP BY-Klausel



### Übung

#### Übung sql.10

Ermitteln Sie, ob die Gruppe der Studenten oder die Gruppe der Studentinnen insgesamt mehr Semester studiert hat. Erstellen Sie dazu eine Tabelle, in der die Gesamtsemesterzahl der jeweiligen Gruppe angezeigt wird.

#### Lösung zeigen

SQL-Anweisung:



```
SELECT Geschlecht, SUM (Semester + UrlaubsSem)
FROM Studierende
```

```
GROUP BY Geschlecht;
```

**SQL Interpreter**[SQLInterpreter](#) 

## 6.2.5 Die HAVING-Klausel

Die HAVING-Klausel selektiert die Ergebnisse der Gruppierung durch GROUP BY. HAVING kann nur nach einem vorangestellten GROUP BY eingesetzt werden.

**Beispiel**

Gesucht ist eine Liste mit dem Gesamtvolumen jeder Ware (Gruppierung). Es sollen jedoch nur die Waren ausgegeben werden, deren Gesamtvolumen größer als 100 ist (Selektion bestimmter Daten).

SQL-Anweisung:



```
SELECT Ware, SUM (Volumen)
FROM Container
GROUP BY Ware
HAVING SUM (Volumen) > 100;
```

Ausgabe:

Ware	SUM (Volumen)
Bananen	220
Baumwolle	150

Die HAVING-Klausel kann auch in Verbindung mit der WHERE-Klausel eingesetzt werden. Dabei wird zunächst mit WHERE eine Selektion auf der Basistabelle ausgeführt, die Zwischenmenge wird dann mit GROUP BY gruppiert. Zum Schluss werden die entstandenen Gruppen mit HAVING selektiert.

**Beispiel**

Zunächst sollen alle Einzellieferungen, deren Volumen unter 100 liegt, ausgeschlossen werden. Dann soll eine Liste mit den Waren erzeugt werden, deren Gesamtvolumen 200 übersteigt.

SQL-Anweisung:



```
SELECT Ware, SUM (Volumen)
FROM Container
WHERE Volumen >= 100
GROUP BY Ware
HAVING SUM (Volumen) > 200;
```

Ausgabe:

Ware	SUM (Volumen)
Bananen	220



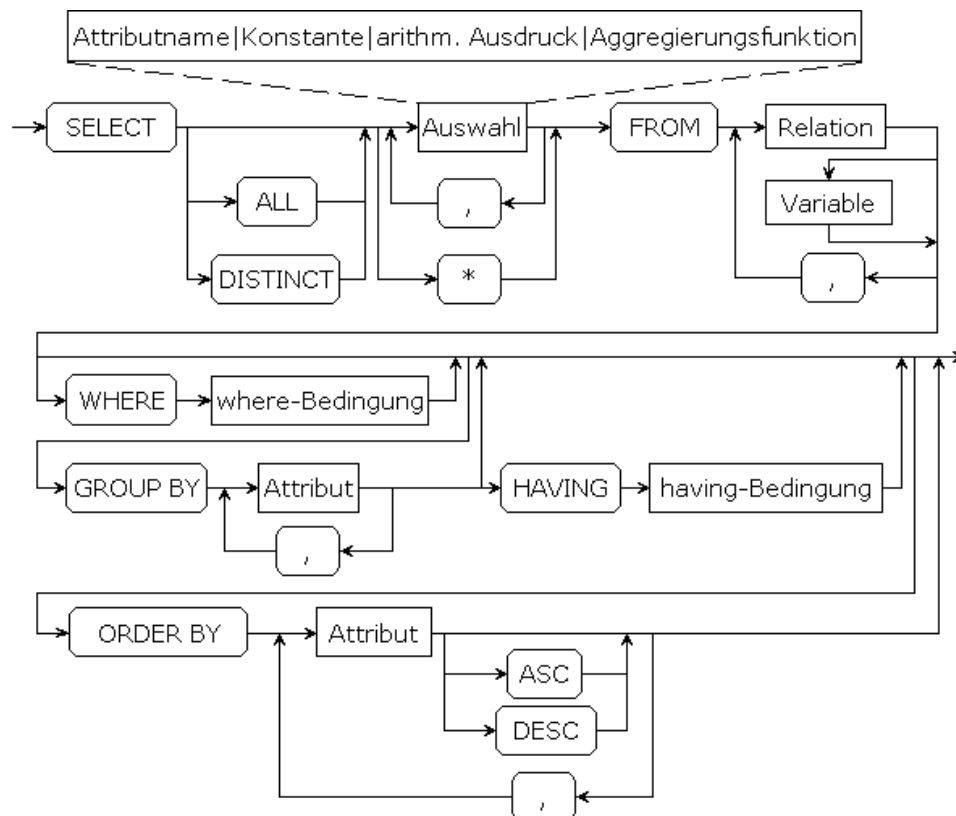
SYNTAX

Syntax der SELECT-Anweisung mit HAVING-Klausel



```
SELECT  [ALL|DISTINCT]  *  |{Attributname  |  Konstante
      | arithmetischer Ausdruck  |  Aggregierungsfunktion}[
      {Attributname  |  Konstante  |  arithmetischer Ausdruck  |
      Aggregierungsfunktion}]*
FROM [Schema-Name.]Relationname
[WHERE -Bedingung]
[GROUP BY Attributliste [HAVING Bedingung] ]
[ORDER BY Attributliste | Referenznummer [ASC|DESC]];
```

**Flussdiagramm zur SELECT-Anweisung mit allen Klauseln**



### Flußdiagramm zur SELECT-Anweisung mit allen Klauseln



In der Online-Version befindet sich an dieser Stelle eine Simulation.

## Animierte Syntax der SELECT-Anweisung mit allen Klauseln



## Übung

## Übung sql.11

Ordnen Sie den unterschiedlichen Leihfristen die Anzahl der Buchseiten aller Buchexemplare mit dieser Leihfrist zu. Ignorieren Sie dabei alle Leihfristen von Büchern, deren Gesamtseitenanzahl aller Exemplare kleiner als 300 ist.

## Lösung zeigen

SQL-Anweisung:



```
SELECT Leihfrist, SUM (Seitenanzahl)
FROM Exemplar e, Buch b
```

```
WHERE e.ISBN = b.ISBN
GROUP BY Leihfrist
HAVING SUM (Seitenanzahl) >= 300 ;
```

**SQL Interpreter**[SQLInterpreter](#) 

## 6.2.6 Joins

### *Datenbankabfragen über mehrere Relationen*

Im Folgenden geht es um Abfragen über mehrere Relationen. Viele der Join-Abfragen lassen sich alternativ auch durch Unterabfragen realisieren. Welche Form bevorzugt wird, muss im Einzelfall entschieden werden. Dabei sollte neben der Übersichtlichkeit der Abfrage vor allem die Geschwindigkeit, mit der diese bearbeitet wird, berücksichtigt werden.

### **Inner Joins**

Im Allgemeinen werden bei einer Datenbankabfrage Informationen aus mehreren Relationen benötigt und zusammengestellt.



Welche Bücher (ISBN, Leihart) sind zur Zeit ausgeliehen?



Aus der Praxis

Dazu werden Informationen aus der Relation "leihen\_aus" und aus der Relation "Exemplar" benötigt. Die SELECT-Anweisung greift auf beide Relationen zu. Dabei wird zunächst das kartesische Produkt der beteiligten Relationen gebildet, d.h. jedes Tupel der einen Relation wird mit jedem Tupel der anderen Relation kombiniert. Da diese Kombination selten sinnvoll ist (jede ISBN der Relation "Exemplar" würde mit jeder "ExemplarNr" der Relation "leihen\_aus" kombiniert werden), wird in den meisten Fällen noch eine Bedingung angefügt. Im Beispiel werden die Attribute "ExemplarNr" und "ISBN" der Relation "leihen\_aus" auf Gleichheit geprüft mit dem Primärschlüssel "ExemplarNr" der Relation "Exemplar".

ExemplarNr	ISBN	...
...	...	...
...	...	...



Relation: leihen\_aus

ExemplarNr	ISBN	Leihart
1	3423101776	N
...	...	...



Relation: Exemplar

Es werden nur die Tupel selektiert, für die eine Gleichheit der Werte in den jeweiligen Attributen "ExemplarNr" bzw. "ISBN" zutrifft. Da die Attributnamen "ExemplarNr" und "ISBN" in den Relationen identisch sind, müssen sie in der SQL-Anweisung eindeutig benannt werden. Dies geschieht, indem ihnen der Relationenname mit einem Punkt vorangesetzt wird.

SQL-Anweisung:



```
SELECT Exemplar.ISBN, Leihart
FROM Exemplar, leihen_aus
WHERE Exemplar.ExemplarNr = leihen_aus.ExemplarNr
AND Exemplar.ISBN = leihen_aus.ISBN;
```

Alternativ:



```
SELECT ISBN, Leihart
FROM leihen_aus
INNER JOIN Exemplar
USING(ExemplarNr, ISBN);
```

Da solche Joins häufig vorkommen, bietet SQL eine Möglichkeit, den Schreibaufwand zu verringern. Dazu werden Variablennamen (im Anschluss an den Relationennamen) vergeben, die dann allerdings auch in allen anderen Klauseln der Anweisung benutzt werden müssen.

In der folgenden SELECT-Anweisung soll die Relation Exemplar mit e und die Relation leihen\_aus mit l abgekürzt werden. Dazu wird in der FROM-Klausel hinter dem Namen der Relation ein Leerzeichen und der Variablenname angegeben oder das Schlüsselwort AS benutzt.

SQL-Anweisung:



```
SELECT e.ISBN, Leihart
FROM Exemplar e, leihen_aus l
WHERE e.ExemplarNr = l.ExemplarNr
AND e.ISBN = l.ISBN;
```

Alternativ:



```
SELECT e.ISBN, Leihart
FROM Exemplar AS e, leihen_aus AS l
WHERE e.ExemplarNr = l.ExemplarNr
AND e.ISBN = l.ISBN;
```



Beispiel

Es soll eine Liste aller ausgeliehenen Bücher erstellt werden, in der neben "ISBN", "ExemplarNr" und "Titel" des Buches auch "Name" und "Vorname" des Lesers/der Leserin enthalten ist.

SQL-Anweisung:



```
SELECT b.ISBN, e.ExemplarNr, Titel, Name, Vorname
FROM Buch b, Studierende s, leihen_aus l, Exemplar e
WHERE l.ExemplarNr = e.ExemplarNr
AND e.ISBN = l.ISBN
AND l.MatrNr = s.MatrNr
```



```
AND e.ISBN = b.ISBN;
```

Ausgabe:

ISBN	ExemplarNr	Titel	Name	Vorname
3423101776	1	Solaris	Schulze	Heiner
3453186834	2	Der Wüstenplanet	König	Mathilde
3423202777	3	Der kleine Hobbit	Baum	Meta
3453140982	4	Das Vogelmädchen	Meier	Siegfried
3453186834	5	Der Wüstenplanet	Meier	Siegfried



Dabei findet ein Join zwischen der Relation "leihen\_aus", der Relation "Buch" und der Relation "Exemplar" statt. Die Attribute "ExemplarNr" und "ISBN" der Relationen "leihen\_aus" und der Relation "Exemplar" werden auf Gleichheit geprüft. Innerhalb der entstandenen Zwischenmenge werden die Attribute "ISBN" der Relationen "Exemplar" und "Buch" auf Gleichheit geprüft. Den zuvor aus der Relation "Exemplar" aussortierten ISBN-Nummern wird der entsprechende Titel zugewiesen. Der Vergleich der Attribute "MatrNr" der Relationen "leihen\_aus" und "Studierende" ordnet jedem ausgeliehenen Exemplar einen Studenten/eine Studentin zu.

Da bei Joins zuerst das kartesische Produkt gebildet wird, können bei Anfragen über mehrere umfangreiche Relationen sehr große Mengen entstehen, die anschließend den WHERE-Bedingungen entsprechend durchsucht werden müssen. In einigen Fällen ist es deshalb sinnvoll, zunächst Teilrelationen zu bilden, um die Suchmenge zu verkleinern.



Welche Studierende haben Bücher ausgeliehen und wie viele Bücher sind es pro Student/Studentin?

SQL-Anweisung:



```
SELECT s.MatrNr, Name, Vorname, COUNT (*) Buchanzahl
FROM Studierende s, leihen_aus l
WHERE s.MatrNr = l.MatrNr
GROUP BY s.MatrNr, Name, Vorname;
```

Ausgabe:

MatrNr	Name	Vorname	Buchanzahl
1562367	Schulze	Heiner	1
2358712	Meier	Siegfried	2
...	...	...	...



Da COUNT verwendet wird und das Ergebnis aus mehreren Tupeln besteht, muss es gruppiert werden. Das Anführen der Attribute Name und Vorname in der GROUP BY-Klausel hat keine Bedeutung für die Ausgabe, da zu jeder MatrNr nur ein Name und ein Vorname existiert. Aufgrund formaler Ansprüche müssen sie jedoch in der GROUP BY-Klausel angegeben werden (s.a. GROUP BY-Klausel).

### Self Joins

In self-joins werden Bezüge innerhalb einer Relation hergestellt. Hierbei wird die Relation virtuell verdoppelt, Relation a und Relation b sind identisch. Damit bei einem self-join die Bezüge stimmen, müssen Variablennamen verwendet werden.



Beispiel

Gesucht sind das Geburtsdatum von Frau König und die Namen aller Studierenden, die am selben Tag Geburtstag haben.

Name	GebDat	...
Meier	07.10.85	...
Schulze	03.05.80	...
König	07.10.85	...
Baum	12.11.82	...

Dreier	25.02.84	...
Hesse	07.10.85	...
...	...	...



Tabelle: Studierende

SQL-Anweisung:



```
SELECT a.Name, a.GebDat
FROM Studierende a, Studierende b
WHERE a.GebDat = b.GebDat
AND b.Name = 'König';
```

Ausgabe:

Name	GebDat
König	07.10.85
Meier	07.10.85
Hesse	07.10.85
...	...

Zunächst werden alle Namen und Geburtsdaten aus der Relation "Studierende a" selektiert, zu denen ein identisches Geburtsdatum in Relation "Studierende b" existiert ( $a.\text{Geburtsdatum} = b.\text{Geburtsdatum}$ ). "Meier" aus "Studierende a" lassen sich drei Tupel aus "Studierende b" zuordnen:

a.Name	a.GebDat	b.Name	b.GebDat
Meier	07.10.85	Meier	07.10.85
Meier	07.10.85	König	07.10.85
Meier	07.10.85	Hesse	07.10.85

Somit würde "Meier" dreimal in der Ergebnismenge erscheinen. Es werden jedoch nur die Tupel aus "Studierende a" ausgewählt, deren Partner in "Studierende b" für das Attribut "Name" den Wert "König" besitzen ( $b.\text{Name} = \text{'König'}$ ). Das Tupel "Meier" aus

"Studierende a" wird also nur einmal ausgewählt. Entsprechend wird mit allen anderen Tupeln verfahren.

### Outer Joins

Die bisher erläuterten Joins werden Inner Joins genannt, weil sie in der Ausgabe nur die Tupel enthalten, die einen passenden Wert in der jeweiligen Partnerrelation besitzen. Sollen z.B. alle ausgeliehenen Bücher mit ISBN, ExemplarNr und Namen der/des Studierenden aufgelistet werden, wird ein Inner Join benutzt.

In der Relation "leihen\_aus" besitzt jedes aufgelistete Buch auch eine Matrikelnummer, zu der es wiederum einen identischen Wert in der Relation "Studierende" gibt.

SQL-Anweisung:



```
SELECT l.ISBN, l.ExemplarNr, s.Name
FROM leihen_aus l, Studierende s
WHERE l.MatrNr = s.MatrNr;
```

Ausgabe:

ISBN	ExemplarNr	Name
3423101776	1	Schulze
3423202777	1	Baum
3453140982	1	Meier
3453186834	1	König
3453186834	2	Meier



Die Studenten Dreier, Hesse, Müller usw. sind nicht aufgeführt, da sie zur Zeit kein Buch geliehen haben, und ihre Matrikelnummern keinen identischen Wert in der Relation "leihen\_aus" besitzen. Für eine Liste, in der auch die Studenten enthalten sind, die zur Zeit kein Buch geliehen haben, muss ein sogenannter Outer Join durchgeführt werden. Dabei werden auch die Tupel berücksichtigt, die kein Partnertupel in der anderen Relation besitzen. In diesem Beispiel sollen alle Tupel der rechten Relation (Tabelle Studierende) aufgelistet werden, deren Matrikelnummer keine Entsprechung in der linken Relation (Tabelle leihen\_aus) hat. Dazu wird ein RIGHT OUTER JOIN angewendet.

SQL-Anweisung:



```
SELECT l.ISBN, l.ExemplarNr, s.Name
FROM leihen_aus l
RIGHT OUTER JOIN Studierende s
ON l.MatrNr = s.MatrNr;
```

Ausgabe:

ISBN	ExemplarNr	Name
3423101776	1	Schulze
3423202777	1	Baum
3453140982	1	Meier
3453186834	1	König
3453186834	2	Meier
		Dreier
		Hesse
		...



Bei einem Left Outer Join werden die Datensätze der linken (erstgenannten) Tabelle mit aufgeführt, die keine Entsprechung in der rechten Tabelle haben.

Bei einem Full Outer Join werden sowohl die Datensätze der linken Relation wie auch die Datensätze der rechten Relation mit angezeigt, die keine Entsprechung in der jeweils anderen Tabelle besitzen.

Bei Oracle werden Outer Joins durch die Zeichenkette "(+)" in der Join-Bedingung gekennzeichnet.

Die Oracle-Anweisung für den Right Outer Join des obigen Beispiels sieht folgendermaßen aus:



```
SELECT l.ISBN, l.ExemplarNr, s.Name
FROM leihen_aus l , Studierende s
WHERE l.MatrNr (+) = s.MatrNr;
```

Die Zeichenkette "(+)" bedeutet, dass allen Datensätzen, denen aus der zweiten Relation (also der rechten Tabelle) kein Datensatz zugeordnet werden kann, ein imaginärer Datensatz mit Nullwerten zugeordnet wird. Die Zeichenkette "(+)" kennzeichnet also die zusätzlichen imaginären Nullwerte und muss bei einem Right Outer Join in der WHERE-Bedingung auf der linken Seite angegeben werden.

Liste der verschiedenen Outer Joins



#### Übung

#### Übung sql.12

Erstellen Sie eine Liste der Studierenden mit ihren MatrNr, ihrem Namen, ihrem Vornamen, ihrer Strasse, ihrer PLZ und ihrem Wohnort.

#### Lösung zeigen

SQL-Anweisung:



```
SELECT s.MatrNr, Name, Vorname, a.Strasse, a.PLZ, a.Ort
FROM Studierende s, wohnen w, Adresse a
WHERE s.MatrNr = w.MatrNr
AND w.PLZ = a.PLZ
AND w.Strasse = a.Strasse
AND w.Nr = a.Nr;
```

#### SQL Interpreter

[SQLInterpreter](#) 

## 6.2.7 Unteranfragen

(Subqueries)

Im Folgenden wird die Schachtelung einzelner Anfragen erläutert. Diese Methode kann oft alternativ zu einer Anfrage mit einem Join verwendet werden. Unteranfragen werden untergliedert in Unteranfragen, die ein Tupel zurückliefern, und solche, die mehr als ein Tupel zurückliefern.

Unteranfragen sind Anfragen, deren Ergebnisse direkt in anderen Anfragen verwendet werden. Jede Anfrage kann eine Unteranfrage sein und diese kann wieder weitere subqueries enthalten. Die Unteranfrage, die für den weiterzuverwendenden Wert/die Werte steht, ist eine in Klammern gesetzte SELECT-Anweisung. Unteranfragen dürfen jedoch nur als Bestandteil einer FROM-, WHERE- oder HAVING-Klausel vorkommen. Weiterhin muss darauf geachtet werden, dass nur gleiche Datentypen einander zugeordnet werden.

### Unteranfragen, die ein Tupel liefern

MatrNr	Name	GebDat
1258964	Meier	07.10.1985
6789534	Schulze	03.05.1980
5349768	König	07.10.1985
6345212	Dreyer	25.02.1984
...	...	...



**Tabelle Studierende**  
aus der Miniwelt Hochschule



#### Beispiel

Gesucht sind alle Studierenden, die das gleiche Geburtsdatum haben wie die Studentin mit der Matrikelnummer 6432753.

SQL-Anweisung:



```
SELECT MatrNr, Name, GebDat
FROM Studierende
WHERE GebDat =
  (SELECT GebDat
   FROM Studierende
```

```
WHERE MatrNr = '6432753');
```

Zunächst wird in der Unteranfrage das Geburtsdatum der Studentin mit der MatrNr 6432753 ermittelt. Die Unteranfrage liefert den Wert 07.10.1985. Der ermittelte Wert wird anschließend in der Hauptanfrage benutzt, um alle Studierenden mit dem gleichen Geburtsdatum zu bestimmen.

Ausgabe:

MatrNr	Name	GebDat
2358712	Meier	07.10.1985
6432753	König	07.10.1985
5236478	Hesse	07.10.1985



Beispiel

Gesucht sind alle Studierenden, deren Fachsemesteranzahl über dem Durchschnitt liegt.

MatrNr	Name	Semester
2358712	Meier	5
6789534	Schulze	10
6432753	König	12
6345212	Dreyer	8
5236478	Hesse	2
...	...	...



**Tabelle Studierende  
aus der Miniwelt Hochschule**

SQL-Anweisung:



```
SELECT MatrNr, Name, Semester  
FROM Studierende
```



```
WHERE Semester >
  (SELECT AVG (Semester)
   FROM Studierende);
```

### Unteranfragen, die mehr als ein Tupel liefern

Wenn die Unteranfrage mehr als einen Wert liefert, müssen in der WHERE-Klausel der Hauptanfrage Mengenoperatoren oder die Booleschen Operatoren EXISTS und IN eingesetzt werden. Mengenoperatoren sind ANY, SOME und ALL in Verbindung mit Vergleichsoperatoren (=, <>, >, <, >=, <=).



Welche Studierenden besuchen die Lehrveranstaltung "Datenbanken"?

SQL-Anweisung:



```
SELECT MatrNr, Name
FROM Studierende
WHERE MatrNr IN
  (SELECT MatrNr
   FROM besuchen
   WHERE LvNr IN
    (SELECT LVNr
     FROM LV
     WHERE LVBezeichnung = 'Datenbanken'));
```

In der letzten Unteranfrage wird zunächst die Lehrveranstaltungsnummer für die Vorlesung "Datenbanken" ermittelt. Mit Hilfe den in dieser Unteranfrage ermittelten Lehrveranstaltungsnummer werden in der ersten Unteranfrage in der Relation "besuchen" die Matrikelnummern ermittelt, die zu der zuvor ausgesuchten Lehrveranstaltungsnummer gehören. In der Hauptanfrage werden dann über diese Matrikelnummern die Namen der Studierenden ermittelt, die die betreffende Lehrveranstaltung besuchen.

Diese Anfrage kann alternativ auch durch einen Verbund erreicht werden. Die SQL-Anweisung dazu lautet:



```
SELECT s.MatrNr, Name
FROM Studierende s, besuchen b, LV
WHERE s.MatrNr = b.MatrNr
AND b.LVNr = LV.LVNr
AND LV.LVBezeichnung = 'Datenbanken';
```

Bei einer Anfrage durch einen Verbund wird zunächst das kartesische Produkt der beteiligten Relationen gebildet und dieses dann auf die Bedingungen hin durchsucht. Bei großen Relationen bedeutet das einen großen Aufwand.

**Beispiel**

Gesucht sind alle Studierenden, die zur Zeit kein Buch geliehen haben.

SQL-Anweisung:



```
SELECT MatrNr, Name
FROM Studierende s
WHERE NOT EXISTS
  (SELECT MatrNr
   FROM leihen_aus l
   WHERE l.MatrNr = s.MatrNr);
```

In der Unteranfrage werden alle Matrikelnummern ausgewählt, die sich sowohl in der Relation "Studierende" als auch in der Relation "leihen\_aus" befinden. In der Hauptanfrage werden die Studierenden selektiert, die sich nicht in der Ergebnismenge der Unteranfrage befinden. Bei Unteranfragen mit EXISTS ist die Unteranfrage immer in Abhängigkeit zur Hauptanfrage zu formulieren. Im Beispiel wurde die Verbindung zwischen Unter- und Hauptanfrage durch den Vergleich der Matrikelnummern hergestellt.

**Der Operator ALL**

Mit ALL (engl.: alle) wird in der Unterabfrage ein Vergleichswert aus einer Gruppe abgefragt.



Beispiel

Welche Studierende (Name, Vorname und MatrNr) haben länger studiert als der Studierende mit der längsten Studiendauer im Studiengang Medieninformatik?

Name	Vorname	MatrNr
König	Mathilde	6432753
Schmitt	Marc	9365461
Meier	Martina	3108642



**Tabelle Studierende**  
aus der Miniwelt Hochschule

SQL-Anweisung:



```
SELECT Name, Vorname, MatrNr
FROM Studierende
WHERE Semester > ALL
(SELECT Semester
FROM Studierende
WHERE Studiengang = 'MI');
```

Die folgende Übersicht zeigt die Wirkungsweise der Vergleichsoperatoren mit ALL allgemein an:

Operatoren:	Auswahl in der Vergleichsgruppe:	Anzeige mit der Hauptabfrage: Alle Tupeln mit ...
> ALL	höchster Wert	größeren Vergleichswerten
>= ALL	höchster Wert	größeren und gleichen Vergleichswerten
< ALL	kleinster Wert	geringeren Vergleichswerten
<= ALL	kleinster Wert	geringeren und gleichen Vergleichswerten



### Die Wirkungsweise des ALL-Operators

#### **Zeichenketten-Vergleich**

Bei Zeichenfeldern tritt an die Stelle größerer und kleinerer numerischer Werte die Sortierordnung im ASCII-Code.

#### **Datumsvergleich**

Bei Datumsfeldern wird das größere oder kleinere Datum verglichen.

#### **Der Operator ANY**

Der Operator ANY (engl.: irgendeiner) fragt bei "> ANY" in der Unterabfrage ab, welcher Wert größer als irgendein Wert in der Vergleichsgruppe ist. Das ist jeder Wert, der größer ist als der kleinste Gruppenwert. Bei "< ANY" lautet die Frage: Welcher Wert ist kleiner als irgendein Wert in der Vergleichsgruppe? Das ist jeder Wert, der kleiner ist als der größte Gruppenwert.



Beispiel

Welche Studierende (Name, Vorname und MatrNr) haben länger studiert als der Studierende mit der kürzesten Studiendauer im Studiengang Medieninformatik?

Name	Vorname	MatrNr
Schulze	Heiner	1562367
König	Mathilde	6432753
Dreier	Magnus	2356984
Schmitt	Marc	9365461
Müller	Udo	4297531
Meier	Martina	3108642
Thiess	Hugo	1230789
Zander	Wolfgang	1286385



**Tabelle Studierende**  
aus der Miniwelt Hochschule

SQL-Anweisung:



```
SELECT Name, Vorname, MatrNr
FROM Studierende
WHERE Semester > ANY
(SELECT Semester
FROM Studierende
WHERE Studiengang = 'MI');
```

Auch der ANY-Operator kann mit dem Zeichenketten- und Datumsvergleich kombiniert werden. Die folgende Übersicht zeigt die Wirkungsweise der Vergleichsoperatoren mit ANY allgemein an:

Operatoren:	Auswahl in der Vergleichsgruppe:	Anzeige mit der Hauptabfrage: Alle Tupeln mit ...
> ANY	kleinster Wert	größeren Vergleichswerten
>= ANY	kleinster Wert	größeren und gleichen Vergleichswerten
< ANY	größter Wert	geringeren Vergleichswerten
<= ANY	größter Wert	geringeren und gleichen Vergleichswerten



Die Wirkungsweise des ANY-Operators

### Unteranfragen mit IN und EXISTS

Die Operatoren IN und EXISTS prüfen in der Unterabfrage, ob eine in der Hauptabfrage gestellte Bedingung erfüllt wird.

#### *Der Operator IN*

Der Operator IN wird in der Regel zur Suche in anderen Relationen eingesetzt. Er prüft nacheinander für jeden Wert aus dem Vergleichsattribut der Hauptrelation, ob dieser Wert in dem Vergleichsattribut der Unterrelation steht, bei NOT IN, ob er nicht darin

steht. Die Namen der Vergleichsattribute in der Hauptabfrage und in der Unterabfrage müssen gleich sein.

Die Hauptabfrage kann sich auf mehrere Attribute und Relationen beziehen. In der Unterabfrage ist auch eine WHERE-Klausel zulässig.

Der Unterabfrage-Operator IN erzeugt ein Abfrageergebnis, das auch mit dem Operator = ANY oder bei Unteranfragen mit einer einzigen Ergebnisreihe mit dem Operator = erreicht wird.

### ***Der Operator EXISTS***

Wenn in der Unterabfrage nur geprüft werden soll, ob darin gültige Vergleichswerte zum aktuellen Wert der Hauptabfrage existieren, so wird der Operator EXISTS verwendet.

Hinter FROM darf nur eine einzige Relation aufgeführt werden, weil der EXISTS-Operator keine Verbund-Bedingungen zulässt.

Mit NOT EXISTS wird die Komplementärmenge der Relation angezeigt. Die Unterabfrage mit EXISTS wird in der Regel mit SELECT \* eingeleitet. An die Stelle des Sterns kann irgendein Attributname aus der Hauptrelation treten.

Da bei EXISTS kein Vergleichsattribut die Verbindung zwischen Haupt- und Unterabfrage herstellt, muss die Verknüpfung in der *WHERE-Bedingung der Unterabfrage* vereinbart werden. Der Operator EXISTS liefert den booleschen Wert "wahr" oder "falsch" als Ergebnis.



Beispiel

Liste aller Bücher (nicht Buchexemplare!), die verliehen sind. Titel, ISBN, Leihfrist sind auszugeben.

Titel	ISBN	Leihfrist
Solaris	3423101776	30
Der Wüstenplanet	3453186834	30
Der kleine Hobbit	3423202777	30
Das Vogelmädchen	3453140982	30



Tabellen Buch und leihen\_aus

SQL-Anweisung:



```
SELECT Titel, ISBN, Leihfrist
FROM Buch b
WHERE EXISTS
(SELECT *
FROM leihen_aus la
WHERE b.ISBN = la.ISBN);
```



## Übung

**Übung sql.13**

Gesucht sind alle Studierenden mit MatrNr, Name und Vorname, die in Bruchtal oder Brewald wohnen.

**Lösung zeigen**

SQL-Anweisung:



```
SELECT MatrNr, Name, Vorname
FROM Studierende
WHERE MatrNr IN
(SELECT MatrNr
FROM wohnen
WHERE PLZ IN
(SELECT PLZ
FROM Adresse
WHERE Ort IN ('Bruchtal', 'Brewald'))
AND Strasse IN
(SELECT Strasse
FROM Adresse
WHERE Ort IN ('Bruchtal', 'Brewald'))
AND Nr IN
```

```
(SELECT Nr  
FROM Adresse  
WHERE Ort IN ('Bruchtal', 'Brewald')));
```

**SQL Interpreter**[SQLInterpreter](#) 

## 6.2.8 Kombination von unabhängigen Unterabfragen

Im Folgenden wird die Bildung von Vereinigungsmengen erläutert.

Der UNION-Operator vereinigt die Ergebnisse mehrerer SELECT-Anweisungen. Doppelte Tupel werden dabei automatisch zusammengefasst (außer bei der Verwendung von UNION ALL). Der Union-Operator wird zwischen den einzelnen SELECT-Anweisungen platziert. Dabei können zwei oder mehrere Relationen miteinander verknüpft werden.

**SYNTAX**

Syntax des UNION-Operators:



```
SELECT-Anweisung  
UNION SELECT-Anweisung  
[UNION SELECT-Anweisung, ... ]  
[ORDER BY-Klausel]
```



Die einzelnen SELECT-Anweisungen können Unterabfragen und alle anderen bekannten Klauseln enthalten. Einzige Ausnahme ist die ORDER BY-Klausel, die nur am Ende verwendet werden darf. Die Anzahl der Attribute und die Datentypen müssen kombinierbar sein.

MatrNr	Name	Vorname
2358712	Meier	Siegfried



1562367	Schulze	Heiner
6432753	König	Mathilde
...	...	...



**Tabelle Studierende**  
aus der Miniwelt Hochschule

<b>PersNr</b>	<b>Name</b>	<b>Vorname</b>
5234260	Zimmer	Monika
9652425	Irrgang	Rolf
...	...	...



**Tabelle Lehrende**  
aus der Miniwelt Hochschule



Beispiel

Es soll eine Liste mit den Namen aller Studierenden und aller Lehrenden erstellt werden.

SQL-Anweisung:



```
SELECT Name, Vorname
FROM Studierende
UNION
SELECT Name, Vorname
FROM Lehrende;
```

Ausgabe:

<b>Name</b>	<b>Vorname</b>
Baum	Meta
Dreier	Magnus
Hesse	Sarah
Irrgang	Rolf
...	...



Die Verknüpfung von unabhängigen Anfragen können ebenfalls mit den Mengenoperationen INTERSECT (Bildung der Durchschnittsmenge) und EXCEPT (Bildung der Differenzmenge) durchgeführt werden.

## 6.3 Datenbankänderungen



Gliederung

### 6.3 Datenbankänderungen

#### 6.3.1 Dateneingabe

#### 6.3.2 Daten ändern

#### 6.3.3 Daten löschen

### 6.3.1 Dateneingabe

#### (INSERT-Anweisung)

Im Folgenden wird gezeigt, wie die zuvor erstellten Relationen mit Daten gefüllt werden. Mit der INSERT-Anweisung werden Daten tupelweise (zeilenweise) in eine Relation eingefügt. Vor dem Speichern des neuen Tupel prüft die Datenbank alle bei der Relationendefinition festgelegten Einschränkungen (UNIQUE, CHECK, NULL). Werden die Einschränkungen nicht eingehalten, bricht das DBMS die Operation ab.



Beispiel

In die Relation "Buch" sollen folgende Werte eingegeben werden:

ISBN	Titel	Exemplare	Seitenanzahl	Leihfrist
3423101776	Solaris	5	236	30



SQL-Anweisung:



```
INSERT INTO Buch
(ISBN, Titel, Exemplare, Seitenanzahl, Leihfrist)
VALUES
('3423101776', 'Solaris', 5, 236, 30);
```



SYNTAX

Syntax einer INSERT-Anweisung:



```
INSERT INTO Relationenname [ Attributliste ]
VALUES (Werteliste);
```



Nach INSERT INTO folgt der Name der Relation, in die ein neuer Datensatz eingetragen werden soll. Danach werden in Klammern und durch Komma getrennt die Namen der Attribute aufgelistet, in die neue Einträge erfolgen sollen. Nach dem Schlüsselwort VALUES folgt in Klammern eine Liste der einzutragenden Werte. Alphanumerische Werte und Werte vom Typ DATE müssen in Hochkommata (') eingeschlossen angegeben werden. Ein Wert muss vom gleichen Datentyp sein wie das Attribut, in das er eingetragen werden soll. Die Liste der Werte muss in Anzahl und Reihenfolge mit der Liste der Attribute übereinstimmen. Die Zuordnung erfolgt über die Reihenfolge (der dritte Wert gehört zu dem an dritter Stelle angegebenen Attribut). Für jedes als NOT NULL definiertes Attribut muss auf jeden Fall ein Wert angegeben werden.

Wenn für alle Attribute ein Wert angegeben wird und dabei die ursprüngliche Reihenfolge der Attributdefinition eingehalten wird, kann die Attributliste entfallen.

SQL-Anweisung:



```
INSERT INTO Buch
VALUES
('3423101776', 'Solaris', 5, 236, 30);
```

Zum Einfügen von Daten, die bereits in anderen Relationen der Datenbank verfügbar sind, wird eine etwas andere INSERT-Anweisung verwendet.



Beispiel

In die folgende Relation sollen die entsprechenden Werte aus der Relation "Buch" eingetragen werden.

ISBN_Neu	Titel_Neu	Seitenanzahl_Neu
...	...	...



Tabelle Buch\_Neu

SQL-Anweisung:



```
INSERT INTO Buch_Neu
  (ISBN_Neu, Titel_Neu, Seitenanzahl_Neu)
SELECT ISBN, Titel, Seitenanzahl
FROM Buch;
```

Nach INSERT INTO werden zunächst der Name der neuen Relation und die Attributliste angegeben. Anschließend werden nach SELECT die zu übertragene Werte angegeben. Nach FROM folgt der Name der Relation, aus der die Werte stammen sollen.

Die Tabelle "Buch\_Neu" sieht dann wie folgt aus:

ISBN_Neu	Titel_Neu	Seitenanzahl_Neu
3802551230	Pustebblume	22
3100101065	Die Pest	362
...	...	...
3546001184	Solaris	236



Tabelle Buch\_Neu



SYNTAX

## Syntax der INSERT-Anweisung für Umspeicherungen



```
INSERT INTO Relationenname  
[Attributliste]  
SELECT-Anweisung;
```



Dabei ist darauf zu achten, dass die Relation, die in der SELECT-Anweisung angesprochen wird, nicht dieselbe ist, die nach INSERT INTO angegeben wird.



Übung

## Übung sql.14

Legen Sie ein Haushaltsbuch an. Erstellen Sie zu diesem Zweck eine Relation "Ausgaben\_IhreMatrikelnummer" mit den Attributen "Nr", "Datum", "Geschäft", "Ware" und "Preis". Füllen Sie die Relation mit Daten.

Die Relation kann durch den Befehl



```
select * from Ausgaben_IhreMatrikelnummer;
```

angezeigt werden.

## Lösung zeigen

SQL-Anweisung 1:



```
CREATE TABLE Ausgaben_45873  
(Nr VARCHAR(20) NOT NULL,  
Datum VARCHAR(10),
```

```

Geschäft VARCHAR(20) NOT NULL,
Ware VARCHAR(20) NOT NULL,
Preis VARCHAR(10) NOT NULL,
PRIMARY KEY (Nr));

```

SQL-Anweisung 2:



```

INSERT INTO Ausgaben_45873
VALUES ('1', '30.12.2002', 'Schröders', 'Socken', '8');

```

**SQL Interpreter**

| [SQLInterpreter](#)

## 6.3.2 Daten ändern

### (UPDATE-Anweisung)

Um Datensätze zu ändern, die bereits in einer Relation vorhanden sind, wird die UPDATE-Anweisung benutzt. Einem oder mehreren Attributen wird ein neuer Wert zugewiesen. Mit Hilfe der WHERE-Klausel wird dabei festgelegt, welche Tupel von der Änderung betroffen sein sollen. Ansonsten werden alle Tupel geändert.

ISBN	Titel	Exemplare	Seitenanzahl	Leihfrist
3802551230	Pusteblume	2	22	15
3100101065	Die Pest	3	362	30
...	...	...	...	...



**Tabelle Buch**  
aus der Miniwelt Hochschule



**Beispiel**

In das Attribut "Titel" der Relation "Buch" soll der Wert "Solaris" eingepflegt werden.

SQL-Anweisung:



```
UPDATE Buch
SET Titel = 'Solaris'
WHERE ISBN = '3423101776';
```

**Beispiel**

Der Eintrag "Die Pest" ist falsch. Der Titel muss geändert werden in "Der Ekel". Die Anzahl der Exemplare beträgt 4. Die Seitenanzahl beträgt 347.

SQL-Anweisung:



```
UPDATE Buch
SET Titel = 'Der EKEL',
Exemplare = 4,
Seitenanzahl = 347
WHERE ISBN = '3100101065';
```

Nach dem Schlüsselwort UPDATE folgt der Name der zu ändernden Relation. Nach dem Schlüsselwort SET folgen die Namen der Attribute, die geändert werden sollen. Die neuen Werte werden den Attributen durch Gleichheitszeichen zugewiesen. Für den Fall, dass nur bestimmte Tupel geändert werden sollen, folgt noch eine WHERE-Bedingung.

**SYNTAX**

Syntax der UPDATE-Anweisung:



```
UPDATE Relationenname
SET Attributname = Wert,
Attributname = Wert, ...
[WHERE -Bedingung];
```





## Übung

**Übung sql.15**

Ändern Sie Ihre Relation "Ausgaben\_IhreMatrikelnummer". Der Wert des Attributes "Preis" soll für den ersten Eintrag 10 betragen. Lassen Sie sich die Relation anzeigen.

**Lösung zeigen**

SQL-Anweisung:



```
UPDATE Ausgaben_45873
SET Preis = '10'
WHERE Nr = '1';
```

**SQL Interpreter**

| [SQLInterpreter](#) 

**6.3.3 Daten löschen**

Mit der DELETE-Anweisung können einzelne Tupel oder Inhalte einer ganzen Relation gelöscht werden.



Wie werden alle Einträge aus der Relation Buch zum Buch Pustebblume gelöscht?

SQL-Anweisung:



```
DELETE FROM Buch
WHERE Titel = 'Pustebblume';
```





SYNTAX

Syntax einer DELETE-Anweisung:



```
DELETE FROM Relationenname  
[WHERE -Bedingung];
```



Ohne die WHERE-Bedingung wird der gesamte Relationeninhalt gelöscht.



Übung

### Übung sql.16

Löschen Sie das erste Tupel der Relation "Ausgaben\_IhreMatrikelnummer".

#### Lösung zeigen

SQL-Anweisung:



```
DELETE FROM Ausgaben_45873  
WHERE Nr = '1';
```

#### SQL Interpreter

[| SQLInterpreter](#)

## 6.4 Weiterführende Informationen

Im Folgenden finden Sie einige weiterführende Informationen, auf die bereits im Laufe der Lerneinheit verwiesen wurde:

"[SQL-Anweisungsgruppen](#)" bietet eine tabellarische Übersicht über alle SQL-Anweisungen.

Unter "SQL-Operatoren" sind alle SQL-Operatoren mit Bedeutung und Priorität aufgelistet.

Unter "WHERE-Bedingung" wird ausführlich der Aufbau einer WHERE-Bedingung erläutert.

Unter "Outer Joins" werden die verschiedenen Typen von Outer Joins, ihr Aufbau und ihr Ergebnis übersichtlich dargestellt.



Gliederung

## 6.4 Weiterführende Informationen

### 6.4.1 SQL-Anweisungsgruppen

#### 6.4.2 SQL-Operatoren

#### 6.4.3 WHERE-Bedingung

#### 6.4.4 Outer Joins

## 6.4.1 SQL-Anweisungsgruppen

### *Anweisungen in der Datendefinitionssprache (DDL)*

CREATE TABLE	erzeugt eine Relation
DROP TABLE	löscht eine Relation
ALTER TABLE	ändert eine Tabellenstruktur
CREATE VIEW	erzeugt eine Sicht
DROP VIEW	löscht eine Sicht
CREATE INDEX	erzeugt einen Index
DROP INDEX	löscht einen Index



### *Anweisungen in der Datenmanipulationssprache (DML und DQL)*

INSERT	trägt einen Datensatz ein
UPDATE	ändert einen Datensatz
DELETE	löscht einen Datensatz
SELECT	stellt eine Anfrage an die Datenbank



### Anweisungen zur Zugriffssteuerung (DCL)

GRANT	gibt Benutzern Erlaubnis
REVOKE	entzieht Benutzern Erlaubnis



### Anweisungen zur Transaktionssteuerung

COMMIT	beendet eine laufende Transaktion
ROLLBACK	macht eine laufende Transaktion rückgängig, falls sie nicht erfolgreich beendet werden kann



## 6.4.2 SQL-Operatoren

Die Priorität gibt an, in welcher Reihenfolge mehrere in einer Bedingung enthaltene Operatoren abgearbeitet werden.

Operator	Bedeutung	Priorität
+	Addition	1
-	Subtraktion	1
•	Multiplikation	0
/	Division	0
=	gleich	2
!=	ungleich	2
>	größer	2
<	kleiner	2
>=	größer gleich	2
<=	kleiner gleich	2
AND	logisches UND	4

OR	logisches ODER	5
NOT	Negation	3
[NOT] IN	[nicht] in der Menge	2
IS [NOT] NULL	[nicht] NULL-Wert	2
[NOT] BETWEEN ... AND ...	[nicht] zwischen ... und ...	2

**SQL-Operatoren****6.4.3 WHERE-Bedingung**

Eine WHERE-Bedingung kann sich aus zwei miteinander verbundenen logischen Ausdrücken zusammensetzen.

logischer Ausdruck 1		Verbindung		logischer Ausdruck 2
-------------------------	--	------------	--	-------------------------



Die Verbindung kann bestehen aus AND oder OR oder AND NOT oder OR NOT.

logischer Ausdruck 1		{AND   OR} [NOT]		logischer Ausdruck 2
-------------------------	--	---------------------	--	-------------------------



Auch die logischen Ausdrücke können verneint sein.

[NOT]	logischer Ausdruck 1		{AND   OR} [NOT]		logischer Ausdruck 2
-------	-------------------------	--	---------------------	--	-------------------------

**Syntax der WHERE-Bedingung**



```
WHERE [NOT] log. Ausdruck 1 [{AND | OR}[NOT] log. Ausdruck 2]
```



Eine WHERE-Bedingung muss mindestens aus einem logischen Ausdruck bestehen. Alle anderen Bestandteile sind optional.



#### Beispiel



```
SELECT Titel
FROM Buch
WHERE Seitenanzahl > 20 AND NOT Titel = 'Die Pest';
```

In diesem Beispiel ist *Seitenanzahl > 20* der erste logische Ausdruck, AND ist die Verbindung und NOT *Titel = 'Die Pest'* ist der zweite logische Ausdruck.

Ein logischer Ausdruck wiederum setzt sich zusammen aus einem Ausdruck 1, einem Vergleichsoperator und einem Ausdruck 2 oder einem Bereichstest oder einem Elementtest oder einem Spaltentest.

Ausdruck 1		Vergleichsoperator		Ausdruck 2
				Bereichstest
				Elementtest
				Spaltentest



Vergleichsoperatoren: <, >, =, <>, <=, >=

Bereichstest: ... [NOT] BETWEEN ... AND ...

Elementtest: ... [NOT] IN Wert [ , Wert, ... ]

Spaltentest: Attributname [NOT] LIKE Muster

Im obigen Beispiel ist *Seitenanzahl* der erste Ausdruck, > ist der Vergleichsoperator und 20 ist der zweite Ausdruck.

### 6.4.4 Outer Joins

#### ***RIGHT OUTER JOIN***

In der als zweite genannten Relation (rechte Tabelle) werden Tupel ohne Entsprechung in der anderen Relation übernommen.

SELECT	Attributliste
FROM	Relationname1
RIGHT OUTER JOIN	Relationname2
ON	Bedingung (z.B. ein Vergleich)

#### ***LEFT OUTER JOIN***

In der zuerst genannten Relation (linke Tabelle) werden Tupel ohne Entsprechung in der anderen Relation übernommen.

SELECT	Attributliste
FROM	Relationname1
LEFT OUTER JOIN	Relationname2
ON	Bedingung (z.B. ein Vergleich)

#### ***FULL OUTER JOIN***

Es werden Tupel aus beiden Relationen, die keine Entsprechung in der Partnerrelation besitzen, mit aufgeführt.

SELECT	Attributliste
FROM	Relationname1
FULL OUTER JOIN	Relationname2
ON	Bedingung (z.B. ein Vergleich)

Alle Klauseln der SELECT-Anweisung (WHERE, GROUP BY usw.) sind auch auf Relationverbünde anwendbar.

## 6.5 Freitextaufgaben zu Structured Query Language

1. Erläutern Sie die Bedeutung des NULL-Wertes.

### Lösung zeigen

Der NULL-Wert bedeutet, dass an dieser Stelle der Wert für das Attribut fehlt. NULL ist nicht identisch mit dem numerischen Wert 0 oder dem Leerzeichen. Der numerische Wert 0 oder ein Leerzeichen werden in Form eines Binärwertes in der Zelle gespeichert. Bei NULL wird nichts gespeichert.

2. Nennen Sie den Datentyp für ganze Zahlen mit Vorzeichen und für eine Zeichenkette mit maximal n Zeichen.

### Lösung zeigen

Der Datentyp für ganze Zahlen mit Vorzeichen heißt integer.

Der Datentyp für eine Zeichenkette mit maximal n Zeichen heißt varchar(n).

3. Warum werden Attributwerte zu den Attributen, die nachträglich in eine Relationendefinition eingefügt werden, nicht mit der INSERT-Anweisung sondern mit der UPDATE-Anweisung belegt?

### Lösung zeigen

Ein angefügtes Attribut, das bereits Daten enthält, bekommt in allen Zellen den Wert NULL. Deshalb werden die Zellen der neu angefügten Attribute (auch wenn diese noch keine Daten enthalten) nicht mit der INSERT-Anweisung gefüllt, sondern mit der UPDATE-Anweisung geändert.

4. Welche Klauseln sind innerhalb einer SELECT-Anweisung möglich?

### Lösung zeigen

Folgende Klauseln sind innerhalb einer SELECT-Anweisung möglich:

- die WHERE-Klausel

- die GROUP BY-Klausel
- die HAVING-Klausel (nur in Verbindung mit der GROUP-BY-Klausel)
- die ORDER BY-Klausel

5. Erläutern Sie, was bei einem Self-Join geschieht.

**Lösung zeigen**

In self-joins werden Bezüge innerhalb einer Relation hergestellt.

Hierbei wird die Relation virtuell verdoppelt, Relation a und Relation b sind identisch. Damit bei einem self-join die Bezüge stimmen, müssen Variablennamen verwendet werden.

6. Erläutern Sie den Unterschied zwischen einem Outer-Join und einem Inner-Join.

**Lösung zeigen**

Inner-Joins enthalten in der Ausgabe nur die Tupel, die einen passenden Wert in der jeweiligen Partnerrelation besitzen. Im Gegensatz dazu werden z.B. in einem Right Outer Join alle Tupel der rechten Relation aufgelistet, auch wenn es hierzu keine Entsprechung in der linken Relation gibt.

7. Unterabfragen dürfen nur als Bestandteile von Klauseln vorkommen. Nennen Sie diese.

**Lösung zeigen**

Unterabfragen dürfen nur als Bestandteil einer FROM-, WHERE- oder HAVING-Klausel vorkommen. Weiterhin muss darauf geachtet werden, dass nur gleiche Datentypen einander zugeordnet werden.

8. Welche Anweisung wird verwendet, um lediglich den Inhalt, nicht jedoch die Definition einer Relation zu löschen?

**Lösung zeigen**

Mit der DELETE-Anweisung wird der Inhalt einer Relation, nicht jedoch ihre Relationendefinition (Attribute, Datentypen, Schlüssel usw.) gelöscht.



## 7 Sichten, Rechtevergabe, Integrität



### Zeitumfang

Die voraussichtliche Bearbeitungsdauer dieser Lerneinheit (ohne Übungsaufgaben!) beträgt ca. 5 Stunden.



### Lernziele

Gemäß der Drei-Ebenen-Schemaarchitektur bietet jedes Datenbankmanagementsystem anwendungsunabhängige Schnittstellen. Eine dieser Möglichkeiten bieten Sichten an. Mit Hilfe von Sichtdefinitionen kann ein Ausschnitt von Relationen festgelegt werden, der von einer Personengruppe gelesen bzw. geändert werden darf. Damit kann Datenschutz erreicht werden. Des weiteren können häufig wiederkehrende und komplizierte SQL-Anfragen vorformuliert werden, um ungeschulten Personen die Nutzung der Datenbank zu erleichtern.

Verstärkt werden Schutzmaßnahmen durch Rechtevergabe ("wer darf was sehen?"). Außerdem wird Datenkonsistenz durch die für jede Anwendung formulierten Integritätsbedingungen (z.B. "das Geburtsdatum darf nicht nach dem aktuellen Datum liegen") gewährleistet.



### Gliederung

#### 7 Sichten, Rechtevergabe, Integrität

##### 7.1 Sichten

##### 7.2 Rechtevergabe

##### 7.3 Integrität

##### 7.4 Syntax-Diagramme

##### 7.5 Freitextaufgaben zu Sichten, Rechtevergabe, Integrität

## 7.1 Sichten

Ein wichtiges Konzept, um ein Datenbanksystem an die Bedürfnisse unterschiedlicher Benutzergruppen anpassen zu können, sind Sichten (engl. views). Eine Sicht beschreibt eine für eine Personengruppe interessante Datenmenge.



Hierbei ist nicht nur wichtig festzustellen, welche Daten ein Benutzer sehen will, sondern auch, welche er nicht sehen darf.

Dies wird im allgemeinen durch Sichten erreicht. Sie bieten virtuelle Relationen an, die nur einen Ausschnitt des gesamten Modells zeigen. Unter "virtuell" ist in diesem Zusammenhang zu verstehen, dass keine neuen Relationen angelegt werden sondern sie bei jeder Verwendung neu generiert werden. Sie spiegeln also stets den aktuellen Zustand der ihnen zugrunde liegenden Relationen wieder.

**Sichten** sind externe Datenbank-Schemata folgend der Drei-Ebenen-Schemaarchitektur. Die Relationenschemata werden implizit verwendet, um Anfragen als Makros zu vereinfachen, und explizit, um auf der externen Ebene einer Datenbank Daten nach bestimmten Kriterien sichtbar zu machen bzw. Daten unsichtbar zu halten. Eine Sicht kann auch eine Berechnungsvorschrift für virtuelle Relationen beinhalten. Z.B. SQL-Anweisungen, die Änderungen auf Sichten bewirken, bewirken gleichzeitig auch Änderungen in den echten Relationen und umgekehrt.

### Sichtdefinition

Sichten können jederzeit aus den Basisrelationen definiert werden. Dazu dient die folgende Syntax:



SYNTAX

Syntax von Sichten:



```
CREATE VIEW sichtname
[(sichtattributname, sichtattributname, ...)] AS
SELECT [attributname, attributname, ...]
FROM relationsname [, relationsname, ...]
[WHERE bedingung]
[GROUP BY attributname, [HAVING bedingung]]
[WITH [CASCADED | LOCAL] CHECK OPTION]
```



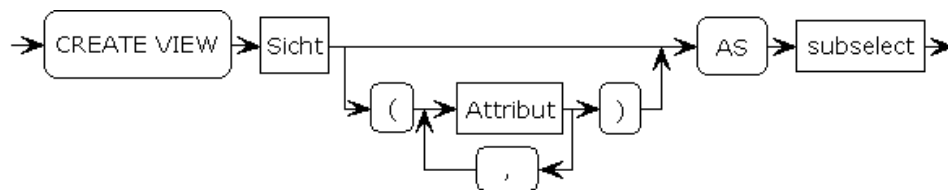
Syntax von Sichten



Die ORDER BY- Klausel kann in Sichten nicht verwendet werden.

### **Syntax-Grafik der CREATE VIEW-Anweisung:**

Die Abbildung *sri.1: CREATE VIEW-Syntax* zeigt eine vereinfachte Syntax zum Einrichten von Sichten.



**Abb.sri.1: CREATE VIEW-Syntax**

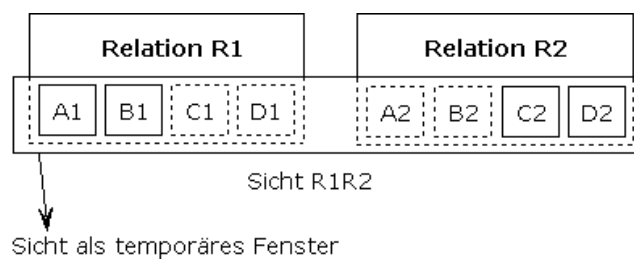
Eine einfache Definition einer Sicht könnte z.B. so aussehen:



```

CREATE VIEW R1R2 AS
SELECT A1 ,B1 ,C2 ,D2
FROM R1 ,R2 ;
  
```

Mit einer Grafik verdeutlicht:



**Abbildung sri.2: Beispielsicht R1R2**

Wie in *Abbildung sri.2* zu sehen ist, bindet die Sicht R1R2 die Attribute A1 und B1 von Relation R1 sowie C2 und D2 von Relation R2 ein und erstellt hieraus eine neue Ansicht für eine Benutzergruppe.

Eine kleine Übersicht der am meisten verwendeten Anweisungen:








SQL-Anweisungen	Beschreibung	Beispiel
 <pre>SELECT * FROM sichtname</pre>	Sicht ausgeben	 <pre>SELECT * FROM Mitarbeiter</pre>
 <pre>INSERT INTO sichtname VALUES (DatenWert1, DatenWert2, ...)</pre>	Tupel in eine Sicht einfügen	 <pre>INSERT INTO Mitarbeiter</pre>
 <pre>UPDATE sichtname SET anweisung WHERE bedingung</pre>	Spaltenwert ändern	 <pre>UPDATE Mitarbeiter SET Gehalt = 4500 WHERE PersNr = 9;</pre>
 <pre>DELETE FROM sichtname WHERE bedingung</pre>	Zeilen löschen	 <pre>DELETE FROM Mitarbeiter</pre>
 <pre>DROP VIEW sichtname</pre>	Sicht löschen	 <pre>DROP VIEW Mitarbeiter;</pre>



Tabelle sri.1: Oracle SQL-Anweisungen bezüglich Sichten

**Vorteile:**

- Vereinfachung von Anfragen  
Häufige/komplizierte Anfragen eines Benutzers können als Sicht implementiert werden, so dass diese über eine Sicht beantwortet werden.
- Strukturierung der Datenbank
- Logische Datenunabhängigkeit  
Die Schnittstelle zwischen Anwendung und Datenbank bleibt stabil. Somit können kleinere Änderungen der Datenbankstruktur vor dem Anwender verborgen werden.
- Beschränkung von Zugriffen

Es ist nicht explizit vorgesehen, auf nur eine Spalte einer Relation spezielle Rechte mit der GRANT SELECT-Anweisung zu vergeben. Um dies dennoch zu ermöglichen, wird eine Sicht auf diese Spalte erzeugt und dann werden die Rechte über die Sicht an den betreffenden Benutzer zugewiesen.

**Nachteile:**

- Sichten benötigen längere Bearbeitungszeiten.
- Sichten legen im Ablauf der Anweisungsdurchführung mehrere temporäre Tabellen an, was bei fehlendem Speicherplatz zu einem nicht ordnungsgemäßen Ausführen einer Anweisung führen kann.

**Problem:**

- Durchführung von Änderungen auf Sichten  
Sichten haben das Problem, dass sie nicht immer änderbar (update-fähig) sind. Ein anschauliches Beispiel ist die folgende Sicht:



```
CREATE VIEW Durchschnittsnote(PersNr, Note) AS
  SELECT PersNr, avg(Note)
  FROM Prüfen
  GROUP BY PersNr;
```

Der Grund, warum diese Sicht nicht veränderbar ist, liegt im Attribut *Note*, das mit der Aggregierungsfunktion "avg" erzeugt wird. Eine Änderungsoperation lässt sich hier nicht mehr auf die Basisrelation *Prüfen* zurückverfolgen. Die folgende Operation würde also vom DBMS abgewiesen werden:



```
UPDATE Durchschnittsnote
SET Note = 1.0
WHERE PersNr =
  (SELECT PersNr
   FROM Lehrende
   WHERE Name = 'Sokrates');
```

Zusammenfassend ist zu sagen, dass Sichten im Allgemeinen veränderbar sind, wenn

- sie weder Aggregierungsfunktionen noch Anweisungen wie DISTINCT, GROUP BY und HAVING enthalten,

- in der SELECT-Liste nur eindeutige Spaltennamen stehen und ein Schlüssel der Basisrelation enthalten ist und
- sie genau eine Tabelle (also Basisrelation oder Sicht) verwenden, die ebenfalls veränderbar sein muss.

## 7.2 Rechtevergabe

Nach der Installation eines DBMSs ist zunächst nur ein/e Benutzer/in bekannt, der/die Systemadministrator/in. Zur Aufgabe eines/r Datenbank-Administrators/in (DBA) gehört die Datenbankimplementierung sowie das Aufrechterhalten des Betriebes der Datenbank. Als Administrator/in hat der/ie Benutzer/in das Recht

- Datenbanken einzurichten und daraus folgend Relationen, Sichten und Indizes anzulegen und wieder zu entfernen, sowie
- Anfrage- und Änderungsanweisungen auf Relationen und Sichten an andere Benutzer weiterzugeben.

Hat ein/e Datenbank-Administrator/in eine Datenbank mit allen Relationen, Sichten und weiteren Objekten eingerichtet, kann noch keine andere Person auf die Datenbank zugreifen. Dies wird durch das sorgfältige Vergeben von Rechten auf Relationen und Sichten erreicht. Zur Rechtevergabe bietet SQL das GRANT-Konstrukt an.



Alle weiteren Benutzer müssen explizit bekannt gegeben werden!

Der/die Systemadministrator/in hat dann die Aufgabe, alle weiteren Personen im Datenbanksystem explizit bekannt zu geben und ihnen Rechte bezüglich des Lesens, Schreibens, Löschens und Ändern von Tupeln auf eine Datenbank zu erteilen. Außerdem kann er/sie für weitere Personen Arbeitsbereiche (Benutzerkonten) einrichten, damit auch sie ihre eigene Datenbank mit Relationen und Sichten erstellen können.



Die Rechtevergabe ist ein technischer Aspekt des Datenschutzes und der Datensicherheit!

Die Rechtevergabe durch den/die Datenbank-Administrator/in auf Relationen und Sichten muss sehr sorgfältig erfolgen, um auch Belange des **Datenschutzes** und der **Datensicherheit** zu berücksichtigen. Hieraus ist zu erkennen, dass nur Zugriff auf Relationen oder Sichten gewährt werden darf, wenn dies nicht gegen die geltenden Bestimmungen verstößt. **Im Zweifelsfall den Zugriff verweigern und erst die Rechtslage überprüfen!**

Auch die Hardware mit den Datenbanken muss durch geeignete Sicherheitsmaßnahmen wie z.B. verstärkte Türen, Schlösser, Wechselfestplatten oder Verschlüsselung von Daten speziell gesichert werden, um den Zugriff durch unbefugte Personen zu erschweren bzw. unmöglich zu machen.

### Zugriffsrechte

Zugriffsrechte können z.B. die Erteilung einer Kennung, die Definition eines DB-Ausschnitts oder eine Liste von erlaubten Operationen sein.

Um mit einer Datenbank arbeiten zu können, benötigt der/die Anwender/in für die Datenbank eine Kennung, auch BenutzerID genannt. Die Kennung besteht aus einem Loginnamen und einem Passwort. Beides wird dem/r Benutzer/in vom Datenbank-Administrator (DBA) zugewiesen. Beim Beantragen der Kennung muss ebenfalls mitgeteilt werden, was er/sie in der Datenbank tun möchte, denn danach richten sich die ihm/ihr erlaubten Datenbank-Operationen (CREATE, SELECT, INSERT, UPDATE, DELETE, usw.). Je nach Rechten kann der/die Anwender/in dann eigene Relationen bzw. Sichten erstellen oder in schon vorhandenen Datenbanken lesen, schreiben, einfügen, ändern oder löschen.



Die Verwaltung der Zugriffsrechte und die Überwachung ihrer Einhaltung erfolgen über das DBMS.

Mit Hilfe der GRANT-Anweisung werden Rechte auf die jeweilige DB vergeben. Die Verwaltung und Überwachung dieser Rechte übernimmt das DBMS.








in SQL:



GRANT <Rechte>





ON <Tabelle>






	 TO <Personenliste>
	 [WITH GRANT OPTION]
 <Rechte>:	 ALL
	oder eine Liste aus
	 {SELECT   INSERT   UPDATE   DELETE   ALTER   INDEX}
 <TABELLE>:	Relation oder Sichtname
 <Personenliste>:	Personen / Personengruppe oder PUBLIC
WITH GRANT OPTION:	Recht auf Weitergabe von Rechten



Erklärungen zu den GRANT-Syntax-Komponenten:

 ALL :	Uneingeschränktes Bearbeitungsrecht
 ALTER :	Neue Spalten können hinzugefügt und Spaltentypen geändert werden



 DELETE :	Zeilen (Tupel) dürfen gelöscht werden
 INSERT :	Neue Zeilen dürfen hinzugefügt werden
 INDEX :	Es darf ein Index auf eine Relation angelegt werden, der aber zusätzlichen Aufwand für das DBS bedeutet
 SELECT :	Daten können gesucht und angezeigt werden
 UPDATE :	Daten dürfen geändert werden. Wenn Spalten genannt werden, dürfen nur die genannten Spalten aktualisiert werden



Beispiel



```
CREATE VIEW Meine_Aufträge AS
SELECT *
  FROM Auftrag
 WHERE KName = USER
GRANT SELECT, INSERT
  ON Meine_Aufträge
  TO PUBLIC;
```

Die zu vergebenden Rechte sind genau an eine Relation oder Sicht gebunden. Es wird bestimmt, welche Operation (SELECT, INSERT, DELETE, UPDATE, usw.) welche Person auf eine Relation oder Sicht durchführen darf. Die UPDATE-Anweisung erlaubt es, noch detaillierter festzuschreiben, welche Attribute aktualisiert werden dürfen. Sollte anstelle einer Liste von Namen hinter TO das Schlüsselwort "PUBLIC" angegeben werden, so sind alle dem Datenbanksystem bekannten Personen mit diesem spezifischen Recht ausgestattet. Außerdem kann bestimmt werden, ob eine Person das vergebene Recht selbst an andere Person weitergeben darf ("WITH GRANT OPTION"). Mit der Anweisung "REVOKE" können vergebene Rechte wieder zurückgenommen werden.



- Die Anweisungen ALTER und INDEX dürfen nicht auf Sichten angewendet werden
- Sind die Rechte auf TO PUBLIC gesetzt, kann WITH GRANT OPTION nicht angewendet werden

### REVOKE (Zurücknahme von Rechten)



```
REVOKE <Rechte>
ON <Tabelle>
FROM <Personenliste>
[RESTRICT | CASCADE]
```

<pre>RESTRICT</pre>	Abbruch bei weitergegebenen Rechten
<pre>CASCADE</pre>	Propagierung bei weitergegebenen Rechten



## 7.3 Integrität

Die Aufgabe eines DBMSs ist nicht nur die Unterstützung bei der Speicherung und Verarbeitung von großen Datenmengen sondern auch bei der Gewährleistung der Konsistenz der Daten.

Hier wird zwischen verschiedenen Integritätsbedingungsarten unterschieden. Semantische Integritätsbedingungen sind solche, die sich aus Eigenschaften der modellierten Realitätsausschnitt (Miniwelt) ableiten lassen. Weiter unterscheidet man zwischen statischen und dynamischen Integritätsbedingungen. *Statische Bedingungen müssen von jedem Zustand der Datenbank erfüllt werden.* Angestellte dürfen z.B. nur die Vergütungsgruppe IIa, III, IVa, IVb, Va oder TV-L 11 bzw. TV-L 13 haben. *Dynamische Bedingungen werden an Zustandsänderungen gestellt.* So dürfen Angestellte befördert, aber nicht herabgestuft werden. Das bedeutet, ihre Vergütungsgruppe darf z.B. nicht von III auf IVa gesetzt werden.

Eine *transitionale Integritätsbedingung* setzt zwei aufeinander folgende Datenbankzustände miteinander in Beziehung und ist daher nicht statisch überprüfbar.

**Integritätsbedingungen** sind Bedingungen für die Zulässigkeit bzw. Korrektheit von einzelnen Datenbankzuständen und -änderungen.

Unter einer **Transaktion** versteht man eine Folge von SQL-Anweisungen, die logisch zusammengehören und bei der Integritätsüberwachung als Einheit angesehen werden.

Bei einer transaktionsorientierten Verarbeitung wird nach dem Alles-oder-nichts-Prinzip dafür Sorge getragen, dass die Anweisungsfolge komplett oder gar nicht ausgeführt wird. Eine Ausführung nur einzelner Anweisungen wird verhindert. Die Datenbank braucht nur vor und nach Transaktionen im zulässigen Zustand zu sein. Transaktionen werden nur dann vollzogen, wenn sie die Datenbasis in einen konsistenten Zustand überführen.

Es sind drei Typen von Integritätsbedingungen zu implementieren:

### 1. Die **"Datentyp"-Integritätsbedingung**

Soll ein Eintrag in einem Feld einer besonderen Bedingung genügen, wird das Schlüsselwort CHECK benutzt, welches hinter den Datentyp geschrieben wird.

z.B.: HK NUMBER(8,2) *CHECK(HK>=0)* oder

Monat NUMBER(2,0) *CHECK(Monat BETWEEN 1 AND 12)*

Im ersten Beispiel werden in HK nur Zahlen zugelassen, die größer oder gleich Null sind, im zweiten wird der Wertebereich des Attributes Monat auf 1 bis 12 gesetzt.

Das nachfolgende Beispiel stammt aus der Relation "besuchen" der Datenbank "Miniwelt Hochschule":

```
Note          CHAR(3)          CHECK(Note          IN
('1.0','1.3','1.7','2.0','2.3','2.7','3.0','3.3','3.7','4.0','5.0'))
```

## 2. Die *referenzielle Integritätsbedingung*

Um die referenzielle Integrität vom System verwalten zu lassen, müssen bei der Erstellung von Relationenstrukturen mit CREATE TABLE oder mit ALTER die Zusätze PRIMARY KEY, FOREIGN KEY und REFERENCES eingesetzt werden, bei Oracle zusätzlich die CONSTRAINT-Klausel.

Mit dieser Struktur können dann Einschränkungen bezüglich des Einsetzens und Löschens von Tupeln in Abhängigkeit von anderen Relationen gemacht werden.

Die referenziellen Integritätsbedingungen sind die wichtigsten innerhalb eines Datenbankschemas. Sie spezifizieren die durch Wertegleichheit realisierten Querverweise zwischen Relationen.



Beispiel



```
CREATE TABLE besuchen
(MatrnR CHAR(7) REFERENCES Studierende,
LVNr CHAR(4) REFERENCES LV,
Note CHAR(3)
CHECK(Note          IN          ( '1.0', '1.3', '1.7',
'2.0', '2.3', '2.7', '3.0', '3.3',
'3.7', '4.0', '5.0' )),
PRIMARY KEY(MatrnR, LVNr));
```

## 3. Die *explizite Integritätsbedingung*

Für Integritätsbedingungen, die nicht den referenziellen Bedingungen zuzurechnen sind, bot der SQL2-Standard die "ASSERTION"-Anweisung (Zusicherung) an.

Hier eine vereinfachte Syntax:



```
CREATE ASSERTION zusicherung CHECK
(where-bedingung)
```

Die Syntax beginnt mit der Anweisung CREATE ASSERTION, worauf ein frei wählbarer Name folgt, unter dem das DBMS die Integritätsbedingung erkennen kann. Als nächstes folgt CHECK und abschließend eine WHERE-Bedingung.



## Beispiel



```
CREATE ASSERTION MatrVsPers
CHECK (NOT EXISTS
(SELECT *
FROM Studierende s, Lehrende l
WHERE s.MatrNr = l.PersNr));
```

Mit der Zusicherung wird verhindert, dass eine MatrNr in der Relation Studierende mit einer PersNr in der Relation Lehrende übereinstimmt.

Die meisten der am Markt verfügbaren SQL-Dialekte bieten statt der CREATE ASSERTION-Anweisung ein Triggerkonzept an. Trigger sind dazu da, um die Konsistenz einer Datenbank zu gewährleisten. Beim Triggerkonzept ist die Bedingung an eine bestimmte Relation oder Änderungsaktion gebunden.



## SYNTAX

```
CREATE [OR REPLACE] TRIGGER trigger_name
[BEFORE | AFTER
]
[
DELETE | INSERT | UPDATE [OF column]]
ON [User.]table
[REFERENCING OLD AS name NEW AS name
]
[
FOR EACH ROW][WHEN condition
]
[
PL/SQL block];
```



Ein Trigger besteht aus der CREATE TRIGGER-Anweisung mit optionalem OR REPLACE. REPLACE gibt an, dass eine eventuell bereits existierende Prozedurdefinition überschrieben werden soll.

Mit den danach folgenden optionalen Anweisungen BEFORE und AFTER wird angegeben, ob der Trigger vor oder nach Ausführung der auslösenden Operation (einschließlich aller referenziellen Aktionen) ausgeführt wird.

DELETE, INSERT und UPDATE bestimmen, bei welcher Aktion der Trigger reagieren soll. Die Angabe OF *column* ist nur für UPDATE erlaubt und gibt an, auf welche Spalte der Trigger reagieren soll. Wird OF *column* nicht angegeben, so wird der Trigger aktiviert, wenn irgendeine Spalte der Relation verändert wird. ON bestimmt die Relation, auf die der Trigger wirken soll.

Mittels der in REFERENCING OLD AS ... NEW AS ... angegebenen Transitions-Variablen kann auf die Zeileninhalte vor und nach der aktivierenden Aktion zugegriffen werden. Mit :OLD und :NEW-Angaben kann die Unterscheidung zwischen dem "alten" Tupel, das sich in der Datenbank befindet, und dem "neuen" Tupel, das in der Datenbank verändert werden soll, hergestellt werden. Die Voreinstellung ist :NEW. Nur der BEFORE-Trigger erlaubt den Schreibzugriff auf :NEW.

Mit FOR EACH ROW wird ein Trigger als Row-Trigger definiert. Sollte diese Zeile fehlen, wird der Trigger als Statement-Trigger definiert. Mit WHEN kann die Ausführung eines Triggers weiter eingeschränkt werden. Insbesondere können die Transitionsvariablen OLD und NEW in WHEN-Bedingungen verwendet werden.

Der PL/SQL-Block eines Triggers darf **keine** Anweisungen zur Transaktionssteuerung enthalten (PL/SQL: Procedural Language/SQL, eine eigene Programmiersprache des Oracle-DBMSs).



Alle Beispiele in diesem Abschnitt wurden auf dem Oracle DBMS getestet.



Beispiel

In den Online-Studiengang Medieninformatik dürfen maximal 30 Studierende aufgenommen werden.



```
CREATE OR REPLACE TRIGGER numerus_clausus
```

```
BEFORE INSERT ON online_Studierende
FOR EACH ROW
DECLARE
anzahl NUMBER;
BEGIN
    SELECT count(*) INTO anzahl
    FROM online_Studierende
    WHERE Studiengang = 'MI';
    IF (anzahl>29)
    THEN
        RAISE_APPLICATION_ERROR(-20250, 'Datensatz
        konnte nicht eingefuegt werden,
        da schon 30 Studierende aufgenommen wurden!');
    ELSE
        DBMS_OUTPUT.PUT_LINE('Datensatz eingefuegt!');
    END IF;
END;
/
```

**Beispiel**

Beispiel in ORACLE PL/SQL: Eine Personalnummer (Attribut PersNr) in der Relation "Lehrende" darf nicht mit einer Matrikelnummer (Attribut MatrNr) in der Relation "Studierende" übereinstimmen.



```
CREATE OR REPLACE TRIGGER T_EX_IN_STUD_IU
AFTER INSERT OR UPDATE ON LEHRENDE
FOR EACH ROW
DECLARE
studentanzahl Number;
BEGIN
    SELECT count(*) INTO studentanzahl FROM Studierende st
    WHERE :new.persnr=st.matrnr;
    IF (studentanzahl > 0)
    THEN
        IF INSERTING
```

```
THEN
RAISE_APPLICATION_ERROR(-20209, 'Datensatz
konnte nicht eingefuegt werden,
da PersNr schon als MatrNr in Studierende vorkommt!');
END IF;
IF UPDATING
THEN
RAISE_APPLICATION_ERROR(-20210, 'Datensatz
konnte nicht veraendert werden,
da PersNr schon als MatrNr in Studierende vorkommt!');
END IF;
ELSE
DBMS_OUTPUT.PUT_LINE('Datensatz eingefuegt!');
END IF;
END;
/
```



Um die Ausgabe über `DBMS_OUTPUT.PUT_LINE` sehen zu können, muss in Oracle die Anweisung *SET SERVEROUTPUT ON* eingegeben werden.

Die Anweisung `RAISE_APPLICATION_ERROR` sendet an Oracle eine Fehlermeldung und sorgt für die Unterbrechung der momentanen Transaktion, womit der Datenbankzustand vor dem Einfügen oder Ändern eines Datensatzes wieder hergestellt wird.

Im oberen Beispiel gibt Oracle den Fehlercode -20250 aus. Erläuterungen dazu sind in der Oracle-Dokumentation zu finden. Allgemein gilt: Fehlercodes die kleiner als -20000 sind, können benutzerdefiniert verwendet werden. Größere Fehlercodes sind Systemfehlern von Oracle vorbehalten.

Die Anweisung `ROLLBACK` dient zum Zurücksetzen der aktuellen Transaktion. Sie trägt, je nach SQL-Dialekt, unterschiedliche Namen, und bei Oracle ist sie in der Anweisung `RAISE_APPLICATION_ERROR` integriert.

### ***Zusammenfassung***



Allgemein ist anzumerken, dass sich referenzielle Integritätsbedingungen bequem mit dem REFERENCES-Konstrukt realisieren lassen. Sollte man einen SQL-Dialekt anwenden, der dieses Sprachmittel noch nicht anbietet, so implementiert man die referenziellen Bedingungen mit dem Triggerkonzept. Bei Datentyp-Integritätsbedingungen bietet sich die "ASSERTION"-Anweisung als sehr elegante und vor allem sehr komfortable Lösung zum Implementieren an. Falls die relationalen Datenbanksysteme, die diese Anweisung aus dem SQL-Standard nicht unterstützen, muss man auch hier die Datentyp-Integritätsbedingungen mit Hilfe eines Triggers verwirklichen. Doch diese Realisierung ist recht unbequem, da man pro Integritätsbedingung meist mehrere Trigger benötigt. Ist eine explizite Bedingung notwendig, hat man keine Wahl und muss diese mit einem Trigger oder ggf. im Anwendungsprogramm implementieren.

## 7.4 Syntax-Diagramme

Im Folgenden werden einige Syntax-Diagramme angegeben, um verschiedene Anweisungen, auch anhand von Beispielen, zu veranschaulichen.



Gliederung

### 7.4 Syntax-Diagramme

#### 7.4.1 Syntax der GRANT-Anweisung

#### 7.4.2 CREATE ASSERTION-Syntax

### 7.4.1 Syntax der GRANT-Anweisung

#### Vereinfachte Syntax

Die Abbildung sri.3 zeigt die vereinfachte Syntax zum Einrichten von Rechten für einen oder mehrere Benutzer.

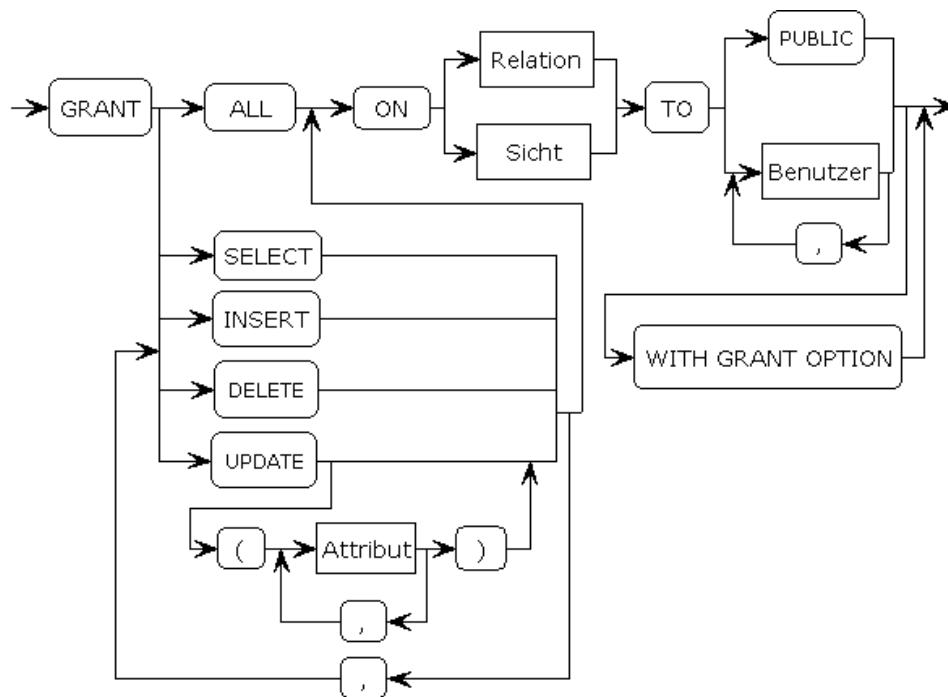


Abbildung sri.3: Vereinfachte GRANT-Syntax

Im nachfolgenden Beispiel wird den Benutzern meier und schmidt das Recht gegeben, auf die Relation "Lehrende" lesend (GRANT SELECT) zuzugreifen. Außerdem dürfen sie dieses Recht an andere weitergeben (WITH GRANT OPTION).



Beispiel



```

GRANT SELECT
ON Lehrende
TO meier, schmidt
WITH GRANT OPTION;
  
```

### 7.4.2 CREATE ASSERTION-Syntax

Die Abbildung sri.5 zeigt die vereinfachte Syntax der expliziten Integritätsbedingung ASSERTION.

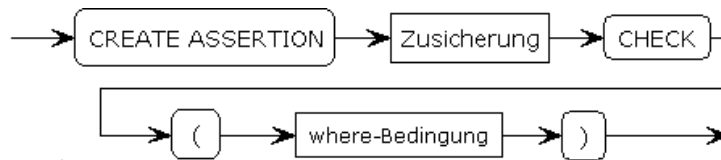


Abb.sri.5: CREATE ASSERTION-Syntax

Im nachfolgenden Beispiel wird festgelegt, dass an einem Kurs nur so viele Teilnehmer teilnehmen dürfen, wie das Attribut "maxTeilnehmer" angibt.

**Beispiel**

```

CREATE ASSERTION EX4 CHECK
  (NOT EXISTS(SELECT *
    FROM Kurs k
    WHERE k.maxTeilnehmer <
      (SELECT COUNT(*)
        FROM Teilnehmer
        WHERE KursNr = k.KursNr)))));
  
```

## 7.5 Freitextaufgaben zu Sichten, Rechtevergabe, Integrität

1. Wie löscht man einen Datensatz in einer Sicht?

**Lösung zeigen**

DELETE FROM sichtname WHERE bedingung.

2. Welchen Einschränkungen unterliegen Sichten bezüglich ihrer Update-Fähigkeit?

**Lösung zeigen**

1. Sichten, die genau eine Tabelle verwenden, die ebenfalls veränderbar sein muss, können unter Umständen update-fähig sein.
2. Sichten sind nur dann update-fähig, wenn in der select-Liste nur eindeutige Attributnamen stehen und ein Schlüssel der Basisrelation enthalten ist.

3. Welche Vorteile bieten Sichten?

**Lösung zeigen**

Vorteile von Sichten:

1. vereinfachte Anfrage
2. die Strukturierung der Datenbank
3. logische Datenbankunabhängigkeit
4. Beschränkung von Zugriffen

4. Geben Sie den Benutzern schmidt und mueller die Leserechte auf die Relation Lehrende und die Rechte, die Leserechte weiterzugeben.

**Lösung zeigen**

```
GRANT SELECT
ON Lehrende
TO schmidt, mueller
WITH GRANT OPTION;
```

5. Mit welcher Anweisung können Rechte wieder zurückgenommen werden?

**Lösung zeigen**

Rechte können mit der Anweisung REVOKE zurückgenommen werden.

6. Wann kann WITH GRANT OPTION nicht angewendet werden?

**Lösung zeigen**

WITH GRANT OPTION kann nicht angewendet werden, wenn die Personenliste auf PUBLIC steht.

7. Was sind Integritätsbedingungen?

**Lösung zeigen**

Integritätsbedingungen sind Bedingungen für die "Zulässigkeit" bzw. "Korrektheit" von einzelnen Datenbankzuständen und -änderungen.

8. Wofür werden Integritätsbedingungen benötigt?

**Lösung zeigen**

Integritätsbedingungen werden benötigt, um die Konsistenz einer Datenbank zu gewährleisten, d.h sie verhindern "unlogische" Zustände und semantische Fehler in den Datensätzen der Datenbank.

9. Was für Arten von Integritätsbedingungen gibt es?

**Lösung zeigen**

Es gibt Datentyp-, implizite/referenzielle und explizite Integritätsbedingungen.

10. Zu welcher Art von Integritätsbedingungen zählt die CREATE ASSERTION-Anweisung?

**Lösung zeigen**

Die CREATE ASSERTION-Anweisung ist eine explizite Integritätsbedingung.

11. Was ist ein Trigger?

**Lösung zeigen**

Ein Trigger ist ein Auslöser mit Folgeaktion(en).

12. Kann auf die Integritätsbedingungen verzichtet werden?

**Lösung zeigen**

Nein, da viele Integritätsbedingungen unabdingbar für die Datenkonsistenz sind. Bei den expliziten Integritätsbedingungen muss der Zeit- und Speicheraufwand für die Überprüfung im Verhältnis zum Nutzen abgewogen werden.

## 8 Anwendungen mit Datenbanken



### Zeitumfang

Die voraussichtliche Bearbeitungsdauer dieser Lerneinheit (ohne Übungsaufgaben!) beträgt ca. 10 Stunden.



### Lernziele

In dieser Lerneinheit erfahren Sie, wo die Grenzen der Datenbanksprache SQL liegen und welche Möglichkeiten es gibt, um Anwendungsprogramme zu schreiben.

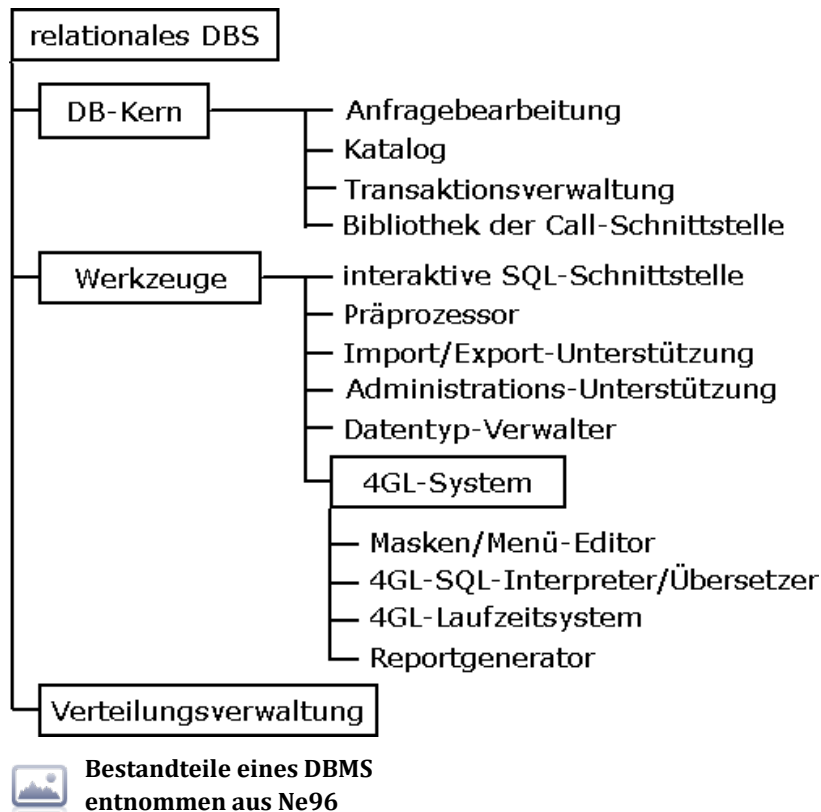
Für Datenbank Anwendungen wird häufig die Mächtigkeit von Programmiersprachen benötigt. Dies kann durch Hinzufügen von datenbankspezifischen Prozeduraufrufen, die sogenannte Call-Schnittstelle, geschehen. In SQL99 wird diese Möglichkeit über das Call-Level-Interface (CLI) standardisiert.

Eine weitere Möglichkeit der Verwendung von Programmiersprachen ist die Einbettung der DML-Anweisungen in Programmiersprachen, wie es bei Embedded SQL und Dynamic SQL der Fall ist. Für Client-/Server-Architekturen bieten die ODBC-, JDBC- und PHP-Schnittstellen die Möglichkeit an, die Anbindung von Anwendungen an ein Datenbanksystem zu realisieren.

Für häufig verwendete Anfragenabläufe können gespeicherte Prozeduren (Stored Procedures) eingesetzt werden. D. h. die SQL-Anfragen werden in einer Datei gespeichert und diese Datei wird jeweils aufgerufen.

Bereits existierende Werkzeuge wie Report-Generatoren können für Standardlösungen eingesetzt werden.

Die Bestandteile eines DBMS werden im folgenden Bild veranschaulicht:



Ein relationales Datenbanksystem besteht im Wesentlichen aus dem Datenbank-Kern und den Werkzeugen, die eine komfortable Handhabung von Datenbanken unterstützen. Für verteilte Datenbanken kommt die Verteilungsverwaltung hinzu, die den Client/Server-Betrieb ermöglicht und die Verteilung der Daten für die Benutzer transparent macht.

Der Datenbank-Kern übernimmt die Anfragebearbeitung, verwaltet den Katalog und die Transaktionen. Des weiteren stellt er die Bibliothek der Call-Schnittstelle zur Verfügung.

Bei Werkzeugen bietet die interaktive SQL-Schnittstelle die Möglichkeit, SQL-Anweisungen interaktiv einzugeben. Der Präprozessor ist für die Analyse der eingebetteten SQL-Anweisungen zuständig. Die Import-/Export- bzw. die Administrations-Unterstützung ermöglichen Datenein-/ausgabe (z.B. bei Massendaten) bzw. Datenbankadministration (z.B. BenutzerInnenverwaltung) komfortabel durchzuführen. Neue Datentypen können mit Hilfe des Datentyp-Verwalters hinzugefügt werden.

Das 4 GL-System bietet an, die vierte Generation der Programmiersprachen für Datenbankanwendungen einzusetzen und Report-Generatoren zur vereinfachten Publizierung von Datenbankinhalten zu benutzen.

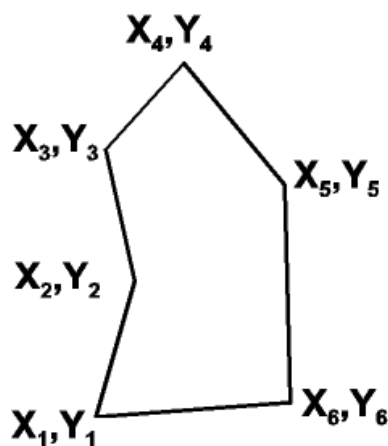


## Gliederung

8 Anwendungen mit Datenbanken8.1 Grenzen von SQL8.2 Call-Schnittstelle8.3 Embedded SQL8.4 ODBC8.5 JDBC8.6 PHP-Anwendungen8.7 PL/SQL8.8 Sicherheit bei datenbankgestützten Webapplikationen - Die SQL-Injection8.9 4GL-Systeme8.10 Ergänzende Informationen8.11 Freitextaufgaben zu Anwendungen mit Datenbanken**8.1 Grenzen von SQL**

Bei der Verwaltung von komplexen Objekten, z.B. geometrischen Objekten mit einer relationalen Datenbank, stößt man auf die Grenzen von SQL.

1. Bei der Realisierung von Algorithmen wird eine anspruchsvolle Datenstruktur benötigt, um z.B. geometrische Objekte darzustellen.



Polygon

ID	Pkt.Nr.	X	Y
----	---------	---	---



P <sub>11</sub>	1	X <sub>1</sub>	Y <sub>1</sub>
P <sub>11</sub>	2	X <sub>2</sub>	Y <sub>2</sub>
P <sub>11</sub>	3	X <sub>3</sub>	Y <sub>3</sub>
P <sub>11</sub>	4	X <sub>4</sub>	Y <sub>4</sub>
P <sub>11</sub>	5	X <sub>5</sub>	Y <sub>5</sub>
P <sub>11</sub>	5	X <sub>6</sub>	Y <sub>6</sub>



**Polygon als Tabelle**  
entnommen Ne98

Sollen z.B. Grundstücke oder Parzellen mit ihren Grenzen und Vermessungspunkten relational gespeichert werden, so ergibt sich typischerweise eine Struktur wie sie in Abbildung anw.1 zu sehen ist. Die geometrischen Daten eines Vielecks (Polygons) werden durch seine Stützpunkte, das sind beliebig viele XY-Koordinaten, in der Ebene dargestellt. Als Beispiel ist hier ein Polygon mit 6 Stützpunkten angegeben. Da in der Relation "Polygone" mehrere Polygone oder Grundstücksgeometrien enthalten sein sollen, muss für jeden Stützpunkt ein Name oder eine Identifikation ("ID") des zugehörigen Polygons angegeben werden.

Außerdem wird die Position jedes Punktes ("PktNr") innerhalb der Liste aller Punkte eines Polygons benötigt, weil Relationen nur ungeordnete Mengen von Tupeln abbilden können.

Der Umfang der so gespeicherten Polygone errechnet sich dann wie folgt:

$$\sum_{i=2}^n \sqrt{(X_i - X_{i-1})^2 + (Y_i - Y_{i-1})^2} + \sqrt{(X_1 - X_n)^2 + (Y_1 - Y_n)^2}$$



Die Anfrage



```
SELECT PktNr, X,Y
FROM Polygone
WHERE Id = 'P11'
ORDER BY PktNr;
```

liefert zwar die XY-Koordinaten des Polygons "P11" in der richtigen Reihenfolge, um die Summe aber berechnen zu können, wird ein **Schleifenkonstrukt** benötigt.

Es gibt auch Aufgaben, die mit SQL-Anweisungen zwar zu lösen sind, wo der Lösungsansatz sich aber als sehr unbequem darstellt.

Zusammengefasst:

Iterations- und Schleifenkonstrukte, allgemeine Prozeduraufrufe und anspruchsvolle Datenstrukturen werden benötigt, um z.B.

- Erben in der Verwandtschaftsdatenbank zu ermitteln,
- Teilbedarfslisten über mehrere Ebenen einer Teil-Subteil-Hierarchie zu erstellen,
- Verbindungen im Straßennetz zu finden.

*Dies alles sind Sprachmittel, die SQL nicht enthält.*

2. Bei der kommandozeilenorientierten Eingabe von Tupeln (SQL-INSERT-Anweisung) müssen alle Attributwerte in der richtigen Reihenfolge und auch im richtigen Format angegeben werden. Dies setzt voraus, dass die eingebende Person mit den Datentypen der Relation vertraut ist. Sollte sich der Bediener auch nur bei einem Datentyp vertippen, würde das Datenbanksystem die Anweisung zurückweisen und die Daten müssten erneut eingegeben werden.

Für den täglichen Betrieb ist die SQL-Lösung bei umfangreichen Relationen nicht realistisch. Benötigt wird eine Eingaberoutine mit komfortabler Oberfläche, die dem Benutzer möglichst viel Eingabearbeit erspart und Eingabefehler frühzeitig anzeigt.

Natürlich ist z.B. eine Eingabe der Werte über ein Formular in Fenstertechnik bequemer als die Benutzung der INSERT-Anweisung über die Kommandozeile. Daraus folgt, dass z.B. für das Ermitteln einer transitiven Hülle und die Berechnung des Umfanges von Polygonen eine maßgeschneiderte Benutzungsoberfläche erstellt werden sollte. Diese kann nicht direkt in SQL implementiert werden. Für beide Aufgabengebiete wird daher eine Kopplung von SQL mit einer Programmiersprache, z.B. Java benötigt.

## 8.2 Call-Schnittstelle

**Call-Level-Schnittstellen (engl. Call Level Interface)** sind eine einfache Art der Kopplung einer Datenbank mit einer Programmiersprache. Dabei werden alle DB-Funktionen aus einem Anwendungsprogramm heraus in Form von externen Prozeduren aufgerufen.

Abhängig vom jeweiligen Datenbanksystem können Bibliotheken, die dem Anwendungsprogrammierer zur Verfügung gestellt werden, hunderte von Prozeduren enthalten. Meist kann man die einzelnen Prozeduren einer der unten aufgeführten Funktionsgruppen zuordnen.

### Funktionsgruppen

- **Kommunikationsprozeduren:** Funktionen zum An- und Abmelden beim DBS, zur Übergabe von Passwörtern und Benutzeridentifikationen
- **Bindungsprozeduren:** Funktionen zum Binden von Programmvariablen an noch offene Parameter in SQL-Anweisungen oder zur Aufnahme von Ergebnisdaten
- **SQL-Anweisungsprozeduren:** Funktionen zum Übersetzen und Ausführen von SQL-Anweisungen
- **Prozeduren zur Ergebnisverarbeitung:** Funktionen zum Bereitstellen, Öffnen, Abarbeiten und Schließen eines Zeigers auf den jeweils aktuellen Datensatz
- **Fehlerbehandlungsprozeduren:** Funktionen zur Entgegennahme von Fehlermeldungen in Form von Zahlenwerten und Texten
- **Transaktionsprozeduren:** Funktionen zum expliziten Abbrechen, Zurücksetzen oder Bestätigen von Transaktionen

Die Call-Schnittstelle ist leider nicht standardisiert. Im SQL-3-Standard ist eine "SQL Call Level Interface (SQL/CLI)" vorgesehen. Die Portabilität auf der Quellprogrammebene ist mit SQL/CLI von der X/Open Group bzw. der ISO gegeben.



#### Beispiel

#### Beispiel

Vorteile:

- geringe Beschaffungskosten (zum Standard-Lieferumfang)
- große Flexibilität

Nachteil:

- Wegen der Vielzahl der bereitgestellten Prozeduren mit

- ihren oftmals komplizierten Parametern ist die Benutzung der
- Call-Level-Schnittstellen schwierig und wenig komfortabel.

## 8.3 Embedded SQL

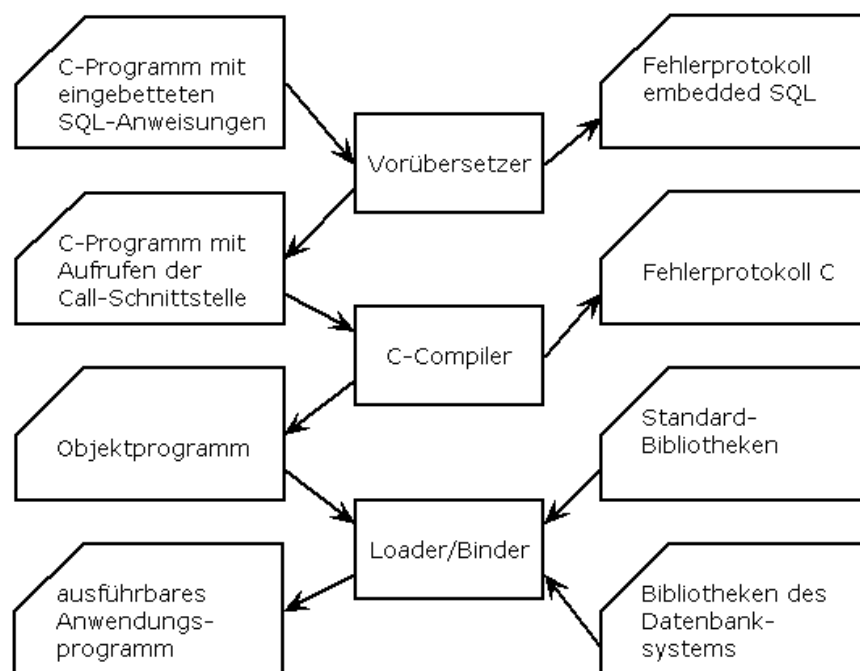
Im Gegensatz zur Call-Level-Schnittstelle sind die Sprachmittel des eingebetteten SQL im SQL-Standard enthalten. Im Folgenden wird speziell auf das Einbetten in C mit Hilfe eines Precompilers (Vorübersetzers) eingegangen.



Anmerkung

Einbettungsansatz: Der Einbettungsansatz kann auch unter den Bezeichnungen Vorübersetzer- oder Preprocessor-Ansatz vorkommen. Gemeint ist damit, dass die Datenbank-Anweisungen, die durch die Zeichenkette "EXEC SQL" gekennzeichnet werden, direkt in den Programmtext geschrieben werden.

Übersetzung eines Embedded SQL C-Programmes:



Übersetzung eines Embedded SQL-C-Programms  
entnommen Ne96



In der Online-Version befindet sich an dieser Stelle eine Simulation.

Animation der Übersetzung eines Embedded SQL-C-Programms

Ein Vorübersetzer, der meist extra zum eigentlichen relationalen Datenbanksystem erworben werden muss, erzeugt aus den markierten SQL-Anweisungen die entsprechenden Aufrufe der Call-Level-Schnittstelle und schreibt diese Aufrufe in das Anwendungsprogramm. Bei diesem Vorgang werden die ursprünglich eingebetteten SQL-Anweisungen in Kommentare umgewandelt; dadurch bleibt das vorübersetzte Anwendungsprogramm lesbar.

Das so vorbereitete Programm, hier in C geschrieben, kann jetzt von einem handelsüblichen Übersetzer, etwa einem normalen C-Compiler, weiterverarbeitet werden. Dem übersetzten Programm werden Standardbibliotheken und die Bibliotheken des Datenbanksystems hinzugegeben. Schließlich kann das fertige Datenbank-Anwendungsprogramm ausgeführt werden.

Vorübersetzer sind von den am meisten am Markt befindlichen relationalen Datenbanksystemen für verbreitete Programmiersprachen wie C, C++ oder SQLJ (Einbettung in Java) zu haben.

Im Sprachumfang können alle SQL-Anweisungen zur Datenhandhabung, wie Anfragen und Änderungen, als eingebettete Anweisungen auftreten. Hinzu kommen noch Deklarations- und Cursor-Anweisungen sowie weitere spezielle Anweisungen, die es nur als eingebettete und nicht als interaktive SQL-Anweisungen gibt.

In einigen SQL-Dialekten ist es möglich, Datendefinitionsanweisungen (wie z.B. das Einrichten und Entfernen von Relationsschemata) in Embedded SQL-Anweisungen einzubetten. Bei den meisten Systemen ist das Einbetten von Datendefinitionsanweisungen aber nicht möglich. Da Anwendungsprogramme überwiegend auf bestehenden Datenbanken arbeiten, sollten diese nur selten zum Ändern eines Datenbankschemas eingesetzt werden. Solche Änderungen werden normalerweise vom Datenbank-Administrator direkt mit einer interaktiven SQL-Schnittstelle ausgeführt.

Anwendungsprogramme mit eingebetteten SQL-Anweisungen greifen in der Regel auf Relationen zu. Beispiel anw.2 zeigt ein Fragment eines Anwendungsprogrammes, das eine Personalnummer von der Tastatur einliest und nach dem dazugehörigen Datensatz in einer Mitarbeiter-Relation sucht. Mit Hilfe der INTO-Anweisung werden Daten aus der SELECT-Anweisung den vorher deklarierten Variablen zugewiesen. Als Ergebnis werden dessen Attribute "Name" und "Vorname" ausgegeben.

**Beispiel**

```
EXEC SQL DECLARE MitarbeiterIn TABLE
(PersNr INTEGER,
Name VARCHAR(40),
Vorname VARCHAR(20),
weiblich CHAR(5),
extern CHAR(5),
Strasse VARCHAR(20),
Hausnr CHAR(4),
Plz CHAR(8),
Ort VARCHAR(40),
TNr CHAR(16),
Urlaubsanspruch INTEGER,
Jahresurlaub INTEGER,
Resturlaub INTEGER,
Institut VARCHAR(60));
```

**Fragment eines Anwendungsprogramms**

**Relation EXEC SQL MitarbeiterIn TABLE, entnommen Ne96**



```
EXEC SQL BEGIN DECLARE SECTION;
DCL PersNr INTEGER;
DCL Name VARCHAR(40);
DCL Vorname VARCHAR(20);
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE MitarbeiterIn TABLE (...);
Get (PersNr)
EXEC SQL SELECT Name, Vorname
INTO :Name, :Vorname
FROM MitarbeiterIn
WHERE extern = 'FALSE'
AND PersNr = :PersNr;
PUT (Vorname, Name);
```

**Fragment eines Anwendungsprogramms**

**Suche eines Datensatzes, entnommen Ne96**

Es werden zunächst drei Programmvariablen deklariert. Die Benennung erfolgt entsprechend der Attribute der MitarbeiterIn-Relation (oberes Listing). Diese Programmvariablen werden so dem Datenbanksystem bekannt gemacht (Zeilen 1-5). Die Relationen-Deklaration wird in Zeile 6 angedeutet. Die Variable "PersNr" wird in Zeile 7 eingelesen. Die in der Zielliste angegebenen Attribute werden mit INTO den Variablen :Name und :Vorname (Zeile 9) zugewiesen, d.h. :Name und :Vorname sind Programmvariablen, die in SQL-Anweisungen benutzt werden. Im Qualifikationsteil (Zeilen 11-12) wird die eingelesene Variable "PersNr" genutzt, um sie mit dem Attribut PersNr der Mitarbeiter-Relation zu vergleichen und um Vorname und Name des Mitarbeiters zu ermitteln. Das Ergebnis der Suche wird auf dem Bildschirm ausgegeben (Zeile 13).

*Rückmeldungen* über den Erfolg einer ausgeführten DB-Anweisung werden vom Datenbanksystem in der Variablen "SQLCODE" abgelegt. Um an diese Variable zu gelangen, muss die Kommunikationsstruktur, SQL Communication AREA ("SQLCA"), geladen werden.

Beispiele für Rückmeldungen: SQL-CODE = 100: kein Tupel wurde gefunden  
 SQL-CODE = 0: fehlerlos  
 EXEC SQL WHENEVER SQLERROR GOTO SQLFehlerbehandlung;

- 
- 
- 

SQLFehlerbehandlung: ...

WHENEVER-Syntaxdiagramm

### Lesen von mehreren Datensätzen

Bei der Verwendung der SELECT-Anweisung in Anwendungsprogrammen muss zwischen zwei Anfragearten unterschieden werden: Anfragen, die höchstens ein Tupel zurückliefern, und Anfragen, die mehrere Tupel (also Relationen) zurückliefern können.

Ein Beispiel für den ersten Fall könnte so aussehen:



```
EXEC SQL SELECT avg (Urlaubssemester)
INTO :avgsem
FROM Studierende;
```

Ziel ist hier die Berechnung des Durchschnitts der Urlaubssemester der Studierenden, wobei *avgsem* eine entsprechend deklarierte Variable sein soll.

Für den zweiten Fall, wenn das Ergebnis eine Menge von Tupeln ist, gestaltet sich die Rückgabe schwieriger. Der Grund liegt in der Tatsache, dass traditionelle Programmiersprachen keine eingebaute Möglichkeit zur Verwaltung von Mengen besitzen. Hier wird das sogenannte Cursor-Konzept verwendet. Mit diesem Konzept kann eine Menge von Tupeln, iterativ und sequenziell bearbeitet werden. Der Cursor zeigt dabei jeweils auf das Tupel, das aktuell in Bearbeitung ist.

In Embedded SQL unterteilt sich die Benutzung des Cursors in vier Schritte:

1. Deklaration des Cursors, um die zugehörige Anfrage festzulegen:



```
EXEC SQL DECLARE peterprofs  
CURSOR FOR  
SELECT Name, RaumNr  
FROM Lehrende  
WHERE Vorname = 'Peter';
```

2. Öffnen des Cursors und damit implizit das Positionieren auf das erste Tupel der Ergebnismenge:



```
EXEC SQL OPEN peterprofs;
```

3. Daten Schritt für Schritt zum Anwendungsprogramm übertragen:



```
EXEC SQL FETCH peterprofs  
INTO <code class="wählbar">:pname, :praum
```

Liegen keine Daten mehr an, wird dies durch entsprechendes Setzen der Statusvariablen angezeigt.

4. Schließen des Cursors mit der Anweisung:



```
EXEC SQL CLOSE peterprofs;
```

### Cursor-Syntaxdiagramm

### Beispiel eines Embedded SQL Programms zur Umfangberechnung von Polygonen



Nachteile einer Einbettung von SQL in Programmiersprachen:

- Traditionelle Programmiersprachen haben keine eingebauten Möglichkeiten zur Mengenverarbeitung; die Datensätze werden sequenziell abgearbeitet während SQL mengenorientiert arbeitet. Dieser Gegensatz heißt *Impedance Mismatch*.
- Da das Cursorkonzept nur einen künstlichen Ausgleich für das tupelorientierte Arbeiten darstellt, entsteht bei komplexen Anwendungen vielfach ein "Reibungsverlust" und zwar durch das wiederholte Schließen und Öffnen eines Cursors oder das Zwischenspeichern von bereits

eingeholten Ergebnissen.



Gliederung

### 8.3 Embedded SQL

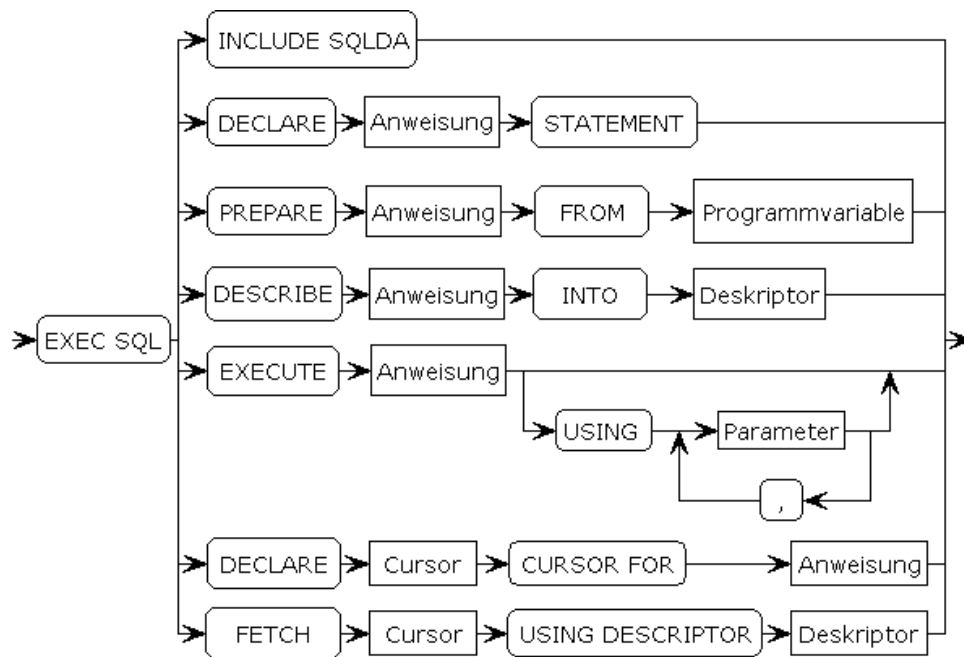
#### 8.3.1 Dynamic SQL

### 8.3.1 Dynamic SQL

Bei den Sprachmitteln von Embedded SQL handelt es sich um ein statisches SQL (Static SQL), da alle in Wirtsprogrammen vorkommenden SQL-Anweisungen bereits zur Kompilierungszeit des Programmes feststehen müssen, also statisch sind. Nun gibt es aber auch Anwendungsfälle, bei denen die eingebetteten SQL-Anweisungen erst zur Laufzeit des Anwendungsprogrammes aufgebaut und ausgeführt werden sollen. Solch ein Anwendungsfall könnte die Realisierung einer flexiblen interaktiven Benutzerschnittstelle für ein Anfragesystem sein, bei dem nicht alle möglichen Anfragen von vornherein feststehen.

Die Sprachkonstrukte, die ein dynamisches Konstruieren und Ausführen von SQL-Anweisungen ermöglichen, heißen "Dynamic SQL". Dynamic SQL ist im Sprachumfang von SQL-2 enthalten. Da manche Hersteller Spezialdialekte von Dynamic SQL anbieten, ist bei der Arbeit mit dem dynamischen Teil von Embedded SQL darauf zu achten, welches Datenbanksystem und welche Programmiersprache konkret verwendet werden. Die Unterschiede beziehen sich auf die Einbindung der Kommunikationsbereiche, nicht auf die Syntax der SQL-Anweisungen. JDBC bietet eine Call-Schnittstelle mit einer dynamischen SQL-Einbettungen.

Die Syntax der wichtigsten Anweisungen von Dynamic SQL wird in der nachfolgenden Abbildung illustriert.



Die wichtigsten Anweisungen von Dynamic SQL sind "PREPARE" und "EXECUTE". "PREPARE" erlaubt das Lesen und Verarbeiten einer Zeichenkette, die als Programmvariable vorliegt. Sollte die Zeichenkette eine konkrete SQL-Anweisung darstellen, so wird die Anweisung übersetzt und an die anzugebende SQL-Variable vom Typ "STATEMENT" gebunden. "EXECUTE" führt eine vorbereitete Anweisung aus. "DESCRIBE" holt Schemainformationen, wie sie bei einer SQL-Anweisung entstehen, in das Anwendungsprogramm.

Anhand eines Beispiels für ein Embedded SQL-Programm mit dynamischen SELECT-Anweisungen wird das Prinzip des Arbeitens mit dynamischen Select-Anweisungen erläutert.

## 8.4 ODBC

ODBC (Open Database Connectivity) ist eine Standardschnittstelle zwischen einer Datenbank und einem Programm, um auf die Daten in der Datenbank zuzugreifen. Über diese Schnittstelle ist es für jedes Programm möglich, mit der Sprache SQL Anweisungen an das DBS abzusenden. Bedingung ist, dass sowohl das Programm als auch das DBS dem ODBC-Standard entsprechen.

Der Zugriff erfolgt durch einen Treiber, mit dem das System ausgestattet wird. Die Anwendungsseite des Treibers muss den ODBC-Standard streng erfüllen. Er sieht auf der Anwendungsseite immer gleich aus, unabhängig davon, wie die anzusprechende

Datenbank aussieht. Die Befehle des Treibers zur Datenbankseite hin werden speziell angepasst.

Durch diese Architektur brauchen die Programme nicht an eine spezielle Datenbank angepasst zu werden. Sie müssen nicht einmal wissen, welches Datenbanksystem die Daten verwaltet.

### **Aufbau der ODBC-Schnittstelle**

Die ODBC-Schnittstelle besteht im Wesentlichen aus einer Reihe von Definitionen, die als Quasi-Standard anerkannt werden. Darin sind alle Definitionen enthalten, die für die Kommunikation zwischen Programm und Datenbank erforderlich sind.

Die Definitionen unterteilen sich in folgende Bereiche:

- Funktionsbibliothek: Die ODBC-Funktionsaufrufe stellen die Mittel bereit, um die Verbindung zu einem DBMS herzustellen, SQL-Befehle auszuführen und die Ergebnisse zurück an das aufrufende Programm zu senden.
- Standard-SQL-Syntax
- Standard-SQL-Datentypen
- Standardprotokoll für die Verbindung zu einer Datenbank-Engine
- Standardfehlercodes

### **Komponenten der ODBC-Schnittstelle**

Die ODBC-Schnittstelle besteht aus vier Komponenten, die jede für sich dazu beiträgt, die Kommunikation zwischen Programm und Datenbank flexibel zu gestalten:

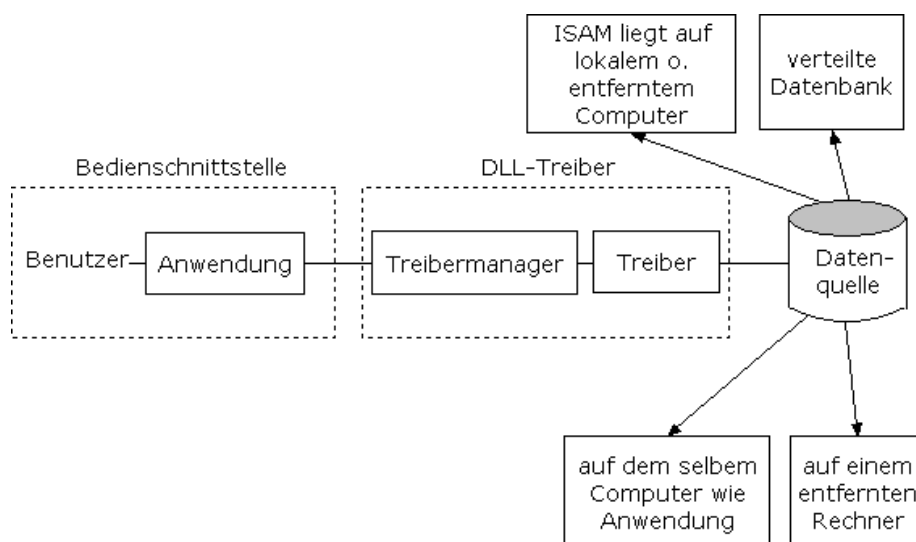
1. *Anwendung*: Diese Komponente ist am dichtesten am Benutzer und sie muss wissen, dass sie mittels einer ODBC mit ihrer Datenquelle kommuniziert, sich reibungslos mit dem Treibermanager verbindet und den ODBC-Standard strikt einhält.
2. *Treibermanager*: Es handelt sich um eine Dynamische Link Library (DLL), die die entsprechenden Treiber für die Datenquellen des Systems lädt und die Funktionsaufrufe, die von der Anwendung kommen, über den passenden Treiber an die entsprechende Datenquelle weiterlenkt. Außerdem führt der Treibermanager einige ODBC-Funktionsaufrufe direkt aus und ist in der Lage, einige Arten von Fehlern selbst zu erkennen und zu behandeln.
3. *Treiber*: Um die sich zum Teil stark voneinander unterscheidenden Datenquellen ansprechen zu können, wird zur Übersetzung die Treiber-DLL verwendet. Die Treiber nehmen die Funktionsaufrufe über die Standard-ODBC-Schnittstelle entgegen und übersetzen sie in Programmtext, den die zugehörige Datenquelle verarbeiten kann. Sollte eine Rückmeldung (Ergebnis) von der Datenquelle erfolgen, übersetzt der Treiber diese umgekehrt in das Standard-ODBC-

Ergebnisformat. Der Treiber ist das Schlüsselement bei der strukturellen und inhaltlichen Manipulation von ODBC-kompatiblen Datenquellen.

4. Die *Datenquelle* kann in verschiedenen Formen auftreten:

- - Es kann sich um ein relationales DBMS mit Datenbank auf dem selben Computer wie die Anwendung handeln
  - Die Datenbank kann auf einem entfernten Computer sein
  - Es kann sich um eine ISAM-Datei (Indexed Sequential Access Method) handeln, die entweder auf dem lokalen oder auf dem entfernten Computer liegt
  - Die Datenquelle liegt als verteilte Datenbank vor

Allerdings erfordert jede der vielen möglichen Formen der Datenquellen, die hier aufgezählt worden sind, einen eigenen Treiber. Das Ganze wird in Abbildung "Komponenten der ODBC-Schnittstelle" nochmal grafisch verdeutlicht.



**Komponenten der ODBC-Schnittstelle**

### ODBC in einer Client/Server-Umgebung

In einer Client/Server-Umgebung wird die Schnittstelle zwischen dem Client und Server als Application Programming Interface (API) bezeichnet. Die API wird in standardmäßige und firmenspezifische unterschieden. Bei den firmenspezifischen Schnittstellen wurde die Client-Komponente dem entsprechenden Datenbanksystem auf einem bestimmten Server hinsichtlich der Effizienz angepasst und optimiert.

### ODBC und Internet

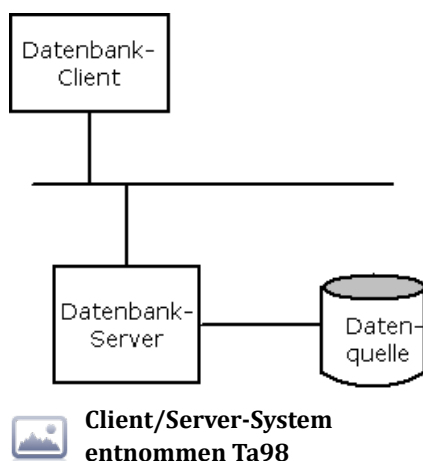
Das Übermitteln von Datenbankoperationen über das Internet unterscheidet sich in mehreren wichtigen Aspekten von Datenbankoperationen in einem Client/Server-System:

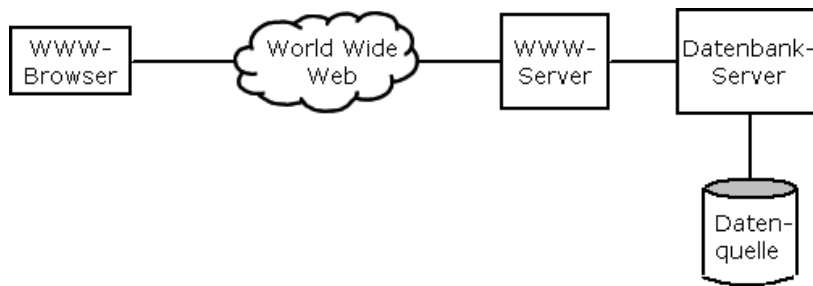
- Aus Sicht einer Benutzerin ist die Client-Komponente des Systems eine Benutzungsoberfläche, über die Datenbankoperationen explizit (z.B. in SQL formuliert) oder implizit (z.B. über eine Formularmaske) eingegeben werden können.
- Da der Zugriff auf die Datenbank über einen Browser von jedem Benutzer durchgeführt werden kann, wird der Vorgang, eine Datenbank im Netz verfügbar zu machen, als *Datenbank-Publishing* bezeichnet.
- Es ist normalerweise nicht möglich zu überprüfen, wer auf die Daten zugreift.



**Durch das Bereitstellen von Daten über das Internet sind in der Regel alle Informationen allgemein zugänglich und öffentlich. Deshalb ist darauf zu achten, dass für vertrauliche und zu schützende Daten (z.B. persönliche Daten) die entsprechenden Maßnahmen, z.B. Passwort, vorgesehen werden.**

In den beiden folgenden Abbildungen ("Client/Server-System" und "WWW-basiertes System") wird die Architektur der beiden Systeme gegenübergestellt.





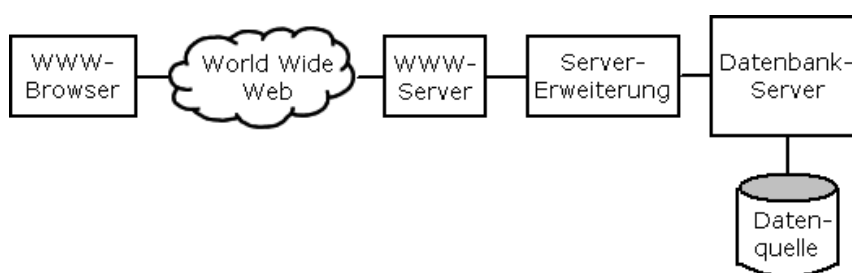
**WWW-basiertes System**  
entnommen Ta98

### Server-Erweiterungen

Bei WWW-basierten Systemen erfolgt die Kommunikation zwischen dem Browser auf der Client-Maschine und dem WWW-Server auf der Server-Maschine mittels HTML. Der HTML-Text wird mit Hilfe einer Systemkomponente, genannt *Server-Erweiterung*, in den ODBC-kompatiblen SQL-Code übersetzt. Die Interpretierung und Ausführung von SQL-Code wird vom Datenbank-Server vorgenommen, der wiederum direkt mit der Datenquelle, z.B. einer Datenbank, arbeitet.

Das Ergebnis einer Abfrage wird von der Datenquelle über den Datenbank-Server zur Server-Erweiterung gesendet, die es in eine Form übersetzt, die der WWW-Server verarbeiten kann. Das Resultat erhält dann der WWW-Browser auf der Client-Maschine über das WWW, wo es auf dem Bildschirm angezeigt wird.

In Abbildung "Web-basiertes Datenbanksystem mit Server-Erweiterung" wird ein solches web-basiertes Datenbanksystem mit Server-Erweiterung grafisch dargestellt.



**Web-basiertes Datenbanksystem mit Server-Erweiterung**  
entnommen Ta98

### Client-Erweiterungen

Web-Browser wurden dafür entwickelt und optimiert, um eine leicht verständliche und einfach zu bedienende Schnittstelle zu Web-Sites aller Art bereitzustellen. Die Browser wurden nicht dazu entwickelt, um als Datenbankanwendungen zu dienen.

Um auf Client-Seite mit einer Datenbank über das Internet kommunizieren zu können, wird eine Funktionalität benötigt, welche von dem Browser nicht angeboten wird. Damit eine Datenbankanbindung über das Internet doch möglich ist, wurden mehrere Arten von Client-Erweiterungen entwickelt:

1. Helper-Anwendungen
2. Client-ErweiterungenPlugIns
3. Client-ErweiterungenActiveX-Controls
4. Client-ErweiterungenJava-Applets
5. Client-ErweiterungenSkripts

Gemeinsam ist allen, dass jeder HTML-Text, der mit Datenbankzugriffen zu tun hat, von der Server-Erweiterung zuerst in ODBC-kompatiblen SQL-Code übersetzt wird, ehe er an die Datenquelle weitergeleitet wird.

### ODBC und Intranet

Ein Intranet ist ein lokales Netzwerk (LAN) oder ein Wide Area Network (WAN), das genau wie das Internet arbeitet. Das Intranet befindet sich vollständig innerhalb einer einzelnen Organisation und ist nur deren Angehörigen zugänglich. Innerhalb eines Intranets wird i.a. auf besondere Sicherheitsmaßnahmen verzichtet.

Bei der Arbeit mit mehreren verschiedenen Datenquellen können Clients, Browser und die entsprechenden Client- und Server-Erweiterungen die Datenquellen benutzen, d.h. damit kommunizieren. Hierbei wird ODBC-kompatibler SQL-Code über die HTML- und ODBC-Stufen an den Treiber weitergeleitet, der den Code in die Befehlssprache der Datenbank übersetzt, die ihn schließlich ausführt.

## 8.5 JDBC

JDBC ist ein eingetragenes Markenzeichen und kein Akronym für Java Database Connectivity, wie oft angenommen.

Die Programmiersprache Java zeichnet sich u.a. durch ihre Plattformunabhängigkeit aus, da auf jedem Computer, auf dem eine **Java Virtual Machine** läuft, Java-Programme ablaufen können.

Um einen Standard für Datenbankzugriffe unter Java festzulegen, wurde JDBC-API entwickelt. Diese API (engl.: application programming interface) stellt eine Sammlung von Klassen und Schnittstellen mit einer bestimmten Funktionalität bereit. JDBC hat weiterhin das Bestreben, eine Anwendung **unabhängig von dem darunter liegenden Datenbanksystem** programmieren zu können. Damit dies gelingen kann, müssen die

verwendeten SQL-Anweisungen dem SQL Entry Level entsprechen, da sonst die Gefahr besteht, dass nur bestimmte Datenbanken angesprochen werden können.

JDBC wird auch als "low-level-Schnittstelle" bezeichnet, da SQL-Anweisungen als Zeichenketten direkt ausgeführt werden. Auf Basis des JDBC-API lassen sich dann "higher-level"-Anwendungen erstellen. Mit JDBC als Grundgerüst sind zwei Arten von "higher-level" APIs in der Entwicklung:

- **Embedded SQL für Java:** Hier können, anders als in JDBC, Variablen in SQL-Statements verwendet werden, um Werte mit der Datenbank auszutauschen. Hierbei bersetzt der Embedded-SQL-Präprozessor die Anweisungen in Java-Anweisungen und JDBC-Aufrufe.
- **Direkte Darstellung von Relationen und Tupeln in Form von Java-Klassen:** Bei dieser objekt-relationalen Darstellung entspricht ein Tupel einer Instanz einer Klasse, und die Attribute einer Relation repräsentieren die Attribute des Objektes. Der Programmierer sieht nur die Java-Objekte. Die SQL-Operationen zu dem Objekt laufen im Hintergrund ab.

## Architektur

Der Zugriff mit JDBC auf die Datenbank erfolgt über entsprechende vom Datenbankhersteller mitgelieferte Treiber, die mit den spezifischen DBMS, die angesprochen werden sollen, kommunizieren können. Dabei werden die SQL-Anweisungen in den entsprechenden SQL-Dialekt übersetzt und an die Datenbank gesendet. Die Ergebnisse werden dann an die aufrufende Instanz zurückgegeben. Die Voraussetzungen, um das zu ermöglichen, sind

1. geeignete Treiber für das darunter liegende Datenbanksystem, sowie
2. geeignete SQL-Befehle in dem SQL-Dialekt des darunter liegenden Datenbanksystems.

## Treiber

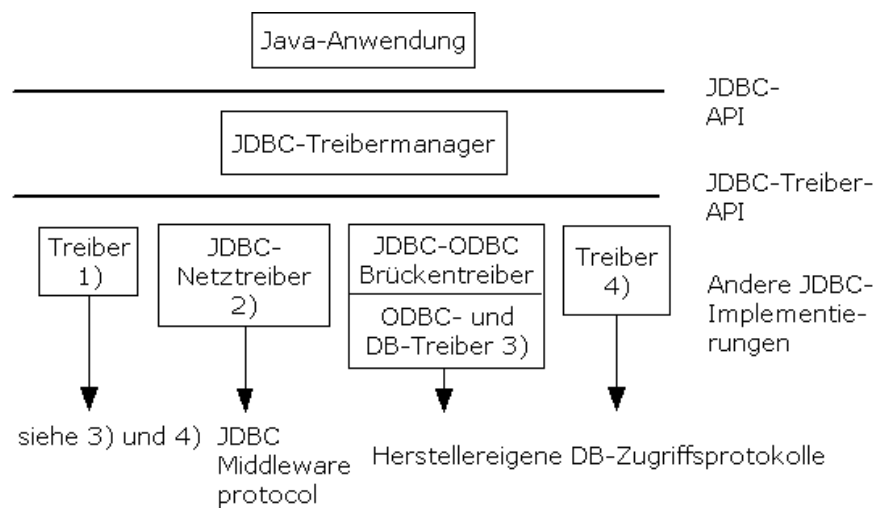
Kern von JDBC ist ein Treibermanager, der für die Verbindung der Java-Anwendung mit einem geeigneten JDBC-Treiber verantwortlich ist. Jedes der verwendeten Datenbanksysteme benötigt die Installation eines JDBC-Treibers. Die JDBC-Treiber, die derzeit verfügbar sind, fallen in eine der vier folgenden Kategorien:

1. *native protocol mit in reinem Java geschriebenen Treiber:* Der Treiber wandelt JDBC-Aufrufe in das vom DBMS verwendete Netzprotokoll um.
2. *JDBC-Netz mit reinem Java-Treiber:* JDBC-Aufrufe werden in ein von dem/den DBMS unabhängiges Netzprotokoll übersetzt, das dann auf einem Server in ein vom DBMS verwendetes Protokoll übersetzt wird.



3. *JDBC-ODBC-Brücke plus ODBC-Treiber*: Das JavaSoft-Brückenprodukt bietet JDBC-Zugriff über ODBC-Treiber.
4. *native API und teilweise in Java geschriebene Treiber*: JDBC-Aufrufe werden in Aufrufe von Client-APIs der entsprechenden Datenbankhersteller (z.B. Oracle, Sybase, Informix, DB2, usw.) übersetzt.

Die in 1) und 2) vorgestellten Treiber sind der bevorzugte Weg, um aus JDBC auf Datenbanken zuzugreifen, da hier die Installation automatisch aus einem Applet erfolgen kann, das vor Benutzung den reinen Java Treiber lädt. Um den Entwicklern eine Übergangslösung zu bieten, bis alle entsprechenden Treiber auf dem Markt sind, werden Treiber 3), 4) benutzt, die auf bereits vorhandenen (non-Java)-Treibern basieren.



**JDBC-Kategorien**  
vgl. KI98

Die Architektur von JDBC-Treiberimplementierungen wird in der Abbildung "JDBC-Kategorien" nochmal gezeigt. Die Tauglichkeit von JDBC-Treibern wird durch eine JDBC-Treiber-Testsuite überprüft und durch die Bezeichnung '**JDBC Compliant**' bestätigt. Diese Art Treiber unterstützt zumindest ANSI SQL-2 Entry Level, also SQL-92 Standard.

## JDBC-Befehle

Die Funktionalität von JDBC lässt sich in drei Bereiche unterteilen:

1. Aufbau einer Verbindung zur Datenbank
2. Senden von SQL-Anweisungen an die Datenbank
3. Verarbeiten der Ergebnismenge

Die Implementierung wird im Wesentlichen durch die Klassen `Connection`, `Statement` (und davon abgeleitete Klassen) und `ResultSet` des JDK (Java Development Kit) vorgenommen.

## Verbindungsaufbau

Die Verwaltung von Treibern wie auch der Aufbau von Verbindungen werden über die Klasse `DriverManager` abgewickelt. Vor dem Aufbau einer Verbindung zu einer Datenbank muss ein entsprechender Treiber geladen werden. Diese Aufgabe übernimmt die Methode `registerDriver` der Klasse `DriverManager`:

```
DriverManager.registerDriver(<driver-name>)
```

Der Klassenname, der in `<driver-name>` eingesetzt werden muss, ist der Installationsanleitung oder dem Handbuch des Datenbankherstellers zu entnehmen.

Um eine Verbindung zu der lokalen Datenbank aufzubauen, wird die Methode

```
DriverManager.getConnection(<jdbc-url>, <user-id>,  
                             <passwd>)
```

der Klasse `DriverManager` benutzt.

Entsprechend zu einer URL (Uniform Resource Locator) des Internets bezeichnet eine JDBC-URL eindeutig eine Datenbank. Eine JDBC-URL besteht aus drei Teilen:

```
<protocol>:<subprotocol>:<subname>
```

wobei

<code>&lt;protocol&gt;:</code>	innerhalb einer JDBC-URL immer für das JDBC Protokoll steht.
<code>&lt;subprotocol&gt;:</code>	bezeichnet den Treiber oder Datenbank-Zugriffsmechanismus, den ein Treiber unterstützt.
<code>&lt;subname&gt;</code>	indentifiziert die Datenbank; Aufbau erfolgt in Abhängigkeit des <code>&lt;subprotocol&gt;</code> .

Durch den Aufruf von

```
DriverManager.getConnection(<jdbc-url>, <user-id>,  
                             <passwd>)
```

wird für jeden registrierten Treiber `<driver>` die Methode `<driver>.connect(<jdbc-url>)` aufgerufen. Der Treiber, der als erster eine Verbindung zur Datenbank herstellen kann, wird genommen. Der Aufruf liefert eine

offene Verbindung zurück, die einem Bezeichner zugewiesen werden muss. Als Beispiel wird hier die Verbindung `conn` aufgebaut:

```
String url="jdbc:oracle:oci8@vfhlsl:1521:vfhl2";  
Connection conn=DriverManager.getConnection  
(url, "dummy", "passwd");
```

Mit der Verbindung an sich ist es noch nicht möglich, direkt SQL-Anweisungen an die Datenbank zu übergeben.

Mit der Methode `close` der Klasse `Connection` kann die Verbindung zur Datenbank wieder geschlossen werden.

#### JDBC-Beispielprogramm

### Versenden von SQL-Anweisungen

Zum Versenden von SQL-Anweisungen wird das `Statement`-Objekt verwendet. `Statement`-Objekte werden durch Aufruf der Methode `createStatement` (und verwandter Methoden) einer bestehenden Verbindung `<connection>` erzeugt. Derzeit existieren drei verschiedene Klassen von SQL-Anweisungen:

- *Statement*: Hier werden Instanzen der Klasse `per <connection>.createStatement()` erzeugt. Mit `Statement` können nur "einfache SQL-Anweisungen" ohne Parameter verarbeitet werden.
- *PreparedStatement*: Eine Instanz dieser Klasse wird `per <connection>.prepareStatement(<string> sql)` erzeugt, wobei die Klasse `PreparedStatement` von der Klasse `Statement` abgeleitet ist. Diese Klasse erlaubt neben der Validierung der SQL-Anweisungen auch das Erstellen von Hinweisen für den Optimierer (Vorübersetzung). Dabei ist die SQL-Anweisung im Objektzustand fest enthalten, was bei häufigem Aufruf des SQL-Statements effizienter ist, auch bei der Verwendung von Anfragen mit Parametern.
- *CallableStatement*: Instanzen dieser Klasse werden `per <connection>.prepareCall` erzeugt. Hiermit können in der Datenbank gespeicherte Methoden aufgerufen werden. Diese Klasse ist von `PreparedStatement` abgeleitet.

Um die erzeugte `Statement`-Instanz später wieder verwenden zu können, wird sie einem Bezeichner zugewiesen:

```
Statement <name> =  
<connection>.createStatement();
```

Die nun erzeugte Instanz der Klasse `Statement` kann verwendet werden, um SQL-Anweisungen an die Datenbank zu übermitteln. Abhängig von der Art des SQL-Statements geschieht dies über verschiedene Methoden. In den folgenden Angaben bezeichnet `<string>` eine SQL-Anweisung *ohne Semikolon*.

- `<statement>.executeQuery(<string>)`: Übergibt die SQL-Anfragen an die Datenbank. Die Ergebnismenge wird an die Instanz der Klasse `ResultSet` zurückgegeben.
- `<statement>.executeUpdate(<string>)`: Unter Update fallen alle SQL-Anweisungen, die eine Veränderung an der Datenbasis vornehmen, insbesondere alle DDL-Anweisungen (`CREATE TABLE`, etc.) und `INSERT`-, `UPDATE`- und `DELETE`-Anweisungen. Sollte man an dem Rückgabewert interessiert sein, kann `executeUpdate` entweder im Stil einer Prozedur (1.) oder einer Funktion (2.) aufgerufen werden.

1. `<statement>.executeUpdate(<string>);`

2. `int n=<statement>.executeUpdate(<string>);`

- `<statement>.execute(<string>)`: Die Methode `execute` wird eingesetzt, wenn ein Statement mehr als eine Ergebnismenge zurückliefert. Dies kann vorkommen, wenn verschiedene gespeicherte Prozeduren und SQL-Anweisungen nacheinander ausgeführt werden.

Einmal erzeugte `Statement`-Objekte können beliebig oft wiederverwendet werden, um SQL-Anweisungen zu übertragen. Es ist darauf zu achten, dass der Objektzustand durch Übermittlung des nächsten Statements verändert wird. Übertragungsdaten eines vorhergehenden Statements, wie z.B. Warnungen, sind dann nicht mehr erreichbar. Zum Schließen eines `Statement`-Objektes dient die Methode `close`.

## Behandlung von Ergebnismengen

Die Ergebnismenge der SQL-Anweisungen wird durch eine Instanz der Klasse `ResultSet` repräsentiert:

```
ResultSet <name>=<statement>.  
executeQuery(<string>);
```

Prinzipiell ist es eine virtuelle Tabelle, auf die von der Programmiersprache Java zugegriffen werden kann. Das `ResultSet`-Objekt benutzt einen Cursor, der mit der Methode `<result-set>.next` auf das nächste Tupel der Ergebnismenge gesetzt wird. Existieren die zugehörigen `ResultSet` und `Statement` Objekte nicht mehr, verliert auch der Cursor seine Gültigkeit. Sind alle Elemente eines `ResultSet`s gelesen, liefert `<result-set>.next` den booleschen Wert `false` zurück.

Um Zugriff auf einzelne Spalten des jeweils unter dem Cursor befindlichen Ergebnistupels zu bekommen, wird die Methode `<result-set>.get<type>(<attribute>)` eingesetzt. `<type>` steht für einen Datentyp und `<attribute>` kann entweder ein Attributname oder eine Spaltennummer (beginnend bei 1) sein. Mit der Methode `close` kann ein `ResultSet`-Objekt explizit geschlossen werden.

## 8.6 PHP-Anwendungen

PHP (rekursives Backronym für **PHP: Hypertext Preprocessor**) ist eine Skriptsprache für die Erstellung von dynamischen Webseiten und Webanwendungen. Sie kommt im Gegensatz zu anderen Skriptsprachen, wie z. B. JavaScript, auf dem Webserver zum Einsatz. PHP generiert auf Benutzeranfrage HTML-Code und liefert diesen zurück. Dabei kann PHP komplexe Funktionen ausführen sowie auf Daten zugreifen, die in einer Datenbank gespeichert sind.

Für die Programmierung einer PHP-Anwendung mit Datenbankanbindung werden ein Webserver mit PHP-Unterstützung sowie ein Datenbankmanagementsystem benötigt. Diese Komponenten stehen alle kostenlos im Internet zum Download bereit. Als Webserver empfiehlt sich der Apache HTTP-Server, der am weitesten verbreitete Webserver im Internet. Das Datenbankmanagementsystem kann nach den eigenen Bedürfnissen ausgewählt werden. Jedoch wird im Internet überwiegend auf die MySQL-Datenbank gesetzt. Die Konfiguration der Komponenten ist eine zeitaufwändige Prozedur und kann nur mit Fachwissen durchgeführt werden. Um sich eine eigene lokale Testumgebung zu erzeugen kann auf fertig konfigurierte Systeme, wie z. B. XAMP, zurückgegriffen werden. Diese lassen sich mit einem Doppelklick installieren und sind sofort einsatzbereit.

### Ablauf eines Datenbankzugriffes über PHP

1. Verbindung zum Datenbankserver aufbauen.
2. Datenbank auf Server auswählen.
3. SQL-Befehl in einer PHP-Variablen speichern.
4. SQL-Query an Datenbankserver schicken.
5. `Ressource_ID` in Variable speichern.
6. In einer Schleife die Daten über die `Ressource_ID` auslesen.
7. Datenbankverbindung schließen.



## Beispiel



```
<?PHP
$db = mysql_connect("$ip", "$username", "$password")
or die (mysql_error());
mysql_select_db("databasename")
or die (mysql_error());

$query = "SELECT
name,
vorname,
studiengang
FROM
studierende";
$result = mysql_query($query);
$result_num = mysql_num_rows($result);

echo "<table border = 1>";
echo " <tr>";
echo " <td>Name</td>";
echo " <td>Vorname</td>";
echo " <td>Studiengang</td>";
echo " </tr>";

for($i = 0; $i < $result_num; $i++) {
$name = mysql_result($result, $i, "name");
$vorname = mysql_result($result, $i, "vorname");
$studiengang = mysql_result($result, $i,
"studiengang");
echo "<tr>";
echo " <td>$name</td>";
echo " <td>$vorname</td>";
echo " <td>$studiengang</td>";
echo "</tr>";
}
```

```
echo "</table>";  
mysql_close($db);  
?>
```

**Codebeispiel:**

In der zweiten Zeile wird versucht, eine Verbindung zum Datenbankserver aufzubauen. Gelingt dies wird der Link in einer Variable gespeichert, ansonsten wird eine entsprechende MySQL-Fehlermeldung ausgegeben. Nachdem die Verbindung zum Datenbankserver hergestellt wurde, wird über den Namen des Datenbankschemas auf die entsprechende Datenbank zugegriffen. Sollte die Datenbank auf dem Server nicht existieren, wird wie bei der Verbindung zum Datenbankserver eine MySQL-Fehlermeldung ausgegeben. Danach wird das SQL-Statement in einer Variablen gespeichert und an den Datenbankserver abgesetzt. Die zurückgegebene Ressource\_ID macht die durch das SQL-Statement angefragten Daten in einer Art Array-Struktur über einen Index zugreifbar. Diese Daten können über eine Schleife in einer Tabelle auf dem Bildschirm ausgegeben werden. Dazu wird im Schleifenkopf lediglich die Anzahl an Durchläufen benötigt, die vorher über die Ressource\_ID ermittelt werden kann. Innerhalb der Schleife wird auf die Daten der Ressource\_ID zugegriffen und in einer Tabellenzeile ausgegeben. Nach Ausgabe der Daten wird die Datenbankverbindung geschlossen.

**Ausgabe**

Name	Vorname	Studiengang
Meier	Siegfried	PI
Schulze	Heiner	MI
König	Mathilde	PI
Baum	Meta	WI
Dreier	Magnus	TI
Hesse	Sarah	PI
...	...	...





## 8.7 PL/SQL

In der von Oracle entwickelten proprietären Programmiersprache PL/SQL wird die Anfragesprache SQL in eine prozedurale Programmiersprache integriert. PL/SQL bedeutet "procedural language extensions to SQL" und gehört zur dritten Generation der Programmiersprachen. Die Syntax von PL/SQL ist an die Programmiersprache

"Ada" vom amerikanischen Verteidigungsministerium angelehnt. PL/SQL erweitert SQL um zusätzliche Variablen, Bedingungen, Schleifen und Ausnahmebehandlungen. Ab der Version 8 der Oracle-RDBMS sind auch objektorientierte Merkmale vorhanden. PL/SQL ist im wesentlichen eine Basistechnologie, die in anderen Softwareprodukten verfügbar ist, aber nicht als freistehende Sprache existiert. Sie ist voll in Oracle integriert, d.h. SQL Anweisungen können direkt aus dem prozeduralen Code heraus aufgerufen werden. Somit ist gewährleistet, dass die SQL-Statements bereits durch Kompilieren syntaktisch überprüft werden können und nicht erst zur Laufzeit. Da PL/SQL eine Datenbanksprache ist, bleibt die Möglichkeit der Bildschirmausgabe auf reine Textausgaben beschränkt. Eine Interaktion mit dem Benutzer ist nicht direkt möglich. Um dies dennoch zu erreichen, muss das Front-End mittels einer weiteren Programmiersprache implementiert werden. Da PL/SQL inzwischen von anderen Datenbankherstellern implementiert wurde, hat das ANSI-Gremium diese SQL-Erweiterung standardisiert.

### Aufbau

Signifikant für ein PL/SQL Programm ist seine Blockstruktur, wobei die Reihenfolge der einzelnen Blöcke auch die Ausführungsreihenfolge des Programms bestimmt. Es existieren die Blöcke

 DECLARE	der Deklarationsblock
 BEGIN	der Ausführungsteil
 EXCEPTION	die Ausnahmeverarbeitung
 END ;	



PL/SQL Blöcke




Im **Deklarationsblock** werden die Variablen, Cursor und Unterblöcke deklariert bzw. initialisiert, auf die dann im Ausführungs- und dem Ausnahmeverarbeitungsteil Bezug genommen wird. Für die Variablen stehen Zeichenketten, numerische Datentypen, der Datentyp DATE, der binäre Datentyp BOOLEAN sowie benutzerdefinierte Datentypen zur Verfügung.

Der **Ausführungsteil** enthält die ausführbaren Anweisungen, die von der Laufzeitumgebung abgearbeitet werden und somit das PL/SQL-Programm steuern. Dieser Block wird durch die Schlüsselwörter BEGIN und END gekennzeichnet. Für sich wiederholende Anweisungen stehen drei verschiedene Schleifenkonstrukte zur Verfügung:

1. die **LOOP-Schleife** (einfache Schleife); hier wird der Schleifenrumpf mindestens einmal durchlaufen.

Die allgemeine Syntax lautet:



```
LOOP
...
EXIT WHEN <Bedingung>
...
END LOOP
```



#### Syntax der LOOP-Schleife

2. die **FOR-Schleife** (numerisch und Cursor); bei dieser Schleife wird der Schleifenrumpf n-mal durchlaufen, wobei eine Indexvariable von einem festgelegten Startwert bis zu einem festgelegten Endwert zählt.

Das Schlüsselwort REVERSE ermöglicht es, dass vom größeren zum kleineren Wert gezählt wird.

Die allgemeine Syntax lautet:



```
FOR <schleifenindex> IN [REVERSE] <kleinste Zahl> ..  
    <groesste Zahl>  
LOOP  
    <ausfuehrbare anweisung(en)>  
END LOOP;
```

Bei der Schleifenalternative **CURSOR FOR** werden Datensätze aus einer Tabelle in einen Cursor ausgelesen. Nach Abarbeitung der Schleife wird der Cursor automatisch geschlossen.

Die allgemeine Syntax lautet:



```
FOR <datensatz_index> IN <cursor_name>  
LOOP  
    <ausfuehrbare anweisung(en)>  
END LOOP;
```

3. die **WHILE-Schleife**; bei dieser Schleife ist die Abbruchbedingung bereits in den Schleifenkopf integriert. Sollte die Abbruchbedingung bereits zu Beginn gültig sein, wird diese Schleife keinmal durchlaufen.

Die allgemeine Syntax lautet:



```
WHILE <bedingung>  
LOOP  
    <ausfuehrbare anweisung(en)>  
END LOOP;
```

Jede dieser Schleifen hat denselben Aufbau. Zunächst erfolgt die Initialisierung; hier werden die Anfangswerte der zu durchlaufenden Zeilen oder Laufvariablen festgelegt. Dann folgt der Rumpf, indem die Anweisungen stehen, die wiederholt werden und die Wertänderung der Abbruchbedingung. Die Abbruchbedingung bestimmt, wann die Schleife enden soll.

Fehler bei der Anweisung zur Änderung des Wertes der Abbruchbedingung führen häufig zu Endlosschleifen.

Eine weitere Möglichkeit das Programm zu steuern ist das Benutzen von Kontrollstrukturen. Zu unterscheiden sind die

### 1. *bedingten Kontrollstrukturen;*

diese ermöglichen es Entscheidungen zu treffen.

a. Die Syntax für die direkte Entscheidung "wenn - dann" lautet:



```
IF <bedingung>  
THEN <ausfuehrbare Anweisung(en)>  
END IF;
```

b. Die Entscheidung mit einer Alternative "entweder - oder" wird mit der nachstehenden Syntax aufgerufen:



```
IF <bedingunge>  
THEN  
bei TRUE <ausfuehrbare Anweisung(en)>  
ELSE  
bei FALSE/NULL<ausfuehrbare Anweisung(en)>  
END IF;
```

c. Eine Fallunterscheidung kann im Quelltext über die Syntax



```
CASE <variable>
WHEN <ergebnis 1> THEN <anweisung 1>
WHEN <ergebnis 2> THEN <anweisung 2>
...
ELSE <anweisung_else>
END CASE;
```

erfolgen.

## 2. *sequentiellen Kontrollstrukturen;*

hiermit können Anweisungen gebildet werden, die nacheinander auftreten.

- a. Mittels einer Sprunganweisung kann zu einer anderen ausführbaren Anweisung im gleichen ausführbaren Abschnitt des PL/SQL Blocks verwiesen werden.

Die allgemeine Syntax lautet:



```
GOTO <label_name>;
...
<<label_name>>
<ausfuehrbare Anweisung(en)>
```

- b. Eine leere Anweisung erzeugt folgende Syntax:



```
NULL;
```

Im **Ausnahmeverarbeitungsblock** werden die Ausnahmen, die während der Laufzeit im Programm auftreten, behandelt. Hierbei wird unterschieden zwischen vordefinierten Exceptions und benutzerdefinierten Exceptions. Die vordefinierten Exceptions werden automatisch von PL/SQL ausgelöst und können anhand von festgelegtem Fehlercode im Programm verwendet werden. Der Fehlercode besteht aus den Buchstaben ORA- und 5 Ziffern z.B. ORA-01017 bedeutet: Benutzername oder Passwort falsch.

Für die Behandlung logischer Programmfehler sind die benutzerdefinierten Exceptions vorgesehen. Sie werden wie folgt definiert:



```
<exception_name> EXCEPTION;  
PRAGMA EXCEPTION_INIT(<exception_name>, -  
<exception_number>);
```

und ausgelöst mit dem Kommando



```
RAISE <exception_name>
```



#### Beispiel

Für die Miniwelt "Hochschule" wird ein PL/SQL - Programm entwickelt, dass das Rückgabedatum der ausgeliehenen Bücher überprüft. Liegt es in der Weihnachtspause, so wird es auf das Ende der Weihnachtspause verlegt.



```
DECLARE  
    CURSOR rueckgabe_cur IS  
        SELECT MatrNr, ExemplarNr, ISBN  
        FROM leihen_aus WHERE Rueckgabedatum IN 24.12.2013 ..  
        01.01.2014;
```

```
BEGIN
  FOR rueckgabe_rec IN rueckgabe_cur
  LOOP
    UPDATE leihen_aus
      SET Rueckgabedatum := 02.01.2014;
    DBMS_OUTPUT.PUT_LINE('Das Rückgabedatum von Matrikelnr '
      || rueckgabe_rec.MatrNr, || 'wurde
      verschoben auf den 02.01.2014');
  END LOOP;
END;
```

Das oben stehende Beispiel enthält einen anonymen Block, in welchem der Cursor `rueckgabe_cur` verwendet wird, um die Matrikelnummer, die Exemplarnr. des Buches und die ISBN herauszufinden, die ein zu frühes Rückgabedatum haben. Dieses wird dann mittels `UPDATE` auf das neue Datum gesetzt.

## 8.8 Sicherheit bei datenbankgestützten Webapplikationen - Die SQL-Injection

Eine **SQL-Injection**, übersetzt als SQL-Einschleusung, tritt dann auf, wenn zur Datenbankabfrage dynamische SQL-Anweisungen verwendet werden. Dies ist der Fall bei datenbankgestützten Webapplikationen, die ihre Eingaben und Parameter vom Browser aus per URL, Formular und der HTTP-Methode POST oder per Cookies übermitteln.

Durch Manipulation dieser dynamischen SQL-Anfrage kann das Verhalten der Webapplikation bewußt verändert werden. Dieser Vorgang wird SQL-Injection genannt.

Um SQL-Injections entgegen wirken zu können, müssen dessen Angriffsmethoden und die Vorgehensweisen bekannt sein.

### Das Ziel einer SQL-Injection

Eine SQL-Injection ist möglich bei jeder Webapplikation, die auf einer Datenbank agiert. Dabei ist die SQL-Injection technologieunabhängig. Ziele des Angreifers sind

- das Verfälschen des Ablaufs der Webanwendung,
- das Lesen und Schreiben von Datenbankeinträgen und Dateien,
- das Ausführen von Systembefehlen, was einer Serverübernahme gleich kommt.

### Angriffsmethoden und Vorgehensweise

Eingeschleuste SQL-Anweisungen des Angreifers, können die dynamische SQL-Anfrage der Webapplikation manipulieren. Wie bereits im Abschnitt Dynamic SQL beschrieben, werden bei einer dynamischen SQL-Anfrage die einzelnen SQL-Anweisungen mit den Eingabeparametern der Benutzer erst während der Programmlaufzeit zusammengesetzt. Dieser Vorgang ermöglicht dem Angreifer die SQL-Anfrage so zu verändern bzw. zu erweitern, dass sein Ziel damit erreicht wird. Erst mit der zusammengesetzten SQL-Anfrage wird die Datenbankabfrage durchgeführt. Nachstehende Vorgehensmuster sind bekannt, wobei SQL-Injection anfälliger SQL-Anweisungen vorauszusetzen sind.

1. Ohne die Kenntnis einer legalen Benutzername-Passwort-Kombination kann ein Benutzer sich mittels eines **Authentication Bypass** einloggen.



Beispiel



```
SELECT Daten FROM Benutzer
WHERE Name = 'Benutzername'
AND Passwort = 'irgendetwas' OR '1' = '1';
```

Mit dieser SELECT-Anweisung werden alle "Daten" des "Benutzernamen" ausgegeben, da der angehängte BOOLEAN "OR '1' = '1'" immer gültig ist.

2. Ein **Informationsdiebstahl** ist möglich, wenn über eine Mengenoperation UNION eine zweite SELECT-Anweisung eingefügt wird, um die benötigten Informationen auszulesen.



Beispiel



Beispiel aus der .

```
SELECT Name, Vorname, MatrNr
FROM Studierende
WHERE GebDat TO_DATE('01.01.1985', 'DD.MM.YYY') AND '1' =
'0'
```

```
UNION
SELECT Name, MatrNr, GebDat
FROM Studierende
```

Die SELECT-Abfrage nach allen Studierenden mit Name, Vorname und MatrNr, die vor 1985 geboren sind, wird durch den BOOLEAN-Ausdruck "AND '1' = '0'" unterdrückt. Der Angreifer kann seine Abfrage nach "Name", "MatrNr" und "GebDat" über eine UNION-Anweisung einfügen.

3. Ein **Denial of Service** ist eine Überlastung des Servers. Dies ist möglich mittels einer Anfrage, die die kompletten Datenbankressourcen aufbraucht.



Beispiel

Beispiel aus der .

Gesucht sind alle "Titel" der Relation "Buch" mit den ISBN 3897212188 und 3453140982.

SQL-Anweisung:



```
SELECT Titel
FROM Buch
WHERE ISBN LIKE '3897212188' OR ISBN LIKE '3453140982';
```

Wird die Benutzereingabe in der LIKE-Anweisung durch ein Wildcard ersetzt und handelt es sich um eine Liste mit vielen Daten, so kann diese Funktion für eine hohe Zeitverzögerung sorgen, und so die durch den Server bereitgestellten Dienste arbeitsunfähig zu machen.

SQL-Anweisung mit SQL-Injection:



```
SELECT Titel
FROM Buch
WHERE ISBN LIKE '%';
```

4. Anweisungen der **Datenmanipulation** sind DROP, DELETE, UPDATE oder INSERT. Bei einigen DBMS z.B. MySQL sind in einer SQL-Anweisung mehrere Anfragen möglich. Diese werden durch ein Semikolon voneinander getrennt.





Beispiel

Beispiel aus der  
SQL-Anweisung:



```
DELETE FROM Buch
WHERE Titel = 'Pustebblume';
```

Durch hinzufügen eines BOOLEAN-Ausdruck kann eigener SQL-Code eingeschleust werden, der es bei diesem Beispiel erlaubt, die Leihfrist aller Bücher auf "NULL" zu löschen.



Beispiel

Beispiel aus der  
SQL-Anweisung mit SQL-Injection:



```
DELETE FROM Buch
WHERE Titel = 'Pustebblume' OR '1' = '1';
UPDATE Buch SET Leihfrist = NULL
WHERE Titel = 'Pustebblume' OR '1' = '1';
```

5. Da die DBMS den Kunden immer mehr Funktionen und Erweiterungen anbieten die oft nicht benötigt werden, kann ein Missbrauch dieser Funktionen einen Zugang zu dem darunter liegenden Betriebssystem ermöglichen und so eine **Serverübernahme** zulassen.



Beispiel

Bei bekannter Datenbankserveradresse und Erreichbarkeit des standardmäßig eingerichteten Datenbankport 1521 reicht es, die Zugangsdaten für den Administrator zu kennen. Diese sind durch Authentication Bypass herauszufinden:



```
SELECT * FROM Benutzer
```

```
WHERE Name = 'admin'  
OR '1' = '1'
```

Die SELECT-Abfrage liest die Logindaten für den Benutzer "admin" aus der Datenbank.

Die Datenbankadresse kann mit folgender SQL-Injection erfahren werden:



```
SELECT Titel  
FROM Buch  
WHERE ISBN = '3897212188' AND 1 =  
UTL_INADDR.GET_HOST_NAME( ( SELECT Benutzer FROM  
DUAL ) ) -- ' ;
```

Die SELECT-Anfrage wird unterdrückt, um mit der SQL-Injection einen Fehler hervorzurufen. Dazu wird versucht, die Datenbankserveradresse eines beliebigen Benutzers aus der Standard ORACLE Tabelle DUAL zu ermitteln.

Im günstigsten Fall für den Angreifer, wird in der Fehlermeldung bereits die gesuchte Datenbankserveradresse enthalten sein. Die Datenbankserveradresse ermöglicht es, sich als Administrator einzuloggen, z.B. mittels eines Javaprogramms und JDBC. Mit Hilfe des Systempakets "UTL\_FILE" können dann Dateioperationen vorgenommen werden.

Eine geeignete Gegenmaßnahme ist es, den Benutzern so wenig Rechte wie möglich zugeben, so dass nur von ausgewählten Funktionen Gebrauch gemacht werden kann.

### Verdeckte SQL-Injection

Bei einer verdeckten SQL-Injection ist für den Angreifer zunächst nicht ersichtlich, dass eine Manipulation der SQL-Anfrage möglich ist. Es wird unterschieden nach:

1. der **Blind SQL-Injection** und
2. den **Stored Procedures Injection**.

Bei der **Blind SQL-Injection** agiert der Angreifer ohne Bildschirmausgaben, d.h. direkte SQL-Fehlermeldungen sind abgeschaltet. So hat der Angreifer die Reaktion der Webanwendung auszuwerten. Im Vergleich mit den üblichen Benutzereingaben kann darauf geschlossen werden, ob SQL-Injection möglich ist.



```
<?PHP

$query = "SELECT
name,
vorname,
studiengang
FROM
studierende
WHERE id = $id";
$result = mysql_query($query);
$result_num = mysql_num_rows($result);

?>
```

**Beispiel aus der Miniwelt****Auszug aus dem PHP-Codebeispiel mit WHERE**

Der PHP-Code gibt den Namen, den Vornamen und den Studiengang aus der Tabelle Studierende aus, der in der Webapplikation durch den "id-Wert" ausgewählt wird.

Der Angreifer überprüft die Reaktion der SQL-Anweisung auf SQL-Injection, indem er den "id-Wert" durch ein Sonderzeichen ersetzt. Daraufhin wird nicht die Relation "Studierende" ausgegeben, sondern der Vorgang wird wegen falscher SQL-Syntax abgebrochen. Dies zeigt dem Angreifer, dass eine SQL-Injection möglich ist und die SQL-Anfrage nach seinen Wünschen manipuliert werden kann.

Die **Stored Procedures** werden wie Funktionen über ihren Namen und eine Parameterliste aufgerufen. Insofern findet auch hier eine Stringverkettung statt und eine SQL-Injection ist möglich. Unter ORACLE werden Stored Procedure in PL/SQL geschrieben.

Jede darin erstellte Stored Procedure wird mit den Rechten ausgeführt, mit denen sie erstellt wurde. Insofern wird eine Stored Procedure mit Administrationsrechten ausgeführt, wenn diese mit diesen Rechten erstellt wurde. So ist, gleichwohl der Nutzung von Stored Procedure, eine SQL-Injection möglich, wie nachstehendes Beispiel zeigt.



```
DECLARE
```

```
CREATE OR REPLACE FUNCTION neu_leihdatum(  
    f_datum IN DATE  
)RETURN DATE  
IS  
    CURSOR rueckgabe_cur IS  
        SELECT MatrNr, ExemplarNr, ISBN  
        FROM leihen_aus  
        WHERE Rueckgabedatum IN f_datum;  
BEGIN  
    EXECUTE IMMEDIATE  
        FOR rueckgabe_rec IN rueckgabe_cur  
        LOOP  
            UPDATE leihen_aus  
            SET Rueckgabedatum := 02.01.2014;  
            DBMS_OUTPUT.PUT_LINE('Das      Rückgabedatum      von  
Matrikelnummer' ||  
rueckgabe_rec.MatrNr || 'wurde  
verschoben auf den 02.01.2014');  
        END LOOP;  
    END;
```

**Beispiel aus der Miniwelt****PL/SQL Codebeispiel mit Variablen zu veränderndem Rückgabedatum**

Im DECLARE-Block wird die Funktion "neu\_leihdatum" angelegt bzw. überschrieben sollte sie schon existieren. Der Funktionsparameter f\_datum ist vom Typ DATE und der Rückgabewert ist vom Typ DATE.

Der Cursor rueckgabe\_cur liest die SELECT-Anweisung mit dem von der Webapplikation ermittelten Datumsbereich ein. Die Anweisung "EXECUTE IMMEDIATE" bewirkt, dass der BEGIN-Block sofort ausgeführt wird.

In einer LOOP-Schleife wird dann das alte "Rueckgabedatum" auf das neue "Rueckgabedatum", den 02.01.2014, gesetzt.

Als Angreifer ist es aufgrund der dynamischen Verkettung möglich, die Rückgabedaten immer wieder auf den 02.01.2014 zu verlegen.

**Gegenmaßnahmen**

An oberster Stelle steht die **Validierung**. Die Eingaben sollten auf Zeichen mit Sonderfunktionen überprüft werden. Dies sind \, ", ', und ;. Bei gefundenen Zeichen muss ein Codeumbau in der Webapplikation erfolgen, da diese für die korrekte Prüfung der Eingabedaten zuständig ist.

Skriptsprachen wie PHP kennen keine Datentypen. Aus diesem Grund nehmen diese Programmentwickler typischerweise keine **Umwandlung in numerische Daten** vor. Eine Typ-Umwandlung der Variablen bietet jedoch bereits einen geringen Schutz vor SQL-Injection-Angriffen: wenn die Datenbank numerische Daten erwartet, gibt sie eine Fehlermeldung aus bei der Eingabe von Daten des Typs String. Insofern ist bei einer Programmentwicklung in einer Skriptsprache auf Typumwandlungen zu achten.

Eine weitere Möglichkeit die unverhüllten Benutzereingaben vom SQL-Interpreter fernzuhalten sind **Prepared Statements**. Hierbei werden die Eingabedaten an eine bereits kompilierte Anfrage weitergeleitet. So brauchen die Benutzerdaten nicht mehr interpretiert werden, und es erfolgt keine dynamische Stringverkettung der SQL-Anfrage.

Eine **Web-Application-Firewall (WAF)** analysiert die Kommunikation des Benutzers mit der Webapplikation. Bei verdächtigen Mustern und Manipulationen wird der Zugriff unterbunden, bevor die Benutzerdaten zum Web-Server gelangen können.

Bei der Verwendung von **Bind Variablen** wird der Zeitpunkt, zu der die Variable durch ihren Wert ersetzt wird, verschoben: in der dynamischen SQL-Anfrage findet die Ersetzung auf Anwendungsseite, d.h. in der Webapplikation, statt, bevor die Datenbank die SQL-Anweisungen syntaktisch überprüfen und interpretieren kann. Werden jetzt Bind-Variablen verwendet, überprüft die Datenbank die SQL-Anweisungen zunächst syntaktisch und ersetzt die Variablen durch ihre Inhalte erst anschließend.

Somit ist es nicht möglich, die Struktur und den Sinn der SQL-Anfrage durch eine Änderung der Parameter zu manipulieren. Dieses Verfahren ermöglicht es auch sichere Stored Procedure zu programmieren.

Ein Nachteil ist jedoch, dass mit Bind-Variablen Relation- oder Attribut-Namen nicht dynamisch gestaltet werden können.

Neben den Möglichkeiten in der Webapplikation eine Codeumstellung vorzunehmen, gibt es auch noch die Alternative dem **Datenbankserver nur die nötigen Rechte für den Benutzer** zu geben.

**Fazit:** Das Risiko einer SQL-Injection kann durch geeigneten Programmcode stark reduziert werden.

## 8.9 4GL-Systeme

4GL ist eine Abkürzung für Fourth Generation Language (Programmiersprachen der vierten Generation).

1. Generation: binäre Maschinensprache 2. Generation: Assembler Sprache (ab 50er Jahre) 3. Generation: ALGOL, COBOL, FORTRAN, PL/I, PASCAL, C 4. Generation: Integration von DB-Sprachen und Programmiersprachen

Das Einsatz-Umfeld der 4GL-Systeme ist fast ausschließlich die kommerzielle Informatik-Anwendungswelt.

Die 4GL-Dialekte unterliegen bisher keinerlei Standardisierungsbemühungen, so dass 4GL-Anwendungen nicht portabel sind und sogar der Funktionsumfang verschiedener 4GL-SQL-Dialekte nur sehr schwer vergleichbar ist.

Um die komfortable Entwicklung von benutzungsfreundlichen Oberflächen zu unterstützen, werden von manchen 4GL-Systemen Prozeduren zur Handhabung von Masken und grafische Editoren zur Layoutgestaltung angeboten. Es gibt auch die Möglichkeit, selbst Prozeduren zu implementieren. Daher sind 4GL-Systeme für Rapid-Prototyping gut geeignet.

#### 4GL-SQL

Ein 4GL-SQL-Dialekt besteht aus SQL, Kontrollkonstrukte wie "IF ... THEN ...", "WHILE" usw. Es ist auch möglich, lokale Variablen zu definieren, die oft nicht streng typisiert sind. Des Weiteren können Prozeduren spezifiziert bzw. externe Prozeduren integriert werden.



#### Beispiel

Bei dem Beispiel handelt es sich um Ausschnitte aus dem Quellcode einer Oberfläche zur Erstellung von Institutsseiten. Die Oberfläche bietet auch die Möglichkeit der Dateneingabe an. Die Auslassungspunkte ([...]) deuten die aus Platzgründen ausgelassenen Teile des Quelltextes an. Beim Anklicken des Menüauswahlpunktes "Mitarbeiter ... Dateneingabe" wird das Frame für die Dateneingabe aufgerufen:



```
ON CLICK menu.mitarb_menu.meinfg_menu =  
BEGIN CALLFRAME Dateneingabe; END;
```

Das Frame "Dateneingabe" bietet die Möglichkeit an, die Mitarbeiterdaten in ein Formular einzugeben und sie dann in die Datenbank durch Anklicken einer Einfügen-Schaltfläche einzufügen:



Beispiel-Frame Dateneingabe (4GL-Systeme)



```
ON CLICK einfuegen_btn =
BEGIN
  IF vorname=' ' OR nachname=' ' OR email=' '
  [...]

  /* Falls eines der obigen Felder
  leer ist, dann Fehlermeldung ... */

  THEN MESSAGE 'Bitte Daten vervollständigen!';

  /* ... sonst Event 'einfuegen' aufrufen */

  ELSE CurFrame.SendUserEvent
(eventname='einfuegen',
messagevarchar='einfuegen');
ENDIF;

END;
```



```
ON USEREVENT einfuegen =
BEGIN
```

```
[...]
INSERT INTO Mitarbeiter
  (vorname, nachname, email, [...])
VALUES (:vorname, :nachname, :email, [...]);

[...];

END
```

Die Erstellung einer solchen Oberfläche erfordert einige Arbeitszeit, aber wenn man die gleiche Funktionalität durch direkte Programmierung ohne ein 4GL-System erreichen wollte, wäre der notwendige Aufwand viel größer.

### Gegenüberstellung von Datenbankverbindungen

	<b>Call-Schnittstelle</b>	<b>EmbeddedSQL</b>	<b>Dynamic SQL</b>	<b>4GL-SQL</b>
Einbettung	statisch (Vorübersetzer-Prinzip)	statisch	dynamisch	statisch
Standard	ja (Call Level Interface)	ja	ja	nein
Einbettung der SQL-Anweisung	zur Übersetzungszeit	zur Übersetzungszeit	zur Laufzeit	zur Übersetzungszeit
Komplexität	hoch	hoch	hoch	hoch
Kosten	gering (bei Beschaffung mitgeliefert)	gering (wird bei den meisten Datenbanken mitgeliefert)	gering	hoch

### Report-Generatoren

Report-Generatoren ermöglichen das Publizieren von Datenbankinhalten im Intra- und Internet. Sie beinhalten u.a. das Erstellen von Berichten und Rechnungen sowie die formatierte Ausgabe. Es gibt Report-Generatoren von verschiedenen Firmen.

*Vorteile von Report-Generatoren:*

1. Qualität und Aktualität per Knopfdruck



2. Formatierung mittels multipler Abfrage
3. Zugriff auf Datenbanken des Typs Microsoft SQL Server, Sybase, Informix, DB2, Oracle und jede ODBC-fähige Datenquelle
4. Skalierbarkeit
5. Einfache Bedienbarkeit
6. Wenig Aufwand bei der Rechnerumgebung
7. Informationen per Selbstbedienung

*Nachteil der Report-Generatoren:* hoher Lizenzpreis

Zum Beispiel bietet das Datenbanksystem Oracle folgende Funktionen an:

- Report Builder: zur Datenbankauswertung
- Graphics Builder: zur grafischen Darstellung der Berichtsdaten
- Translation Builder: Übersetzung von Berichten in anderen Sprachen
- Reports Server: skalierbare Umgebung für zentralen Einsatz und Ausführung von Berichten

## 8.10 Ergänzende Informationen

Im Folgenden finden Sie einige ergänzende bzw. weiterführende Informationen, auf die bereits im Laufe der Lerneinheit verwiesen wurde:

"Beispiel zur Call-Schnittstelle" enthält ein kommentiertes Beispiel (bezogen auf das Datenbanksystem Sybase), wie die Call-Schnittstelle zur Anwendung kommt.

"Relation für das Beispiel zu Embedded SQL" ist die Relation, auf die sich das Beispiel anw.2 bezieht.

"Syntax von EXEC SQL WHENEVER" enthält das Syntax-Diagramm der Anweisung zur Fehlerbehandlung in Embedded SQL.

"Syntax von EXEC SQL Cursor" beinhaltet das Syntax-Diagramm zur Cursor-Benutzung (wenn mehrere Tupel einer SELECT-Anweisung über eine Embedded SQL-Anweisung von einer Datenbank eingelesen werden sollen).

"Beispiel: Umfangberechnung eines Polygons" ist ein Embedded SQL-Programm, mit dessen Hilfe der Umfang eines Polygons berechnet werden kann. Das Beispiel ist kommentiert.

"Client-Erweiterungen" listet die Erweiterungen auf, die auf der Seite des Clients notwendig sind, um mit einer Datenbank über das Internet kommunizieren zu können.



## Gliederung

- 8.10 Ergänzende Informationen
- 8.10.1 Beispiel zur Call-Schnittstelle
- 8.10.2 Relation für das Beispiel zu Embedded SQL
- 8.10.3 Syntax von EXEC SQL WHENEVER
- 8.10.4 Syntax von EXEC SQL Cursor
- 8.10.5 Beispiel: Umfangberechnung eines Polygons
- 8.10.6 Beispiel zu Dynamic SQL
- 8.10.7 Client-Erweiterungen
- 8.10.8 JDBC-Beispielprogramm

## 8.10.1 Beispiel zur Call-Schnittstelle

*in der Programmiersprache C für das Datenbanksystem Sybase geschrieben*



```
# include <sybfont.h>
# include <sybdb.h>
# include <syberror.h>
int err_handler();
int msg_handler();
main()
{
    DBPROCESS *dbproc;
    LOGINREC *login;
    DBINT  persnr;
    DBCHAR  name[40];
    if (dbinit() == FAIL) exit (ERREXIT);
    dberrhandle (err_handler);
    dbMSGhandle (msg_handler);
    login = dblogin ();
    DBSETLPWD(login,"p72");
    DBSETLAPP (login,"call_bspl");
    dbproc = dbopen (login, NULL);
    dbcmd (dbproc, "SELECT PersNr, Name
```

```
FROM MitarbeiterIn");
dbcmd (dbproc, "WHERE extern = 'False'");
dbsqlxec (dbproc);
if (dbresults(dbproc) == SUCCEED)
{
    dbbind (dbproc, 1, INTBIND, (DBINT)0, persnr);
    dbbind (dbproc, 2, STRINGBIND, (DBINT)0, name);
    while (dbnextrow(dbproc) != NO_MORE_ROWS)
    { printf ( "%d, %s\n", persnr, name ); }
}
dbexit ();
}
```

**Benutzung der Call-Schnittstelle  
entnommen Ne96**

In den Zeilen 1-3 werden Bibliotheksdateien des Datenbanksystems (sogenannte Headerfiles) mit vordefinierten Konstanten und Dateistrukturen zugeladen. Die Datei "sybfont.h" enthält Konstanten, die von Datenbank-Prozeduren als Rückgabewert benutzt werden. In "sybdb.h" sind wichtige Typdefinitionen enthalten z.B. auch die Definition der zentralen Kommunikationsstruktur "DBPROCESS", die von zahlreichen Prozeduren benutzt wird, so z.B. beim Ausführen von Anfragen (Zeile 22). Die Datei "sysberror.h" schließlich enthält Fehlermeldungen.

Nach den Deklarationen der Kommunikations- und Login-Struktur folgen noch die der Variablen, die später die Personalnummer und Namen der gesuchten Mitarbeiter aufnehmen sollen (Zeile 10, 11). Mit dem obligaten Prozeduraufruf "dbinit()" muss die Datenbank-Bibliothek initialisiert werden. Die Zeilen 13 und 14 sind Vorwärtsdeklarationen, d.h. Vorweckaufrufe, von Funktionen zur Fehler- und Nachrichten-Behandlung. Diese Prozeduren werden automatisch ausgeführt, sobald das Datenbanksystem einen Fehler oder eine Meldung an das laufende Anwendungsprogramm übergibt. Die Implementierung der Routinen "dberrhandle" und "dbMSGhandle" muss vom Anwendungsprogrammierer vorgenommen werden, da sie nicht in der Datenbank-Bibliothek enthalten sind.

Die Zeilen 15-18 dienen zum Anmelden des Benutzers und der Anwendung beim Datenbanksystem. Mit der Anweisung "login=dblogin()" wird zunächst eine leere Login-Struktur in der Variablen "login" zur Verfügung gestellt. In dieser werden dann durch zwei weitere Funktionsaufrufe (Zeile 16 und 17) das Passwort und der Name der Anwendung eingetragen. Schließlich wird die so aufgefüllte Struktur als Argument der Funktion "dbopen" übergeben, die ihrerseits ein Ergebnis in "dbproc" zurückliefert.

War der Anmeldevorgang erfolgreich, kann die eigentliche Anfrage zusammengestellt werden (Zeile 19 und 21).

Hier kommt die Funktion "dbcmd" ins Spiel, die die zu übergebenden Argumente bei wiederholten Aufrufen in einen Kommandopuffer innerhalb der Struktur "DBPROCESS" hintereinander einträgt. Die dadurch erzeugte Datenbank-Anweisung wird mit Hilfe der Funktion "dbsqlxec" dem Datenbanksystem übergeben, das die Anweisung übersetzt und ausgeführt (Zeile 22).

Sollte die Anweisung erfolgreich ausgeführt worden sein (Zeile 23) werden die beiden Attribute der Ergebnisrelation an die oben deklarierten Variablen "persnr" und "name" weitergegeben (Zeile 25 und 26). Die zugehörige Routine "dbbind" kann dabei auch zu Konvertierungen zwischen Datentypen benutzt werden.

Die Ergebnistupel werden durch den Aufruf von "dbnextrow" (Zeile 27) nacheinander in die angegebene Variable geschrieben und mit "printf" auf dem Bildschirm ausgegeben. Abschließend wird in Zeile 30 die Verbindung zum Datenbanksystem wieder geschlossen.

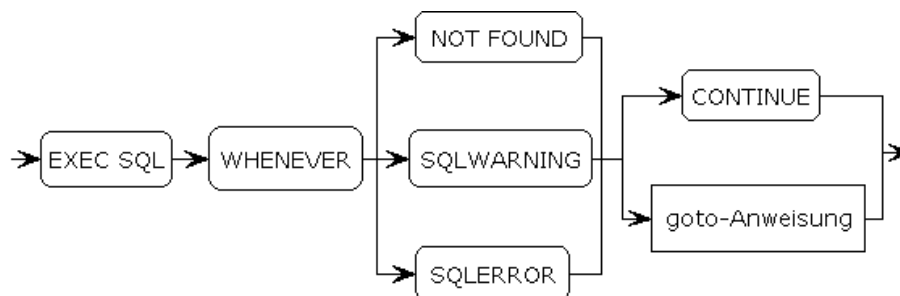
## 8.10.2 Relation für das Beispiel zu Embedded SQL



```
EXEC SQL DECLARE MitarbeiterIn TABLE
    (PersNr INTEGER,
    Name VARCHAR(40),
    Vorname VARCHAR(20),
    weiblich CHAR(5),
    extern CHAR(5),
    Strasse VARCHAR(20),
    Hausnr CHAR(4),
    Plz CHAR(8),
    Ort VARCHAR(40),
    TNr CHAR(16),
    Urlaubsanspruch INTEGER,
    Jahresurlaub INTEGER,
    Resturlaub INTEGER,
    Institut VARCHAR(60));
```

### 8.10.3 Syntax von EXEC SQL WHENEVER

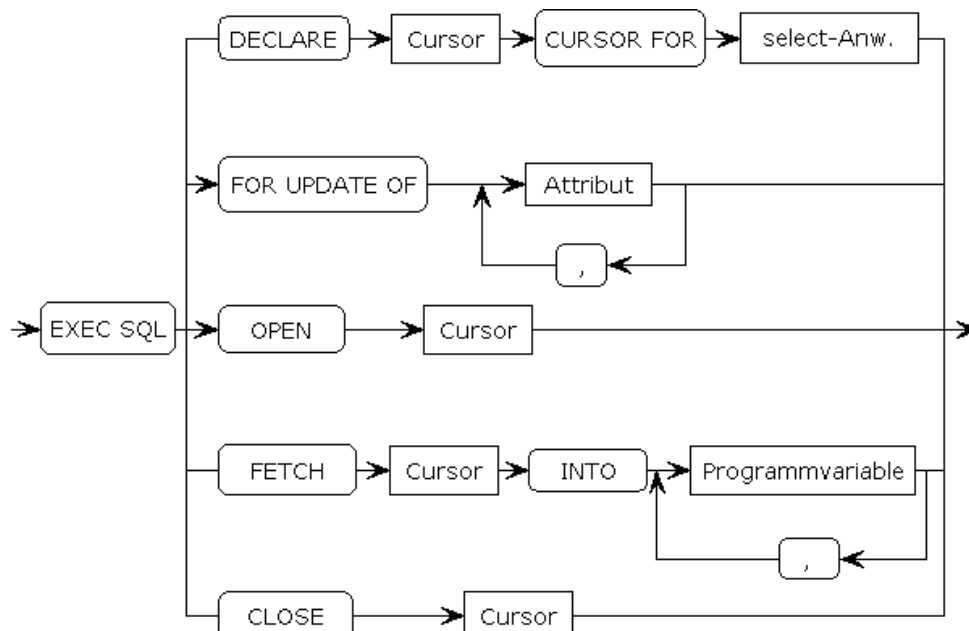
In der Abbildung sehen Sie die Syntax, um eine Fehlerbehandlung mit einer Embedded SQL-Anweisung durchzuführen.



Syntax der WHENEVER-Anweisung

### 8.10.4 Syntax von EXEC SQL Cursor

In der Abbildung sehen Sie die Syntax, um mehrere Tupel eines SELECT-Befehls über eine Embedded SQL-Anweisung von einer Datenbank einzulesen.



Syntax der Cursor-Anweisung bei Embedded SQL

### 8.10.5 Beispiel: Umfangberechnung eines Polygons



```
EXEC SQL INCLUDE SQLCA;
EXEC SQL BEGIN DECLARE SECTION;
  DCL PktNr, X1, Y1, Xi, Yi INTEGER;
  DCL Id CHAR(5);
EXEC SQL END DECLARE SECTION;
EXEC SQL DECLARE Polygone TABLE
  (ID CHAR(5),
  PktNr INTEGER,
  X INTEGER,
  Y INTEGER);
EXEC SQL WHENEVER SQLERROR
  GOTO SQLFehlerbehandlung;
  DCL Xminus1, Yminus1 INTEGER;
  DCL Umfang:= 0.0 REAL;
  DCL weitereKoordinaten:=
    TRUE BOOL;
EXEC SQL DECLARE Stuetzpunkte CURSOR
FOR SELECT PktNr, X, Y
  FROM POLYGONE
  WHERE Id= :Id
  ORDER BY PktNr;
PUT ('Umfang von Polygon mit Id = ?');
GET (Id);
EXEC SQL OPEN Stuetzpunkt;
EXEC SQL FETCH Stuetzpunkt INTO
  :PunktNr,:X1,:Y1;
Xminus1:= X1;
Yminus1:= Y1;
WHILE weitereKoordinaten = TRUE DO
BEGIN
  EXEC SQL FETCH Stuetzpunkte
    INTO :PunktNr, :Xi,:Yi;
  IF SQLCODE = 100
```

```
THEN weitereKoordinaten = FALSE;
ELSE Umfang := Umfang +
SQRT((Xi-Ximinus1)^2
+ (Yi-Yiminus1)^2);
Ximinus1 := Xi;
Yiminus1 := Yi;
ENDIF;
END;
Umfang := Umfang + SQRT((X1-Ximinus1)^2 +
(Y1-Yiminus1)^2);
PUT(Umfang);
EXEC SQL CLOSE Stuetzpunkt;
Umfangberechnung von Polygonen
Das Programm wurde Ne99 entnommen.
```

Die "SQL Communication Area" wird geladen um später auf die Variable "SQLCODE" zugreifen zu können, und alle Variablen werden deklariert, die in SQL-Anweisungen benutzt werden. Dabei enthalten "X1", "Y1" den ersten und "Xi", "Yi" immer den i-ten Stützpunkt des Polygons (Zeilen 1-5). In den Zeilen 6-10 erfolgt die Deklaration der Relation "Polygone". Die Sprungmarke zur Fehlerbehandlung wird in Zeile 11 angegeben.

In den Zeilen Zeilen 13-16 erfolgt die Deklaration von Programmvariablen, die nicht in SQL-Anweisungen benutzt werden. "Ximinus1", "Yiminus1" dienen zur Aufnahme des (i-1)ten Stützpunktes. "Umfang" enthält zum Schluß den berechneten Umfang, und die boolsche Variable "weitereKoordinaten" gibt an, ob noch weitere Stützpunkte in der Ergebnismenge des Cursors vorhanden sind.

Die Deklaration des Cursors für die Stützpunkte findet in den Zeilen 17-21 statt. Nach der Wahl des Benutzers, für welches Polygon der Umfang berechnet werden soll, wird der Cursor geöffnet, und die Koordinaten des ersten Stützpunktes werden in die Variablen "X1", "Y1" gelesen.

Da die ersten Stützpunkte auch die Startkoordinaten zur Berechnung der fortlaufenden Summe sind, werden sie auch in die Variablen "Ximinus1" und "Yiminus1" gespeichert (Zeilen 22-28).

Es wird bei der Berechnung des Umfangs für das Polygon davon ausgegangen, dass mindestens drei Stützpunkte angegeben werden. In einer Schleife wird der nächste Stützpunkt gelesen (Zeilen 29-41).

Sollten alle Datensätze des Cursors gelesen worden sein, wird die Verarbeitungsschleife verlassen. Die Berechnung des Summanden wird mit der Anweisung in den Zeilen 35-37 vorgenommen. Danach erfolgt die Zwischenspeichern des bislang letzten Stützpunktes zur Berechnung des nächsten Summanden (Zeilen 38-39). In den Zeilen 42-43 wird der letzte Summand mit Hilfe des ersten und letzten Stützpunkt gebildet.

### 8.10.6 Beispiel zu Dynamic SQL



```
EXEC SQL INCLUDE SQLDA;  
EXEC SQL BEGIN DECLARE SECTION;  
    DCL Anfrage CHAR(200);  
    DCL Deskriptor SQLDA;  
EXEC SQL END DECLARE SECTION;  
PUT ('SELECT-Anweisung eingeben:');  
GET (Anfrage);  
EXEC SQL DECLARE SQLObjekt STATEMENT;  
EXEC SQL DECLARE Cursor FOR SQLObjekt;  
EXEC SQL PREPARE SQLObjekt FROM :Anfrage;  
EXEC SQL DESCRIBE SQLObjekt  
INTO :Deskriptor;  
FOR i=1 to Deskriptor.SQLN DO  
BEGIN  
    Deskriptor.SQLVAR[i].SQLDATA :=  
    ALLOCATE (Deskriptor.SQLVAR[i].SQLLEN);  
END;  
EXEC SQL OPEN Cursor;  
weitereTupel:= TRUE;  
WHILE weitereTupel= TRUE DO  
BEGIN  
    EXEC SQL FETCH Cursor  
    USING DESKRIPTOR :Deskriptor  
    IF SQLCODE = 100  
    THEN weitereTupel:= FALSE;  
    ELSE FOR i=1 TO Deskriptor.SQLN DO
```



```
BEGIN
PUT (Deskriptor.SQLVAR[i].-&gtSQLDATA);
END;
ENDIF;
END;
EXEC SQL CLOSE Cursor;
```

**Programm: Prinzip des Arbeitens mit dynamischen SELECT-Anweisungen, entnommen Ne96**

In den Zeilen 1-5 wird die "SQL Description Area"-Struktur mit "INCLUDE" dem Anwendungsprogramm bekannt gemacht und die Variablen "Anfrage" zur Aufnahme der Anfrage sowie "Deskriptor" zur Aufnahme der Schemainformationen deklariert.

Die Eingabe der eigentlichen SQL-Anfrage als Zeichenkette wird mit dem Befehl in Zeile 7 vorgenommen. Danach folgt in Zeile 8 die Deklaration einer SQL-Variablen zur Aufnahme des Objektcode der SELECT-Anweisung. Zur Abarbeitung der Ergebnismenge ist ein Cursor notwendig. Dieser wird nicht direkt an die SQL-Anweisung gebunden sondern an die zugeordnete SQL-Variable "SQLObjekt" (Zeile 9).

Die als Zeichenketten-Variable vorliegende Anfrage wird mit "PREPARE" übersetzt, und der SQL-Objektcode wird der angegebenen SQL-Variable zugewiesen. Schemainformationen werden in der folgenden "DESCRIBE"-Anweisung in die Variable "Deskriptor" geschrieben (Zeilen 10-12). Die Anzahl der Attribute der Ergebnisrelation ist in "Deskriptor.SQLN" gegeben. Damit kann in der Schleife Speicherplatz zur Aufnahme jeweils eines Attributwertes reserviert werden. Dazu wird die Funktion "ALLOCATE" verwendet. Diese reserviert den Speicherplatz in Bytes und gibt einen Zeiger auf den Bereich zurück (Zeilen 13-17).

Das Öffnen des zugeordneten Cursors wird mit dem Befehl in Zeile 18 erreicht. Die Tupel der Ergebnismenge werden mit einer modifizierten "FETCH"-Anweisung verarbeitet (Zeile 20). Anstelle der sonst anzugebenen Programm-Variablen werden implizit über den Deskriptor die Adressen der zuvor reservierten Datenbereiche benutzt.

Die Attributwerte werden in einzelne Tupel geschrieben (Zeilen 22-27). Die Verarbeitung der Tupel erfolgt durch deren Ausgabe (Zeile 28).

## 8.10.7 Client-Erweiterungen

### *Helper-Anwendungen*

- erste Generation von Client-Erweiterungen
- eigenständige Programme, die auf dem PC des Benutzers laufen
- nicht in einer Web-Seite integriert

- werden nicht in einem Browser-Fenster angezeigt
- Haupteinsatzgebiet: Anzeige von Grafikdateien in Dateiformaten, die vom Browser nicht unterstützt werden

#### *Navigator PlugIns*

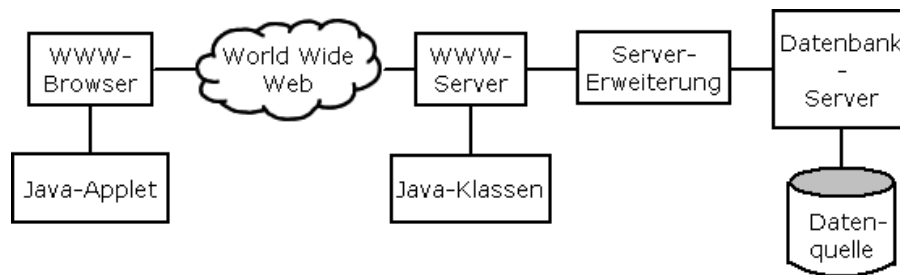
- sind mit Helper-Anwendungen in der Hinsicht vergleichbar, dass sie die Verarbeitung und Anzeige von Informationen unterstützen, die ein Browser allein nicht verarbeiten kann
- arbeiten nur im Browser
- sind sehr eng in den Browser integriert; dadurch können schon Teile einer Datei angezeigt werden, bevor alles geladen ist
- Einsatzzwecke:
  - Sound
  - Chats mit ähnlich ausgestatteten Benutzern
  - Animationen
  - Video
  - Interaktionen in virtuellen 3D-Welten
  - Erleichterung des Zugriffs auf entfernte Datenbanken

#### *ActiveX-Controls*

- bieten ähnliche Funktionalitäten wie PlugIns
- basieren auf Microsofts früherer OLE-Technologie
- Internet Explorer ist ActiveX kompatibel

#### *Java-Applets*

- Java ist eine C++ ähnliche Sprache, die von Sun speziell für das Schreiben von Web-Client-Erweiterungen entwickelt wurde.
- werden bei einer Server/Client Web-Verbindung auf den Client heruntergeladen und ausgeführt
- sind in einer HTML-Seite eingebettet
- stellen dem Client die datenbankspezifischen Funktionalitäten zur Verfügung, um auf die Daten des Servers zugreifen zu können
- sind meistens auf dem neuesten Stand
- Die Abbildung *WWW-Datenbankanwendung mit einem Java-Applet* zeigt schematisch, wie eine Web-Datenbankanwendung mittels eines Java-Applets auf einer Client-Maschine arbeitet

**WWW-Datenbankanwendung mit einem Java-Applet**

### *Skripts*

- flexibles Werkzeug zum Erstellen von Client-Erweiterungen
- Skript-Sprachen sind z.B. JavaScript oder Microsofts JScript und VBScript, was unter ActiveScript zusammengefasst wird
- dienen zur Formulkontrolle auf Client-Seite und können dadurch viel Zeit bei der Arbeit mit einer Datenbank sparen
- werden wie Java-Applets in eine HTML-Seite eingebettet und ausgeführt, wenn diese Seite angezeigt wird

## 8.10.8 JDBC-Beispielprogramm

Dieses Programm liest aus den Tupeln der Relation Studierende, die in einer Oracle-Datenbank stehen, die Attribute Vorname und Name aus und zeigt sie an.



```
import java.sql.*;

class dbtest {

    public static void main (String args[])
        throws SQL Exception {

        // Oracle-Treiber laden

        DriverManager.registerDriver(
            new oracle.jdbc.driver.OracleDriver());
```

```
// Verbindung zur Datenbank herstellen

String url="jdbc:oracle:oci8@vfhlsl:1521:vfhl2";
Connection
    conn=DriverManager.getConnection(url,"dummy","passwd");

// Anfrage an die Datenbank stellen

Statement stmt=conn.createStatement();
ResultSet rset=stmt.executeQuery("Select Name,Vorname FROM
    Studierende");

// Verarbeiten der Ergebnismenge

while(rset.next()) {
    String s=rset.getString(2); // Attribut Name
    String t=rset.getString("Vorname");
    System.out.println(t+" "+s+"\n");}}
DB-Anbindung mit JDBC
```

## 8.11 Freitextaufgaben zu Anwendungen mit Datenbanken

1. In wieviele Funktionsgruppen lassen sich die externen Prozeduren grob einteilen?

### Lösung anzeigen

Sechs Funktionsgruppen:

1. Kommunikationsprozeduren
2. BindungsprozedurenSQL-Anweisungsprozeduren
3. Fehlerbehandlungsprozeduren
4. Transaktionsprozeduren
5. Prozeduren zur Ergebnisverarbeitung

2. Wie werden Datenbankfunktionen aus einem Anwendungsprogramm heraus aufgerufen?

**Lösung anzeigen**

Die Datenbankfunktionen werden aus dem Anwendungsprogramm als externe Prozeduren aufgerufen.

3. Wozu dient die Prozedur DBSETLPWD?

**Lösung anzeigen**

Die Prozedur DBSETLPWD wird benutzt, um ein Passwort in eine leere Login-Struktur einzutragen.

4. Welche Aufgaben übernimmt ein Vorübersetzer beim Übersetzen eines Embedded SQL C-Programmes?

**Lösung anzeigen**

Ein Vorübersetzer erzeugt aus den markierten SQL-Anweisungen die entsprechenden Aufrufe der Call-Schnittstelle und schreibt diese Aufrufe in das Anwendungsprogramm. Die ursprünglich eingebetteten SQL-Anweisungen werden in Kommentare umgewandelt.

5. Welche drei Komponenten werden zum Übersetzen eines Embedded SQL C-Programmes benötigt?

**Lösung anzeigen**

1. Vorübersetzer
2. C-Compiler
3. Loader/Binder

6. In welche vier Schritte unterteilt sich die Benutzung des Cursors in Embedded SQL?

**Lösung anzeigen**

1. Deklaration des Cursors, um die zugehörige Anfrage festzulegen
2. Öffnen des Cursors
3. Daten Schritt für Schritt zum Anwendungsprogramm übertragen
4. Schließen des Cursors

7. Wozu dient die ODBC-Schnittstelle?

**Lösung anzeigen**

Die ODBC-Schnittstelle ist eine Standardschnittstelle zwischen Datenbank und einem Programm, die versucht, auf die Daten in einer Datenbank zuzugreifen.

8. Welche Mittel stellt die Funktionsbibliothek der ODBC-Schnittstelle zur Verfügung? Nennen Sie die Komponenten einer ODBC-Schnittstelle.

**Lösung anzeigen**

Mittel der Funktionsbibliothek der ODBC-Schnittstelle:

1. Standard-SQL-Syntax
2. Standard-SQL-Datentypen
3. Standardprotokoll für die Verbindung zu einer Datenbank-Engine
4. Standardfehlercodes

9. Wie erfolgt der Zugriff auf Datenbanken mit JDBC?

**Lösung anzeigen**

Der Zugriff mit JDBC erfolgt durch entsprechende von den Datenbankherstellern mitgelieferte Treiber.

10. In wieviele Bereiche lässt sich die Funktionalität von JDBC unterteilen?

**Lösung anzeigen**

Die JDBC-Treiber lassen sich in vier Funktionalitätsklassen einteilen.

11. In wieviele Klassen von SQL-Anweisungen lassen sich Statement-Objekte einteilen?

**Lösung anzeigen**

Statement-Objekte lassen sich in drei Klassen von SQL-Anweisungen einteilen.

12. Welche Aufgaben erfüllt ein Report-Generator?

**Lösung anzeigen**

Er ermöglicht das Publizieren von Datenbankinhalten im Intra- und Internet.

13. Nennen Sie die Oracle Report-Funktionen.

**Lösung anzeigen**

1. Report Builder

2. Graphics Builder
3. Translation Builder
4. Report Server

14. Was sind die Vor- und Nachteile von 4GL-SQL?

#### **Lösung anzeigen**

Vorteile:

- hoher Komfort, ist produktivitätssteigernd,
- eignet sich für Rapid Prototyping

Nachteile: ist nicht standardisiert, verursacht hohe Kosten, hat eine relativ geringe Flexibilität und hohe Komplexität

## 9 Transaktionsverwaltung und Wiederherstellung



### Zeitumfang

Die voraussichtliche Bearbeitungsdauer dieser Lerneinheit (ohne Übungsaufgaben!) beträgt ca. 8 Stunden.



### Lernziele

In vollständigen Datenbanksystemen sind Komponenten wie Transaktions- und Recovery-Manager feste Bestandteile. Sie tragen dazu bei, die Datenkonsistenz zu gewährleisten und die Datensicherheit zu erhöhen. Es geht darum, wie konsistente Datenbankzustände sichergestellt und wie konkurrierende Datenbankzugriffe synchronisiert werden können.

Diese Lerneinheit behandelt die Aspekte der Transaktionsverwaltung, Sperralgorithmen und Wiederherstellungsalgorithmen.



### Gliederung

#### 9 Transaktionsverwaltung und Wiederherstellung

##### 9.1 Grundbegriffe der Transaktionsverwaltung

##### 9.2 Sperrtechniken

##### 9.3 Transaktionsunterstützung in SQL

##### 9.4 Wiederherstellung

##### 9.5 Wiederherstellungsverfahren

##### 9.6 ARIES-Algorithmus

##### 9.7 Freitextaufgaben zu Transaktionsverwaltung und Wiederherstellung

### 9.1 Grundbegriffe der Transaktionsverwaltung

Eine Transaktion ist eine logische Verarbeitungseinheit mit Konsistenzbewahrung, d.h. sie muss einen konsistenten Datenbankzustand in einen ebenfalls konsistenten Datenbankzustand überführen.



Mögliche Ursachen für Inkonsistenzen sind der Abbruch nach Systemfehlern oder die gegenseitige Beeinflussung von Transaktionen im Mehrbenutzerbetrieb. Ein Beispiel hierfür ist Zimmerreservierung für Hotels. Benötigt wird eine Ablaufintegrität der Transaktionen durch eine Mehrbenutzersynchronisation (engl. concurrency control).

In diesem Zusammenhang sind einige Begriffe hier aufgeführt.

#### ACID-Prinzip:

- **Atomicity** (Ununterbrechbarkeit)

Alle Datenbankoperationen, die zu einer Transaktion gehören, werden ganz oder gar nicht ausgeführt ("Alles oder Nichts-Prinzip").

- **Consistency** (Integritätserteilung)

Die Datenbankkonsistenz muss gewährleistet sein.

- **Isolation** (Isolierter Ablauf)

Transaktionen laufen isoliert ab. Parallel ablaufende Transaktionen können sich nicht gegenseitig beeinflussen. Zu erreichen ist dies durch Serialisierbarkeit.

- **Durability** (Dauerhaftigkeit oder Persistenz der Daten)

Alle Operationen haben eine dauerhafte Wirkung.



In der Praxis ist das ACID-Prinzip schwer zu erreichen.

Ein **Datenelement** (engl. **item**) ist ein Teil einer Datenbank, z.B. Relation, Tupel, Attributwerte im Relationenmodell oder Seiten auf Speicherebene.

Die **Granularität der Datenelemente** ist der Grad der Verarbeitungsebene. Hohe Granularität der Datenelemente bedeutet einen geringen Verwaltungsaufwand der Transaktionen und feine Granularität bringt den Vorteil des hohen Parallelitätsgrades. Typische Datenelemente für die Transaktionsverarbeitung sind Tupel oder Relationen.

**read A:** liest ein Datenelement der Datenbank namens A in eine Programmvariable (einfachheitshalber heißt die Programmvariable auch A). Folgende Schritte sind dazu notwendig:

1. Suche die Adresse des Plattenblocks, der das Datenelement A enthält
2. Kopiere diesen Block in einen Puffer im Arbeitsspeicher
3. Kopiere das Datenelement A vom Puffer in die Programmvariable namens A

**write A:** schreibt den Wert der Programmvariable A in das Datenelement namens A in der Datenbank. Folgende Schritte sind dazu notwendig:

1. Suche die Adresse des Plattenblocks, der das Datenelement A enthält
2. Kopiere diesen Block in einen Puffer im Arbeitsspeicher
3. Kopiere das Datenelement A von der Programmvariable A an die richtige Stelle im Puffer
4. Schreibe den aktualisierten Block aus dem Puffer auf die Platte zurück.

Bei der gleichzeitigen Ausführung von Transaktionen können mehrere Probleme auftreten, die nachfolgend aufgelistet und beschrieben werden:

#### 1. Das Lost-Update-Problem

Gegeben seien zwei Transaktionen:

T1: read A; A:=A-1; write A; read B; B:=B+1; write B;

T2: read A; A:=A+2; write A;

Die beiden Transaktionen greifen auf dasselbe Datenelement zu.

Falls die Transaktionen folgendermaßen ablaufen (die Zeitachse verläuft von oben nach unten):

T <sub>1</sub>	T <sub>2</sub>
read A	
A:=A-1	
	read A
	A:=A+2
write A	
read B	
	write A
B:=B+1	
write B	

erhält A einen falschen Wert, weil die Aktualisierung durch T1 überschrieben wurde.

#### 2. Das Dirty-Read-Problem (Problem des unsauberen Lesens)

Dieses Problem kommt dann vor, wenn eine Transaktion ein Datenelement aktualisiert und dann aus irgendeinem Grund fehlschlägt.

T <sub>1</sub>	T <sub>2</sub>
read A	
A:=A-1	
write A	
	read A
	A:=A+2
	write A
read B	
.	
.	
.	

T<sub>1</sub> schlägt fehl. Der Wert von A muss auf den alten Wert zurückgesetzt werden. T<sub>2</sub> hat bereits den falschen Wert gelesen.

#### 3. Das Problem der falschen Summenbildung



Beispiel

Eine Transaktion T<sub>3</sub> berechnet die Gesamtzahl von Zimmerreservierungen.

T <sub>1</sub>	T <sub>3</sub>
	sum:=0
	read X
	sum:=sum+X
	read A
	A:=A+2
	write A
read B	
.	
.	
.	
read A	

A:=A-1	
write A	
	read A
	sum:=sum+A
	read B
	sum:=sum+B
read B	
B:=B+1	
write B	

$T_3$  liest A, nachdem 1 abgezogen wurde, und B, bevor 1 addiert wird, was zu falschen Ergebnissen führt.

#### 4. Das Problem des nicht wiederholbaren Lesens

Dieses Problem entsteht, wenn eine Transaktion T ein Datenelement zweimal liest, während der Wert des Datenelements in der Zwischenzeit von einer anderen Transaktion geändert wird.

Die Zugriffskontrolle erfolgt durch die Vergabe von Sperren. Die Anweisungen, die dazu verwendet werden, sind **lock** zum Sperren und **unlock** zum Freigeben/Entsperren.



Beispiel

$T_1$ : lock A; read A; A:=A-1; write A; unlock A;

Bei der Vergabe von Sperren können **Verklemmungen** (engl. deadlocks) entstehen.



Beispiel

$T_1$ : lock A; ... ; lock B; ... unlock A; unlock B;

$T_2$ : lock B; ... ; lock A; ... unlock B; unlock A;

Die Aufgabe des Datenbankmanagementsystems ist, solche Verklemmungen zu erkennen und durch Abbruch einer beteiligten Transaktion aufzulösen.

## 9.2 Sperrtechniken

### Sperrtechniken zur Nebenläufigkeitskontrolle

#### 1. *Binäres Sperren*

Diese Technik benutzt die Operationen lock und unlock, ähnlich wie die kritischen Abschnitte bei Betriebssystemen.

Eine Sperrtabelle hält die Information darüber, welche Datenelemente momentan gesperrt sind. Dafür gibt es folgende Regeln:

1. lock-Operationen werden vor read- oder write-Operationen ausgeführt,
2. unlock-Operationen werden erst ausgeführt, nachdem alle read- und write-Operationen abgeschlossen sind,
3. keine lock-Operation ausführen, wenn eine Transaktion bereits über eine Sperre auf das Datenelement verfügt und
4. nur dann unlock-Operation ausführen, wenn eine Transaktion bereits über eine Sperre auf das Datenelement verfügt.

Nachteil: binäres Sperren ist restriktiv, da höchstens eine Transaktion eine Sperre halten kann.

#### 2. *Gemeinsames/Exklusives Sperren*

Das Ziel ist, mehreren Transaktionen Lesezugriff und einer Transaktion Schreibzugriff zu gewähren. Dafür werden drei Sperroperationen benötigt: read\_lock, write\_lock und unlock.

Ein lesegesperrtes Datenelement gilt als gemeinsames Datenelement.

Ein schreibgesperrtes Datenelement wird exklusiv von einer Transaktion gesperrt.

Die Implementierung erfolgt durch Mitzählen der Anzahl Transaktionen, die eine Lese- oder Schreibsperre auf ein Datenelement in der Sperrtabelle halten. Dabei darf es nur eine einzige Transaktion mit Schreibsperre geben.

Beim gemeinsamen/exklusiven Sperren müssen folgende Regeln beachtet werden:

1. read\_lock oder write\_lock-Operationen müssen vor der read-Operation ausgeführt werden,
2. write\_lock-Operationen müssen vor write-Operationen ausgeführt werden,
3. unlock-Operationen dürfen erst ausgeführt werden, wenn read- und write-Operationen abgeschlossen sind,
4. es darf keine read\_lock-Operation ausgeführt werden, wenn eine Transaktion bereits eine Schreibsperre auf Datenelement X hält,

5. es darf keine write\_lock-Operation ausgeführt werden, wenn eine Transaktion bereits eine Lese- oder Schreibsperre auf Datenelement X hält,
6. es darf nur eine unlock-Operation ausgeführt werden, wenn eine Transaktion bereits eine Lese- oder Schreibsperre auf dem Datenelement X hält.

### Sperränderung

Eine Transaktion, die bereits eine Sperre auf dem Datenelement X hält, darf unter bestimmten Bedingungen die Sperre vom gesperrten in einen anderen Zustand ändern, z.B.:

1. Nach der Ausführung der read\_lock-Operation wird die write\_lock-Operation ausgeführt (Verschärfung der Sperre)
2. Nach der Ausführung der write\_lock-Operation wird die read\_lock-Operation ausgeführt (Abschwächung der Sperre)

### Serialisierbarkeit von Ausführungsplänen

Ein **Ausführungsplan** (AP; engl.: schedule) für eine Menge von Transaktionen ist eine geordnete Folge ihrer Aktionen (lock, read, write, unlock, ...), wobei die Aktionen in der gleichen Reihenfolge wie in der Transaktion selbst aufgeführt sind. Mehrere Transaktionen können verschränkt ausgeführt werden.

Ein Ausführungsplan ist **seriell**, wenn die Aktionen je einer Transaktion direkt aufeinander folgen. Für gegebene n Transaktionen gibt es n! mögliche Ausführungspläne.

Ein Ausführungsplan heißt **serialisierbar**, wenn sein Ergebnis äquivalent zu dem eines seriellen Ausführungsplans ist.



Beispiel

T<sub>1</sub>: read A; A:=A-1; write A; read B; B:=B+1; write B;

T<sub>2</sub>: read B; B:=B-2; write B; read C; C:=C+2; write C;

AP <sub>1</sub>		AP <sub>2</sub>		AP <sub>3</sub>	
T <sub>1</sub>	T <sub>2</sub>	T <sub>1</sub>	T <sub>2</sub>	T <sub>1</sub>	T <sub>2</sub>
read A		read A		read A	
A:=A-1			read B	A:=A-1	
write A		A:=A-1			read B
read B			B:=B-2	write A	

B:=B+1		write A			B:=B-2
write B			write B	read B	
	read B	read B			write B
	B:=B-2		read C	B:=B+1	
	write B	B:=B+1			read C
	read C		C:=C+2	write B	
	C:=C+2	write B			C:=C+2
	write C		write C		write C

$AP_1$  ist seriell, da die Aktionen direkt aufeinander folgen.  $AP_2$  ist nicht seriell, aber serialisierbar, da sein Ergebnis äquivalent zu  $AP_1$  ist.  $AP_3$  ist nicht serialisierbar, da der Wert von B von  $T_1$  überschrieben wird.

Wir haben also  $n!$  serielle Ausführungspläne für  $n$  Transaktionen, wobei es viel mehr mögliche nicht-serielle Ausführungspläne gibt.

Von den nicht-seriellen Ausführungsplänen können zwei disjunkte Gruppen gebildet werden:

1. diejenigen Ausführungspläne, die mit einem (oder mehreren) seriellen Ausführungsplan (Ausführungsplänen) äquivalent und somit serialisierbar sind, und
2. diejenigen Ausführungspläne, die mit keinem seriellen Ausführungsplan äquivalent und somit nicht serialisierbar sind.

Zwei Ausführungspläne sind **resultatsäquivalent**, wenn sie den gleichen endgültigen Zustand der Datenbank produzieren.



Beispiel

**Gegenbeispiel:**  $AP_1$ : read A; A:=A+10; write A;

$AP_2$ : read A; A:=A\*1.1; write A;

Mit einem Anfangswert  $A=100$  bekommt man den gleichen Datenbankzustand, mit anderen Anfangswerten jedoch nicht. Deshalb sind die beiden Ausführungspläne nicht resultatsäquivalent.

Zwei Ausführungspläne gelten als **konfliktäquivalent**, wenn die Reihenfolge von zwei in Konflikt stehenden Operationen in beiden Ausführungsplänen gleich sind.

Zwei Operationen in einem Ausführungsplan stehen **in Konflikt**, wenn sie zu unterschiedlichen Transaktionen gehören, auf das gleiche Datenbankelement zugreifen und mindestens eine der beiden Operationen eine write-Operation ist, z.B.:

in  $AP_i$ :  $r_1(X)$ ,  $w_2(X)$

in  $AP_j$ :  $w_2(X)$ ,  $r_1(X)$

Der von  $r_1(X)$  gelesene Wert in den beiden Ausführungsplänen kann unterschiedlich sein.

Ein Ausführungsplan ist **konfliktserialisierbar**, wenn er mit einem serialisierbaren Ausführungsplan konfliktäquivalent ist.

### Test eines Ausführungsplans auf Konfliktserialisierbarkeit

Gegeben sei ein gerichteter Präzedenzgraph  $G=(N,E)$  mit der Knotenmenge  $N=\{T_1, T_2, \dots, T_n\}$  und der Kantenmenge  $E=\{e_1, e_2, \dots, e_m\}$ .  $e_i$  hat die Form  $(T_j \rightarrow T_k)$ , mit  $1 \leq j \leq n$ ,  $1 \leq k \leq n$ .

$e_i$  wird erzeugt, wenn im Ausführungsplan eine der Operationen von  $T_j$  vor einer mit ihr in Konflikt stehenden Operation in  $T_k$  vorkommt.

Algorithmus:

1. Erzeuge für jede Transaktion  $T_i$ , die im Ausführungsplan auftritt, einen Knoten  $T_i$  im Präzedenzgraphen
2. Erzeuge für jeden Fall im Ausführungsplan, bei dem  $T_j$  eine read-Operation ausführt, nachdem  $T_i$  eine write-Operation ausgeführt hat, eine Kante  $(T_i \rightarrow T_j)$  im Präzedenzgraphen
3. Erzeuge für jeden Fall im Ausführungsplan, bei dem  $T_j$  eine write-Operation ausführt, nachdem  $T_i$  eine read-Operation ausgeführt hat, eine Kante  $(T_i \rightarrow T_j)$  im Präzedenzgraphen
4. Erzeuge für jeden Fall im Ausführungsplan, bei dem  $T_j$  eine write-Operation ausführt, nachdem  $T_i$  eine write-Operation ausgeführt hat, eine Kante  $(T_i \rightarrow T_j)$  im Präzedenzgraphen
5. Der Ausführungsplan ist serialisierbar, falls der Präzedenzgraph keinen Zyklus aufweist. ("Zyklus" bedeutet: es gibt eine Kantenfolge  $K=((T_j \rightarrow T_k), (T_k \rightarrow T_p), \dots, (T_p \rightarrow T_j))$ )



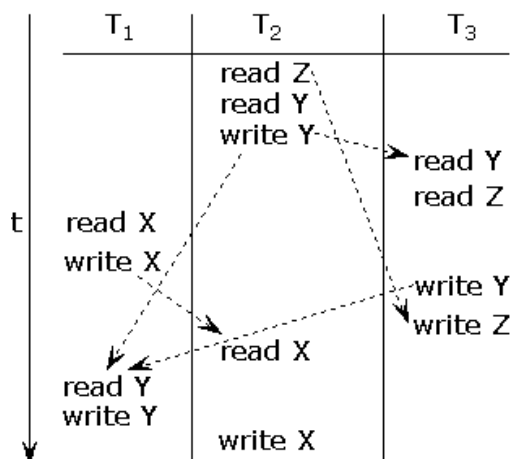


Beispiel

für Tests von Ausführungsplänen auf Konfliktserialisierbarkeit:

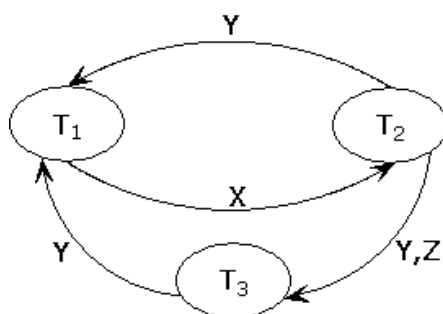
T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	
read X	read Z	read Y	
write X	read Y	read Z	
read Y	write Y	write Y	
write Y	read X	write Z	
	write X		

AP<sub>1</sub> (die gestrichelten Pfeile bedeuten, dass dort jeweils eine Kante im Präzedenzgraphen erzeugt wird):



Test eines Ausführungsplans auf  
Konfliktserialisierbarkeit

Eine Kante ( $T_1 \rightarrow T_2$ ) wird für das Datenelement X erzeugt, da read X in T<sub>2</sub> ausgeführt wird, nachdem write X in T<sub>1</sub> ausgeführt wurde. Ebenso werden die Kanten ( $T_2 \rightarrow T_1$ ), ( $T_2 \rightarrow T_3$ ) und ( $T_3 \rightarrow T_1$ ) zwischen den Knoten generiert.

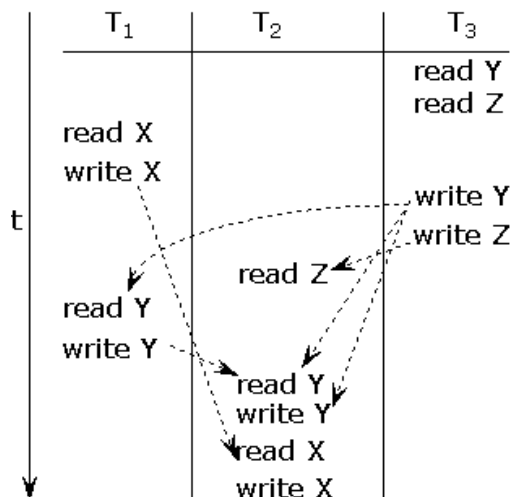




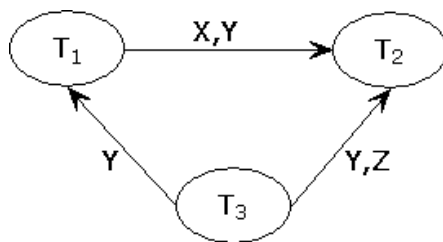
### Test eines Ausführungsplans auf Konfliktserialisierbarkeit

AP<sub>1</sub> ist nicht serialisierbar, da im Präzedenzgraphen zwei Zyklen  $((T_1 \rightarrow T_2), (T_2 \rightarrow T_1))$  bzw.  $((T_1 \rightarrow T_2), (T_2 \rightarrow T_3), (T_3 \rightarrow T_1))$  existieren.

AP<sub>2</sub> (die gestrichelten Pfeile bedeuten, dass dort jeweils eine Kante im Präzedenzgraphen erzeugt wird):



### Test eines Ausführungsplans auf Konfliktserialisierbarkeit




### Test eines Ausführungsplans auf Konfliktserialisierbarkeit

AP<sub>2</sub> ist serialisierbar, da es einen äquivalenten seriellen Ausführungsplan  $T_3 \rightarrow T_1 \rightarrow T_2$  im Präzedenzgraphen gibt.

Die meisten Systeme verfolgen den Ansatz, die Serialisierbarkeit sicherzustellen, ohne die Ausführungspläne testen zu müssen. Protokolle für die Nebenläufigkeitskontrolle, die die Serialisierbarkeit garantieren, sind:

- Zwei-Phasen-Sperrprotokoll (kommt in den meisten kommerziellen Datenbankmanagementsystemen zum Einsatz)
- Zeitstempel-Verfahren

- Multiversionsprotokolle
- Zertifizierungs- und Validierungsprotokolle

Die ersten beiden Protokolle werden zu einem späteren Zeitpunkt vorgestellt und erläutert, die letzten beiden können in [ElNa99](#)  nachgelesen werden.

### Weitere Äquivalenzart für Ausführungspläne

Es gibt Ausführungspläne, die korrekt sind, indem sie weniger strenge Bedingungen als die Konflikt- und View-Serialisierbarkeit erfüllen.



#### Beispiel

#### Debit-/Kredit-Transaktionen

Man hat zwei Transaktionen zur Überweisung eines Betrages zwischen zwei Banken (Addition und Subtraktion sind kommutativ):

$T_1$ :  $\text{read}_1 X$ ;  $X := X - 100$ ;  $\text{write}_1 X$ ;  $\text{read}_1 Y$ ;  $Y = Y + 100$ ;  $\text{write}_1 Y$ ;

$T_2$ :  $\text{read}_2 Y$ ;  $Y = Y - 200$ ;  $\text{write}_2 Y$ ;  $\text{read}_2 X$ ;  $X = X + 200$ ;  $\text{write}_2 X$ ;

Betrachten wir den APk für die beiden Transaktionen:

$AP_k$ :  $\text{read}_1 X$ ;  $\text{write}_1 X$ ;  $\text{read}_2 Y$ ;  $\text{write}_2 Y$ ;  $\text{read}_1 Y$ ;  $\text{write}_1 Y$ ;  $\text{read}_2 X$ ;  $\text{write}_2 X$ ;

$AP_k$  ist nicht serialisierbar, aber korrekt.

### View-Äquivalenz und -Serialisierbarkeit

Zwei Ausführungspläne AP und AP' gelten als view-äquivalent, wenn die folgenden drei Bedingungen erfüllt sind:

1. Die gleiche Transaktionsmenge nimmt in AP und AP' teil und die beiden Ausführungspläne beinhalten die gleichen Operationen dieser Transaktionen.
2. Wenn der gelesene Wert von A für jede Operation  $r_i(A)$  in AP von einer Operation  $w_j(A)$  von  $T_j$  geschrieben wurde, muss die gleiche Bedingung für den von Operation  $r_i(A)$  von  $T_i$  in AP' gelesenen Wert gelten.
3. Wenn die Operation  $w_k(B)$  von  $T_k$  die letzte Operation ist, die das Datenelement B in AP schreibt, dann muss  $w_k(B)$  von  $T_k$  ebenfalls die letzte Operation sein, die das Datenelement B in AP' schreibt.

Die Leseoperationen haben die gleiche Sicht in beiden Ausführungsplänen.

Ein Ausführungsplan ist view-serialisierbar, wenn er mit einem seriellen Ausführungsplan view-äquivalent ist.

Das Problem mit dem Test auf View-Serialisierbarkeit hat sich als NP-vollständig erwiesen. Es ist also sehr unwahrscheinlich, für dieses Problem einen effizienten polynomialen Algorithmus zu finden.



Gliederung

## 9.2 Sperrtechniken

### 9.2.1 Das 2PL-Protokoll

## 9.2.1 Das 2PL-Protokoll

### Verklemmungserkennung

Um Verklemmungen erkennen zu können, gibt es einen Wartegraphen, in dem für jede Transaktion ein Knoten erzeugt wird. Wenn  $T_i$  wartet, um ein Datenelement  $X$  zu sperren, das momentan von  $T_j$  gesperrt ist, wird eine gerichtete Kante ( $T_i \rightarrow T_j$ ) im Graphen eingefügt.

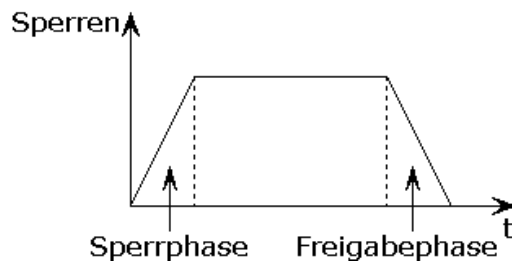
Ein Verklemmungszustand entsteht nur dann, wenn der Wartegraph einen Zyklus enthält.

Falls das System sich in einem Verklemmungszustand befindet, müssen einige Transaktionen abgebrochen werden. Die Auswahl wird bei den Transaktionen getroffen, die noch nicht viele Änderungen durchgeführt haben (Opferauswahl).

### Das Zwei-Phasen-Sperrprotokoll

Das Zwei-Phasen Sperrprotokoll (engl.: 2PL-protocol: two-phase-locking-protocol) garantiert die Serialisierbarkeit von Ausführungsplänen. Sperren der Datenelemente vor jedem Zugriff (1. Phase, Wachstumsphase) kommen vor allen Freigaben (2. Phase, Schrumpfungsphase).

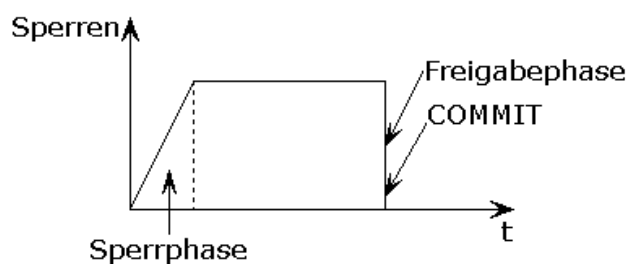
Falls sich alle Transaktionen an das Zwei-Phasen-Sperren halten, können keine nicht serialisierbaren Ausführungspläne auftreten. Also ist jeder Ausführungsplan, der ausschließlich aus Zwei-Phasen-Transaktionen besteht, serialisierbar.



Zwei-Phasen-Transaktion

Das Problem der Verklemmung ist jedoch noch nicht gelöst, da zwei Transaktionen gegenseitig auf die Freigabe einer Sperre warten.

Eine weitere Verschärfung des Zwei-Phasen-Sperrprotokolls ist das **strikte Zwei-Phasen-Sperrprotokoll**. Die Idee dabei ist, dass alle Sperren zusammen mit der COMMIT-Anweisung freigegeben werden. Dies hat als Ziel die Vermeidung kaskadierender Abbrüche.



Das strikte Zwei-Phasen-Sperrprotokoll

### Zeitstempelverfahren

Jede Transaktion erhält einen eindeutigen Zeitstempel (engl. timestamp)  $TS(T)$ . Wenn  $T_1$  vor  $T_2$  beginnt, dann gilt  $TS(T_1) < TS(T_2)$ , d.h.  $T_1$  ist die ältere Transaktion.

Das Ziel des Verfahrens ist, Verklemmungen zu vermeiden.

Es gibt zwei Schemata:

- Wait/Die  
Wenn  $TS(T_i) < TS(T_j)$ , dann darf  $T_i$ , also die ältere Transaktion, warten; anderenfalls wird  $T_i$  abgebrochen und kann später mit dem gleichen Zeitstempel wieder starten. Je älter die Transaktion wird, desto größer ist die Wahrscheinlichkeit fürs Warten.
- Wound/Wait  
Wenn  $TS(T_i) < TS(T_j)$ , dann wird  $T_j$  abgebrochen und kann später mit dem gleichen Zeitstempel wieder starten; anderenfalls darf  $T_i$  warten. Die jüngere Transaktion wartet solange, bis die ältere Transaktion fertig ist.

Ein weiteres Verfahren zur Verklemmungserkennung bieten Timeouts. Eine Transaktion wird abgebrochen, wenn sie über eine vom System definierte Zeitschranke bereits gewartet hat.

### 9.3 Transaktionsunterstützung in SQL

In SQL gibt es keine expliziten Anweisungen BEGIN TRANSACTION und END TRANSACTION. Stattdessen gibt es in SQL99 die Anweisungen ROLLBACK (Transaktion rückgängig machen), COMMIT (Transaktion abschließen) und SAVEPOINT (vorzeitiges Schreiben zu einem bestimmten Zeitpunkt einer Transaktion).

Jeder Transaktion werden bestimmte Eigenschaften zugeschrieben, die durch eine Anweisung SET TRANSACTION spezifiziert werden.

Eigenschaften sind:

1. Zugriffsart: READ ONLY oder READ WRITE, wobei letztere standardmäßig voreingestellt ist. Bei der Isolationsstufe READ UNCOMMITTED ist die Zugriffsart READ ONLY.
2. Diagnosebereichsgröße: DIAGNOSTIC SIZE n, wobei n die Anzahl der Bedingungen ist, die gleichzeitig im Diagnosebereich gehalten werden können.
3. Isolationsstufe: ISOLATION LEVEL <isolation>  
<isolation>:: READ UNCOMMITTED|READ COMMITTED|REPEATABLE READ|SERIALIZABLE

Voreinstellung ist SERIALIZABLE, d.h. keine Verletzung. Einige Systeme benutzen jedoch READ COMMITTED als Voreinstellung.



Beispiel

Beispiel für eine Transaktion in SQL:



```
EXEC SQL WHENEVER SQLERROR GOTO UNDO
EXEC SQL SET TRANSACTION
    READ WRITE
    DIAGNOSTICS SIZE 5
    ISOLATION LEVEL SERIALIZABLE;

EXEC SQL
    INSERT INTO Person (VNAME, NNAME, PERSID)
```

```
VALUES ( 'Max', 'Schmidt', '20034711' );  
EXEC SQL UPDATE Person  
SET NNAME='Meier' WHERE PERSID='20034711';  
  
EXEC SQL COMMIT;  
GOTO THE_END;  
UNDO: EXEC SQL ROLLBACK;  
THE_END: ... ;
```

Folgende drei Arten von Verletzungen können auftreten:

1. "dirty read"
2. "non repeatable read"
3. "phantom read"

$T_1$  liest Tupel aus einer Relation, die einer in der WHERE-Klausel spezifizierten Bedingung erfüllen

$T_2$  fügt eine Zeile ein, die ebenfalls dieser Bedingung genügt.

Wird  $T_1$  wiederholt, bekommt  $T_1$  ein Phantom, d.h. ein Tupel, das vorher nicht existierte.

Mögliche Verletzungen:

Isolationsstufe	Art der Verletzung		
	unsauberes Lesen	unwiederholbares Lesen	Phantom-Lesen
READ UNCOMMITTED	ja	ja	ja
READ COMMITTED	nein	ja	ja
REPEATABLE READ	nein	nein	ja
SERIALIZABLE	nein	nein	nein

SERIALIZABLE entspricht der Standardeinstellung und verhindert alle Verletzungen. Datenbankadministratoren können die Isolationsstufe nach Bedarf einstellen.

## 9.4 Wiederherstellung

### Gründe für das Misslingen einer Transaktion

1. Rechnerfehler (Systemabsturz): Hardware-, Software-, Netzfehler.
2. Transaktions- oder Systemfehler, z.B. Division durch Null, fehlerhafte Parameterwerte, logischer Programmfehler, Transaktionsabbruch durch den Benutzer.
3. Lokale Fehler oder von der Transaktion erkannte Ausnahmebedingungen, z.B. Daten für eine Transaktion wurden nicht gefunden.
4. Nebenläufigkeitskontrolle, z.B. Verklemmungssituation oder weil die Serialisierbarkeit verletzt wurde.
5. Plattenfehler, z.B. Lese-/Schreibkopf defekt.
6. Physikalische Probleme und Katastrophen, z.B. Ausfall der Netzversorgung oder Lüftung, Brand, Blitzschlag, falsche Bedienung

Die ersten vier Fehlerquellen kommen häufig vor, eine Wiederherstellung ist notwendig, während die beiden letzten Fehlerquellen nicht so oft vorkommen.

#### Das Systemlog

Das Systemlog ist die Protokolldatei und wird manchmal auch als DBMS-Journal bezeichnet. Darin werden alle Datenbank-Operationen aufgezeichnet, um später nach einem Systemabsturz den konsistenten Datenbankzustand wiederherstellen zu können.

Das Systemlog:

- wird auf Platte gespeichert
- wird periodisch in einen Archivspeicher (Band) kopiert (Sicherungskopie)
- enthält folgende Arten von Einträgen:
  - 3.1. [start\_transaction, T]
  - 3.2. [write\_item, T, X, alter\_wert, neuer\_wert], wobei X das Datenbankelement ist
  - 3.3. [read\_item, T, X]
  - 3.4. [commit, T]
  - 3.5. [abort, T]

Um mehr über Wiederherstellung zu erfahren, sollen zunächst einige Begriffe definiert werden.

**UNDO** (rückgängig machen): die Log-Einträge werden rückwärts durchsucht und alle von einer write-Operation geänderten Exemplare werden wieder auf alter\_wert zurückgesetzt.

**REDO** (wiederholte Ausführung): die Operationen einer Transaktion können wiederholt ausgeführt werden, indem die Log-Einträge vorwärts durchsucht und alle geänderten Exemplare auf neuer\_wert gesetzt werden.



Folgende Wiederherstellungsverfahren sind zu finden:

	REDO	NO REDO
UNDO	Immediate Update	Immediate Update
NO UNDO	Deferred Update	Shadow Paging (Schattenspeicherkonzept)

Ein **REDO-Eintrag** beinhaltet den neuen Wert des Datenelements (AFIM, After Image)

Ein **UNDO-Eintrag** beinhaltet den alten Wert des Datenelements (BFIM, Before Image)

**Steal (stehlen)/No Steal** ist das Schreiben eines aktualisierten Puffers vor dem COMMIT der Transaktion, z.B. bei Seitenauslagerung.



Bei No Steal ist keine UNDO-Operation notwendig.

Bei **Force/No Force** werden alle von einer Transaktion aktualisierten Seiten sofort beim COMMIT der Transaktion auf die Platte geschrieben.



Es ist keine REDO-Operation notwendig. Nach einem Systemabsturz erfordert No Force jedoch eine REDO-Operation.

Der **DBMS-Cache** speichert nicht nur Datenblöcke sondern auch Index- und Log-Blöcke, um einen schnelleren Zugriff zu ermöglichen.

Das **Veränderungs-Bit** (engl.: dirty bit) hat den Wert 0 beim Laden auf Cache und 1 nach einer Veränderung.

Das **Pin/Unpin-Bit** wird benötigt, um eine Seite im Cache festzuhalten (Bitwert 1), falls sie noch nicht auf die Platte zurückgeschrieben werden darf.

### Checkpoints

Ein **Log-Eintrag [checkpoint]** wird periodisch in das Log geschrieben. Alle Datenbankänderungen bis zum Zeitpunkt "checkpoint" sind bereits auf der Platte nachvollzogen worden. Der Recovery Manager legt die Intervallgröße fest, z.B. alle m

Minuten, oder nach  $t$  bestätigten Transaktionen, die seit dem letzten Checkpoint gezählt wurden ( $m$  und  $t$  sind Systemparameter).

Zum Checkpoint werden folgende Aktionen ausgeführt:

1. Temporäres Anhalten der Ausführung von Transaktionen
2. Zurückschreiben (Force) aller Puffer, die auf der Platte modifiziert wurden
3. Schreiben eines Checkpoint-Eintrags in das Log und Schreiben (Force) des Logs auf Platte
4. Wiederaufnahme der Ausführung der Transaktionen

**Fuzzy-Checkpointing** bedeutet, dass das System die Transaktionsverarbeitung wieder aufnimmt, nachdem der Checkpoint-Eintrag in das Log geschrieben wurde, ohne auf die Beendigung des Schrittes 2 warten zu müssen. Der vorherige Checkpoint-Eintrag bleibt zunächst gültig. Das System führt einen Zeiger auf den gültigen Checkpoint, der auf den vorherigen Checkpoint-Eintrag im Log zeigt.

### Write-Ahead Logging (WAL)-Protokoll

Dieses Protokoll hat als Ziel, dass der Log-Eintrag auf die Platte geschrieben wird, bevor der alte Wert durch den neuen Wert in der Datenbank überschrieben wurde. Es setzt sowohl UNDO als auch REDO voraus:

1. Der alte Wert eines Datenelements kann erst durch seinen neuen Wert auf der Platte überschrieben werden, wenn alle UNDO-Logeinträge für die aktualisierende Transaktion permanent auf Platte geschrieben wurden
2. Die COMMIT-Operation kann erst fertig werden, wenn alle REDO- und UNDO-Logeinträge dieser Transaktion auf Platte geschrieben wurden (Force)

## 9.5 Wiederherstellungsverfahren

### Wiederherstellung mit Deferred Update

Das Deferred-Update-Protokoll:

1. Eine Transaktion kann die Datenbank auf der Platte erst ändern, wenn sie ihren COMMIT-Punkt erreicht hat.
2. Eine Transaktion erreicht ihren COMMIT-Punkt erst, wenn alle ihre Änderungsoperationen im Log aufgezeichnet und das Log permanent auf Platte geschrieben wurde (ähnlich wie beim WAL-Protokoll).

Es ist also nicht notwendig, irgendwelche Operationen rückgängig zu machen. Dies entspricht dem NO UNDO/REDO-Wiederherstellungsalgorithmus.

Für den **Einbenutzerbetrieb** wird der Algorithmus **RDU\_S** (Recovery using Deferred Update in a Singleuser Environment) eingesetzt:

- Es werden zwei Transaktionslisten verwendet, und zwar:
  - 1.1. die bestätigten Transaktionen seit dem letzten Checkpoint (COMMIT-Liste)
  - 1.2. die aktiven Transaktionen, wobei hier nur eine vorhanden ist, wegen des Einbenutzerbetriebs (aktive Liste)
- REDO-Operation auf alle write-Operationen der bestätigten Transaktionen aus dem Log werden in der Reihenfolge ausgeführt, in der sie in das Log geschrieben wurden.

Eine REDO (write)-Operation besteht aus:

- Prüfung des Logeintrages [write, T, X, neuer\_wert]
- Setzen des Werts von Datenelement X in der Datenbank auf neuer\_wert (AFIM-Wert)



Beispiel

Beispiel für den Einbenutzerbetrieb:

Gegeben seien zwei Transaktionen  $T_1$  und  $T_2$ :

$T_1$	$T_2$
read A	read B
read D	write B=5
write D=0	read D
	write D=15

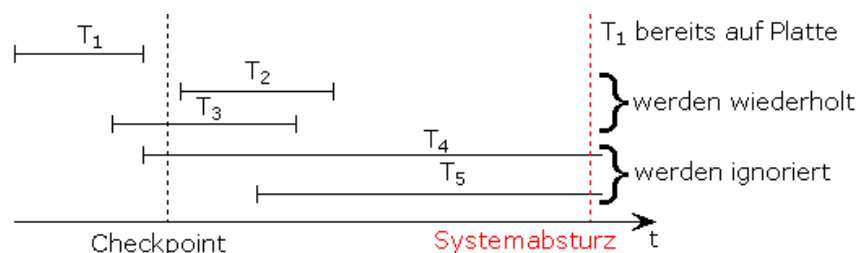
Die Einträge im Systemlog sehen wie folgt aus:

```
[start_transaction, T1]
[write, T1, D, 0]
[commit, T1]
[start_transaction, T2]
[write, T2, B, 5]
[write, T2, D, 15]
```

Vor der letzten write-Operation stürzt das System ab. Die Transaktion  $T_1$  wird wiederholt, da sie zum Zeitpunkt des Systemabsturzes bereits mit COMMIT abgeschlossen war, die Änderungen der Transaktion  $T_2$  werden beim Wiederaufsetzen jedoch ignoriert.

Beim **Mehrbenutzerbetrieb** besteht das Problem, dass je höher der angestrebte Nebenläufigkeitsgrad ist, desto zeitaufwändiger das Wiederaufsetzen ist. Dabei wird der Algorithmus RDU\_M (**R**ecovery using **D**eferred **U**pdate in a **M**ultiuser Environment) mit Checkpoints eingesetzt:

- es gibt zwei Transaktionslisten, wie beim RDU\_S
- die REDO-Operationen werden wie beim RDU\_S ausgeführt
- die aktiven Transaktionen werden annulliert und müssen erneut abgearbeitet werden



Wiederherstellung mit Deferred Update

### Wiederherstellung mit Immediate Update

Diese Art der Wiederherstellung besitzt zwei Varianten:

1. UNDO/REDO-Algorithmus: Die Transaktion kann ihren COMMIT-Punkt erreichen, bevor alle ihre Änderungen in die Datenbank geschrieben wurden
2. UNDO/NO REDO-Algorithmus: Alle Änderungen einer Transaktion werden auf Platte aufgezeichnet, bevor die Transaktion ihren COMMIT-Punkt erreicht.

Für UNDO/REDO in einem Einbenutzerbetrieb wird der Algorithmus RIU\_S (**R**ecovery using **I**mmEDIATE **U**pdate in a **S**ingleuser Environment) eingesetzt:

1. Es gibt zwei Transaktionslisten (wie RDU\_S).
2. Die UNDO-Operation wird auf alle write-Operationen der aktiven Transaktion vom Log ausgeführt.

- Die write-Operationen der bestätigten Transaktionen aus dem Log werden mit Hilfe der REDO-Operation in der Reihenfolge wiederholt, in der sie in das Log geschrieben wurden.

Eine UNDO (write)-Operation besteht aus:

- Prüfung des Logeintrags [write, T, X, alter\_wert, neuer\_wert]
- Setzen des Werts des Datenelements X auf alter\_wert (BFIM-Wert) in der umgekehrten Reihenfolge, in der die Operationen in das Log geschrieben wurden.

Für UNDO/REDO in einem Mehrbenutzerbetrieb wird der Algorithmus RIU\_M (**R**ecovery using **I**mmEDIATE **U**pdate in a **M**ultiuser **E**nvironment) eingesetzt:

- Es gibt zwei Transaktionslisten (wie RIU\_S)
- Die UNDO-Operation wird auf alle write-Operationen der aktiven Transaktion vom Log ausgeführt (wie RIU\_S). Die Operationen sollen in der umgekehrten Reihenfolge, in der sie in das Log geschrieben wurden, rückgängig gemacht werden.
- wie RIU\_S

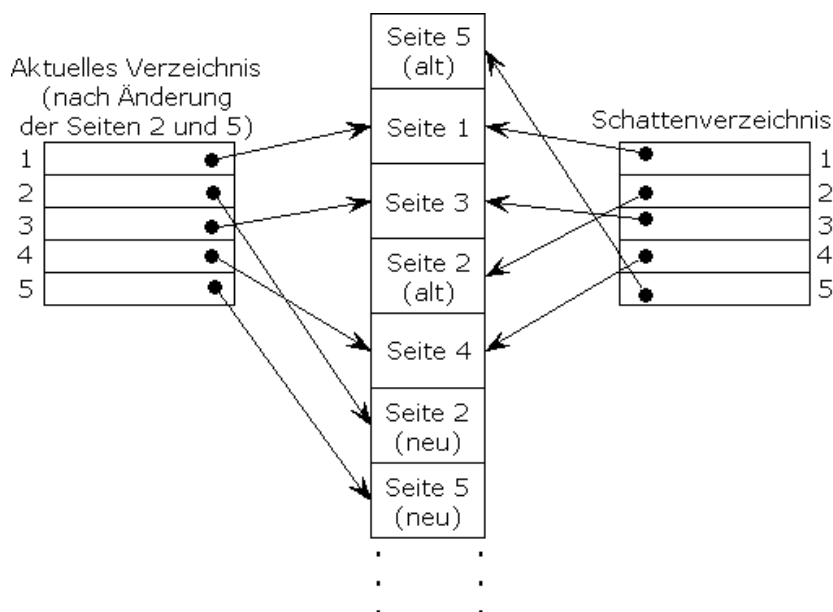
### Schattenspeicherkonzept (NO REDO/NO UNDO)

Eine Datenbank besteht aus einer Reihe von Plattenseiten oder -blöcken mit fester Größe n.

Wenn eine Transaktion beginnt, wird das aktuelle Verzeichnis in ein Schattenverzeichnis kopiert. Geänderte Seiten werden in einen freien Bereich geschrieben.



Beispiel





### Das Schattenspeicherkonzept (NO REDO/NO UNDO)

Zur Wiederherstellung wird das aktuelle Verzeichnis verworfen, und die geänderten Seiten werden freigegeben. Das COMMIT einer Transaktion entspricht dem Verwerfen des vorherigen Schattenverzeichnisses.

## 9.6 ARIES-Algorithmus

### Wiederherstellungsalgorithmus ARIES

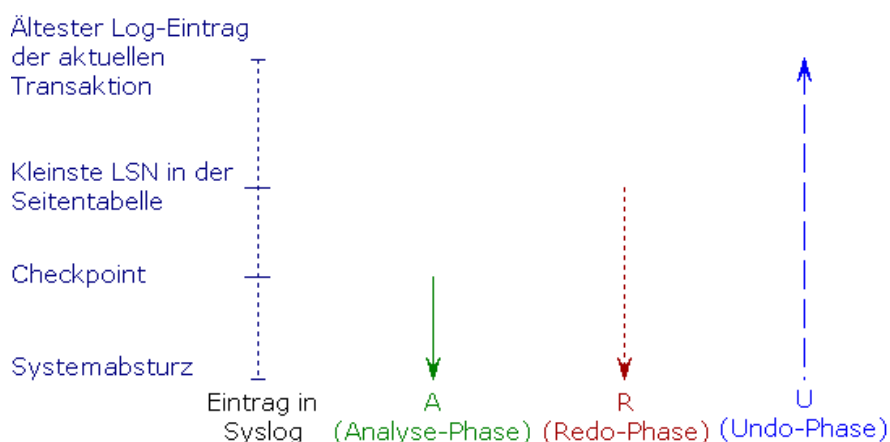
ARIES (Algorithm for **R**ecovery and **I**solation **E**xploiting **S**emantics) basiert auf der STEAL/NO-FORCE-Strategie und

1. Write-Ahead-Logging
2. Wiederholungshistorie während der Wiederholung
3. Logänderungen während des Wiederaufsetzens

Ein Logeintrag besteht aus:

- LSN (Logsequenznummer)
- Aktion
- LSN des letzten Logeintrages, der einer Änderung dieser Seite entspricht
- TID
- SeitenID
- Informationen

Grafische Darstellung des ARIES-Algorithmus:



### Grafische Darstellung des ARIES-Algorithmus

Algorithmus:

1. In der Analysephase werden die geänderten Seiten im Puffer und die aktiven Transaktionen identifiziert.
2. In der REDO-Phase werden die REDO-Operationen auf alle write-Operationen ausgeführt.
3. In der UNDO-Phase werden die Operationen von aktiven Transaktionen in der umgekehrten Reihenfolge rückgängig gemacht.



Beispiel

Gegeben seien drei Transaktionen  $T_1$ ,  $T_2$  und  $T_3$ .  $T_1$  aktualisiert die Seite 3,  $T_2$  die Seiten 2 und 3 und  $T_3$  die Seite 1.

Log (mögliche Typen sind update, commit, abort, end, CLR=compensation log records):

LSN	Letzte LSN	TID	Typ	SeitenID	Info
1	0	$T_1$	update	3	
2	0	$T_2$	update	2	
3	1	$T_1$	commit		
4	begin checkpoint				
5	end checkpoint				
6	0	$T_3$	update	1	
7	2	$T_2$	update	3	
8	7	$T_2$	commit		

Nach LSN=8 stürzt das System ab.

Zum Zeitpunkt "Checkpoint" sehen die Transaktions- und Seitentabelle folgendermaßen aus:

Transaktionstabelle:				Seitentabelle (dirty page table)	
TID	Letzte LSN	Status		SeitenID	LSN
$T_1$	3	commit		3	1
$T_2$	2	in progress		2	2

Die Analysephase beginnt ab LSN=4 und verläuft bis LSN=8. Bei LSN=6 ist ein neuer Eintrag für  $T_3$ . Nach der Analyse von LSN=8 wird der Status von  $T_2$  in der Transaktionstabelle auf COMMIT geändert, wie in der Transaktions- und Seitentabelle zum Zeitpunkt nach der Analyse gezeigt wird:

Transaktionstabelle:				Seitentabelle (dirty page table)	
TID	Letzte LSN	Status		SeitenID	LSN
$T_1$	3	commit		3	1
$T_2$	8	commit		2	2
$T_3$	6	in progress		1	6

Für die REDO-Phase ist die kleinste LSN in der Seitentabelle gleich 1. Somit beginnt REDO mit der Wiederherstellung von Aktualisierungen ab LSN=1.

Die UNDO-Phase bezieht sich nur auf die aktive Transaktion  $T_3$  und beginnt mit der LSN=6. Sie verläuft rückwärts im Log, d.h. die Update-Anweisung der  $T_3$  wird rückgängig gemacht.

## 9.7 Freitextaufgaben zu Transaktionsverwaltung und Wiederherstellung

1. Was ist das ACID-Prinzip?

### Lösung zeigen

ACID steht für atomicity (Ununterbrechbarkeit), consistency (Integritäts-erhaltung), isolation (isolierter Ablauf) und durability (Dauerhaftigkeit oder Persistenz der Daten). Dabei handelt es sich um Anforderungen, die an Transaktionen gestellt werden.

2. Wie kann Datenkonsistenz nach einem Systemabbruch gewährleistet werden?

### Lösung zeigen



Durch Rückgängigmachung (UNDO) bzw. Wiederholung (REDO) der Operationen einer nicht zu Ende ausgeführten Transaktion.

3. Erläutern Sie das Lost-Update- und das Dirty-Read-Problem.

**Lösung zeigen**

$T_1$ : read C;  $C=C+4$ ; write C; read B;  $B=B+1$ ; write B;

$T_2$ : read C;  $C=C/2$ ; write C;

Das Lost-Update-Problem kann eintreten, weil die Aktualisierung von  $T_1$  durch  $T_2$  überschrieben wurde.

Das Dirty-Read-Problem kann eintreten, wenn  $T_1$  fehlschlägt. Der Wert von C muss auf den alten Wert zurückgesetzt werden.  $T_2$  hat ggf. bereits den falschen Wert gelesen.

4. Welche Mechanismen sind notwendig für die Transaktionsverwaltung?

**Lösung zeigen**

Synchronisation (Concurrency Control), Logging, Recovery, Commit-Behandlung und Integritätskontrolle.

5. Was ist eine Verklemmung?

**Lösung zeigen**

Eine Verklemmung ist eine gegenseitige Blockierung zweier Transaktionen, wenn diese darauf warten, dass gesperrte Daten der jeweils anderen Transaktion freigegeben werden.

6. Warum ist die Serialisierbarkeit wesentlich?

**Lösung zeigen**

Serialisierbarkeit ist wesentlich, damit Verklemmungen erkannt werden können.

7. Wann sind Ausführungspläne serialisierbar?

**Lösung zeigen**

Ein Ausführungsplan heißt serialisierbar, wenn sein Ergebnis äquivalent zu dem eines seriellen Ausführungsplans ist.

8. Warum sind Sperrprotokolle wesentlich?

**Lösung zeigen**

Sperrprotokolle sind wesentlich, um konsistenten Datenbankzustände bei Mehrbenutzer-Synchronisation zu gewährleisten.

9. Was sind Checkpoints?

**Lösung zeigen**

Checkpoints sind periodische Einträge in der Protokolldatei (Systemlog).

10. Warum ist das Systemlog wesentlich? Wo muss es gespeichert werden?

**Lösung zeigen**

Das Systemlog ist wichtig für Wiederherstellungen von Transaktionen.

Es wird auf der Festplatte gespeichert, muss jedoch regelmäßig in einen Archivspeicher (Band) kopiert werden (Sicherungskopie).

11. Welche Gründe sprechen für den Einsatz des ARIES-Algorithmus in der Praxis?

**Lösung zeigen**

Für den Einsatz des ARIES-Algorithmus in der Praxis spricht die Tatsache, dass nicht nur der konsistente Datenbankzustand nach einem Systemabsturz wiederhergestellt wird, sondern auch Logänderungen während des Wiederaufsetzens aufgezeichnet werden.

## 10 Miniwelt Hochschule



### Zeitumfang

Die voraussichtliche Bearbeitungsdauer beträgt ca. 3 Stunden.



### Lernziele

Im Modul Datenbanken wird durchgehend die "Miniwelt Hochschule" benutzt, um die Theorie durch Beispiele zu illustrieren. An dieser Stelle finden Sie einen Überblick über die Miniwelt: sie wird textuell beschrieben, eine EER-Modellierung und das dazugehörige Relationenmodell werden angegeben, außerdem werden einige mögliche Belegungen der Relationen angegeben, um die Ergebnismengen der begleitenden Beispiele verdeutlichen zu können.



### Gliederung

- 10 Miniwelt Hochschule
- 10.1 Textuelle Beschreibung
- 10.2 Eine EER-Modellierung
- 10.3 Das Relationenmodell
- 10.4 Die CREATE TABLE-Anweisungen
- 10.5 Datensätze

### 10.1 Textuelle Beschreibung

Es werden Informationen über Studierende, Lehrende, Lehrveranstaltungen und Bücher gesammelt.

Von den Studierenden sind folgende Daten bekannt: Matrikelnummer (eindeutig), Name, Vorname, Geburtsdatum, Geschlecht, Adresse (bestehend aus Straße, Hausnummer, Postleitzahl und Ort), die Anzahl Urlaubssemester und Semester sowie der Studiengang. Als Studiengänge sind zugelassen: MI (Medieninformatik), PI (Praktische Informatik), TI (Technische Informatik) und WI (Wirtschaftsinformatik).

Von den Lehrenden sind die Personalnummer (eindeutig), der Name, der Vorname, die Telefonnummer, das Fach, die Raumnummer und die Gebäudenummer bekannt. Lehrende können ProfessorInnen, Lehrbeauftragte oder Lehrkräfte für besondere Aufgaben sein. Bei ProfessorInnen ist zusätzlich die Besoldungsgruppe (C2 oder C3 oder W2, bzw. B3 für die Hochschulleitung), bei Lehrbeauftragten die Anzahl der Semesterwochenstunden des Lehrauftrags und die Stufe der Vergütung (je nach Hochschulabschluss), bei Lehrkräften die Vergütungsgruppe (z.B. BAT IIa, BAT IIa/2 oder TVL Gruppe 14) bekannt.

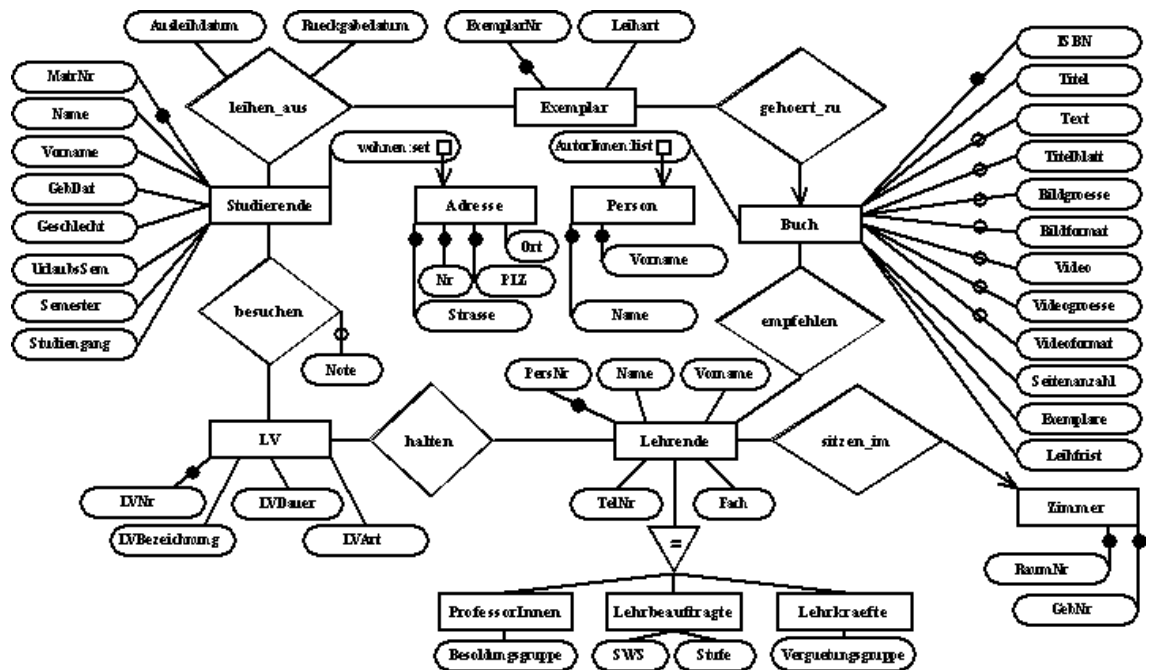
Die Bücher werden durch die ISBN eindeutig identifiziert. Von einem Buch können jedoch auch mehrere Exemplare vorhanden sein, daher ist jedem Exemplar auch eine Exemplarnummer zugeordnet, die in Verbindung mit der ISBN eindeutig ist. Um sicherzustellen, dass dasselbe Exemplar eines Buches nicht von zwei Studierenden gleichzeitig ausgeliehen werden kann, gilt zusätzlich die Integritätsbedingung, dass die Tupel (ExemplarNr, ISBN) einmalig sind (UNIQUE). Außerdem sind der Titel und der Autor/die Autoren bekannt. Ferner kann ein Beschreibungstext, das eingescannte Titelblatt samt Bildgröße und Bildformat (Dateiformat der Bilddatei) und ein Video samt Videogröße und Videoformat (Dateiformat der Videodatei) abgespeichert werden. Die Seitenanzahl, die Anzahl der vorhandenen Exemplare und die Leihfrist (in Tagen) sowie das Ausleih- und Rückgabedatum werden ebenfalls abgespeichert. Pro Buchexemplar wird außer der Exemplarnummer auch die Leihart abgespeichert. Als Leihart sind zugelassen: K (Kurzausleihe), N (normal), P (Präsenz).

Die Lehrveranstaltungen (LV) haben eine eindeutige Lehrveranstaltungsnummer, einen Titel und eine bestimmte Anzahl Semesterwochenstunden (SWS). Die Anzahl SWS kann zwischen 1 und 4 sein. Außerdem wird auch die Art der Lehrveranstaltung, nämlich VL (Vorlesung) oder L (Labor) abgespeichert.

Die Studierenden besuchen Lehrveranstaltungen, die von den Lehrenden gehalten werden. Wird die Lehrveranstaltung mit einer Prüfung abgeschlossen, erhalten die Studierenden dafür eine Note. Die Lehrenden empfehlen in den Lehrveranstaltungen Bücher, und Studierende können Exemplare dieser Bücher ausleihen.

## 10.2 Eine EER-Modellierung

(Durch Anklicken des Bildes bekommen Sie eine vergrößerte Ansicht)



Miniwelt Hochschule  
als EER-Diagramm

## 10.3 Das Relationenmodell

Attribut	Datentyp	Anmerkungen
MatrNr	char(7)	Schlüssel (primary key)
Name	varchar(30)	
Vorname	varchar(20)	
GebDat	date	im Format: TT.MM.JJJJ
Geschlecht	char(1)	m (männlich) oder w (weiblich)
UrlaubsSem	number(1,0)	Anzahl der Urlaubssemester (0=kein) $0 \leq \text{Urlaubssemester} \leq 2$
Semester	number(2,0)	Anzahl der Semester $1 \leq \text{Semester} \leq 50$
Studiengang	char(2)	MI, PI, TI oder WI



Tabelle Studierende

Attribut	Datentyp	Anmerkungen
Strasse	varchar(30)	Schlüssel (primary key)
Nr	varchar(10)	Schlüssel (primary key)
PLZ	varchar(5)	Schlüssel (primary key)
Ort	varchar(30)	



Tabelle Adresse

Attribut	Datentyp	Anmerkungen
MatrNr	char(7)	Schlüssel (primary key), references Studierende(MatrnNr)
Strasse	varchar(30)	Schlüssel (primary key), references Adresse(Strasse)
Nr	varchar(10)	Schlüssel (primary key), references Adresse(Nr)
PLZ	varchar(5)	Schlüssel (primary key), references Adresse(PLZ)



Tabelle wohnen

Attribut	Datentyp	Anmerkungen
ISBN	varchar(20)	Schlüssel (primary key)
Titel	varchar(100)	
Text	varchar(1000)	optional
Titelblatt	blob	optional

Bildgroesse	varchar(20)	optional
Bildformat	varchar(10)	optional
Video	blob	optional
Videogroesse	varchar(20)	optional
Videoformat	varchar(10)	optional
Seitenanzahl	number(4,0)	1<=Seitenanzahl<=9999
Exemplare	number(3,0)	1<=Exemplare<=999
Leihfrist	number(5,0)	

**Tabelle Buch**

Attribut	Datentyp	Anmerkungen
ExemplarNr	varchar(2)	Schlüssel (primary key)
ISBN	varchar(20)	Schlüssel (primary key), references Buch(ISBN)
Leihart	char(1)	N (normal), K (Kurzausleihe), P (Präsenz)

**Tabelle Exemplar**

Attribut	Datentyp	Anmerkungen
MatrNr	char(7)	Schlüssel (primary key), references Studierende(MatrnNr)
ExemplarNr	varchar(2)	Schlüssel (primary key), references Exemplar(ExemplarNr), UNIQUE(ExemplarNr,ISBN)

ISBN	varchar(20)	Schlüssel (primary key), references Exemplar(ISBN), UNIQUE(ExemplarNr,ISBN)
Ausleihdatum	date	
Rueckgabedatum	date	



Tabelle leihen\_aus

Attribut	Datentyp	Anmerkungen
Name	varchar(30)	Schlüssel (primary key)
Vorname	varchar(20)	Schlüssel (primary key)



Tabelle Person

Attribut	Datentyp	Anmerkungen
ISBN	varchar(20)	Schlüssel (primary key), references Buch(ISBN)
Position	number(2,0)	Schlüssel (primary key)
Name	varchar(30)	references Person(Name)
Vorname	varchar(20)	references Person(Vorname)



Tabelle AutorInnen

Attribut	Datentyp	Anmerkungen
RaumNr	varchar2(5)	Schlüssel (primary key)
GebNr	varchar2(5)	Schlüssel (primary key)





Tabelle Zimmer

Attribut	Datentyp	Anmerkungen
PersNr	char(7)	Schlüssel (primary key)
Name	varchar2(30)	
Vorname	varchar2(20)	
TelNr	varchar2(15)	
Fach	varchar2(30)	
RaumNr	varchar2(5)	references Zimmer(RaumNr)
GebNr	varchar2(5)	references Zimmer(GebNr)



Tabelle Lehrende

Attribut	Datentyp	Anmerkungen
PersNr	char(7)	Schlüssel (primary key), references Lehrende(PersNr)
Besoldungsgruppe	varchar2(3)	C2, C3, B3



Tabelle ProfessorInnen

Attribut	Datentyp	Anmerkungen
PersNr	char(7)	Schlüssel (primary key), references Lehrende(PersNr)
SWS	number(3,0)	Anzahl der Semesterwochenstunden
Stufe	varchar(3)	Vergütungsstufe (Uni, HS)



Tabelle Lehrbeauftragte

Attribut	Datentyp	Anmerkungen
PersNr	char(7)	Schlüssel (primary key), references Lehrende(PersNr)
Verguetungsgruppe	varchar(10)	z.B. BAT IIa, BAT IIa



Tabelle Lehrkraefte

Attribut	Datentyp	Anmerkungen
LVNr	char(4)	Schlüssel (primary key)
LVBezeichnung	varchar(30)	
LVDauer	smallint	$1 \leq \text{LVDauer} \leq 4$
LVArt	varchar(2)	VL (Vorlesung) oder L (Labor)



Tabelle LV

Attribut	Datentyp	Anmerkungen
LVNr	char(4)	Schlüssel (primary key), references LV(LVNr)
PersNr	char(7)	Schlüssel (primary key), references Lehrende(PersNr)



Tabelle halten

Attribut	Datentyp	Anmerkungen
----------	----------	-------------

PersNr	char(7)	Schlüssel (primary key), references Lehrende(PersNr)
ISBN	varchar(20)	Schlüssel (primary key), references Buch(ISBN)



Tabelle empfehlen

Attribut	Datentyp	Anmerkungen
MatrNr	char(7)	Schlüssel (primary key), references Studierende(MatrnNr)
LVNr	char(4)	Schlüssel (primary key), references LV(LVNr)
Note	char(3)	1.0, 1.3, 1.7, 2.0, 2.3, 2.7, 3.0, 3.3, 3.7, 4.0, 5.0



Tabelle besuchen

## 10.4 Die CREATE TABLE-Anweisungen



```
CREATE TABLE Studierende
(MatrnNr CHAR(7) PRIMARY KEY,
Name VARCHAR(30) NOT NULL,
Vorname VARCHAR(20) NOT NULL,
GebDat DATE NOT NULL,
Geschlecht CHAR(1) NOT NULL,
UrlaubsSem NUMBER(1,0) NOT NULL
CHECK (UrlaubsSem BETWEEN 0 AND 2),
Semester NUMBER(2,0) NOT NULL
CHECK (Semester BETWEEN 1 AND 50),
```

```
Studiengang CHAR(2) NOT NULL
CHECK (Studiengang IN ('MI','PI','TI','WI')));
```

```
CREATE TABLE Adresse
(Strasse VARCHAR(30),
Nr VARCHAR(10),
PLZ VARCHAR(5),
Ort VARCHAR2(30) NOT NULL,
PRIMARY KEY (Strasse,Nr,PLZ));
```

```
CREATE TABLE wohnen
(MatrnNr CHAR(7) REFERENCES Studierende,
Strasse VARCHAR(30),
Nr VARCHAR(10),
PLZ VARCHAR(5),
PRIMARY KEY (MatrnNr,Strasse,Nr,PLZ),
FOREIGN KEY (Strasse,Nr,PLZ)
REFERENCES Adresse(Strasse,Nr,PLZ));
```

```
CREATE TABLE Buch
(ISBN VARCHAR(20) PRIMARY KEY,
Titelblatt BLOB,
Bildgroesse VARCHAR(20),
Bildformat VARCHAR(10),
Titel VARCHAR(100) NOT NULL,
Text VARCHAR(1000),
Video BLOB,
Videogroesse VARCHAR(20),
Videoformat VARCHAR(10),
Seitenanzahl NUMBER(4,0) NOT NULL CHECK(Seitenanzahl>0),
Exemplare NUMBER(3,0) NOT NULL CHECK(Exemplare>=1),
Leihfrist NUMBER(5,0) NOT NULL);
```

```
CREATE TABLE Exemplar
(ExemplarNr VARCHAR(2),
ISBN VARCHAR(20) REFERENCES Buch,
Leihart CHAR(1) NOT NULL
CHECK (Leihart IN ('K','N','P'))),
```

```
PRIMARY KEY(ExemplarNr,ISBN));
```

```
CREATE TABLE leihen_aus
(MatrNr CHAR(7) REFERENCES Studierende,
ExemplarNr VARCHAR2(2),
ISBN VARCHAR(20),
Ausleihdatum DATE NOT NULL,
Rueckgabedatum DATE NOT NULL,
PRIMARY KEY(MatrNr,ExemplarNr,ISBN),
FOREIGN          KEY          (ExemplarNr,ISBN)          REFERENCES
  Exemplar(ExemplarNr,ISBN),
CONSTRAINT              Exemplar_bereits_ausgeliehen
  UNIQUE(ExemplarNr,ISBN));
```

```
CREATE TABLE Person
(Name VARCHAR(30),
Vorname VARCHAR(20),
PRIMARY KEY (Name,Vorname));
```

```
CREATE TABLE AutorInnen
(ISBN VARCHAR(20) REFERENCES Buch,
Position NUMBER(2,0),
Name VARCHAR(30) NOT NULL,
Vorname VARCHAR(20) NOT NULL,
PRIMARY KEY (ISBN,Position),
FOREIGN          KEY          (Name,Vorname)          REFERENCES
  Person(Name,Vorname));
```

```
CREATE TABLE Zimmer
(RaumNr VARCHAR(5),
GebNr VARCHAR(5),
PRIMARY KEY (RaumNr,GebNr));
```

```
CREATE TABLE Lehrende
(PersNr CHAR(7) PRIMARY KEY,
Name VARCHAR(30) NOT NULL,
Vorname VARCHAR(20) NOT NULL,
TelNr VARCHAR(15) NOT NULL,
```

```
Fach VARCHAR(30) NOT NULL,  
RaumNr VARCHAR(5) NOT NULL,  
GebNr VARCHAR(5) NOT NULL,  
FOREIGN          KEY          (RaumNr,GebNr)          REFERENCES  
    Zimmer(RaumNr,GebNr));
```

```
CREATE TABLE ProfessorInnen  
(PersNr CHAR(7) REFERENCES Lehrende PRIMARY KEY,  
Besoldungsgruppe VARCHAR(3) NOT NULL  
CHECK (Besoldungsgruppe IN ('C2','C3','B3')));
```

```
CREATE TABLE Lehrbeauftragte  
(PersNr CHAR(7) REFERENCES Lehrende PRIMARY KEY,  
SWS NUMBER(3,0),  
Stufe VARCHAR(3)  
CHECK (Stufe IN ('Uni','HS')));
```

```
CREATE TABLE Lehrkraefte  
(PersNr CHAR(7) REFERENCES Lehrende PRIMARY KEY,  
Verguetungsgruppe VARCHAR(10));
```

```
CREATE TABLE LV  
(LVNr CHAR(4) PRIMARY KEY,  
LVBezeichnung VARCHAR(30) NOT NULL,  
LVDauer SMALLINT NOT NULL  
CHECK (LVDauer BETWEEN 1 AND 4),  
LVArt VARCHAR(2) NOT NULL  
CHECK (LVArt IN ('VL','L')));
```

```
CREATE TABLE halten  
(LVNr CHAR(4) REFERENCES LV,  
PersNr CHAR(7) REFERENCES Lehrende,  
PRIMARY KEY(LVNr,PersNr));
```

```
CREATE TABLE empfehlen  
(PersNr CHAR(7) REFERENCES Lehrende,  
ISBN VARCHAR(20) REFERENCES Buch,  
PRIMARY KEY(PersNr,ISBN));
```

```

CREATE TABLE besuchen
(
  MatrNr CHAR(7) REFERENCES Studierende,
  LVNr CHAR(4) REFERENCES LV,
  Note CHAR(3)
CHECK (Note IN ('1.0','1.3','1.7','2.0',
                '2.3','2.7','3.0','3.3','3.7','4.0','5.0')),
PRIMARY KEY(MatrNr, LVNr));

```

## 10.5 Datensätze

ISBN	Titel	Seitenanzahl	Exemplare	Leihfrist
3802551230	Pusteblume	22	2	15
3499225085	Der Ekel	347	4	30
3100101065	Die Pest	362	3	30
3451280000	Die Bibel	1863	5	60
3423101776	Solaris	236	5	30
3453186834	Der Wüstenplanet	1256	3	30
3423202777	Der kleine Hobbit	331	3	30
3453140982	Das Vogelmädchen	221	3	30
3897212013	Perl 5 kurz und gut	70	3	30
3897212188	CGI kurz und gut	104	3	30

MatrNr	ExemplarNr	ISBN	Ausleihdatum	Rueckgabedatum
1562367	1	3423101776	01.01.2008	30.01.2008
6432753	1	3453186834	02.01.2008	31.01.2008
7564258	1	3423202777	02.01.2008	31.01.2008

2358712	1	3453140982	01.01.2008	30.01.2008
2358712	2	3453186834	01.01.2008	30.01.2008

MatrNr	Name	Vorname	GebDat	Geschlecht	Urlaubs-Sem	Semester	Studiengang
2358712	Meier	Siegfried	07.10.1985	m	1	5	PI
1562367	Schulze	Heiner	03.05.1980	m	2	10	MI
6432753	König	Mathilde	07.10.1985	w	2	12	PI
7564258	Baum	Meta	12.10.1982	w	0	4	WI
2356984	Dreier	Magnus	25.02.1984	m	1	8	TI
5236478	Hesse	Sarah	07.10.1985	w	0	2	PI
9812964	Meier	Hans	05.12.1985	m	0	1	PI
9252425	Müller	Karla	12.10.1984	w	0	5	TI
9365461	Schmitt	Marc	27.08.1981	m	2	11	PI
7654321	Müller	Hans	12.03.1986	m	0	5	MI
4297531	Müller	Udo	24.07.1988	m	1	6	MI
3108642	Meier	Martina	18.11.1983	w	2	12	PI
1230789	Thiess	Hugo	22.04.1984	m	0	6	TI
1286385	Zander	Wolfgang	12.10.1979	m	0	6	TI

Strasse	Nr	PLZ	Ort
Löwenwall	34	40456	Bruchtal
Neue Straße	22	23005	Brewald
Rheinring	12	40453	Bruchtal
Moorkamp	123	72345	Weiler
Bergfeld	47a	40876	Borstel
Wipperstraße	45	38459	Anstedt
Rudolfplatz	23b	38502	Braunschweig
Kreuzstraße	56	38840	Wolfsburg



Bültenweg	345	38234	Lüneburg
Fasanenkamp	9	44001	Peine
Ackerweg	110	38302	Wolfenbüttel
Berliner Straße	67	34234	Vordorf
Theaterwall	101	38498	Stöckheim

PersNr	Name	Vorname	TelNr	Fach	RaumNr	GebNr
5234260	Zimmer	Monika	92345	Mathematik	120a	5a
9652425	Irrgang	Rolf	12432	Datenbanken	20	1
1234567	Müller	Hans	3456	Physik	12	1
1357924	Müller	Udo	45367	BWL	34	2
2468013	Meier	Martina	28786	Informatik	28	2
1133556	Thein	Siegfried	38574	Mathematik	67	4
9870321	Schmidt	Manfred	23145	Elektrotechnik	220	8
1523345	Schulze	Hans	74625	Mediendesign	64	4
2314856	Maier	Franziska	52456	VLSI	300	10
3495067	Müller	August	75362	Marketing	123	5b
4526748	Sommer	Michaela	76472	Programmieren	45	3
2563172	Rehmer	Franz	27122	Informatik	25	2
3649443	Wolitz	Petra	88122	Physik	10	1
8511227	Kehr	Wolfgang	12238	Datenbank-systeme	18	1
7281128	Wagner	Wilhelm	78222	Elektrotechnik	230	8
3975982	Prell	Verena	19784	Marketing	124	5b

LVNr	LVBezeichnung	LVDauer	LVArt
111	Mathematik I	3	VL
112	Mathematik II	3	VL
113	Physik	3	VL

122	BWL	4	VL
123	Informatik	4	VL
124	Mediendesign	4	VL
125	Labor Informatik	2	L
126	Labor Datenbanken	2	L
127	Labor Elektrotechnik	2	L
161	Marketing	2	VL
411	Datenbanken	4	VL
521	Datenbanksysteme	2	VL

MatrNr	Strasse	Nr	PLZ
2358712	Löwenwall	34	40456
1562367	Neue Straße	22	23005
6432753	Rheinring	12	40453
7564258	Moorkamp	123	72345
2356984	Bergfeld	47a	40876

ExemplarNr	ISBN	Leihart
1	3423101776	N
1	3453186834	P
1	3423202777	K
1	3453140982	N
2	3453186834	N
1	3897212188	P

Name	Vorname
Lustig	Peter
Sartre	Jean-Paul

Camus	Albert
unbekannt	unbekannt
Lem	Stanislaw
Herbert	Frank
Tolkien	J.R.R.
Wallis	Velma
Vromans	Johan
Mui	Linda

ISBN	Position	Name	Vorname
3802551230	1	Lustig	Peter
3499225085	2	Sartre	Jean-Paul
3100101065	3	Camus	Albert
3451280000	4	unbekannt	unbekannt
3423101776	5	Lem	Stanislaw

RaumNr	GebNr
120a	5a
20	1
12	1
34	2
28	2
67	4
220	8
64	4
300	10
123	5b
45	3
25	2

10	1
18	1
230	8
124	5b

PersNr	Besoldungsgruppe
5234260	C2
9652425	C3
1234567	C3
1357924	C2
2468013	C2

PersNr	SWS	Stufe
9870321	6	Uni
1133556	6	HS
1523345	6	Uni
2314856	6	HS
3495067	6	Uni

PersNr	Verguetungsgruppe
2563172	BAT IIa
3649443	BAT IVa
8511227	BAT IIa/2
7281128	BAT IIa
3975982	BAT IVb

LVNr	PersNr
111	5234260
112	1133556

113	1234567
122	1357924
123	2468013
124	1523345
125	2563172
126	9652425
411	9652425
521	8511227

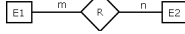
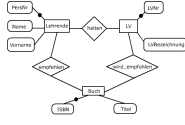
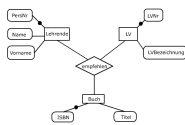
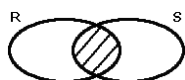
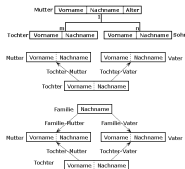
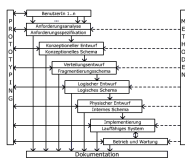
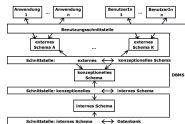
PersNr	ISBN
5234260	3897212013
9652425	3897212188
1234567	3897212013
1357924	3897212188
2468013	3897212188
2468013	3423101776
4526748	3423101776

MatrNr	LVNr	Note
7564258	122	
2356984	123	
5236478	123	
1562367	123	
5236478	124	
9812964	125	
9252425	126	
6432753	411	
7564258	521	
1286385	521	

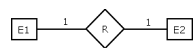
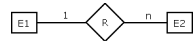

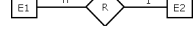


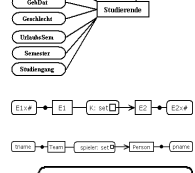
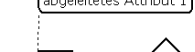

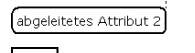

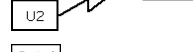
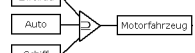

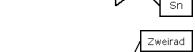

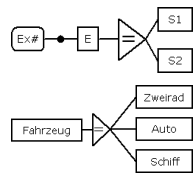
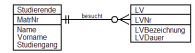
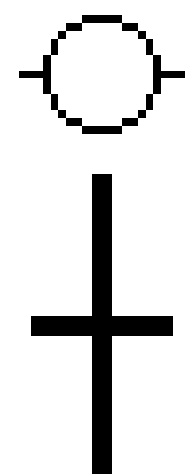
**I Literaturverzeichnis**

- FP 99** Steven Feuerstein, & Bill Pribyl Oracle PL/SQL - Grundlagen, 1. Aufl. 1999, O'Reilly
- GP 99** Peter Gulutzan & Trudy Pelzer SQL-99 Complete, Really An Example-Based Reference Manual of the New Standard 1999, R&D Books
- HMDf 09** M. Heidrich, C. Matthies, J. Dahse, fukami Sichere Webanwendungen: das Praxisbuch, 1. Aufl. 2009, Galileo Press
- KE 99** A. Kemper, A. Eickler Datenbanksysteme: Eine Einführung, 7. Auflage 2009, Oldenbourg-Verlag
- KI 98** Rainer Klute JDBC in der Praxis 1998, Addison Wesley Longman Verlag GmbH
- Ma 99** Wolfgang May Einführung in SQL (Skript) 1999, Institut für Informatik, Universität Freiburg
- Ne 96** K. Neumann Datenbank-Technik f&#252;r Anwender 1996, Carl Hanser Verlag
- Sc 96** Edwin Schicker Datenbanken und SQL 1996, B. G. Teubner
- SH 99** G. Saake, A. Heuer Datenbanken - Implementierungstechniken, 1. Auflage 1999, MITP
- SSH 10** G. Saake, K-U. Sattler, A. Heuer Datenbanken - Konzepte und Sprachen, 4. Auflage 2010, MITP
- Ta 98** Allan G. Taylor SQL für Dummies 1998, Internat. Thomson Publ.
- Vo 00** G. Vossen Datenmodelle, Datenbanksprachen und Datenbankmanagement-Systeme, 4., korr. u. erg. Aufl. 2000, Oldenbourg-Verlag

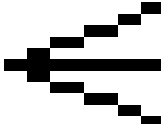
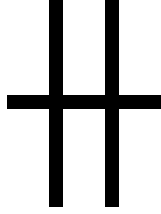
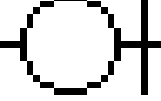
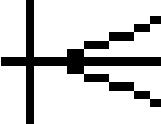
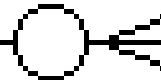
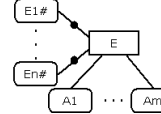
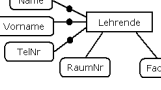


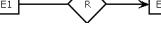






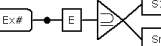
## II Abbildungsverzeichnis

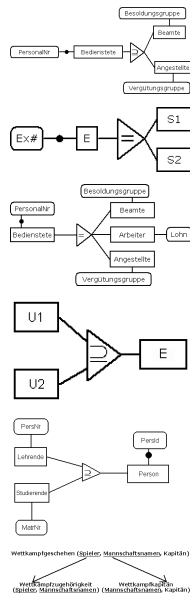


.....	5
Lochkarte.....	8
Festplatte der neueren Generation.....	9
Drei-Ebenen-Schemaarchitektur.....	15
Phasenmodell (Datenbank-Lebenszyklus).....	23
Beispiel zum hierarchischen Modell.....	30
Beispiel für ein einfaches Netz.....	31
Beispiel für ein Netz mit Zyklus.....	31
Vereinigung zweier Relationen (Relationenmodell).....	36
Differenz zweier Relationen (Relationenmodell).....	37
Durchschnitt zweier Relationen (Relationenmodell).....	37
Grundlegende Modellierungskonzepte des ER-Modells.....	43
Ausschnitt aus der VFH-Welt (ER-Modell).....	43
Dreistellige Beziehung (ER-Modell).....	44
Zweistellige Beziehungen (ER-Modell).....	45
m:n-Beziehung.....	45

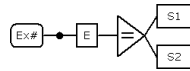
	.....	46
	.....	46
	Funktionale Beziehung (EER-Modell).....	47
	n:1-Beziehung (EER-Modell).....	47
	Einfache Komponente (EER-Modell).....	48
	Kardinalität (EER-Modell).....	48
	Abbildung einer mehrwertigen Komponente auf das Relationenmodell (Beispiel).....	49
	Mengenwertiges Attribut (EER-Modell).....	49
	Beispiel für mengenwertiges Attribut (EER-Modell).....	50
	Abgeleitetes Attribut (EER-Modell).....	50
	Generalisierung (EER-Modell).....	51
	Beispiel für Generalisierung (EER-Modell).....	51
	Spezialisierung (EER-Modell).....	51
	Beispiel für Spezialisierung (EER-Modell).....	51
	Partitionierung (EER-Modell).....	52
	Beispiel für Partitionierung (EER-Modell).....	52
	CrowFoot Beispiel.....	53
	.....	53
	.....	53



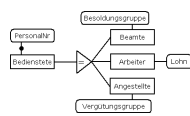
	..... 54
	..... 54
	..... 54
	..... 54
	..... 54
	Objekttyp..... 59
	Beispiel für die Abbildung eines Objekttyps auf das Relationenmodell..... 59
	Beziehungstyp..... 59
	Beispiel für die Abbildung eines Beziehungstyps auf das Relationenmodell... 60
	Funktionale Beziehung..... 60
	Beispiel für die Abbildung einer funktionalen Beziehung auf das Relationenmodell..... 61
	Beispiel für die Abbildung einer einfachen Komponenten auf das Relationenmodell..... 62
	Einfache Komponente..... 62
	Beispiel für die Abbildung einer einfachen Komponenten auf das Relationenmodell..... 62
	Mehrwertige Komponente..... 63
	Beispiel für die Abbildung einer mehrwertigen Komponenten auf das Relationenmodell..... 63
	Spezialisierung..... 64



Beispiel für die Abbildung der Spezialisierung auf das Relationenmodell.....64



Partitionierung..... 65



Beispiel für die Abbildung der Partitionierung auf das Relationenmodell.....65

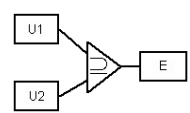


Abbildung der Generalisierung auf das Relationsmodell..... 66

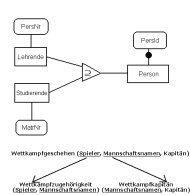


Abbildung der Generalisierung auf das Relationenmodell als Beispiel..... 66



Boyce-Codd'sche Normalform (BCNF)..... 74

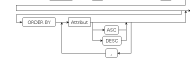
.....114



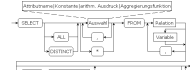
Syntax der SELECT-Anweisung (Grundform).....116



Flußdiagramm zur SELECT-Anweisung mit ORDER BY..... 119



Flußdiagramm zur SELECT-Anweisung mit WHERE-Bedingung.....124



Flußdiagramm zur SELECT-Anweisung mit GROUP BY..... 129



Flußdiagramm zur SELECT-Anweisung mit allen Klauseln.....132

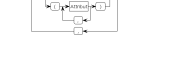
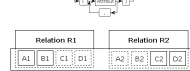
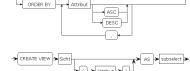
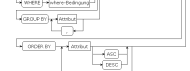
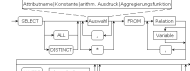


Abb.sri.1: CREATE VIEW-Syntax..... 171

Abbildung sri.2: Beispielsicht R1R2..... 171

Abbildung sri.3: Vereinfachte GRANT-Syntax..... 185

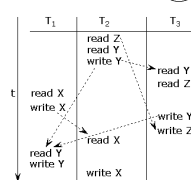
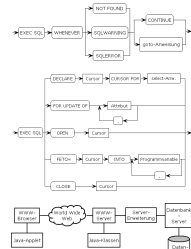
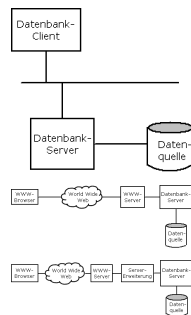
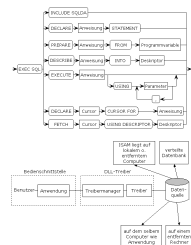
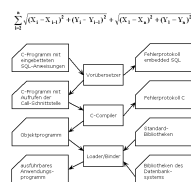
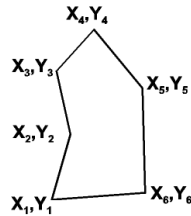
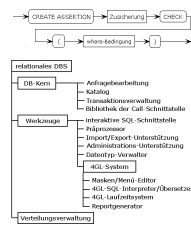


Abb.sri.5: CREATE ASSERTION-Syntax.....	186
Bestandteile eines DBMS.....	190

Polygon.....	192
--------------	-----

Übersetzung eines Embedded SQL-C-Programms.....	193
.....	196

.....	201
-------	-----

Komponenten der ODBC-Schnittstelle.....	204
---	-----

Client/Server-System.....	205
---------------------------	-----

WWW-basiertes System.....	205
---------------------------	-----

Web-basiertes Datenbanksystem mit Server-Erweiterung.....	206
---	-----

JDBC-Kategorien.....	209
----------------------	-----

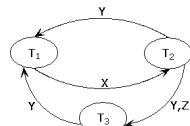
Beispiel-Frame Dateneingabe (4GL-Systeme).....	230
--	-----

Syntax der WHENEVER-Anweisung.....	237
------------------------------------	-----

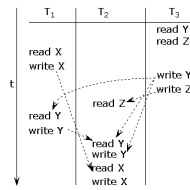
Syntax der Cursor-Anweisung bei Embedded SQL.....	237
---	-----

WWW-Datenbankanwendung mit einem Java-Applet.....	243
---	-----

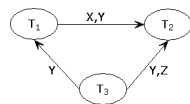
Test eines Ausführungsplans auf Konfliktserialisierbarkeit.....	257
---	-----



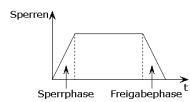
Test eines Ausführungsplans auf Konfliktserialisierbarkeit..... 257



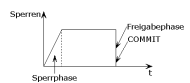
Test eines Ausführungsplans auf Konfliktserialisierbarkeit..... 258



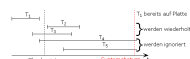
Test eines Ausführungsplans auf Konfliktserialisierbarkeit..... 258



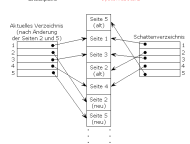
Zwei-Phasen-Transaktion..... 260



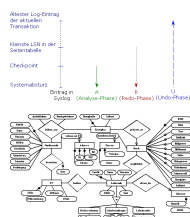
Das strikte Zwei-Phasen-Sperrprotokoll..... 261



Wiederherstellung mit Deferred Update..... 268


























Das Schattenspeicherkonzept (NO REDO/NO UNDO)..... 269




Grafische Darstellung des ARIES-Algorithmus..... 270

Miniwelt Hochschule..... 276

## III Tabellenverzeichnis

	Verschiedene Datentypen.....	32
	Tabelle Studierende.....	34
	Tabelle Lehrende.....	34
	Tabelle der Relation R.....	38
	Tabelle der Relation S.....	39
	Kartesisches Produkt der Relationen R und S (interaktiv).....	39
	Tabelle der Relation R.....	40
	Tabelle der Relation S.....	40
	Natürlicher Verbund der Relationen R und S (interaktiv).....	40
	.....	88
	.....	89
	.....	91
	Tabelle Studierende.....	92
	Tabelle Studierende.....	93
	Relation Buch.....	98
	Relation leihen_aus.....	98
	.....	111
	.....	112
	.....	113
	.....	114
	Syntax der SELECT-Anweisung (Grundform).....	116
	Relation: leihen_aus.....	135
	Relation: Exemplar.....	135

	.....	137
	.....	138
	Tabelle: Studierende.....	138
	.....	140
	.....	141
	Tabelle Studierende.....	143
	.....	144
	Tabelle Studierende.....	144
	Tabelle Studierende.....	147
	Die Wirkungsweise des ALL-Operators.....	147
	Tabelle Studierende.....	148
	Die Wirkungsweise des ANY-Operators.....	149
	Tabellen Buch und leihen_aus.....	150
	.....	152
	Tabelle Studierende.....	152
	Tabelle Lehrende.....	153
	.....	153
	.....	154
	.....	155
	Tabelle Buch_Neu.....	156
	Tabelle Buch_Neu.....	156
	.....	157
	Tabelle Buch.....	158
	.....	159
	.....	161




































	.....	162
	.....	162
	.....	163
	.....	163
	SQL-Operatoren.....	163
	.....	164
	.....	164
	.....	164
	.....	164
	.....	165
	Syntax von Sichten.....	170
	Tabelle sri.1: Oracle SQL-Anweisungen bezüglich Sichten.....	171
	.....	175
	.....	176
	.....	178
	.....	181
	Polygon als Tabelle.....	192
	PL/SQL Blöcke.....	216
	Syntax der LOOP-Schleife.....	217
	Tabelle Studierende.....	277
	Tabelle Adresse.....	278
	Tabelle wohnen.....	278
	Tabelle Buch.....	278
	Tabelle Exemplar.....	279
	Tabelle leihen_aus.....	279
























	Tabelle Person.....	280
	Tabelle AutorInnen.....	280
	Tabelle Zimmer.....	280
	Tabelle Lehrende.....	281
	Tabelle ProfessorInnen.....	281
	Tabelle Lehrbeauftragte.....	281
	Tabelle Lehrkraefte.....	282
	Tabelle LV.....	282
	Tabelle halten.....	282
	Tabelle empfehlen.....	282
	Tabelle besuchen.....	283



## IV Medienverzeichnis

	Animierte Darstellung der Drei-Ebenen-Schemaarchitektur.....	15
	Animation des Phasenmodells (Datenbank-Lebenszyklus).....	24
	.....	44
	Animation der Umsetzung einer dreistelligen Beziehung in drei zweistellige Beziehungen.....	45
	Animierte Syntax der SELECT-Anweisung (Grundform).....	116
	Animierte Syntax der SELECT-Anweisung mit ORDER BY.....	119
	Animierte Syntax der SELECT-Anweisung mit WHERE-Bedingung.....	125
	Animierte Syntax der SELECT-Anweisung mit GROUP BY-Klausel.....	130
	Animierte Syntax der SELECT-Anweisung mit allen Klauseln.....	133
	Animation der Übersetzung eines Embedded SQL-C-Programms.....	196

**V Aufgabenverzeichnis**

	.....	12
	.....	19
	.....	28
	.....	57
	.....	84
	Übung sql.1.....	90
	Übung sql.2.....	95
	Übung sql.3.....	99
	Übung sql.4.....	105
	Übung sql.5.....	109
	Übung sql.6.....	117
	Übung sql.7.....	120
	Übung sql.8.....	125
	Übung sql.9.....	126
	Übung sql.10.....	130
	Übung sql.11.....	133
	Übung sql.12.....	142
	Übung sql.13.....	151
	Übung sql.14.....	157
	Übung sql.15.....	160
	Übung sql.16.....	161
	.....	167
	.....	187



..... 244



..... 272

**VI Index****A**

API 202  
Abhängigkeit, funktionale 67  
Abhängigkeit, mehrwertige 67  
Abhängigkeit, transitive 67  
Application Programming Interface 202  
Attribut 42

**B**

B-Baum 23, 29  
Baum 29  
Baumstruktur 29  
Beziehung 42

**C**

Call-Level-Schnittstellen 194

**D**

Dateisystem 8  
Datenbank 11  
Datenbank, deduktive 54  
Datenbank, objektorientierte 54  
Datenbank, objektrelationale 8  
Datenbanksprache 11  
Datenbanksystem 11  
Datenbasis 11  
Datenintegrität 11  
Datenmodell 11  
Datenschutz 13  
Datensicherheit 174  
Datenunabhängigkeit 8  
Deduktive Datenbanken 54  
Die 4GL-SQL 229  
Drei-Ebenen-Schemaarchitektur 15  
Dynamic SQL 201

**E**

EXEC SQL 196  
Embedded SQL 196  
Entity 42

**F**

FOREIGN KEY 100  
Fremdschlüssel 100  
Funktionale Abhängigkeit 67  
Funktionale Beziehung 47

**G**

Generalisierung 47  
Gleichverbund , 34

**H**

Hash-Funktion 8  
Hash-Verfahren 23

**I**

Informationssystem 21

**J**

JDBC 207

**M**

Mehrwertige Abhängigkeit 67  
Metadaten 13, 15

**N**

Nichtschlüsselattribut 67  
Normalisierung 67

**O**

Objektorientierte Datenmodelle 54

**P**

Primärschlüssel 97  
Projektion , 34  
Prototyping 23

**R**

Rechtevergabe 174  
Redundanz 8  
Relation 32  
Report-Generator 229

**S**

Schlüsselattribut 32  
Selektion , 34  
Sicht 169  
Spezialisierung 47, 58  
Synchronisation 13

**T**

Theta-Verbund , 34  
Transaktion 17, 178, 248  
Transaktionskontrolle 178  
Transaktionsorientierte Verarbeitung 178

Transitiver Abhängigkeit 67

**V**

Verbund , 34, 76, 142

Verifikation 23

Vorübersetzer- oder Preprocessor-Ansatz  
196