Fakultät Informatik/Mathematik

HOCHSCHULE FÜR TECHNIK UND WIRTSCHAFT DRESDEN
UNIVERSITY OF APPLIED SCIENCES

Lehrstuhl für Betriebliche Informations- und Kommunikationssysteme/Datenbanksysteme

Vorlesungsmitschrift

DATENBANKSYSTEME 1

Mitschrift von

Falk-Jonatan Strube

Vorlesung von

Prof. Dr.-Ing. Axel Toll

26. März 2018

INHALTSVERZEICHNIS

1 BETRIEBLICHE INFORMATIONS- UND KOMMUNIKATIONSSYSTEME - UNTERNEHMENSMODELL - DATENBANK

1.1 DATEN ALS UNTERNEHMENSRESSOURCE

1.1.1 DATEN UND INFORMATIONEN

Redundante Daten bergen Gefahr von Inkonsistenz ⇒ Ziel: Schaffen von Datenbank mit folgenden Eigenschaften:

- ohne Inkonsistenzen (redundanzarm)
- Zugriffsschutz
- Mehrfachzugriff
- Backup-Möglichkeiten (mit Widerspruchsfreier Wiederherstellung)

Kap1.pdf
Folie 1

	Daten	Informationen
Zweck	zweckneutral	zweckgebunden
Verarbeitung	maschinell	Interpretation durch Menschen
Speicherform	vergegenständlicht	an Menschen gebunden

Betriebliche Produktionsfaktoren

- klassische Faktoren
  - Betriebsmittel
  - Werkstoffe
  - Arbeitskraft
- Daten + Informationen

Kap1.pdf
Folie 2

Große Datenbestände ⇒ Maßnahmen zur Datenorganisation

Eine mögliche Organisationsform (logisches Konzept): Ablage in Relationen (=Tabelle)

Eine Zeile in dieser Tabelle nennt man DATENSATZ (Tupel, Record, . . .).
Eine Spalte nennt man DATENFELD.

1.1.2 KLASSIFIKATION VON DATEN

Mögliche Kriterien für Datenfeld

- Zeichenart
  - ganze Zahl ⇒ für Aufzählungen
  - reelle zahl ⇒ numerische Berechnungen
  - Währung ⇒ finanztechnische Berechnungen
  - Datum ⇒ kalendarische Berechnungen/Werte
  - Text ⇒ Beschreibung
  - Bitmuster ⇒ Video, Bilder, ...
- Erscheinungsform
  - sprachlich
  - bildlich
  - schriftlich
- Stellung im Verarbeitungsprozess (E - V - A)
  - Eingabe
  - Verarbeitung
  - Ausgabe
- Verarbeitbarkeit mittels IT
  (Umwandlung in digitale Daten: analog → diskret → digital)
- Verwendungszweck

	Charakterisierung	Beispiel
Stammdaten	selten zu verändern (über längeren Zeitraum in Struktur und Inhalt konstant)	Personalstammdaten (Name, Adresse)
Änderungsdaten	Aktualisierung der Stammdaten	Änderung der Adresse
Bestandsdaten	Periodische Änderung des wertes (Inhalt) von Feldern, Datenstruktur besteht über längeren Zeitraum konstant	Lagerbestände, Kassenbestände
Bewegungsdaten	Daten zur Aktualisierung des Wertes von Bestandsdaten	Lagerzugänge und -abgänge
Archivdaten	vergangenheitsbezogene Daten die über langeren Zeitraum aufbewahrt werden	Rechnungen, Buchungen der vergangenen 5 Jahre
Transferdaten	Daten, die von einem anderen Programm erzeugt wurden und an ein anderes transferiert werden	Verkauf von Kundenadresson
Vormerkdaten	Daten, die solange existieren, bis ein genau definiertes Ereignis eintritt	Reservierung einer Materialmenge im Lager

1.1.3 DATENVERSCHLÜSSELUNG

Gemeint ist nicht die Codierung und Decodierung von Daten, sondern das Zuweisen von Schlüsseln zu Datensätzen.

Kap1.pdf
Folie 3

Identifizierender Schlüssel
kennzeichnet Objekteindeutig
Bsp.:

- Personal-Nr.
- Material-Nr.

Klassifiziernder Schlüssel
ordnet Objekt einer Klasse zu
Bsp.:

- Länderkennung: D, C, CH, ...
- Geschlecht: M, W

Hierarchischer Verbundschlüssel
identifizierender Teil hängt vom klassifizierenden Teil ab
Bsp.:

- Autokennzeichen: $\underbrace{\text{DD}}_{\text{klass.}}\ \underbrace{\text{XY 715}}_{\text{ident.}}$

Parallelschlüssel
zwei unabhängige Schlüsselteile
Bsp.:

- Flugnummer $\underbrace{\text{LH 283}}_{\text{Flugnr.}}\ \underbrace{\text{AB3}}_{\text{Flugzeug}}$

spezielle Schlüssel in Datenbanksystemen

- PRIMÄRSCHLÜSSEL (primary key PK): Datenfeld oder die Kombination aus Datenfeldern, die den Datensatz in der Tabelle eindeutig identifizieren.
  Bsp. Vereinsdatenbank:
  Primärschlüssel als einzelnes Datenfeld (Mitgliedertabelle): Migtlieds-ID
  Primärschlüssel als eine Kombination von Datendfeldern (Betragstabelle): ID mit Jahr (für Vereinsbeitrag abhängig von Jahr)
- FREMDSCHLÜSSEL (foreign key FK): Datenfeld, oder Kombination aus Datenfeldern, der (die) auf den PK einer anderen Tabelle zeigt.
  Bsp.: Mitglieds-ID in Tabelle mit Datenfelder-Primärschlüssel kommt aus der ersten Tabelle

- REFERENTIELLE INTEGRITÄT: Jeder Wert eines FK muss gleich dem Wert des PK sein, auf den der FK zeigt.
  Bsp.: Neuer Eintrag in Beitragstabelle kann nur neue Einträge bekommen, die Mitglieder aus Mitgliedertabelle enthält. Anders herum kann aus der Mitgliedertabelle kein Mitglied gelöscht werden, das noch in der Beitragstabelle genutzt wird.

Kap1.pdf
Folie 4

1.1.4 SPEICHER- UND ZUGRIFFSFORMEN

- SEQUENTIELLE SPEICHERUNG (fortlaufend)
  Bsp.: Bandlaufwerk

101	102	103	...

- VERKETTETE SPEICHERUNG
  Bsp.: verkette Listen (vgl. Programmierung I)
- INDEXVERKETTETE SPEICHERUNG
  Trennung: Datenspeicherung und „Weg“ zu den Daten
  - Indexdatei (sortiert nach entsprechendem Index)
    - Primärindex zeigt auf physische Adresse
    - Sekundärindex zeigt auf Primärindex
  - Hauptdatei

Kap1.pdf
Folie 5

Unterschied Primärschlüssel-Primärindex:

- Primärschlüssel dient dem Identifizieren
- Primärindex zum schnellen Suchen

1.2 DATENMODELLE ALS INFORMATIONELLES ABBILD DER UNTERNEHMENSREALITÄT

Kap1.pdf
Folie 6

Informationssystem

- FUNKTIONSMODELL (was soll das System leisten: Produktion, Lager, Beschaffung, . . .) $\Rightarrow$
  Kernfrage: „Was will ich machen“
  Strukturen, Abläufe
  Technik: Programm-Ablauf-Plan (PAP), Ereignisorientierte Prozessketten (EPK), . . .
- DATENMODELL
  Daten und deren logische Struktur
  Technik: Entity-Relationship-Modell (ERM)

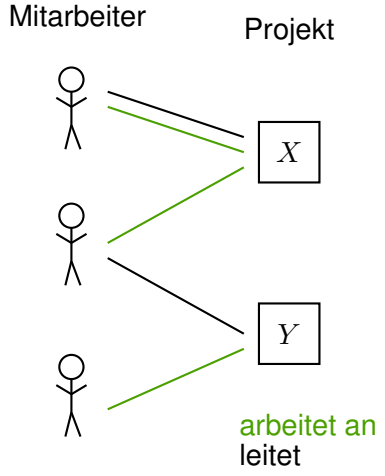
Kap1.pdf
Folie 7

Kap1.pdf
Folie 8

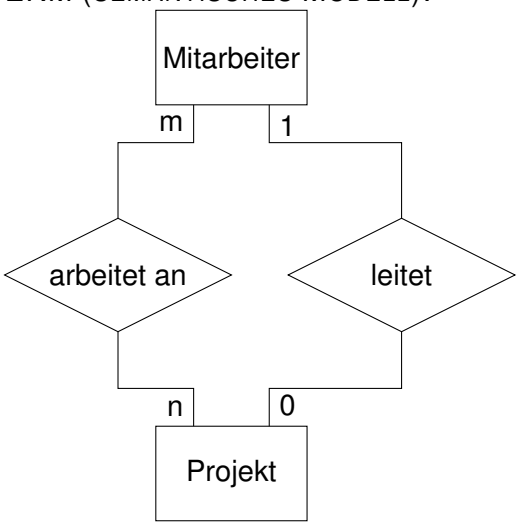
Kap1.pdf
Folie 9

Bsp.:

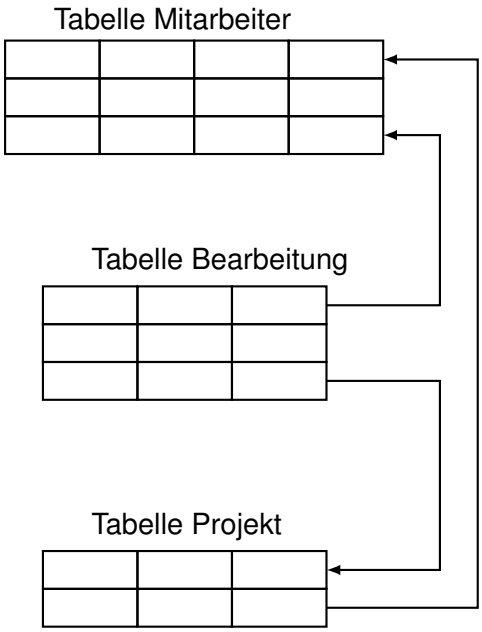
REALE WELT:



ERM (SEMANTISCHES MODELL):

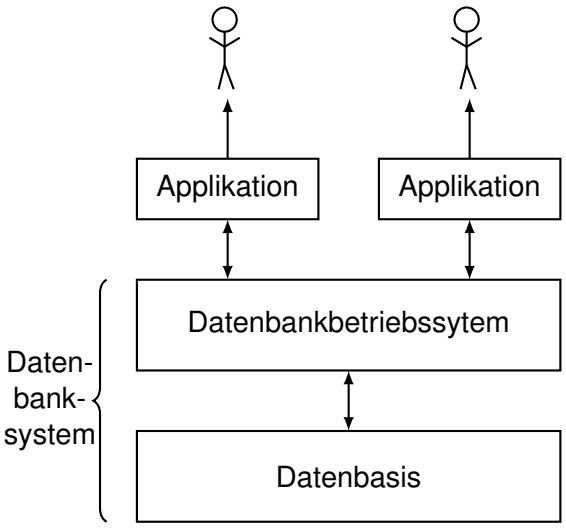


RM (RELATIONALES/LOGISCHES MODELL):



1.3 DATENBANKSYSTEME ALS TECHNOLOGISCHE GRUNDLAGE DER DATENVERWALTUNG

Kap1.pdf
Folie 10

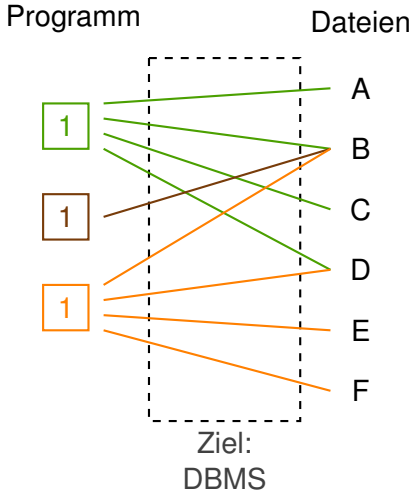


Datenbasis: Tabellen mit Metadaten
Datenbankbetriebssystem (DBMS): Software, die mit Datenbasis kommuniziert

2 GRUNDLAGEN UND ARCHITEKTUR EINES DATENBANKSYSTEMS (DBS)

2.1 DEFEKTE DES KONVENTIONELLEN ANSATZES DER DATENVERWALTUNG / ZIELSTELLUNG DES DATENBANKORIENTIERTEN ANSATZES

konventionell



konventionelle Datenorganisation

MERKMALE

- Datenspeicherung je Anwendung
- Datenspeicherung auf physischem Niveau

NACHTEILE

- mangelnde Passfähigkeit (Zugriffskonflikte usw.)
- Redundanz
- Konsistenzprobleme
- mangelnde Flexibilität
- Daten-Programm-Abhängigkeit (kurz: Datenabhängigkeit)

Kap2.pdf
Folie 1

Kap2.pdf
Folie 2

Kap2.pdf
Folie 3

Zielsetzung des Datenbankeinsatzes

Kap2.pdf
Folie 4

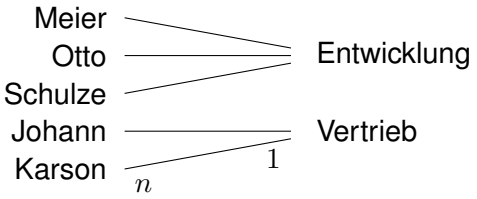
1. Bsp. für gewollte Redundanz: Sekundärindex
2. Datensicherheit:
   - physisch, falls bspw. der Server abbrennt
   - logisch, dass bspw. alle Daten den richtigen Typ haben

2.2 ARCHITEKTUR VON DATENBANKSYSTEMEN

2.2.1 GRUNDLEGENDE BEGRIFFE

Am Beispiel der Objekte der Datenmodellierung mittels ERM

Begriff	Erklärung	Beispiel
Entity	Objekt der realen Welt	Max Meier, Arbeitsaufgabe Reportgenerator
Entity-Typ	Objektklasse (-Menge), enthält Elemente mit struktureller Ähnlichkeit	Mitarbeiter, Arbeitsaufgabe, Abteilung
Merkmale / Attribut / Prädikat	Beschreibungen eines Entity-Typs	Name, Vorname, Gehalt
Wert	Ausprägung des Merkmals je Entity, aus einem bestimmten Wertevorrat (Domain)	„Meier“, „Max“, 3800,-
Beziehung, Set	Logischer Zusammenhang zwischen Entity-Typen	Mitarbeiter – arbeitet an – Arbeitsaufgabe
Beziehungstyp, Settyp	Art der Beziehung (mögliche Anzahl an Entitäten, die in Beziehung treten)	$n:1$ Mitarbeiter – gehört zu – Abteilung



2.2.2 3-EBENEN-ARCHITEKTUR

gemäß ANSI x3/SPARC (1975)

- Architekturebene
  - externe Ebene
  - konzeptionelle Ebene
  - interne Ebene
- Modell
  - externes Modell
  - konzeptionelles Modell
  - internes Modell
- Schema (konkrete Ausprägung des Modells)
  - externes Schema
  - konzeptionelles Schema
  - internes Schema

2.2.2.1 KONZEPTIONELLE EBENE

Gegenstand: logisches Modell des gesamten Systems

Beschreibungselemente:

- Entity-Typen
- Beziehungen
- Attribute
- Wertevorrate (bspw. Einschränkung von Alter: nur Zahlen zwischen 1 und 100)
- Integritätsbedingung (bspw. NOT NULL, vgl. Wertevorrat)

2.2.2.2 EXTERNE EBENE

Gegenstand: Beschreibung AUSGEWÄHLTER Elemente der konzeptionellen Ebene aus Sicht des jeweiligen Endbenutzers

Kap2.pdf
Folie 5

Element: Sicht (View)

2.2.2.3 INTERNE EBENE

Gegenstand: Form/Art der Ablage der Elemente der konzeptionellen Ebene im physischen Speicher

Element: Index

Kap2.pdf
Folie 7

2.3 AUFGBAU UND ARBEITSWEISE VON DBMS

5 Grundfunktionen eines DBMS

Kap2.pdf
Folie 8

2.3.1 ZUGRIFFSVERMITTLUNG

Kap2.pdf
Folie 9

2.3.2 UNTERSTÜTZUNG DATENBESCHREIBUNG-ENTWICKLUNG

Kap2.pdf
Folie 10

2.3.3 INTEGRITÄTSSICHERUNG

Kap2.pdf
Folie 11

Bsp. operationale Integrität:
Gehaltserhöhungen sowohl für Organisatoren (O) und Programmierer (P) um €50,-.
Gehaltserhöhung darf nicht doppelt erfolgen $\Rightarrow$ Sperren von Gehalt, solange ein Nutzer das Gehalt ändert (bei Gefahr bezgl. Deadlock, muss das System das Problem erkennen und entsprechend auflösen).

2.3.4 ZUGRIFFSSCHUTZ

Kap2.pdf
Folie 12

2.3.5 DIENSTPROGRAMMFUNKTIONEN

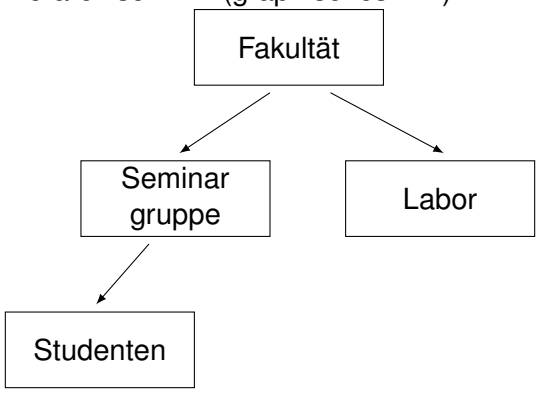
Kap2.pdf
Folie 13

2.4 DATENORGANISATION

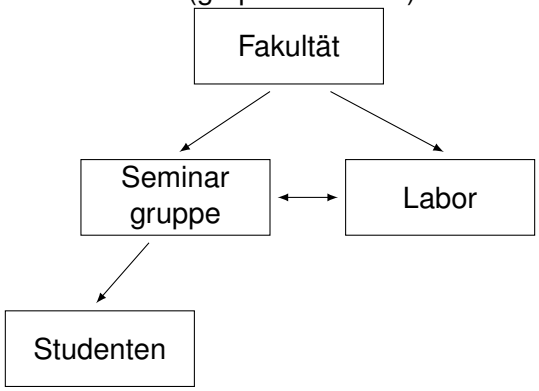
- logische Datenorganisation (DO)
  - externe Ebene
  - konzeptionelle Ebene
- physische DO
  - interne Ebene

klassische Datermodelle (logisch)

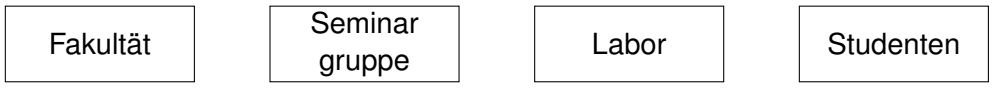
- hierarchisch DM (graphisches DM)



- Netzwerk DM (graphisches DM)



- relationales DM (behandelt in DBS I+II)



weitere DM

- objektorientiertes DM (DBS II)
- objektrelationales DM (DBS II)
- XML-DM / NoSQL DM . . . (DBS III)

Kap2.pdf
Folie 14

	Hierarchisches DM	Netzwerk DM	relationales DM
Einstiegspunkt	ein Entity-Typ	mehrere Entity	beliebig
strukturelle Beschränkung	Hierarchie	keine	keine
Zeitpunkt des Aufbau der Beziehung	zur Entwicklungszeit	zur Entwicklungszeit	zur Laufzeit
Performance	+	+	**−**
Flexibilität bzgl. Änderung	–	–	**+**

3 RELATIONALES DATENMODELL

3.1 TERMINOLOGIE IM RELATIONENMODELL

Kap3.pdf
Folie 1

Bsp.:
Entitytyp:

- Zeugnis

Attribute:

- A_1 Fach
- A_2 Note

Wertebereiche:

- W_1 {Ma, Ph}
- W_2{1 ,2 ,3 , 4 , 5}

$n = 2$, d.h. 2-stellige Relation ableitbar (Grad = degree = 2)
$PM = W_1 * W_2 = W_1 \times W_2$

Fach	Note
Ma	1
Ma	2
Ma	3
Ma	4
Ma	5
Pd	1
Pd	2
Pd	3
Pd	4
Pd	5

Teilmenge 1 = Relation 1:

Fach	Note	
Ma	1	gültig
Ph	2	

Teilmenge 2 = Relation 2:

Fach	Note
Ma	1
Ph	1
Ph	4

gültige Relation (unabhängig von der semantischen Sinnhaftigkeit)

Kap3.pdf
Folie 2

Weitere Kernaussagen zum relationalen Modell:

- Darstellung der Relation als Tabelle
- Identifikation der Relation über Namen
- Anzahl an Attributen (Spalten) ist fest (degree)
- Anzahl der Tupel (Zeilen) ist variabel (Mächtigkeit)
- Wertebereiche der Attribute = Domain
- Im Kreuzungspunkt von Attribut und Tupel stehen ATOMARE Werte

3.2 DEFINITION UND MANIPULATION IM RELATIONALEN DATENMODELL

3.2.1 DATENDEFINITION

⇒ Definition von Relationen

Kap3.pdf
Folie 3

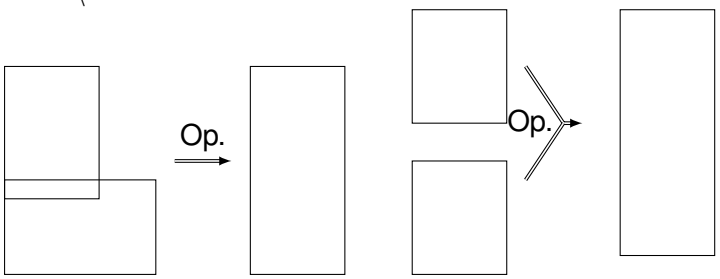
Kap3.pdf
Folie 4

3.2.2 DATENMANIPULATION / RELATIONENALGEBRA

Relationenalgebra nach: Codd
Grundidee:
Operationen auf Relationen
⇒ Ergebnis ist wieder eine RELATION
D.h. mengenweise Arbeit NICHT satzweise.

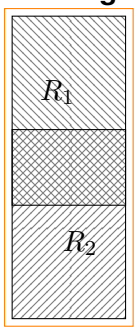
3.2.2.1 MENGENOPERATIONEN

$\cup \cap \setminus \times$



Kap3_Beispiele.pdf
Folie 1, scale=0.8

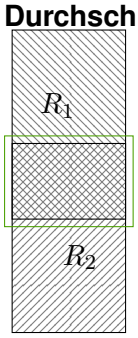
Vereinigung $\cup$



UNION

Kap3_Beispiele.pdf
Folie 2, scale=0.8

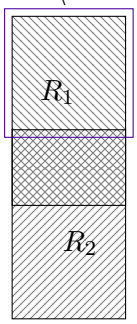
Durchschnitt $\cap$



INTERSECTION

Kap3_Beispiele.pdf
Folie 3, scale=0.8

Differenz $\setminus$



$R_1 \setminus R_2$

Bedingung für $\cup, \cap, \setminus$ (VEREINIGUNGSVERTRÄGLICHKEIT):

- Anzahl an Attributen ist gleich
- unzugeordnete Attribute besitzen gleiche Domain (Domainverträglichkeit)

$R_1 \cup R_2 = R_2 \cup R_1$
$R_1 \cap R_2 = R_2 \cap R_1$
$R_1 \setminus R_2 \neq R_2 \setminus R_1$
DIFFERENCE

Kap3_Beispiele.pdf
Folie 4, scale=0.8

Kartesisches Produkt $\times$

$R_1 \times R_2$
Ergebnisrelation enthält:

- alle Attribute aus R_1 und R_2.
- alle Kombinationen an Tupeln aus R_1 und R_2.

R_1

xxxxxxxx
yyyyyyyy

R_2

aaaaaaaaaaaaa
bbbbbbbbbbbbb
ccccccccccccc

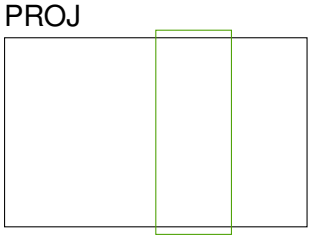
$\Downarrow R_1 \times R_2$

xxxxxxxx	aaaaaaaaaaaaa
xxxxxxxx	bbbbbbbbbbbbb
xxxxxxxx	ccccccccccccc
yyyyyyyy	aaaaaaaaaaaaa
yyyyyyyy	bbbbbbbbbbbbb
yyyyyyyy	ccccccccccccc

Kap3_Beispiele.pdf
Folie 7, scale=0.8

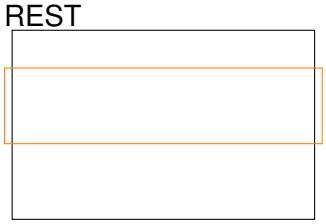
3.2.2.2 RELATIONALE OPERATIONEN

Projektion Spaltenauswahl
PROJ



Kap3_Beispiele.pdf
Folie 5, scale=0.8

Selektion Tupelauswahl (laut Bedingung)
REST



Kap3_Beispiele.pdf
Folie 6, scale=0.8

Verbund Verbindung zwischen zwei Relationen bezüglich der Gleichheit der Attributwerte in einer Verbindungsspalte
JOIN
intern:

1. Kartesisches Produkt der Relation
2. auf Ergebnisrelation Selektion nach Gleichheit der Werte in der/den Verbindungsspalten

Merkmale des JOIN:

- Attribute über die den JOIN ausgeführt wird, müssen
  - KEINE Schlüsselspalten sein
  - gleiche Domain besitzen
  - NICHT die gleichen Namen besitzen

  Jede Relation ist mit jeder Relation via JOIN verbindbar (auch mit sich selbst).

Kap3_Beispiele.pdf
Folie 8, scale=0.8

3.3 NORMALFORMENLEHRE

Ziele der Normalisierung:

- Vermeidung unerwünschter Abhängigkeiten beim Ändern, Löschen und Einfügen

- Reduzierung der Umbildung von Relationen bei Einführung neuer Attribute
- Erhöhung der Transparenz und Aussagekraft für den Nutzer (Trennung der unterschiedlichen Konzepte der realen Welt)
- Gewährung der Korrektheit der Datenbakn (zu jedem Zeitpunkt)

Vorteile der Normalisierung:

- Sicherung von relativ einfachen, überschaubaren und einfach handhabbaren Relationen
- Beseitigung von Update-/Insert- und Delete-Anomalien
- Einfachere Überprüfung von Konsistenzbedingungen

Nachteile:

- größere Redundanz (Schlüsselredundanz)
- höherer Aufwand bei komplexen Auswertungen

Codd (1970)
Normalform (NF): $\underbrace{\text{1. NF} \Rightarrow \text{2. NF} \Rightarrow \text{3. NF}}_{\text{praktisch relevant}} \Rightarrow \text{4. NF} \Rightarrow \text{5. NF}$

3.3.1 1. NORMALFORM

Kap3.pdf
Folie 5

⇒ Relation
- atomare Werte
- PS erweitern

3.3.2 2. NORMALFORM

Kap3.pdf
Folie 6

Abhängigkeiten:

PS	Nichtschlüssel-Attribute	2. NF
Mitnr, Projnr	Anteil	MiPro
Mitnr	Name, Beruf, Gehalt, Abtnr, Abtbez	Mitarbeiter
Projnr	Projbez	Projekt

⇒ Zerlegung
- volle funktionale Abhängigkeit

3.3.3 3. NORMALFORM

Kap3.pdf
Folie 7

für Mitarbeiter (M): (x→y: von x kann man auf y schließen)
M.Mitnr $\rightarrow$ M.Abtnr
M.Abtnr $\not\rightarrow$ M.Mitnr
M.Abtnr $\rightarrow$ M.Abtbez
Also:
M.Mitnr $\rightarrow$ M.Abtnr $\rightarrow$ M.Abtbez
aber:
M.Abtnr $\not\rightarrow$ M.Mitnr

$\Rightarrow$ weitere Zerlegung

Abtnr $\rightarrow$ weitere Tabelle Abteilung mit PS=Abtnr.

Kap3.pdf
Folie 8

Kap3_Beispiele.pdf
Folie 9, scale=0.8

Kap3_Beispiele.pdf
Folie 10, scale=0.8

3.4 VERGLEICH RELATIONALER DBMS

Kap3.pdf
Folie 9

NULL $=$ missing value (kein Wert)
$\neq$ ' '
$\neq \varnothing$

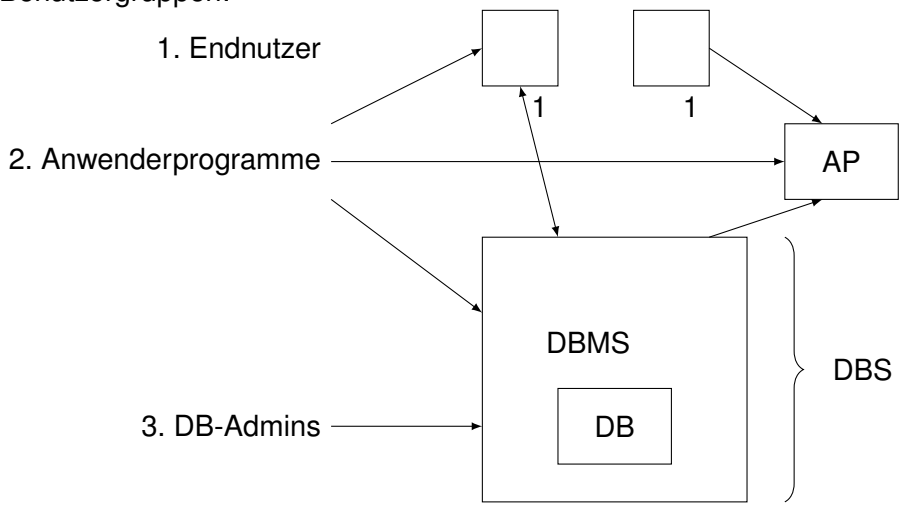
Kap3.pdf
Folie 10

Kap3.pdf
Folie 11

4 DATENBANKSPRACHEN FÜR RELATIONALE DBMS

4.1 BENUTZERGRUPPEN UND DATENBANKSPRACHEN

Benutzergruppen:



Einordnung der Datenbanksprache SQL Einteilung der Programmiersprachen:

- klassisch nach Gen. 1-4
- weitere Einteilung

Kap4.pdf
Folie 1

Kap4.pdf
Folie 2

3. Generation of Language (GL) (prozedural)

```
opne (buecher);
while (not EOF(buecker)){
  read (buch);
  if (buch.leihfrist > 21)
    print (buch.titel);
}
close (buecher);
```

⇒ WIE

4. GL (descriptiv, NICHT prozedural)

```
SELECT titel
FROM buecher
WHERE leihfrist > 21
```

⇒ WAS

Kap4.pdf
Folie 3

4.2 SQL ALS STANDARDSPRACHE FÜR RELATIONALE DBMS

4.2.1 ÜBERBLICK

SQL - Structured Query Language

- Structured – „oft etwas übertrieben“
- Query – „bescheiden“
- Language – „unanfechtbar“

Kap4.pdf
Folie 4

Kap4.pdf
Folie 5

Klassen an SQL-Befehlen:

- DDL – Data Definition Language
- DCL – Data Control Language
- DML – Data Manipulation Language

Dialekte:
z.B.:

- pl SQL ⇒ Oracle (Postgresql)
- t SQL ⇒ MS SQL-Server, Sybase (SAP)

4.2.1.1 SQL-DATENTYPEN (AUSWAHL)

- exakte Numerische Werte
  - INT ganze Zahl mit VZ
  - SMALLINT ganze Zahl mit VZ (kleinerer Wertebereich als INT)
  - DECIMAL (m,n) Dezimalzahl mit m Stellen, davon n EXAKTE Nachkommastellen
- numerische Näherungswerte (Gleitkommazahl)

  - FLOAT (n) Gesamtstellenanzahl n
  - DOUBLE PRECISION i.A. größerer Wertebereich als FLOAT
- Datum/Uhrzeit (abhängig vom System)
  - DATE
  - TIME
  - DATETIME
- Zeichenketten (Zeichenkette der Länge n)
  - CHAR (n) (Speichert gesamte Zeichenketten)
  - VARCHAR (n) (Speichert Zeichenkette ohne Leerzeichen)
- weitere spezielle Typen
  - BIT (n) Bitkette der Länge n
  - BLOB (n) Binary-Array
  - TEXT Zeichenkette
  - CLOB (n) Zeichenkette
  - IMAGE Bilder
  - BOOLEAN Boolsche Werte
  - MONEY Währung
  - XML
  - ...

4.2.1.2 SYMBOLE DER SYNTAXBESCHREIBUNG:

(EBNF)

$\|$	Alternative (XOR)
$[\,]$	optionales Element (ein- oder keinmal)
$\{\,\}$	Gruppierungen (ein- oder mehrmals)
...	Wiederholung
$<>$	Sprachbeschreibende Variablen
$::=$	Definitionssymbol

4.2.2 ANWEISUNGEN ZUR DEFINITON (DDL-BEFEHLE)

4.2.2.1 TABELLENDEFINITION

Anlegen einer Tabelle

Kap4.pdf
Folie 6

```
CREATE TABLE Kunde(
  KuNr          INT PRIMARY KEY,
  Name          CHAR(10) NOT NULL,
  Vorname       CHAR(10),
  Ort           CHAR(20),
  Gebdatum      DATETIME NULL,
  Geschlecht    CHAR(1) NULL,
  TelNr         INT,
  CHECK( 999 < TelNr < 10000)
)
```

- PRIMARY KEY: Primärschlüssel
- NULL: missing value (empty) $\neq 0$ $\neq$' ' (typübergreifend)
- NOT NULL: Werteingabe zwingend
  (wenn weder NULL noch NOT NULL da steht: wird vom System bestimmt $\rightarrow$ wird entweder NULL oder NOT NULL gesetzt. Empfehlung: für jeden Eintrag selber festlegen!)
- UNIQUE: eindeutiger Wert

```
CREATE TABLE Artikel (
  Artnr SMALLINT PRIMARY KEY,
  Bezeichnung CHAR(15) NOT NULL,
  Beschreibung CHAR(30) NULL,
  EPreis DECIMAL(7,2) NOT NULL
)
```

```
CREATE TABLE Kauf (
  Kunr INT,
  Artnr SMALLINT,
  Menge INT NOT NULL,
  VPreis decimal(8,2) NOT NULL,
  PRIMARY KEY(Kunr, Artnr),
  FOREIGN KEY(Kunr) REFERENCES Kunde(Kunr),
  FOREIGN KEY(Artnr) REFERENCES Artikel(Artnr)
)
```

Bei zusammengesetzten Primärschlüsseln muss dieser am Ende von CREAT TABLE definiert werden. Bei einem einfachen kann dies INLINE passieren oder auch wie beim zusammengesetzten am Schluss:

```
CREATE TABLE Artikel (
  Artnr SMALLINT,
  Bezeichnung CHAR(15) NOT NULL,
  Beschreibung CHAR(30) NULL,
  EPreis DECIMAL(7,2) NOT NULL,
  PRIMARY KEY (Artnr)
)
```


Ändern der Tabellendefinition

Kap4.pdf
Folie 7

Bsp.: Spalte Email an Tabelle Kunde anfügen.

```
ALTER TABLE Kunde ADD email CHAR(20) NULL
```

Fall: neue Spalte soll NOT NULL sein. Entweder DEFAULT-Wert übergeben oder erst als NULL einführen, Inhalte befüllen und dann NOT NULL SETZEN.

Löschen einer Tabelle

```
DROP TABLE tab_name
```

Beispiel:

```
DROP TABLE Kauf
```

Befehl wird ohne Nachfrage ausgeführt (wenn Tabelle nicht als Referenz verwendet wird)!

4.2.2.2 ARBEITEN MIT INDEX

(Primär-)Index dient dem schnellen Suchen, Primärschlüssel dient nur dem Identifizieren.

Anlegen des Index

```
CREATE INDEX index_name ON tab_name(index)
```

Bsp.: Zusammengesetzter Index (Kunde nach Ort und dann nach Name indexiert)

```
CREATE INDEX iOrtName ON Kunde(Ort,Name)
```

Löschen eines (Sekundär-)Index

```
DROP INDEX tab_name.index_name
```

```
DROP INDEX Kunde.iOrtName
```

4.2.2.3 VIEW-DEFINITION

Kap4.pdf
Folie 8

Eine View kann auch Daten aus mehreren Tabellen nehmen.

Anlegen einer View

```
CREATE VIEW view_name [ ( <Spaltenliste> ) ]
AS <select_anweisung>
```

Bsp.:

```
CREATE VIEW Beruf_Inf
AS (
  SELECT Kunr, Name, Vorname, Ort
  FROM Kunde
  WHERE Beruf = 'Informatiker'
)
```

Löschen einer View

```
DROP VIEW view_name
```

Bsp.:

```
DROP VIEW Beruf_Inf
```

Zugriff auf eine View

```
SELECT Kunr, Name
FROM Beruf_Inf -- (View Name)
```

4.2.3 ANWEISUNGEN ZUR ABFRAGE (DML-BEFEHLE)

4.2.3.1 STANDARDABFRAGE

Kap4.pdf
Folie 9

```
SELECT   Bilbereich          <Attributliste>   Was?
FROM     Definitionsbereich  <Relation>        Wovon?
WHERE    Auswahlkriterien    <Bedingungen>     Bedingung?
```

4.2.3.2 EINFACHE ABFRAGE AUF EINE RELATION

```
SELECT < attributliste >
FROM < tab_name > | < view_name >
[ WHERE < auswahlbedingung > ]
```

einfachste Abfrage:

```
SELECT *  -- * ... alle Attribute (Spalten)
FROM Kunde
```

Beim Programmieren am besten nicht * verwenden. Nur für's Abfragen. * ist Fehleranfällig bei Veränderungen!

```sql
SELECT Name, Vorname
FROM im15s12345.dbo.Kunde      -- (Datenbank.Eigentümer.Tabellenname)
```

```sql
SELECT Name, Vorname
FROM Kunde
WHERE Ort='Dresden'
```

Abfrage mit Vergleichs- und logischen Operatoren

```sql
SELECT * FROM Kunde
  WHERE Geschl = 'M'

SELECT * FROM Kunde
  WHERE (Ort='Pirna' OR Ort='Dresden') AND Beruf = 'Projektant'

SELECT * FROM Artikel
  WHERE NOT Artnr = '1234'     -- Alternativ: Artnr != '1234'
```

Abfrage mit Patternsuche

```sql
... WHERE <spaltenname> LIKE 'muster'
```

```sql
SELECT * FROM Kunde
WHERE Name LIKE '%GmbH'

SELECT * FROM Kunde
WHERE Name LIKE 'M_$\,$_er' -- sucht bspw. Maier, Meyer oder Mayer
```

Funktion

```sql
SUBSTR ( <spaltenname>, <startpos>, <länge> )
```

```sql
SELECT * FROM Kunde
WHERE SUBSTR(Name, 2,2)='ei'  -- ACHTUNG: substring fängt bei 1 an zu zählen: 1. Zeichen ist nicht an der Stelle 0, sondern tatsächlich 1
-- das gleiche wie:
WHERE Name LIKE '_ei%'
```

Bereich

```sql
SELECT * FROM Kunde
WHERE Name LIKE '[E-N]%'  -- alle Namen die mit Buchstaben zwischen E und N beginnen
WHERE Name LIKE '[^E-N]%' -- alle, die nicht mit Buchstaben zwischen E und N beginnen
```


Attributzuweisung in Abfrage

```
AS <spaltenname>
```

```
SELECT Kunr, Name, AS Kundenname
FROM Kunde

SELECT Kunr, Vorname | ', ' | Vorname AS Kundenname
FROM Kunde
```

zweiteres würde liefern:

Kunr	Kundenname
123	Maier, Uwe
234	Adler, Sabine
...	

```
SELECT Kunr, Artnr, Menge*vPreis AS Umsatz
FROM Kauf
```

liefert:

Kunr	Artnr	Umsatz
123	1234	3600
234	5555	500
...		

Datumsfunktionen

1.

```
DAY(), MONTH(), YEAR(), HOUR(), ...
```

```
SELECT YEAR(GebDat)
FROM Kunde
```

2.

```
DATEDIFF ( <datepart>, <begin>, <end> ) -- <datepart>: dd, mm,
   yy
```

```
SELECT DATEDIFF (dd, GETDATE(), '24/12/2016') -- Achtung
   Formatierung des Datums ist von DBMS abhängig (ob / oder .
   und Reihenfolge)
```

3.

```
DATEADD ( <datepart>, <number>, <date> )
```

```
DATEADD ( dd, 50, GETDATE() )
```


4.2.3.3 SORTIERUNG / GRUPPENBILDUNG

Kap4.pdf
Folie 10

Sortierung

```sql
SELECT * FROM Kunde
ORDER BY Name, Vorname

SELECT * FROM Kunde
ORDER BY GebDat DESC  -- jüngste zuerst
```

Gruppierung

```sql
SELECT Ort, SUM( Kredit ) AS Kreditsumme
-- Achtung! Auswahl kann nicht auf bspw. Name erweitert werden, weil dann die Tabelle für einen Ort mehrere Namen hätte
-- Ähnlich: Nicht mit bspw. Kredit erweitertbar. ABER: Möglich ist SUM(Kredit), was dann alle Werte für den Ort aufsummieren würde
FROM Kunde
GROUP BY Ort  -- Jeder Ort taucht nur einmal auf
```

4.2.3.4 BUILT-IN-FUNKTIONEN

(Aggregatfunktionen, Statistikfunktionen)

```sql
SUM ()
AVG ()
MAX ()
MIN ()
COUNT ()
```

```sql
SELECT SUM( Kredit )
FROM Kunde    -- Summe aller Kredite von allen Kunden (eine Tabelle mit nur einem Eintrag: der Summe)

-- weiteres Bsp. siehe Gruppierung

SELECT Artnr, SUM( Menge*vPreis ) AS UmSum
FROM Kauf
GROUP BY Artnr
```

Count:

```sql
COUNT (<spaltenname>) -- zählt belegte Werte einer Spalte. Einträge mit NULL werden nicht gezählt.
COUNT (*)   -- zählt auch NULL-Werte, also einfach alle Zeilen
DISTINCT    -- zählt keinen Wert mehrfach
```


```sql
SELECT COUNT(*)    -- Anzahl der Kunden
FROM Kunde

SELECT COUNT(DISTINCT Ort)  -- Anzahl der Orte ohne Dopplungen
FROM Kunde

SELECT DISTINCT Ort -- Zählt Orte ohne Dopplungen auf
FROM Kunde
```

4.2.3.5 ABFRAGE MIT BEREICHSGRENZEN UND WERTAUFZÄHLUNGEN

Bereich

```sql
BETWEEN x AND y
```

```sql
SELECT * FROM Kauf
WHERE vPreis BETWEEN 400 AND 1000
```

Aufzählung

```sql
IN (<liste>)
```

```sql
SELECT * FROM Kunde
WHERE Beruf IN('Maler', 'Schlosser', 'Tischler')
```

4.2.3.6 EINFACHE ABFRAGESCHACHTELUNG

(hinter dem WHERE steht eine Unterabfrage)

```sql
SELECT * FROM <tab_name>
WHERE <spaltenname> [ = | IN ] (  -- = benutzen, falls Unterabfrage
    einzelnen Wert zurück gibt, IN-Operator benutzen, falls eine
   Wertliste durch Unterabfrage erzeugt wird
  SELECT <sp_name>
  FROM <tab_name>
  [WHERE]
)
```

Abfrage von Name, Ort, GebDat des ältesten Dresdner Kunden:

```sql
SELECT Name, Ort, GebDat FROM Kunde
WHERE GebDat = (
  SELECT MIN(GebDat)
  FROM Kunde
  WHERE Ort='Dresden' -- genau ein Wert
) -- Achtung: hätten zwei älteste am gleichen Tag Geburtstag und
   einer würde nicht aus Dresden kommen, dann würde diese mit
   angegeben. Also:
AND Ort='Dresden'
```

Alle Kunden, die etwas gekauft haben (bzw. in Kauf eingetragen sind) aus Dresden:

```sql
SELECT * FROM Kauf
WHERE Kunr IN (
  SELECT KuNr FROM Kunde
  WHERE Ort='Dresden'
)
```

4.2.3.7 ABFRAGE ÜBER MEHRERE RELATIONEN

⇒ JOIN

Kap4.pdf
Folie 11

Bsp.: Anzeige der Artikeldaten und deren Käufe Mehrere Varianten:

1. WHERE:

```sql
SELECT  Artikel.Artnr, Bezeichnung, Menge, VPreis
FROM Artikel, Kauf
WHERE Artikel.Artnr = Kauf.Artnr
```

2. mit Alias:

```sql
SELECT a.Artnr, Bezeichnung, Menge, VPreis
FROM Artikel a, Kauf k
WHERE a.Artnr = k.Artnr
```

3. JOIN:

```sql
SELECT a.Artnr, Bezeichnung, Menge, VPreis
FROM Artikel a
JOIN Kauf k ON a.Artnr = k.Artnr
```

Zusätzlich: Name des Kunden anzeigen und nur Kunden aus Dresden:

1. WHERE:

```sql
SELECT A.Artnr, Bezeichnung, Menge, U.Kunr, Name
FROM Artikel A, Kauf K, Kunde U
WHERE A.Artnr = K.Artnr    -- implizite Bedingung(en)
  AND K.Kunr = U.Kunr      -- implizite Bedingung(en)
  AND Ort = 'Dresden'      -- explizite Bedingung(en)
```

2. JOIN:

```sql
SELECT A.Artnr, Bezeichnung, Menge, U.Kunr, Name
FROM Artikel A
JOIN Kauf K ON A.Artnr = K.Artnr  -- JOIN: implizite Bedingung(
   en)
JOIN Kunde U ON K.Kunr = U.Kunr   -- JOIN: implizite Bedingung(
   en)
WHERE Ort='Dresden'               -- WHERE: explizite Bedingung
   (en)
```


⇒ JOIN ist der ersten Variante zu bevorzugen, da die impliziten und expliziten Bedingungen klar und auf den ersten Blicks erkennbar sind.

4.2.3.8 WEITERE ABFRAGEMÖGLICHKEITEN

Kap4.pdf
Folie 12

- Equi-JOIN

```
SELECT Kunde.*, Kauf.*
FROM Kunde, Kauf
WHERE Kunde.Kunr = Kauf.Kunr
```

- Natural JOIN
  (man nimmt MANUELL die Spalten raus, die verglichen werden [also doppelt wären])

```
SELECT Kunde.*, Artnr, Menge, VPreis -- Kauf.*-Kunr oder
   vergleichbares nicht möglich! Es müssen alle Spalten manuell
    angegeben werden
FROM Kunde, Kauf
WHERE Kunde.Kunr = Kauf.Kunr

-- alternativ in JOIN-Schreibweise:
SELECT Kunde.*, Artnr, Menge, VPreis
FROM Kunde
JOIN Kauf ON Kunde.Kunr = Kauf.Kunr
```

- Kartesisches Produkt

```
SELECT *
FROM Kunde, Kauf
```

- Theta JOIN

```
SELECT *
FROM Kunde, Kauf
WHERE Kunde. Kunr != Kauf.Kunr
```

  Entspricht (Kartesisches Produkt) − (Equi-Join)

- Outer JOIN
  - Right
  - Left
  - Full

```
SELECT *
FROM Kunde K
FULL OUTER JOIN Kauf F  -- Schaut in beide Tabellen (Kauf und
   Kunde) und fügt die hinzu, die keinen Zusammenhang haben
-- bei LEFT OUTER JOIN: Schaut in die "linke" Tabelle (Kunde)
   und prüft, welche im Zusammenhang mit Kauf nicht auftauchen
   (sprich: welcher Kunde nichts gekauft hat)
```


```
-- bei RIGHT OUTER JOIN: Schaut in die "rechte" Tabelle (Kauf)
   (sprich: welcher Kauf keinen Kunden hat: nicht möglich)
  ON K.Kunr = F.Kunr
```

4.2.3.9 ABFRAGESCHACHTELUNGEN

Arten von Unterabfragen(UA):

- einfach/unkorreliert
  - UA wird einmal ausgeführt und in die Hauptabfrage (HA) eingesetzt
- abhängig/korreliert
  - UA wird für jeden Tupel der HA einmal ausgeführt
  - UA (innere Abfrage) bezieht sich auf eine (oder mehrere) Tabellen, die NUR in der FROM-Klausel der HA aufgeführt sind

Bsp.: unkorrelierte UA

- Alle Käufe in Dresden (Kunden).

```
SELECT *
FROM Kauf
WHERE Kunr IN (
  SELECT Kunr
  FROM Kunde
  WHERE Ort='Dresden'
)
```

- Anzeige Artnr, Bezeichnung und Einkaufspreis, sowie maximal erzielter VPreis je Artikel.

```
SELECT UA.Artnr, VPreisMax, Bezeichnung, EPreis
FROM (SELECT Artnr, MAX(VPReis) AS VPreisMax  -- Alias sehr
   wichtig, damit Spalte in HA wieder verwendet werden kann!
  FROM Kauf
  GROUP BY Artnr
) UA  -- Unterabfrage für Tabelle von max. VPreis
JOIN Artikel ON Artikel.Artnr = UA.Artnr
```

Diese Schachtelung kann beliebig weiter fortgesetzt werden.

- Anzeige aller Kunden, deren Umsatz (= Menge * VPreis) größer als deren Kreditlimit ist.

```
SELECT K.*
FROM (SELECT Kunr, SUM(Menge*VPreis) AS Umsatzsumme
  FROM Kauf
  GROUP BY Kunr
) UA
JOIN Kunde K ON UA.Kunr = K.Kunr
WHERE Umsatzsumme > Kredit
```


Bsp.: korrelierte UA

- Gleiches Beispiel, wie letztes unkorreliertes.

```
SELECT *          -- hier HA
FROM Kunde
WHERE Kredit < (
  SELECT SUM(Menge*VPreis)     -- ab hier UA
  FROM Kauf
  WHERE Kauf.Kunr = Kunde.Kunr
)
```

Ergebnis ist das gleiche wie in der unkorrelierten Abfrage.

Bsp.: korrelierte, unkorrelierte UA
Anzeige von Name und Ort der Kunden deren Umsatz (Umsatzsumme) größer als deren Kreditlimit ist.
unkorreliert:

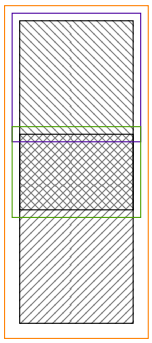
```
SELECT K.Kunr, K.Name, K.Ort
FROM (
  SELECT Kunr,
  SUM (Menge*vPreis) AS U-Summe
  FROM Kauf
  GROUP BY Kunr) UA
JOIN Kunde K
ON K.Kunr = UA.Kunr
WHERE U-Summe > Kredit
```

korreliert:

```
SELECT Kunr, Name, Ort
FROM Kunde
WHERE Kredit < (
  SELECT SUM (Menge * vPreis) AS U.Summe
  FROM Kauf -- Kunde hier nicht enthalten!
  WHERE Kauf.Kunr = Kunde.Kunr )  -- Kunden-Info aus übergeordneter
      Abfrage
```

4.2.3.10 MENGEN-OPERATIONEN

Kap4.pdf
Folie 13



UNION
EXCEPT
INTERSECT

Allgemein in SQL:

```
SELECT ...
FROM ...
[WHERE ...]
UNION | INTERSECT | EXCEPT
SELECT ...
FROM ...
[WHERE ...]
```

Bsp.:

```
SELECT VNum, VName
FROM Verkaeufer
WHERE Ort='Dresden'
UNION
SELECT KNum, KName
FROM Kunden
WHERE Ort='Dresden'
```

4.2.3.11 EXISTS / NOT EXISTS-OPERATOR

Kap4.pdf
Folie 14

Bsp.:

```
SELECT Name, Ort
FROM Kunde
WHERE EXISTS (
  SELECT *
  FROM Kunde
  WHERE Ort LIKE 'Dre%')
```

4.2.3.12 SOME/ANY, ALL

Bsp.:

```
SELECT *
FROM Kunde
WHERE Kunr = ANY (
  SELECT Kauf.Kunr
  FROM Kauf
  WHERE Kunde.Kunr = Kauf.Kunr) -- Kunde-Info wieder aus ü
     bergeordneter Abfrage
```


Bsp.: Ausgabe aller Daten des jüngsten Kunden

```
SELECT *
FROM Kunde
WHERE Gebdat >= ALL(
  SELECT Gebdat
  FROM Kunde)
```

4.2.3.13 HAVING-KLAUSEL (IM UNTERSCHEID ZU WHERE)

HAVING „filtert“ auf Gruppen: „HAVING ist das WHERE vom GROUP BY“.

Bsp.:

```
SELECT Ort, SUM (Kredit)
FROM Kunde
GROUP BY Ort
HAVING SUM(Kredit) > 2000
```

```
SELECT Ort, SUM (Kredit)
FROM Kunde
WHERE Ort LIKE 'D%' -- Filter auf Einzeldatensätzen
GROUP BY Ort
HAVING SUM(Kredit) > 2000 -- Filter auf den Gruppen
```

4.2.4 ANWEISUNG ZUR DATENMANIPULATION

4.2.4.1 EINFÜGEN VOR DATENSÄTZEN

```
INSERT INTO <tab_name> [ ( <spalte1>, <spalte2>, ... ) ]
  {[VALUES (wert1, wert2, ...)] |
  [SELECT <spalte1>, <spalte2>, ... FROM <tab_name1>]}
```

INSERT-Varianten:

- 1 Datensatz:
  VALUES
- Datensatzmenge:
  SELECT

Bsp.:

```
INSERT INTO Kunde
VALUES (110, 'Strobl', 'Frank', 'Dresden', '1977-2-27', 'm', '
   Student', 500)
-- oder auch (nicht in Microsoft SQL aber MySQL)
VALUES (110, 'Strobel', 'Frank',,,,, 500)
-- besser:
VALUES (110, 'Strobel', 'Frank', null, null, null, null, 500)
-- oder:
INSERT INTO Kunde (Kunr, Name, Vorname, Ort)
VALUES (110, 'Strobel', 'Frank', 'Dresden')
```


Datensatzmenge einfügen:

```
INSERT INTO Kunde(Kunr, Name, Vorname, Ort)
  SELECT Kunr, AName, AVorname, AOrt
  FROM Adresse
  WHERE AOrt LIKE 'Dre%'
```

4.2.4.2 ÄNDERN VON DATENSÄTZEN

```
UPDATE <tab_name> SET <spalte1> <ausdruck1> [, <spalte2> <ausdruck2
   >, ...]
```

Bsp.:

```
UPDATE Kunde SET Ort='Pirna'
WHERE Ort='Dresden' -- geht auch ohne WHERE, würde dann bloß in
   ALLEN Datensätze Ort auf Pirna setzen.

UPDATE Artikel SET EPreis = EPreis*1.1  -- 10 % Preiserhöhung
WHERE ... --korrelierte/unkorrelierte UA
```

4.2.4.3 LÖSCHEN VON DATENSÄTZEN

```
DELETE <tab_name> [WHERE <bedingung> ]
```

Bsp.:

```
DELETE FROM Kauf
WHERE KuNr IN (
  SELECT KuNr
  FROM Kunde
  WHERE Ort LIKE 'Dre%')

DELETE FROM Kunde -- Kann erst ausgeführt werden, wenn oberer
   Befehl ausgeführt werden. Wegen referenzieller Integrität!
WHERE Ort LIKE 'Dre%'
```

Befehlsübersicht Diese Befehle beziehen sich auf ...

- Metadaten (Objekt) (z.B. Tabellendefinition, View-Definition)
  - CREATE
  - ALTER
  - DROP
- Dateninhalt (der Tabelle/View)
  - INSERT
  - UPDATE
  - DELETE

4.3 QUERY BY EXAMPLE (QBE)

graphische Oberfläche
Bsp.: Access

4.4 DBMS-SPEZIFISCHE ERWEITERUNGEN VOM STANDARD-SQL (T-SQL)

4.4.1 STATEMENTS ZUR ABLAUFKONTROLLE

Kap4.pdf
Folie 15

4.4.1.1 IF

```
IF SELECT AVERAGE(vPreis) FROM Kauf < 40
  UPDATE Kauf SET vPreis=vPreis*1.1 -- für Produkte mit vPreis <40
     wird der Preis um 10% erhöht
```

4.4.1.2 IF EXISTS

```
IF EXISTS (SELECT COUNT(*) FROM Kunde WHERE Name='Adler')
  PRINT 'Adler vorhanden'
```

4.4.1.3 WHILE / WAITFOR

```
WHILE 2>1 -- Endlosschleife
  BEGIN
    WAITFOR DELAY '1:00:00' -- eine Stunde warten
    SELECT GETDATE()
    EXEC sp_who -- Liste aktiver Nutzer auf Server
  END
```

4.4.2 BENUTZUNG VON VARIABLEN

Kap4.pdf
Folie 16

	global	**lokal**
Deklaration:	System	im Programmcode
Bezeichnung:	@@name	@myname
Wertzuweisung:	System	select-Befehl (Konstante/Wert aus Relation übergeben)
Zugriff:	Lesend	Lesend + Schreibend

4.4.3 ANZEIGEN VON NACHRICHTEN

PRINT	⇒	ASCII-Zeichen
SELECT	⇒	Dateninhalte ausgeben
	⇒	Zuweisung von Variablen
RAISERROR	⇒	Fehlermeldungen ausgeben

Bsp.:

```
DECLARE @myName CHAR(10),
  @myKnr INT
SELECT @myKunr = 345  -- Wertzuweisung  braucht SELECT!!!
SELECT @myName = Name -- Wert aus Relation
  FROM Kunde
  WHERE Kunr = @myKunr
-- diese beiden SELECTS liefern keine Ergebnismenge: sind nur
   Wertzuweisungen
PRINT 'Kunr= ' + CONVERT(CHAR(10), @myKunr)
PRINT 'Name= ' + @myName  -- Achtung: 'test'+NULL gäbe insgesamt
   NULL, also keine Ausgabe, Lösung:
PRINT 'Name= ' + ISNULL(@myName, 'kein Name')
```

zu Bsp. 1: (aus Folie 4.16)

```
DECLARE @Anz int
SELECT @Anz = COUNT(*)
  FROM Kauf
  WHERE Artnr=5555
IF @Anz > 3
  PRINT 'Anzahl ist ' + CONVERT(CHAR(10), @Anz)
ELSE
  BEGIN
    PRINT 'einzelne Käufe von 5555:'
    SELECT K.Name, K.Vorname
      FROM Kunde K
      JOIN Kauf F ON K.Kunr = F.Kunr
      WHERE Artnr = 5555
  END
```

zu Bsp. 2:

```
WHILE (SELECT SUM(Kredit) FROM Kunde) < 100000
  BEGIN
    UPDATE Kunde
      SET Kredit = Kredit * 1.1
    IF (SELECT MAX(Kredit) FROM Kunde) > 50000
      BEGIN
        PRINT 'Abbruch > 50000'
        BREAK
      END
```

```
  END
```

zu Bsp. 3:

```
DECLARE @dKredit MONEY,
  @zKredit MONEY
SELECT @zKredit = 10000
SELECT @dKredit = AVG (Kredit)
  FROM Kunde
IF (SELECT Kredit FROM Kunde WHERE Kunr=123) < @dKredit
 UPDATE Kunde SET Kredit = Kredit+@zKredit
```

4.4.4 ARBEITEN MIT BATCH

Kap4.pdf
Folie 17

4.4.5 ARBEITEN MIT STORED PROCEDURE

Kap4.pdf
Folie 18

Kap4.pdf
Folie 19

zu Bsp. 1: (aus Folie 4.19)

```
CREATE PROCEDURE Erhoehe_Preis (@Prozent int = 5) AS
  UPDATE Kauf SET vPreis = vPreis + vPreis * @Prozent/100
```

zu Bsp. 2:

```
-- Aufruf bspw in einem Batch:
EXECCUTE Erhoehe_Preis 10 -- keine Klammern um Paramter im Aufruf!
```

zu Bsp. 3:

```
CREATE PROC LoescheKu (@Kunr int, @Zaehler int OUTPUT) AS -- CREATE
    PROC: Abkürzung für CREATE PROCEDURE
  SELECT @Zaehler = COUNT(*)
    FROM Kauf
    WHERE Kunr = @Kunr
  DELETE FROM Kauf WHERE Kunr = @Kunr
  DELETE FROM Kunde WHERE Kunr = @Kunr  -- Referentielle Integrität
     : Erst Kauf, dann Kunde löschen!

-- Aufruf bspw. im Batch:
```

```
DECLARE @Anz int
EXEC LoescheKu 123, @Anz OUTPUT -- OUTPUT muss hier auch noch mal
   angegeben werden!
PRINT 'Anzahl gelöschter Käufe: ' + CONVERT(CHAR(10), @Anz)
```

Kap4.pdf
Folie 19

Kap4.pdf
Folie 20

4.4.6 CURSOR

Kap4.pdf
Folie 21

Ablauf:

```
-- Cursor vereinbaren:
DECLARE <cname> FOR <SQL-Statement>
  SELECT <Spaltenliste>   -- muss gleiche Spalten haben wie die Var
     -Liste im fetch
  FROM <table>
-- Cursor öffnen
OPEN <cname>
-- Cursor nutzen
FETCH <cname> INTO <Var-Liste>
-- Cursor schließen
CLOSE <cname>
-- Cursor freigeben
DEALLOCATE <cname>
```

4.5 SEMANTISCHE DATENMODELLIERUNG – DAS ENTITY RELATIONSHIP MODELL

4.5.1 ÜBERBLICK ÜBER DATENMODELLE/SEMANTISCHE DATENMODELLE

Kap5.pdf
Folie 1, scale=0.8

Kap5.pdf
Folie 2, scale=0.8

4.5.2 ENTITIES UND RELATIONSHIPS (ERM-DIAGRAMM)

Kap5.pdf
Folie 3, scale=0.8

Kap5.pdf
Folie 4, scale=0.8

4.5.3 KARDINALITÄT UND KOMPLEXITÄT VON BEZIEHUNGEN

Kap5.pdf
Folie 5, scale=0.8

4.5.4 SEMANTISCHE BEZIEHUNGEN AM BEISPIEL

Kap5.pdf
Folie 6, scale=0.8

4.5.5 KONSTRUKTION SEMANTISCHER OBJEKTE

Kap5.pdf
Folie 7, scale=0.8

Kap5.pdf
Folie 8, scale=0.8

Kap5.pdf
Folie 9, scale=0.8

Kap5.pdf
Folie 10, scale=0.8

Kap5.pdf
Folie 11, scale=0.8

Kap5.pdf
Folie 12, scale=0.8

4.5.6 ABBILDUNG DES ENTITY-RELATIONSHIP-MODELLS AUF NORMALISIERTE RELATIONEN

Kap5.pdf
Folie 13, scale=0.8

Kap5.pdf
Folie 14, scale=0.8

Kap5.pdf
Folie 15, scale=0.8

Kap5.pdf
Folie 16, scale=0.8

Kap5.pdf
Folie 17, scale=0.8

Bsp. 1: Vertrieb

Kap5_BspVertrieb.pdf
Folie 1, scale=0.8

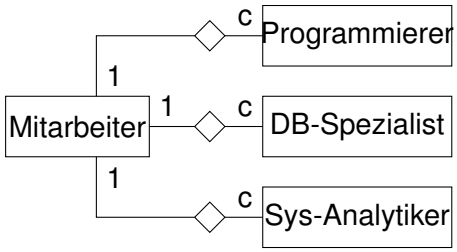
Bsp. 2: Rechnung

Kap5_BspRechnung.pdf
Folie 1, scale=0.5

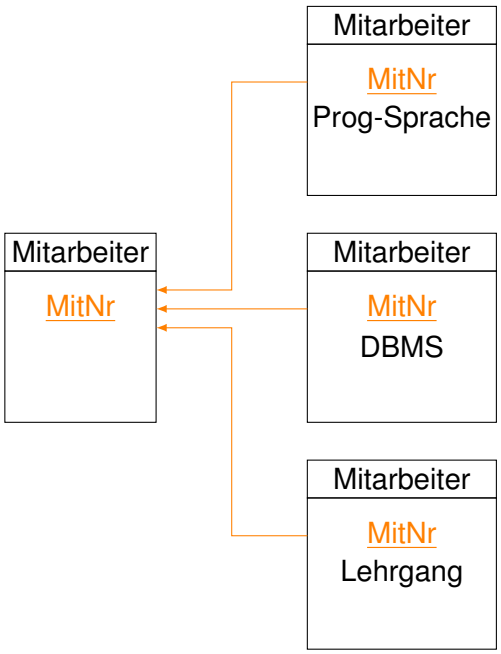
Bsp. 3: Projekt-Ableitung der Relationen

Kap5.pdf
Folie 18, scale=0.8

Mitarbeiter ist. . .



Wird zu:



(Vergleich: Praktikum)

PRÜFUNGSSCHWERPUNKTE

- Schlüssel (Aufbau, Funktion) (hierarchisch, nicht-hierarchisch usw.) → identifizieren, klassifizieren
- Daten-Kategorien (Stamm-Daten, Bewegungsdaten, . . .)
- Datenbanksysteme (Programmabhängige Datenorganisation → Probleme? Redundante Daten, usw.)
- (Grund-)Funktionen des DB-Betriebssystems (5) An Beispiel erläutern.
- DB-Architektur nach Ansi Sparc
- Normalisierung: Normalformen, Anomalien. Überführen in Normalform.
- Modellierung: (vgl. letzten Text-Beispiele) → Text in ERM und Relation überführen (Schrittweise: erst ERM, dann Relation)
- SQL: keine Erläuterungen sondern Beispiele an Tabellen: Abfragen ((left) join, in, . . . [Praktikum]), stored procedure (cursor nicht :-)), Tabellen erweitern/ändern
- Access: keine Abfragen