


Informatik Überblick

- 4 Bereiche:
 - Praktische Informatik
 - Technische Informatik
 - Angewandte Informatik
 - Theoretische Informatik

Kap.1 Daten

1.1 Daten, Nachrichten, Informationen

reale Welt \leftrightarrow Informatik

Liegenstand	Daten	Konto	\Rightarrow Daten sind die Gegenstände die verarbeitet werden und Algorithmen
Aktion	Algorithmus	Einführung	die Handlungen, die „verarbeiten“/verarbeitet werden.
Eigenschaft (kein Synonym)	Attribut	Kontostand	

Eine Nachricht ist eine endliche Menge an Zeichen

die von A nach B übertragen wird

$$A \xrightarrow{\text{Nachricht}} B$$

! Weine Änderungen können Informationen verfälschen
verschiedene Nachrichten können dieselbe Information enthalten

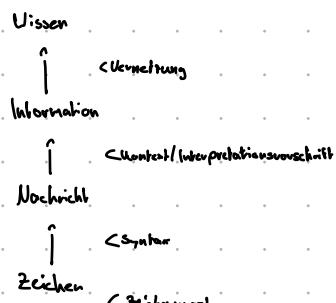
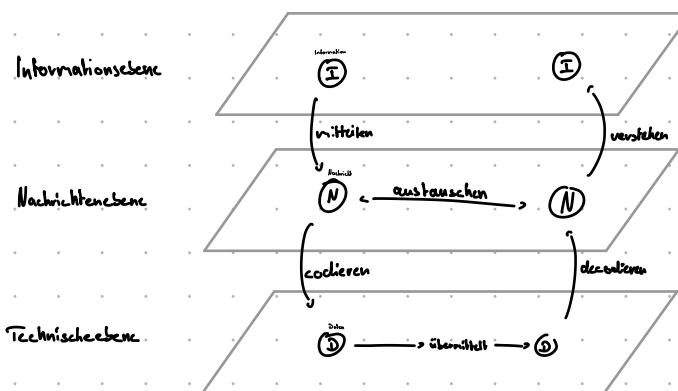
Nachricht wird Information, wenn A und B

dieselbe Interpretationsvorschau (I) haben

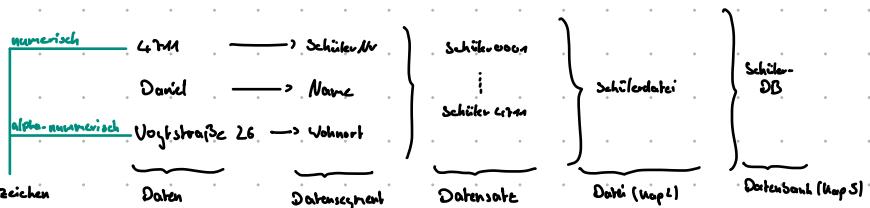
$$N \xrightarrow{\text{I}} I$$

$$H_2O \xrightarrow{\text{chemie}} \text{wasser}$$

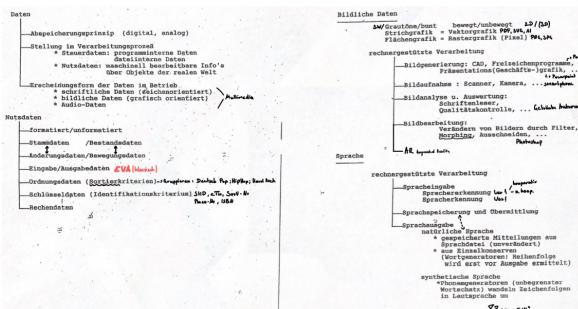
Curatisch:



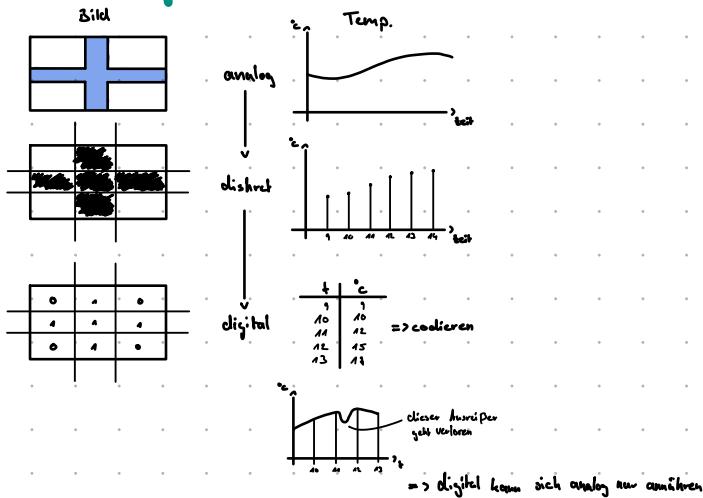
1.3 Byte zu Datei



1.2 Klassifikation von Daten



Daten Abspeicherungsprozess



formatiert/unformatiert

unformatiert:

Würde 1 ist 10cm breit, formatieren, 10cm hoch, 20cm lang.

formatiert:

Würde	h	b	l
1	10cm	10cm	20cm
2

Stammdaten, ...

Stammdaten
persönlichbezogene Daten
z.B. Konto

Bestandsdaten
z.B. Kontostand

Änderungsdaten
fallen ein bei
Veränderung von Stammdaten
z.B. Umtag, Heimat
→ Formular

Beregungsdaten
verbinden Bestandsdaten
z.B. Transaktion
Daten die sich auf dauerhaft ändernde
zweckd. Services und einer Substanz richten

relativ elenziert

relativ häufige Änderungen

sequentielle Dateiorganisation

- Standardoperationen:**
- 1) Lesen/Schreiben
 - 2) einfügen
 - 3) entfernen
 - 4) löschen/finden von Sätzen

Bearbeitungskriterien: - Aufwand: Lesen und Schreiben eines Satzes (wahlweise und sequentiell)

- Aufwand: einfügen und entfernen → (+ Aufwand Verarbeitung d. Datengröße)
- Speicherplatzausnutzung (→ Speicherplatz für Verwaltungsinformationen)

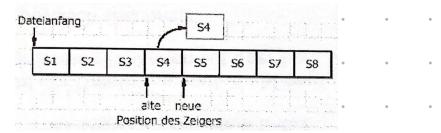
... mit sequentiellen Zugriff

sequentiell: zusammenhängende Folge der (Daten-) Sätze

... starr-sequentiell

↳ Reihenfolge der Sätze: direkt hintereinander (z.B. Magnetband)

Grundlegender Aufbau:



- einzige Operationstechnik für Dateien auf Magnettändern
- es gibt einen Zeiger, der durch bestimmte Operationen ex- oder implizit bewegt werden kann

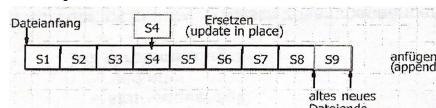
↳ Operationen (explizit):

- next: Zeiger um einen Satz. nach vorne
- previous: ... nach hinten
- rechts: Zeiger auf Dateiende setzen

Standard-Operationen:

- Zugriff (z.B. Lesen) berichtet sich auf Position des Trägers
- Datensatzsuche erfordert durchlaufen der Datei bis DS gefunden
- Schreiben i.d.R. nur durch Anlegen am Ende der Datei
- Ausnahme: Satz durch anderen Satz gleicher Länge ersetzen
→ dann auch zwischen innerhalb Datei möglich ("update in place")

Ersetzten (gleiche Größe):



Ersetzten (verschiedene Größe):

~~S1|S2|S3|S4|SS|S6~~

↓ Kopiere, lösse S4 aus

~~S1|S2|S3|~~S4~~|SS|S6~~

alles Band

~~S1|S2|S3|X|SS|S6~~

↓ Kopiere, lösse S4 aus

~~S1|S2|S3|SS|S6~~

neues Band

... verketten

↳ nur logisch sequentiell (Zeiger auf den Nachfolger)

Kettendatensatz-Aufbau:

- Jeder Datensatz hat Zeiger auf Nachfolger.



- Physische Abspeicherung der Sätze nicht zwangsläufig sequentiell

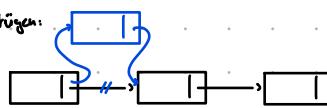
Nutzdaten: faktische Informationen

Speicherdaten: verweisen auf den nächsten Datensatz

Standard-Operationen:

- Datensatzsuche durch Durchlaufen der Datei bis DS gefunden.

- Einfügen:



- Löschen:



- Ändern: - einfache Änderung z.B. S1 → S2

- oder Kombi aus einfügen und löschen

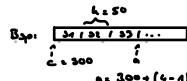
... mit direktem Zugriff

Voraussetzungen:

- Sätze physisch sequentiell gespeichert
- Satzlänge fix
- Speichermedium erlaubt direkten Zugriff

direkter Zugriff: inter-Satz wird gesucht

$$a \in \{1 \dots n\} \cdot k \quad \text{mit } a \text{ ist d. num. Satz} \\ \text{z. Anfangsadresse} \\ k \text{ = Satzlänge}$$

Bsp.: 

$$a = 300 \quad (a-1) * k = 299 * 500 = 149500 \\ a * k = 300 * 500 = 150000$$

Bewertungskriterien:

Variante	Starr-sequentiell	Verkettet	Direktem Zugriff
Organisation	Sequentiell	Logisch sequentiell sortiert	Sequentiell sortiert (nach Satznr.)
Seq. Lesen	Sehr gut	Gut	Gut
Seq. Schreiben	Sehr gut	Gut	Gut
Wahlfreies Lesen	Sehr schlecht	Schlecht, außer Adresse ist bekannt	Sehr gut, wenn Zugriff über SatzNr
Wahlfreies Schreiben	Sehr schlecht	Sehr schlecht	Sehr gut
Satz einfügen	Gut (hinten), ansonsten Sehr schlecht	Gut	Schlecht, außer hinten einfügen ↳ o.w.
Satz entfernen	Sehr schlecht	Gut	Logisch: gut (Löschermarkierung →) Physisch (Reorg.): schlecht
Finden	Schlecht	Schlecht	Schlecht, außer Satznummer
Speicherplatznutzung	Sehr gut	Sehr gut	Gut
Typische Anwendungen	Sicherung od. Vorspeicherung für seq. Verarbeitung (z.B. Auftragseingang)	„Temporäre“ Datei, Zwischendaten (vor allem, wenn einsortiert werden muss)	Dateien ohne Zu- und Abgänge Beachte: Voraussetzungen

Standard-Operationen:

- Lesen/Schreiben durch direkten oder sequentiellen Zugriff (a.a.)
- beim Löschen entsteht (logisch) ein „Loch“, Datengröße unverändert
- Änderungen möglich (satu. Satzlänge und satzlin. gleichbleibt)

Bäume

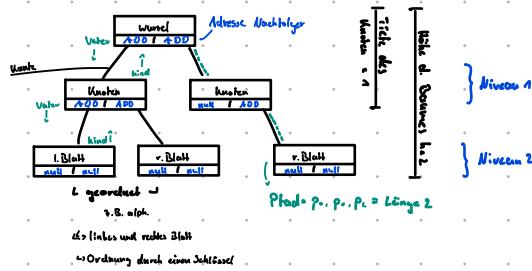
Anwendungsbereich:

Relevant für Programmierung:

- Datenstrukturen und Algorithmen werden nicht direkt von erfinden sondern aus Bibliotheken ausgewählt.
- grundlegendes Verhältnis wichtig: damit man weiß, wonach man suchen muss

Binärbaum:

- besteht aus Knoten
- ein Knoten kann maximal 2 Nachfolger haben
- " " " höchstens 1 Vorgänger haben
- erster Knoten (keine Vorgänger): Wurzel / root
- letzter Knoten (keine Nachfolger): Blatt / leaf
- von jedem Knoten ein Pfad zur Wurzel



Zusammenhang:

- Listen gut geeignet um dynamisch sortierte Datenbestände aufzubauen
- > einsortieren einfach
- > Zugriff auf bestimmtes Element aufwendig
- > Liste nicht gut geeignet für effiziente aufinden sortierter Elemente

Bäume:

- für solche Probleme: Bäume
- verzweigte, nichtlineare, Listenstrukturen
- wichtige Unterguppe: Binärbäume

Pfad: - Folge von Knoten $P_0 \dots P_n$ eines Baums

(\Rightarrow Bedeutung: P_k ist Kind von P_i)

- Pfad der p_0 mit p_k verbindet, Länge k

Höhe: - Maximaler Abstand aller Blätter zur Wurzel

Tiefe eines Knotens:

- ist sein Abstand zur Wurzel. \Rightarrow Anzahl der Knoten auf Pfad zur Wurzel

Niveau: - Knoten gleicher Tiefe. \Rightarrow Knoten, Tiefe i auf Niveau i

Vollständig: - wenn auf jedem Niveau die maximale Knotenzahl hat

- Blätter dieselbe Tiefe haben.

rechteckige Struktur: - Binärbaum besteht aus Wurzel, linker Teilbaum, rechter Teilbaum

\hookrightarrow Teilbaum hat „Wurzel“, linker ...

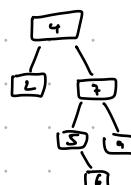
Belebung: - geeignete Struktur zur Speicherung von Schlüsseln

- jedes Datenelement genau ein Schlüssel
- Schlüssel (paarweise) verschieden (Kollisionsproblem)
- werden so gespeichert, dass sie schnell und effizient gefunden werden können

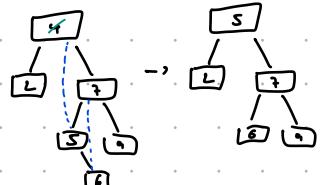
Basis-/Wörterbuchoperationen:

- Suchen (nach Schlüssel)
- Einfügen eines Knotens mit geg. Schlüssel \rightarrow bei geordneten Bäumen anhand Schlüssel
- Entfernen
- Entfernen Wurzknoten: entferntes ersetzen mit
 - größtem aus linkem Teilbaum oder
 - kleinstem aus rechtem Teilbaum

Sop: 4, ?, 2, ?, 5, 6



Löschen von '4':



Spezialisierungen:

balancierte Binärbäume: Ziel: Verbindung der Entartung zur Liste

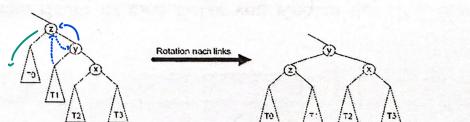
AVL-Bäume: - Adelson-Velsky & Landis

- Degenerieren durch Forderung an Höhendifferenz der beiden Teilstämme eines Knotens verhindert

-> höhenbalancierte Binärbäume

Restrukturierung nach dem Einfügen/Entfernen eines Knotens:

Z: Erster Knoten auf dem Weg vom eingefügten / entfernten Knoten zur Wurzel, bei dem die Balance nicht mehr stimmt
y: Sohnknoten von Z mit größter Höhe
x: Sohnknoten von Z mit kleinerer Höhe



Breiter-Bäume: - Alle Blätter selben Abstand zur Wurzel

- Bedingen für Verzweigungsgrad

gewichtsbalancierte Bäume:

- für jeden Knoten müssen die Gewichte der Teilstämme
im einem bestimmten Verhältnis zueinander stehen

Allgemeine Bäume: - Knoten haben endliche, begrenzte Anzahl von Kindern (nicht auf 2 begrenzt)

- Ordnung: maximale Anzahl von Kindern

-> d=2: Binärbäume

-> d>2: Vielwegbäume

B-Bäume: - Anzahl der Kinder eines Knoten zwischen fester Untergrenze und Obergrenze

- Sicher: Daten vollständig im Arbeitsspeicher verwaltet

- ist dies nicht der Fall: Verwaltung des Schlund in „Indextabelle“

mit Verweis wo die zugehörigen Daten gespeichert sind

- Indextabelle zu groß für Arbeitsspeicher: Teile des Baumes ausserhalb

-> Knoten kann hierarchisch als B-Baum organisiert werden

-> Knoten in einzelne Seiten

↳ Seiten zusammenhängend, extern

↳ 1 Seite passt in Arbeitsspeicher

↳ 1 Seite enthält Indexteil + Information

welche Seite zu laden ist wenn Schlüssel nicht in A3

↳ Knoten < 1 Seite

↳ Knoten enthält Schlüssel und tragen auf weitere Knoten
merkmale

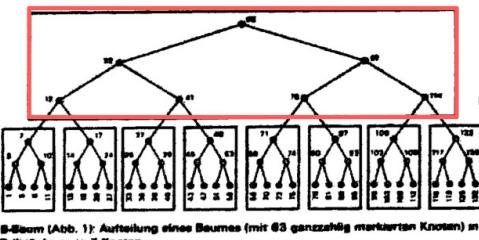
B-Baum (Balanciert)

Bauwurkungsbereich

- Datenstruktur zur Verwaltung oft sehr großer Datenmengen auf Massenspeichern, z.B. Festplatten
- Zeiger eines Knotens zeigen auf Plattspeicheradressen statt Hauptspeicheradressen
+ geringere Suchzeit

Aufteilung in einen B-Baum

Bei der grafischen Darstellung von B-Bäumen fasst man alle Knoten eines Teilstücks in einem Knoten zusammen.



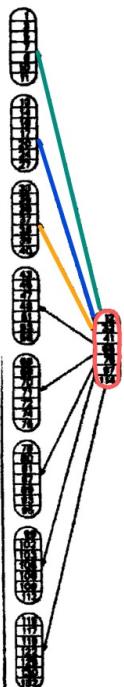
Die Ordnung des rechts dargestellten B-Baumes ist $m=4$.

Überprüfen Sie folgende allg. Regeln:

- (1) Jeder Knoten hat höchstens $2m$ Schlüssel
 $\leq 2 \cdot m$ hier $2 \cdot 4 = 8$
- (2) Jeder Knoten (mit Ausnahme der Wurzel) enthält mindestens m Schlüssel
 $\geq m$
- (3) Jeder Knoten mit k Schlüsseln hat genau $k+1$ Söhne oder keinen Sohn.

Was besagen die 3 obersten Kanten, die vom Wurzelknoten abgehen?

- ... kleiner als 1. Schlüssel der Wurzel
- ... 1. Schlüssel < Schlüssel d. Kindes < 2. Schlüssel
- ... 2. Schlüssel < Schlüssel d. Kindes < 3. Schlüssel
- usw. bis > letzter Schlüssel



Löschen (Komplex → Grundprinzip)

Ausgangspunkt Abb. 5

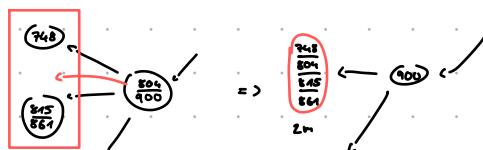
713 wird gelöscht?

R2 >= ? „Unterlaut“

Alt: 680 wird gelöscht
↳ Nächst größeren Schlüssel nach oben / oder nächst kleineren
713 656 („Vergleichswert“)



776 löschen?



Rekurrenz

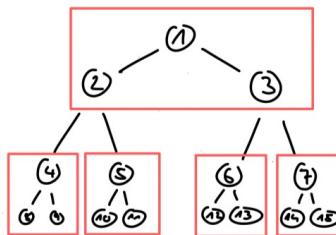
Funktion ist/reicht rekursiv, wenn sie sich selbst wieder aufruft oder eine andere Funktion aufruft die wiederum sie aufruft

Wichtig: Abbruchbedingung, sonst endlos

Versionen der Attribute werden auf einen Stack gelegt, dieser bildet Ebenen der Verpackung ab

Komplett	Speicherplatz und Zeit nicht weniger Endlosschleusengetaktet
einfach zu lesen	
Eleganter	
Eignen sich für Baumstrukturen ☞ rekursiv	

Baum



Anzahl Knoten	Zugriffe
1	1
3	2
7	3
15	4

Fazit bei 10 Mio:

Zugriffe = $\log_2(10 \text{ Mio}) \Rightarrow 20$ Zugriffe à 0,1 sek

= 2 sec (zu langsam) ⇒ Plattspeicherminimieren

Teilstücke d. Baumes in Hauptspeicher übertragen und dort suchen (vol)

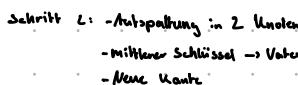
Pramiss: Blöcke von 255 bis 1023 Knoten

$\log_2(10 \text{ Mio}) \Rightarrow$ bei 10 Mio ca. 2,5 Zugriffe
(Die Suchzeit im HS vernachlässigbar vgl. Zugriffsschleife)

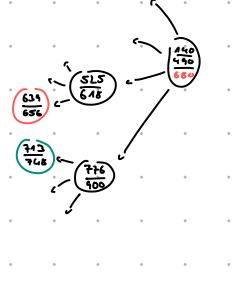
HEBENBEDINGUNG: Funktioniert nur, wenn balanciert

entfernen + suchen s.S. 17

Überlauf bei Entfernen



Hinweis: Theoretisch auch Wurzelüberlaut möglich
→ neue Wurzel (vgl. R2 Ausnahme)
„Wurzel Wurzel“



Versionen der Attribute werden auf einen Stack gelegt,

dieser bildet Ebenen der Verpackung ab

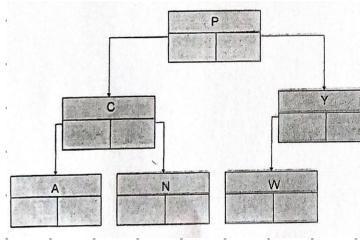
Traversierung

Durchlaufen der Knoten eines Baumes

Hauptreihenfolge (preorder): Wurzel \rightarrow linker Teilbaum \rightarrow rechter Teilbaum

Nebenreihenfolge (postorder): linker Teilbaum \rightarrow rechter Teilbaum \rightarrow Wurzel

Symmetrische Reihenfolge (inorder): linker Teilbaum \rightarrow Wurzel \rightarrow rechter Teilbaum



Durchlaufen des Baumes rekursiv:

pre-order:	P; C; A; N; Y; W	M Hauptreihenfolge
post-order:	A; N; C; W; Y; P	N Nebenreihenfolge
inorder:	A; C; N; P; W; Y	S Symmetrisch

Zweck

Problemlösungskonzept: Gerüst für spezifische Aufgaben wie z.B.

- Drucken, Markieren, Kopieren aller in einem binären Suchbaum
- vorhandene Knoten in einer bestimmten Reihenfolge
- Berechnen Summe, Durchschnitt, Anzahl
- Höhe des Baumes und Tiefe eines Knotens

Stack, Queue

Stack

Last In First Out - Prinzip

- Verbstamm der Speicherung, bei dem zuletzt gespeicherte
- Elemente zuerst wieder entnommen werden
- Push(E): legt drauf / fügt E. an hinten
- Pop(): entfernt oberstes E.
- Peek(): liest oberstes E., ohne entfernen

Queue

First In First Out - Prinzip

- Verbstamm der Speicherung, bei dem zuerst gespeicherte
- Elemente zuerst wieder entnommen werden
- enqueue(a): stellt an / fügt E. an hinten an
- dequeue(): entfernt erstes E.
- front(): liest erstes E., ohne entfernen

Anwendungssicht

BLU: Bestandsbereitung

FIFO in Herstellung, „ältere“ waren sollen zuerst verbraucht werden

LIFO bei Schüttgätern in Mühlen

Inf: Daten ablegen und abrufen, Vorfeschlange (Druckqueue)

Datenstrukturen

Array

Aneinanderreihung von gleichartigen Elementen

Zugriff auf Komponenten mithilfe eines Index

0	1	2	3

Rest siehe Java

Strukturen / struct

Verbind unterschiedlicher Elemente

Schüler Nr.
Vorname
Nachname
Geb. Datum

häufig mehrere gleichartige Strukturen

Zsp: struct adresse { einzelne Elemente }
↳ Straße, Nr., Ort, PLZ

Bubble Sort $O(n^2)$

Algorithmus

1. nimmt erstes Zahlenpaar
 2. links > rechts? \rightarrow tauschen
 3. nächstes Zahlenpaar; Wenn 2 Zahlen \rightarrow nicht am Ende
 - \hookrightarrow nur eine Zahl? \rightarrow Ende
- Wenn min. 1x getauscht wurde sonst \rightarrow alle an richtiger Position

Optimierung: Wenn nur das letzte Zahlenpaar getauscht wurde, kann davon ausgegangen werden, dass alle anderen Zahlen bereits an richtigen Platz stehen.

Jedes Mal den einen Eintrag mehr von hinten „entfernen“/nicht betrachten
 $\text{for } i \leftarrow 1 \text{ to } \text{Arr.length} - 1 \text{ do}$
 $\quad \text{for } j \leftarrow i + 1 \text{ to } \text{Arr.length} - 1 \text{ do}$

1 2 3 4 5 6 \rightarrow Täusche

Quick-Sort $O(n \log n)$

Algorithmus

```
quicksort(links; rechts)
  if links < rechts;
    gewebe z.B. Mitte / „Pivot“
    if i < j { Wenn i links von j ist
      while i <= j { Wiederhole bis i rechts neben j steht
        while (Arr[i] < pivot) { bis Arr[i] größer/gleich
          i++; i < rechts wie gewebe ist
          wie gewebe ist
        }
        while (Arr[j] > pivot) { bis Arr[j] kleiner/gleich
          j--; j > rechts
        }
        if (i == j) { Wenn i links neben j oder auf j steht
          tausche (Arr[i]; Arr[j]) tausche position i mit position j
          i++;
          j--;
        }
      }
      quicksort(links; i) linker Teil
      quicksort(i; rechts) rechter Teil
    }
  }
```

$$qs[0, 5] \quad p = \text{Arr}[[(i+j)/2]] \text{ (int)}$$

i p i

1 2 3 4 5 6

qs(0, 5)
 qs(0, 1)
 qs(0, -1) \rightarrow Ende
 qs(1, 1) \rightarrow Ende

qs(1, 5)
 qs(1, 3)
 qs(1, 1) \rightarrow Ende
 qs(3, 3) \rightarrow Ende

qs(3, 5)
 qs(3, 4)
 qs(3, 2) \rightarrow Ende
 qs(4, 4) \rightarrow Ende

qs(5, 5) \rightarrow Ende

Komplexitätstheorie

untersucht die Komplexität und Effizienz von Algorithmen in Abhängigkeit von Umfang der Eingabedaten
 \rightarrow Aufwand an Zeit / Speicherplatz

unabhängig von Sprache + verwendetem System

Nutzen

- Schätzung ob Probleme mit vorhandenen Ressourcen lösbar sind
- Beurteilung von Zeit- und Ressourceneffizienz
- Analyse gibt Hinweise auf mögliche Optimierung

$O(f(n))$: Ordnung von f(n) [Sekundär dominante Größe]

ist die asymptotische Zeitschärfkomplexität an

(\rightarrow Umfang von n geht gegen 0)

$$\text{Sap: } f(n) = 2^n - 10 \cdot n$$

$$O(f(n)) = 2^n$$

weglassen: - alle konstanten Terme ($\sim n^0$)

- alle schwachen Terme ($\sim n^1$)



Bsp. für f(n): Minimalsuche

int min = array[0]	konstant (c)
von i=1 bis n-1	
wenn array[i] < min	
min = array[i]	

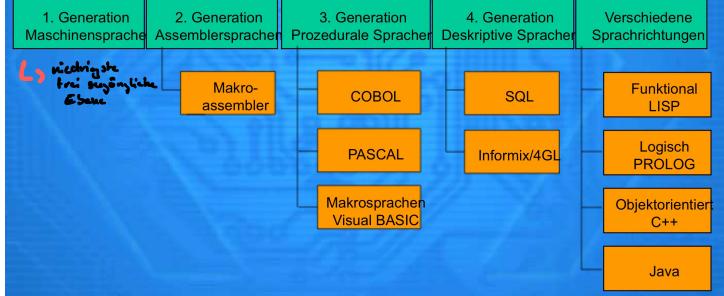
$$f_{\text{Minimalsuche}}(n) = n - 1 + c$$

$$O(f_{\text{Minimalsuche}}(n)) = n \rightarrow \text{linearer Aufwand}$$

Algorithmus	Laufzeit	1 s	1 min	1 h	
A ₁	$O(n)$	1000	$6 \cdot 10^6$	$3.6 \cdot 10^5$	sequentielle Suche
A ₂	$O(n \log n)$	128	4615	$2 \cdot 10^5$	quicksort
A ₃	$O(n^2)$	31	244	1897	bubblesort
A ₄	$O(n^3)$	10	39	153	
A ₅	$O(2^n)$	9	15	21	Häufig von kanni

Metasprachen

Ges. PSprachen



Sprachkonstrukte Metasprache

- > Terminalsymbol
 - Alphabet: nicht leerer, geordneter, endlicher Zeichenraum
 - Menge von Schlüsselsymbolen/Schlüsselwörtern: z.B. VAR, END, WHILE, ...
- > Metadeinitionen: Metavariablen und Metaausdruck
- > Metavariablen (Nonterminalsymbol): Klassen von Zeichenketten
- > Metaausdrücke: Verknüpfung von Tsymbolen und Metavariablen

ZDNF

Definition eines Bezeichners:

```

<Bezeichner> ::= <Buchstabe> { <Buchstabe> | <Ziffer> }

<Buchstabe> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|
                  P|Q|R|S|T|U|V|W|X|Y|Z|
                  a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|
                  r|s|t|u|v|w|x|y|z

<Ziffer> ::= 1|2|3|4|5|6|7|8|9|0
  
```

Metavariablen ::= Metaausdruck (Metavariable + Terminalsymbol + Det. Symbols)

Definition einer Variablen:

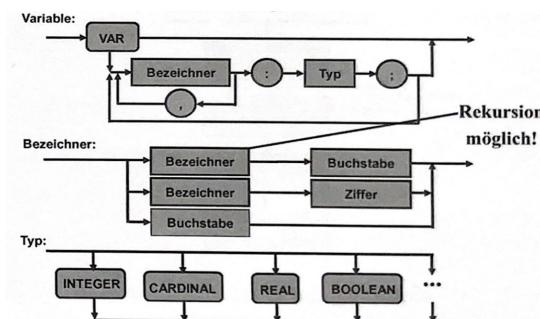
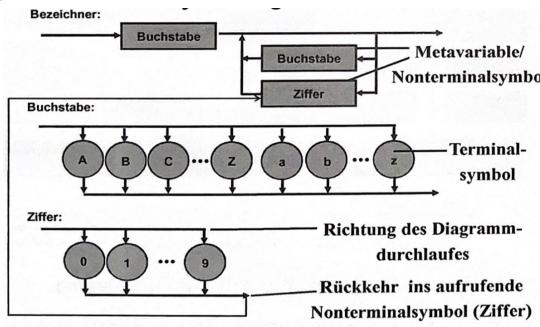
```

<Variable> ::= VAR {<Deklaration>}

<Deklaration> ::= <Bezeichner> { , <Bezeichner> } : <Typ>

<Typ> ::= INTEGER | CARDINAL | REAL | BOOLEAN
  
```

Syntaktische Strukturen



CODASYL-Normalisierung

bezeichner	variable
buchstabe	$\{ \text{buchstabe} \}$
	$\{ \text{ziffer} \}$
	VAR [bezeichner [, bezeichner] ... : typ ;] ...
buchstabe	$\{ A, B, C, \dots, Z, a, b, \dots, z \}$
	$\{ 1, 2, 3, \dots, 9 \}$
ziffer	$\{ 0 \}$
	$\{ \text{INTEGER}, \text{CARDINAL}, \text{REAL}, \text{BOOLEAN} \}$

Zsp:

- > Alternative:


```
<Ziffer> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```
- > Sequenz:


```
<Zweistellige Zahl> ::= <Ziffer> <Ziffer>
<Zehn bis Neunzehn> ::= 1 <Ziffer>
<Zweiundvierzig> ::= 4 2
```
- > positive Zahl:


```
<Positive Zahl> ::= <Ziffer> | <Ziffer> <Positive Zahl>
```

BNF Erkennung + Regeln

$\hookrightarrow \dots$: Zuordnung

\dots : Nichtterminal

| : Auswahl

[...]: Optional

{...}*: Wiederholung 0/1 bis n

	BNF	Syntaxdiagramm
Grundsymbol Terminalesymbol	A	
Nichtterminales Symbol	(a)	
Symbolkette	(a) A B (b) ...	
Produktionsregel	(x) ::= (x) A	
Alternativen	(a1) (a2) (a3)	
Optionen	(x) ::= [(opt)] (x)	
„Leeres“ Produktion als Alternative	(x) ::= [()	
Wiederholung	(x)* (x)? B	
Wiederholung als Option	(x)* (x)? B	

5. Datenbanken

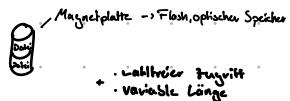
5.1. Einführung

Von Datei zur Datenbank

a) Beginn



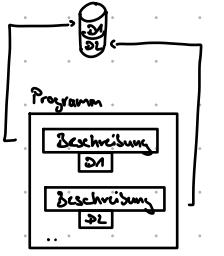
b) gestern/heute



c) heute/morgen? Optische Speicher, Nanotechnik, DNA?!

In Memory-Technik (z.B. SAP, IBM, HANA)

Bsp: gestern/heute



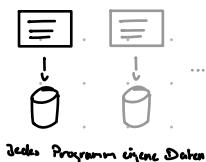
Je mehr Dateien,
=> desto komplizierter
das Programm

↓
Folge?
↓
Aus vielen kleinen Dateien
eine große (vgl. Versicherungsbeispiel)

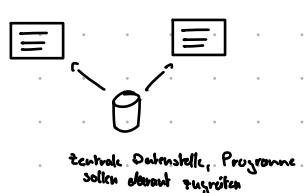
Fazit: Zugriff vereinfacht sich,
aber Redundanzen
↓
Anomalien

=> Wandel in der Denkweise

Alt: Funktionsorientierter Ansatz



Neu: Datenorientierter Ansatz



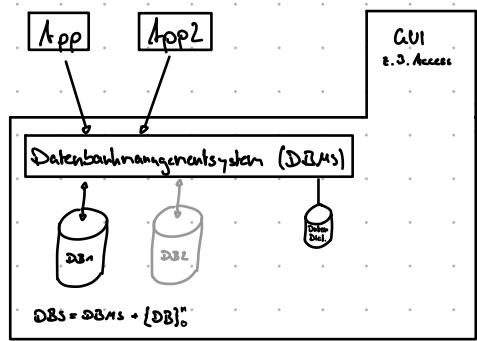
Fazit: Redundanz + Inkonsistenz



Ansatz für die Lösung:

DB

Die Überlegungen von Archibat und Codd (S. 72-78) mündeten in folgendes Konzept für Datenbanksysteme (DBS):



Kapitel Einführung in die Welt der Datenbanken**Weshalb werden Datenbanken eingesetzt?****Probleme konventioneller Dateiverarbeitung:**

1. Es gibt keinen logischen **Datenschutz** (Datengeheimnis)

d.h. es fehlt der Schutz vor unberechtigtem Zugriff.

Bei einer Realisierung mit konventioneller Dateiverarbeitung hat jede Anwendung Zugriff zum ganzen Satz! Es wird nicht unterschieden, ob eine Anwendung nur Lese- oder auch Schreibberechtigung hat!

Dabei müssen bei der Realisierung unbedingt die gesetzlichen Vorschriften erfüllt werden (z.B. Gehaltsdaten)

Außerdem gibt es innerbetriebliche Regeln, die auch eingehalten werden sollten (Wer darf Überweisung anlegen?).

2. Problem **Datensicherung**

muss allgemein für den Betrieb bzw. für die betroffene EDV-Anlage gelöst werden.

(Datensicherung, d.h. Vorsorge, dass Daten nicht durch Brand, Überschreiben, Hardwarefehler oder derartiges verloren gehen)

3. **Parallelbetrieb mehrerer Programme**, die auf die gleiche Datei zugreifen, ist selten möglich.

Zum Beispiel müssen für verschiedene Anwendungen die Daten nach anderen Kriterien sortiert sein:

Gehaltsabrechnung	:	Personalnummer
Adressen	:	Postleitzahl
Auswertung	:	Abteilung

4. Daten werden mehrfach abgespeichert (**Redundanz**)

Die Redundanz wird nicht kontrolliert, d.h. die mehrfach abgespeicherten Daten „wissen“ nichts voneinander.

=> mehr Speicherplatzbedarf und
 Probleme beim Löschen oder Ändern der Daten
 (inkonsistente Daten sind oft die Folge)

Begriff **inkonsistent**: d.h. redundant gespeicherte Daten haben unterschiedliche Werte z.B. in einer mehrfach abgespeicherten Adresse sind unterschiedliche Postleitzahlen für denselben Ort eingetragen.

5. mangelnde Aktualität der Daten

aus 4. kann folgen, dass die Änderung von mehrfach abgespeicherten Daten erst zeitlich stark versetzt in allen Dateien nachgezogen wird (Durch Konsolidierung -> Aufwand).

6. Programme sind datenabhängig (dies ist eine unerwünschte Eigenschaft!)

Datenabhängigkeit ist ein hohes Maß der Bindung zwischen Programm und Daten zur Entwurfszeit.

~~(z.B. bei Anwendung der Jackson-Methode)~~

In einem konventionellem DV-System werden die Datenbestände vom Programmierer schon zur Entwurfszeit an das Programm gebunden, d.h. Aufbau und Abspeicherung und Zugriff auf die Daten werden in der Kontrolllogik und in den programmierten Teifunktionen berücksichtigt.

Das Resultat sind anwendungsbezogene Datenstrukturen.

7. Datenintegrität muss von jedem Programm selbst verwaltet werden. Daten sind integer, wenn sie der Realität entsprechen.

Man unterscheidet:

a) die lokale Integrität

d.h. die abgespeicherten Daten einer Tabelle entsprechen der Realität
Bsp.: ein Feldelement enthält ein Datum das Datum muss einen gültigen Wert besitzen, sonst ist die lokale Integrität verletzt

b) die referentielle Integrität

d.h. Daten, die an unterschiedlichen Stellen (in verschiedenen Tabellen) abgespeichert sind, aber miteinander in Beziehung stehen, müssen zusammen der Realität entsprechen.

Bsp.: abgespeichert seien Personen und ihre Kinder in zwei verschiedenen Tabellen. Kinder ohne Eltern sollen nicht gespeichert werden. Wird jetzt eine Person gelöscht, so sollen auch ihre Kinder gelöscht werden; falls die Kinder in der Tabelle verbleiben, so ist die referentielle Integrität verletzt

8. Abfragen sind nur mit Programmen möglich Ad-hoc-Abfragen sind nicht möglich

ANFORDERUNGEN AN DATENBANKEN

(sie leiten sich aus den oben genannten Problemen der konventionellen Dateiverarbeitung ab)

Die zwei wichtigsten:

- a) Datenunabhängigkeit
- b) Vermeidung von Redundanz

zu a) Datenunabhängigkeit

Datenunabhängigkeit ist die Isolierung der Daten von den Programmen, ODER das geringe Maß der Bindung zwischen Programm und Daten zur Entwurfszeit.

Die erste Stufe ist die Unabhängigkeit vom ZUGRIFF auf die Daten
Der Benutzer hat keine Kenntnisse, wie die Daten gefunden werden, wie sie modifiziert werden, wie sie eingefügt werden oder wie sie gelöscht werden.
Die Datenübergabe erfolgt über einen Puffer, in dem die gewünschten Daten übergeben werden. Jede Änderung der Zugriffspfade lässt die Programme unberührt.
Die Datenstruktur muss dem Benutzer bekannt sein.

zu b) Vermeidung von Redundanz

Daten werden nicht speziell für ein Programm abgespeichert, sondern stehen mehreren Programmen oder auch für interaktive Abfragen zur Verfügung.

D.h. zusätzlich zum Gedanken a) Datenunabhängigkeit kommt (logisch daraus folgend) die Idee, Daten EINMAL ZU SPEICHERN und MEHRFACH ZU VERWENDEN. Damit erreicht man eine erhebliche Reduzierung des Erfassungs-, Speicherungs- und **Änderungsaufwandes**.

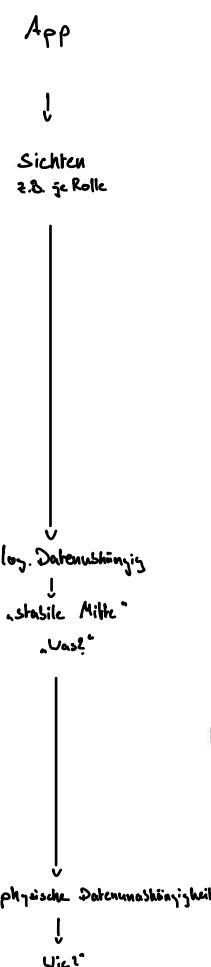
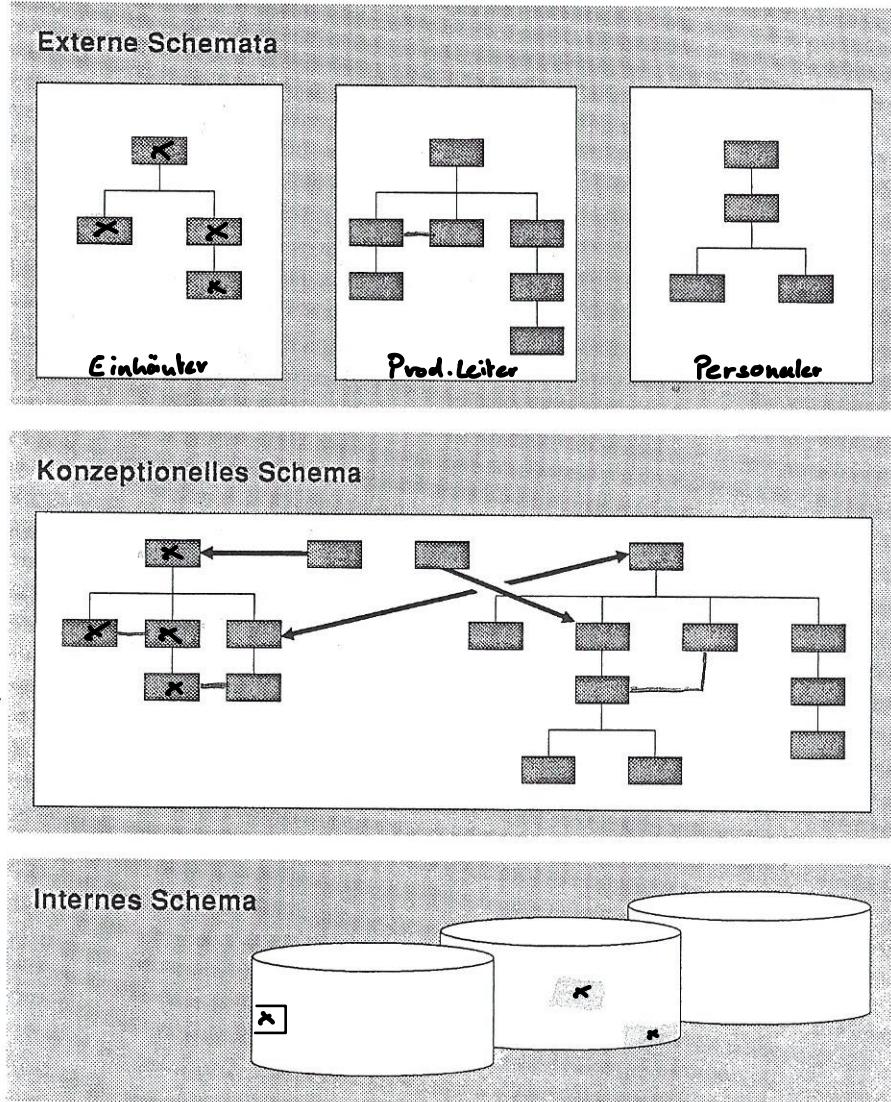
Weitere Anforderungen an eine ideale Datenbank

- Verbesserung der Datenkompatibilität und des Datenaustauschs mit 'fremden' Systemen
- problemlose Verarbeitung von 'Kann'- und 'Muss'-Feldern
- Zugriffsschutz kann variabel definiert werden (einzelne Datenelemente können geschützt werden, ebenso ganze Datengruppen; es wird zwischen Leseschutz und Schreibschutz unterschieden)
- Trennung zwischen der logischen und der physikalischen Datenstruktur
- unnötige Redundanz (Mehrfachspeicherung) wird vermieden (Schlüsselredundanz wird notwendig (Fremdschlüssel -> Primärschlüssel))
- die Datenintegrität wird von der Datenbank mitüberwacht
- Daten sind jederzeit aktuell und verfügbar
- die Benutzung der Datenbank ist einfach zu erlernen, jeder berechtigte Benutzer kann auf die Daten zugreifen
- die Datenbank hat ein Konzept für die Datensicherung
- es sind jederzeit spontane Fragen an die Datenbank möglich (ohne aufwendige Programmierung)
- auf die Daten kann variabel zugegriffen werden (keine Probleme mit dem Schlüssel, keine Sortierungen)

Architektur, Komponenten eines Datenbanksystems Schemata

Drei-Ebenen-Architektur

Die Drei-Ebenen-Architektur nach ANSI/SPARC ist ein allgemeiner Gestaltungsrahmen für Datenverwaltungssysteme:



* konzeptionelle Ebene

Logische Gesamtsicht eines Realitätsausschnitts ohne Rücksicht auf hardwaremäßige oder applikatorische Charakteristika mit Hilfe einer geeigneten Struktur. Angestrebt wird die Schaffung eines zentralen, stabilen Bezugspunktes mit langer Gültigkeitsdauer. *hierarchisch -> hierarchisch*
 \Rightarrow konz. Ebene bleibt gleich

* interne Ebene

Mit Hilfe einer geeigneten Struktur (physischen Datenstruktur) werden die Informationen auf Speichermedien festgehalten.
 Implementierungssicht der Daten durch Zugriffspfade und Speicherungsverfahren. (Speicherort, Index (Masch., logisch, phys., ...), Verschlüsselung)

* externe Ebene

Mit Hilfe einer geeigneten Struktur (logische oder externe Datenstrukturen) werden die einzelnen Benutzersichten und/ oder Programmsichten repräsentiert.
 Auf das Realitätsmodell der konzeptionellen Ebene wird Bezug genommen.
vgl. Bsp. Linke + Anpassungen (Kunde, Debitor)

Beispiel:

Die Struktur des "Kursbuch der Bundesbahn" kann als konzeptionelles Schema angesehen werden. Davon lassen sich verschiedene benutzerspezifische externe Schemata ableiten:

"Städteverbindungen", ein Taschenfahrplan mit IC- und D-Zugverbindungen zwischen den größeren Städten.

"Abfahrts- und Ankunfts fah rplan" für eine bestimmte Stadt.

"Karte des Streckennetzes" (ohne Abfahrts- und Ankunftszeiten).

"Zugbegleiter", ein Fahrplan für einen bestimmten Zug mit Ankunfts- und Abfahrtszeiten für die einzelnen Bahnhöfe und Angabe der Anschlußverbindungen ("Sicht" des im Zug Sitzenden).

"Ortliches Kursbuch" für eine bestimmte Region.

Codd'sche Regeln (nach Vorlesung Prof. Jeckle, www.jeckle.de)

1986 definiert Codd in einem Artikel für die Zeitschrift *Computer World* zwölf strenge Regeln, die ein R(elationales)DBMS aus seiner Sicht zwingend erfüllen muss um als solches eingestuft werden zu können. Die erhobenen Forderungen sind jedoch so streng, dass sie bis heute kein System vollständig erfüllt. **Ausgewählte Regeln** sind nachfolgend mit ihren englischsprachigen Originalbezeichnungen wiedergegeben.

Regel 1: The Information Rule:

Alle Daten, die in einer Datenbank gespeichert werden sind auf dieselbe Art dargestellt, nämlich **durch Werte in Tabellen** (*Anmerkung*: häufig als **Relation** bezeichnet).

Regel 2: Guaranteed Access Rule:

Jeder gespeicherte Wert muss über **Tabellenname, Spaltenname und Wert des Primärschlüssels zugreifbar sein**, wenn der **Anwender über hinreichende Zugriffsrechte verfügt**.

Regel 3: Systematic Treatment of Null Values:

Nullwerte müssen datentypunabhängig zur Darstellung fehlender Werte unterstützt werden.

Regel 4: Dynamic On-line Catalog Based on the Relational Model:

Forderung nach einem Datenkatalog (*data dictionary*). Dieser Katalog **beschreibt** die in der Datenbank abgelegten **Tabellen** hinsichtlich ihrer Struktur und zugelassenen Inhaltsbelegungen.

Regel 5: Comprehensive Data Sublanguage Rule:

Für das DBMS muss mindestens eine Sprache existieren durch die sich die verschiedenen Inhaltstypen (Tabelleninhalte, Sichten, Integritätsstrukturen (Schlüsselbeziehungen, Wertebereichseinschränkungen, Aufzählungstypen) sowie Zugriffsrechte) definieren lassen.

Regel 7: High-level Insert, Update, and Delete:

Innerhalb einer Operation können beliebig viele Datensätze bearbeitet werden. Es sind somit **keinerlei Informationen** über die **systeminterne Darstellung der Datensätze** notwendig.

Regel 8: Physical Data Independence:

Änderungen an der **internen Ebene** dürfen keine Auswirkungen auf die auf den abgespeicherten Daten operierenden Anwendungsprogramme besitzen. Werden **Daten reorganisiert** oder **beispielsweise durch Indexe zugriffsbeschleunigt** (vgl. Kapitel Dateiorganisation), so darf eine solche Änderung die auf die Datenbank zugreifenden **Anwendungsprogramme nicht beeinträchtigen**.

Regel 9: Logical Data Independence:

Änderungen des **konzeptuellen Schemas** dürfen **keine Auswirkung auf die Anwendungsprogramme** besitzen, solange diese nicht direkt von der Änderung betroffen sind.

Regel 10: Integrity Independence:

In Verfeinerung der fünften Regel wird gefordert, dass alle Integritätsbedingungen ausschließlich durch die Sprache des DBMS definieren lassen müssen. **Definierte Integritätsbedingungen** müssen in Tabellen abgespeichert werden und durch das DBMS zur Laufzeit abgeprüft werden, z.B. Primärschlüssels darf nicht NULL sein.

Regel 11: Distribution Independence:

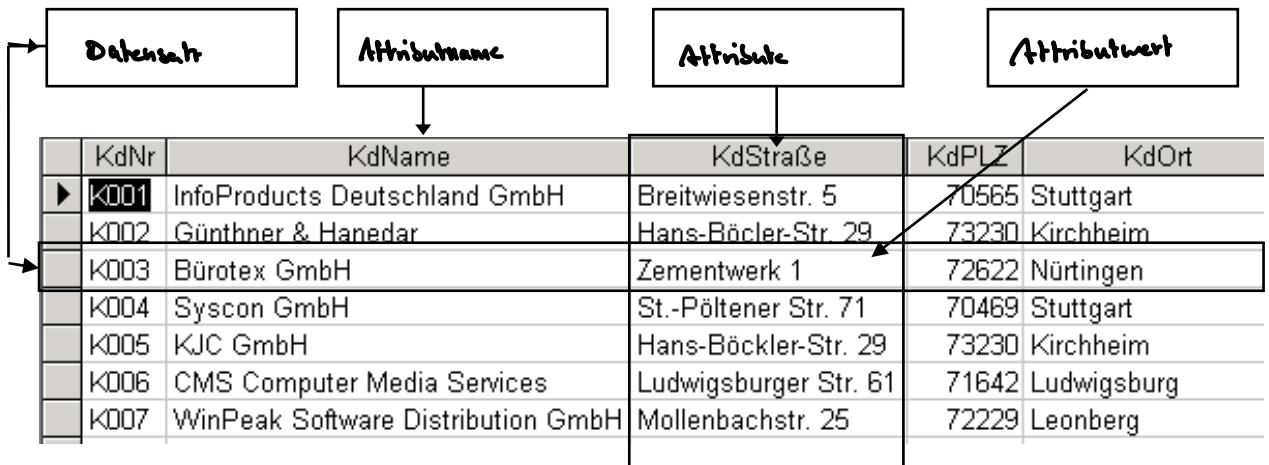
Die Anfragesprache muss so ausgelegt sein, dass Zugriffe auf lokal gehaltene Daten identisch denen auf verteilt gespeicherte Daten formuliert werden können. Hieraus lässt sich auch die Ausdehnung der Forderungen nach logischer und physischer Datenunabhängigkeit für verteilte Datenbanken ableiten.

Derzeit existiert kein am Markt verfügbares kommerzielles System welches alle zwölf Regeln vollständig umsetzt. Insbesondere sind die Regeln 6, 9, 10, 11 und 12 in der Praxis schwer umzusetzen. Darüber hinaus greifen die CODD'schen Regeln nicht alle Gesichtspunkte des praktischen Datenbankeinsatzes auf. So bleiben Fragestellungen des Betriebs (wie Sicherungs-, Wiederherstellungs- und Sicherheitsaspekte) eines DBMS völlig ausgeklammert.

Der Aufbau einer Datenbanktabelle

Im Tabellenaufbau (Tabellenstruktur) wird festgelegt, durch welche Merkmale (=Attribute) ein Datensatz (=Tupel) beschrieben wird.

- **Elemente der Tabelle:**



Infobox

- ⇒ Eine Tabelle besteht aus Zeilen und Spalten.
- ⇒ Eine Zeile (=ein **Datensatz**) enthält die Daten (eines Kunden). Alle Zeilen sind gleich aufgebaut. Das heißt, sie besitzen die gleichen **Attribute**.
- ⇒ Die Identifizierung eines Datensatzes erfolgt über einen eindeutigen **Primärschlüssel** (hier die Kundennummer).
- ⇒ Die Identifizierung einer Spalte erfolgt über einen eindeutigen Spaltennamen (=**Attributname**). Jede Spalte enthält einen bestimmten Typ von Attributen (z.B. Zahlen, Text, Datum).
- ⇒ Ein Datenfeld ist der Schnittpunkt einer Zeile und einer Spalte. Der Inhalt dieses Datenfeldes wird als **Attributwert** bezeichnet. Weniger formal wird auch die Bezeichnung „Attribut“ verwendet.

Kontrollfragen:

- 1) Welchen Attributwert für das Attribut KdStraße hat der Datensatz mit dem Primärschlüssel K005? **Hans-Böckler Str 29**
- 2) Welche Datensätze haben die Ziffernfolge 30 enthalten?
K002, K005
- 3) Welche Datenfelder haben die Ziffernfolge 29 als Datenfeldinhalt enthalten? (Farblich markieren)
Keine Verweichung, eindeutig identifizierbar

Die Bedeutung des Primärschlüssels

Wird einem Attribut die **Eigenschaft des Primärschlüssels** festgelegt, kann mit Hilfe des jeweiligen Wertes des Datenfeldes der einzelne Datensatz **eindeutig identifiziert** werden.

Bedeutung des Primärschlüssels:

- *In unserem Beispiel: Jeder Kunde hat eine Kundennummer, welche nur einmal vergeben werden kann.*
- Ein Primärschlüssel ist erforderlich, **wenn Beziehungen zwischen mehreren Tabellen hergestellt werden.** (Kommt noch!)
- Der Primärschlüssel ist in die Speicherorganisation mit einbezogen und darum für direkte Zugriffe besonders effizient zu benutzen.

Kontrollfrage:

- 1) Welches ist der Primärschlüssel für die Tabelle „Kunde“ aus dem Einführungsbeispiel? **KuNr**
- 2) Durch welchen konkreten Schlüssel wird der Kunde KJC GmbH eindeutig identifiziert? **KoNr**
- 3) Warum ist hier die Verwendung eines zusätzlichen Attributes als „künstlicher“ Primärschlüssel notwendig?



Feldeigenschaften

?

Neben dem Felddatentyp (siehe Entwurfsansicht und Hilfe) können für jedes Attribut noch weitere Feldeigenschaften festgelegt werden.

Die nachfolgende Aufstellung zeigt diese zusätzlichen Feldeigenschaften mit Beispielen für die Kundentabelle:

Feldeigenschaft	Beschreibung	Beispiel
Feldgröße	Länge des Textfeldes (5 Stellen) bzw. Zahlentyp des Felddatentyps "Zahl" (Byte, Int, Double usw.)	
Format	Ausgabeformat der Daten	Währungsformat €, \$ oder DM beim Umsatz
Dezimalstellen	Anzahl der Stellen rechts vom Dezimalzeichen	bei Zahlenfeldern
Eingabeformat	Schablonen für die Dateneingabe. Vordefinierte Muster können genutzt werden	Kundennummer: >L000 (Bedeutung: ACCESS-Hilfe)
Beschriftung	Eigene Feldbezeichnungen in einer Tabelle, einem Formular oder Bericht sind möglich	Kund.-Nr. für Kundennummer
Standardwert	Wert, mit dem in ein Feldinhalt bei der Eingabe vorbelegt werden kann.	Zahlungsziel von 30 Tagen
Gültigkeitsregel	Ausdruck, der Regeln für die Dateneingabe festlegt	Bei Zahlungsziel: <=30
Gültigkeitsmeldung	Fehlermeldung bei ungültiger Dateneingabe (Verstoß gegen die Gültigkeitsregel)	„Zahlungsziel kann nicht länger als 30 Tage sein!“
Eingabe erforderlich	Einstellung, die festlegt, ob Daten eingegeben werden müssen	
Leere Zeichenfolge (bei Text)	Vor allem als Ergänzung zu „Eingabe erforderlich“. Legt fest, ob die Eingabe einer leeren ZF: erlaubt ist.	(I{Leerzeichen}n Enter) erlaubt?
Indiziert	Einfache Indizes zur Beschleunigung von Such- und Sortievorgängen (vgl. LPE 2: Dateiorganisation)	Die Kundennummer ist indiziert (Ohne Duplikate). Wieso?

Die aus meiner Sicht wichtige Attribute sind markiert!!

Kontrollaufgabe:

- 1) PLZ des Kunden: ~~0{Ziffer}5~~ **immer 5**
- 2) Ist es sinnvoll dem Attribut KdNr in der Tabelle AufKopf folgende Eigenschaften zuzuweisen: **Eingabe erforderlich: ja** und **Indiziert: Ja (Ohne Duplikate)? ja**
- 3) Setzen Sie die Beispiele zum Zahlungsziel des Kunden in der Tabelle um. Wählen Sie einen geeigneten Felddatentypen und beachten Sie die gültigen Werte.
- 4) Erweitern Sie die Tabelle ARTLIEF um ein Attribut Lieferzeit. Wählen Sie einen geeigneten Datentyp. Annahme: Die Lieferzeit kann zwischen einem und 280 Tagen liegen. **Gültigkeitsregel: >0 Und <= 280**
- 5) Hat der Kunde Bürotex GmbH bereits etwas bestellt?
Nein
- 6) Welche Lieferanten liefert uns den Artikel mit der Artikelnummer IN311524?
**Looptech...
Loopt Computer...**
- 7) Wie viele Aufträge haben wir insgesamt?

Die Konzeption der relationalen Datenbank

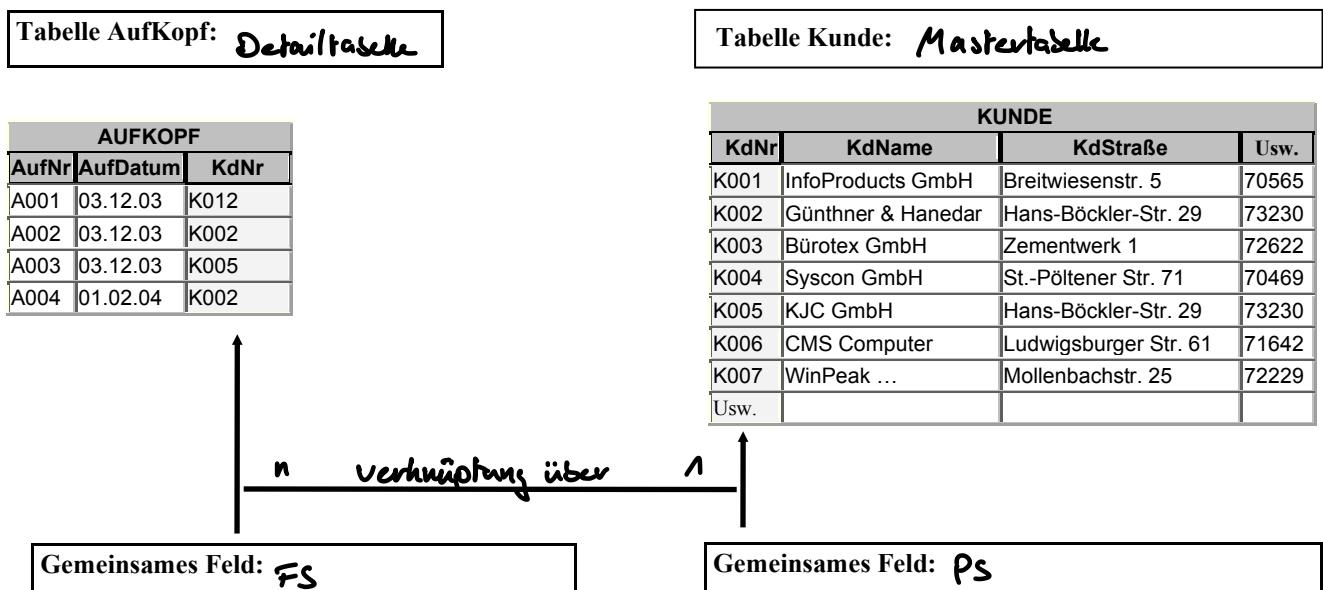
- am Beispiel zweier Tabellen

In relationalen Datenbanken arbeitet man mit mehreren Datenbanktabellen. Dies vermeidet:

- die redundante Speicherung von Daten [z.B. die Kundenanschrift in jedem Auftrag(-skopf) (vgl. Schaubild)] und vermeidet somit
- Dateninkonsistenzen (z.B. KdName für K003: Bürotex, Bürotex GmbH, Büro Tex usw.).

Zum Auswerten und zum Arbeiten mit mehreren Tabellen der Datenbank können Verknüpfungen erstellt werden. Diese Verknüpfungen werden über gemeinsame Schlüsselfelder zwischen den Tabellen hergestellt (vgl. Schaubild):

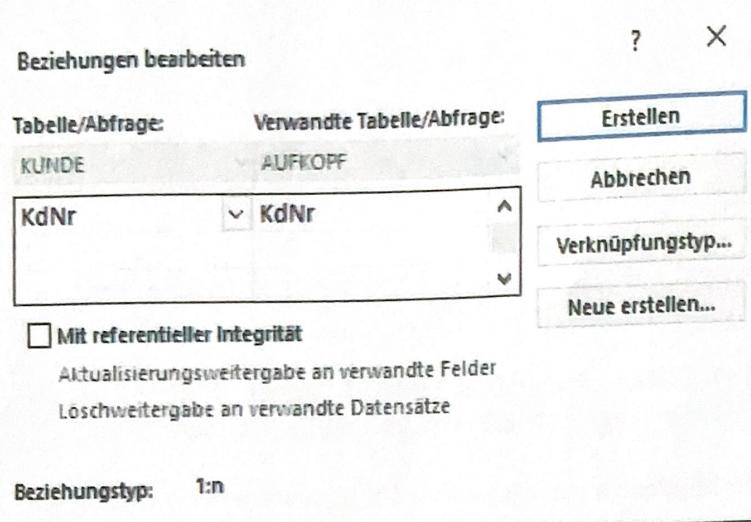
Übung: Erarbeiten Sie die untere Tabelle und ordnen Sie die Begriffe dem folgenden Schaubild zu:



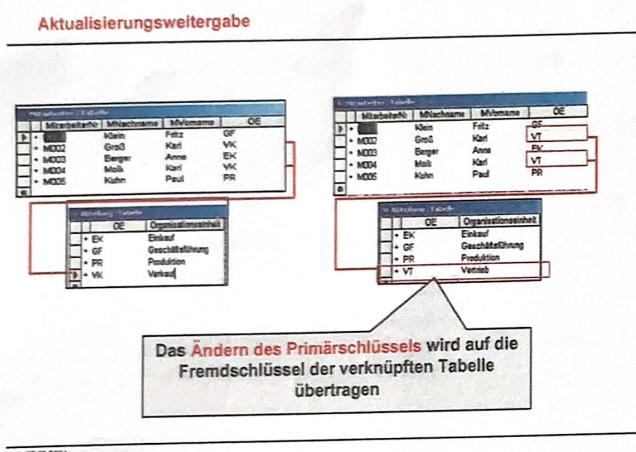
Begriff	Erklärung
<ul style="list-style-type: none"> Mastertabelle und Detailtabelle 	Beim Verknüpfen von Tabellen wird/werden jedem Datensatz der Mastertabelle (=Haupttabelle) je nach Beziehungsart ein oder mehrere Datensätze der Detailtabelle (=verknüpfte Tabelle) zugeordnet. Das für die Verknüpfung erforderliche Verbindungsfeld (=Schlüsselfeld) in der Mastertabelle muss beim Planen der Tabellenstruktur immer als <u>Primärschlüssel</u> festgelegt werden.
Primärschlüssel (PS)	In einer Tabelle muss ein Attribut als Primärschlüssel festgelegt werden, wenn Beziehung(en) zu anderen Tabellen hergestellt werden sollen. Durch die Eindeutigkeit des Primärschlüssels können sich diese nicht wiederholen.
Fremdschlüssel (FS)	Verbindungsfeld in der <u>Detailtabelle</u> , welches den (<u>fremden</u>) <u>Primärschlüssel</u> der verknüpften <u>Mastertabelle</u> enthält.
<ul style="list-style-type: none"> Beziehungsarten: <ul style="list-style-type: none"> 1 : N - Beziehung 1 : 1 – Beziehung (M:N – Beziehungen) 	<p>Die Beziehungsart gibt die Anzahl der möglichen Verknüpfungen an:</p> <ul style="list-style-type: none"> wenn für jeden Datensatz aus der <u>Mastertabelle</u> mehrere Datensätze in der verknüpften <u>Detailtabelle</u> existieren (können). wenn zu jedem Datensatz aus der <u>Mastertabelle</u> nur ein Datensatz aus der <u>Detailtabelle</u> zugeordnet werden kann diese können so nicht abgebildet werden (Eine weitere Tabelle ist einzufügen). (Kommt noch!!)

Die Referentielle Integrität

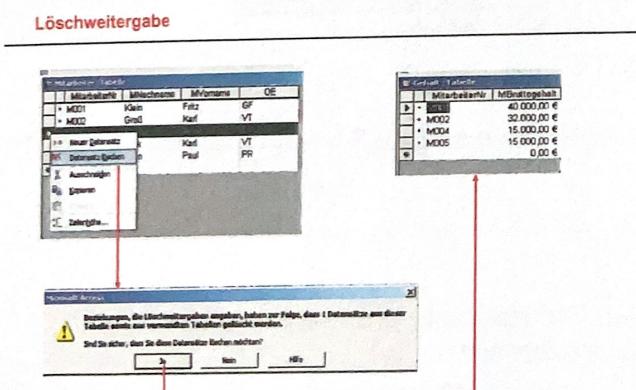
In ACCESS:



Zur Aktualisierungsweitergabe: Wird beispielsweise die Kundennummer eines Kunden in der Mastertabelle geändert, werden in den verknüpften Detailtabellen (z.B. AufKopf), die entsprechenden Fremdschlüssel der betroffenen Datensätze angepasst.



Zur Löscheitertagbe: Beim Löschen eines Datensatzes aus der Mastertabelle werden alle Datensätze mit demselben Fremdschlüsselwert aus der Detailtabelle gelöscht bzw. als gelöscht gekennzeichnet.

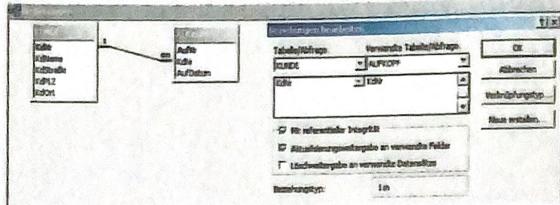


Hinweis: Die Löscheitertagbe an Detaildatensätze sollte *nicht als Standardeinstellung* beim Festlegen von Beziehungen gewählt werden.

Die Referentielle Integrität

Referentielle Integrität

Voraussetzungen:
Das verknüpfte Schlüsselattribut der Mastertabelle muss Primärschlüssel sein.



Die **referentielle Integrität** ist eine spezielle „Eigenschaft der Beziehung“ zwischen Tabellen.

2.08.02.2004
Burger

Die referentielle Integrität lässt sich im Fenster „Beziehung bearbeiten“ aktivieren. Um in dieses Fenster zu gelangen, müssen Sie mit der rechten Maustaste die zu bearbeitende Beziehung anklicken.

Auswirkungen der referentiellen Integrität:

Durch die Festlegung der **referentiellen Integrität**, z. B. zwischen der Tabelle Aufkopf und der Tabelle Kunde, besteht eine feste Verknüpfung zwischen diesen beiden Tabellen. Diese hat folgende Wirkungen:

1. Wirkungen der referentiellen Integrität

Es kann in der Detailtabelle kein Datensatz mit einem Fremdschlüssel aufgenommen werden, der nicht in der Mastertabelle existiert.

Tabelle Aufkopf		
AufNr	KdNr	AufDatum
A001	K012	03.12.03
A002	K002	03.12.03
A003	K005	03.12.03
A004	K002	10.02.04
A005	K002	11.02.04
A006	KNIX	

KdNr	KdName	KdStraße
K001	InfoProducts Deutschland GmbH	Breitwiesenstr. 5
K002	Günther & Hanedar	Hans-Böcler-Str. 29
K003	Bürotex GmbH	Zementwerk 1
....
K011	Comp & Phone	Neue Weilheimer Str. 14
K012	C&S Computer und Service	Katharinenstr. 21

Fehlt in Tabelle
KUNDE

4.08.02.2004
Burger

2. Wirkungen der referentiellen Integrität

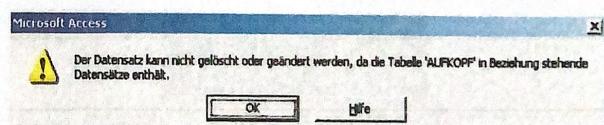
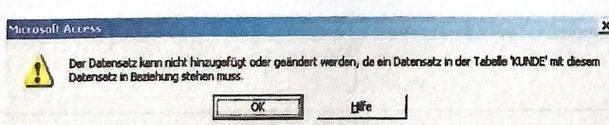
Es kann kein Datensatz der Mastertabelle gelöscht werden, solange noch Datensätze mit dem entsprechenden Fremdschlüssel in der Detailtabelle existieren.

Tabelle Aufkopf				
AufNr	KdNr	AufDatum		
A001	K012	03.12.03		
A002	K002	03.12.03		
A003	K005	03.12.03		
A004	K002	10.02.04		
A005	K002	11.02.04		

KdNr	KdName	KdStraße	KdPLZ	KdOrt
K001	InfoProducts Deutschland GmbH	Breitwiesenstr. 5	70565	Stuttgart
K002	Günther & Hanedar	Hans-Böcler-Str. 29	73230	Kirchheim
K003	Bürotex GmbH	Zementwerk 1	72622	Nürtingen

Kunde K002 kann nicht gelöscht werden!

3.08.02.2004
Burger



Berechnete Felder

Feld:	VKPreis	VKPreis(brutto): [VKPreis]*1,19
Tabelle:	ARTIKEL	

Mathematische Operatoren

- * Multiplikation
- + Addition
- Subtraktion bzw. Vorzeichen
- / Division

Punkt-vor-Strich-Rechnung: Ansonsten Klammern verwenden

Besonderheiten bei Berechnungen

Anzeigeformat ändern
→ Kontextmenü EIGENSCHAFTEN

Rechte Mausklick auf einzustellendes Feld
in der Entwurfsansicht -> Eigenschaften ->

Textfelder verknüpfen mit (&)

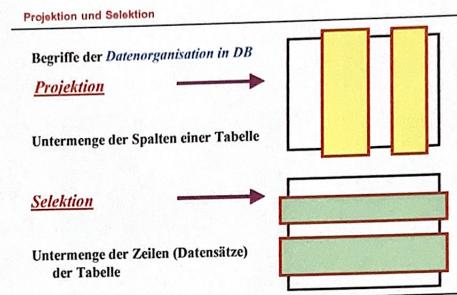
Feld:	VerkäuferName	Anschrift: [Straße] & ", "& [PLZ] & " "& [Wohnort]
Tabelle:	Verkäuferdaten	

VerkäuferName	Anschrift
Lorenz, Sophia	Schloßstr. 5, 55163 Mainz
Richter, Hans-Otto	Goetheplatz 6, 55173 Mainz

Zu den Begriffen Projektion und Selektion:

Bisher wurden nur einzelne Felder ausgewählt und die Werte aller Datensätze angezeigt. Man spricht hier von einer **Projektion**.

Daneben gibt es die **Selektion**. Hierbei werden, einzelne Datensätze, nach *anzugebenden Kriterien* (z.B. Artikelgruppe Drucker) selektiert. Häufig ist eine Mischung der beiden Formen zweckmäßig.



Selektion

Die Selektion hat den Zweck, nur diejenigen Datensätze anzuzeigen, die den aktuellen Anforderungen entsprechen (z.B. alle Ausgänge eines Tages). Die Kriterien lassen sich in der entsprechenden Spalte eintragen. Die aus der Programmierung bekannten logischen Operatoren UND, ODER, NICHT stehen zur Eingabe zur Verfügung (z.B. #30.06.99# ODER #02.02.50#).

Wie am Beispiel ersichtlich, gibt es spezielle Anforderungen an die Syntax der Kriterien.

Feld:	Ausgangsdatum
Tabelle:	Ausgänge
Sortierung:	
Anzeigen:	<input checked="" type="checkbox"/>
Kriterien:	#30.06.99#
oder:	

Besondere Möglichkeiten zur Formulierung von Abfragen

Syntax für

- Feldnamen [NAME] z.B. [Artikelnummer] oder [Menge][Einzelpreis]
- DATUM/UHRZEIT #12.04.99#
- TEXT "TEXT"

Vergleichsoperatoren =,<,> und

Vergleichsoperator WIE

- * Beliebige Anzahl von Zeichen
- ? Ein beliebiges Zeichen
- # Eine beliebige Ziffer

Feld:	ArtikelNr	Eingangsdatum
Tabelle:	Einläge	Einläge
Sortierung:		
Anzeigen:	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Kriterien:	Wie "EDV"	Wie "#06.99"
oder:		

ArtikelNr	Eingangsdatum
EDV-003	03. Jun. 99
EDV-002	06. Jun. 99
EDV-003	06. Jun. 99
EDV-001	12. Jun. 99
EDV-002	20. Jun. 99

Während mit = auf exakte Übereinstimmung geprüft wird (z.B. EDV-003) lassen sich mit dem Wie-Operator auch Abfragen nach einem Teil des Datenfeldwertes formulieren.
Vergleichen Sie die unterschiedlichen Aufbau der SQL-Anweisung !!

SQL (structured query language) -Anweisungen

DDL

**Data
Definition
Language**

**Definition von
Tabellen und
Feldern**

z. B.

CREATE (Erzeugen),
ALTER (Ändern),
DROP (Löschen)

DML

**Data
Manipulation
Language**

**Zugriff auf
Datensätze**

z. B.

SELECT (Auswahl),
INSERT (Einfügen),
UPDATE (Aktualisieren),
DELETE (Löschen)

DCL

**Data
Control
Language**

**Zur Vergabe
von
Berechtigungen
z.B.**

GRANT,
COMMIT,
REVOKE



Wie arbeitet SQL?

SQL fragt nicht, wie die gesuchten Daten ermittelt, sondern ausschließlich welche Daten benötigt werden.

Das Arbeiten mit relationalen Datenbanken geschieht **mengenorientiert**. Dabei wird das gewünschte Ergebnis deskriptiv angegeben, indem seine Eigenschaften in SQL beschrieben werden. Ein prozedurales Vorgehen, bei dem durch die einzelnen Datensätze navigiert werden muss, ist nicht erforderlich.

SQL Standard?

**SQL grundsätzlich normiert,
ABER
es gibt unterschiedliche Stände und
(wie bei den Menschen auch)
jedes DBMS spricht etwas Dialekt.**

Beispiel:

ACCESS	Andere Schreibweise
Platzhalter *	Platzhalter %
[Mittlerer Einstandspreis]	Nicht möglich, bzw. M_EP

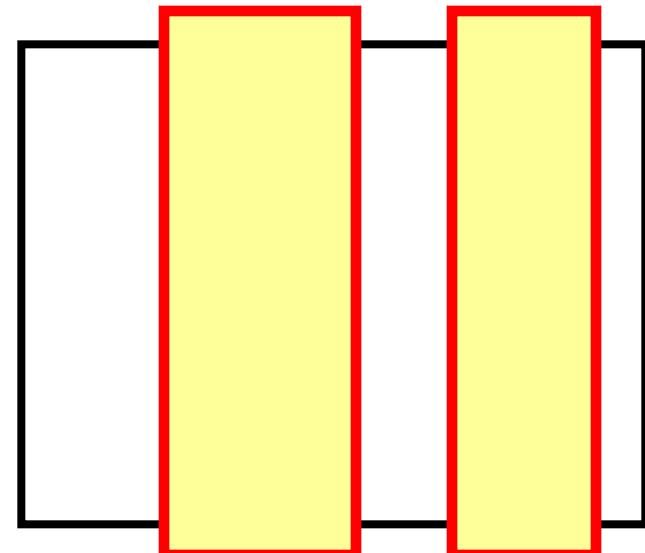
Projektion und Selektion

Begriffe der
Datenorganisation

Projektion



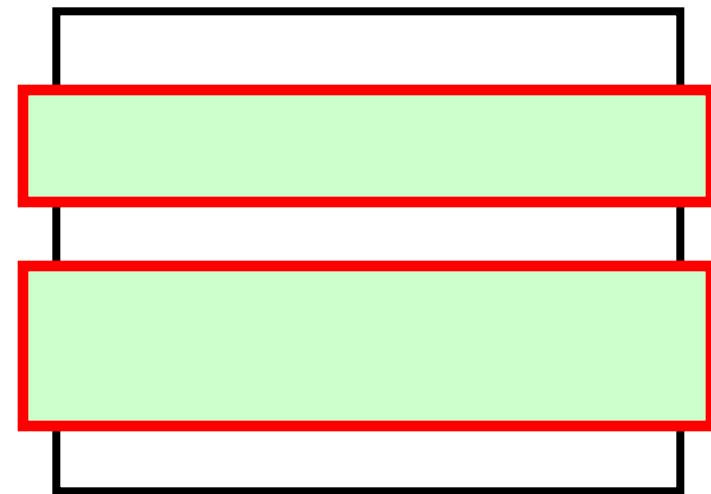
**Untermenge der Spalten
einer Tabelle**



Selektion



**Untermenge der Zeilen
(Datensätze) der
Tabelle**



6. SQL (-Abfragen)

Structured Query Language

wird zur Abfrage von Daten aus Datenbanken benutzt

- 4 „Hauptbefehle“
 - C**reate (insert)
 - R**ead (select)
 - U**pdate (update)
 - D**elete (delete)

Einfügen

INSERT INTO	Befehl: Daten sollen eingefügt werden
tabelle	Auswahl der Tabelle
(spalteA, spalteB, ...)	Auswahl der Spalten
VALUES	Befehl: jetzt folgen einzufügende Daten
(,wertA', ,wertB', ...)	Daten

Abfragen

SELECT	Befehl: Daten sollen ausgewählt werden
spalteA, spalteB, ... oder * (alle Spalten)	Auswahl der Spalten
FROM tabelle	Auswahl der Tabelle
WHERE spalteX = 5	Angabe eines Suchfilters (optional)
ORDER BY spalteZ DESC	Angabe der Sortierung (optional) (descending)

Löschen

DELETE FROM	Befehl: Daten sollen gelöscht werden
tabelle	Auswahl der Tabelle
WHERE spalteA = 7	Angabe eines Suchfilters (optional)

Aktualisieren

UPDATE	Befehl: Aktualisiere die Daten
tabelle	Auswahl der Tabelle
SET spalteZ='wert'	Setze neue Werte
WHERE spalteX='wert'	optional: Angabe eines Filters

JOINS

allg: man kann mehrere Tabellen miteinander verknüpfen.

Bsp: ohne: `SELECT * FROM Produkte, Anbieter
WHERE Anbieter.ID = Produkte.ID
AND Anbieter.Name = "Amazon"`

mit: `SELECT * FROM Produkte
JOIN Anbieter ON (Anbieter.ID = Produkte.ID)
WHERE Anbieter.Name = "Amazon"`

NATURAL JOIN

wie JOIN, nur werden Datensätze direkt anhand „passender“ Attribute miteinander verknüpft

\Rightarrow `SELECT * FROM Produkte
NATURAL JOIN Anbieter
WHERE Anbieter.Name = "Amazon"`

weitere Befehle

AVG() Durchschnitt **MIN(spalte)** kleinste Zahl der Spalte

MAX(spalte) größte Zahl der Spalte

... AS... eine Tabelle bekommt einen anderen Namen

COUNT() berechnet Anzahl der Einträge

SUM() berechnet Summe der Einträge (Inhalte) (Beispiel dazu \rightarrow)

GROUP BY Gruppiert anhand Filter

ORDER BY sortiert anhand Filter

LIKE sucht nach etwas indem Bsp: ... LIKE "%2014%"
das folgende verknüpft \Rightarrow 2014... wird gesucht

\Rightarrow Party...
 \Rightarrow alle Sachen mit Party

emp_id	emp_name	working_date	working_hours
1	Rjeet	2015-01-24	18
2	Milan	2015-01-24	18
3	Ryan	2015-01-24	6
1	Rjeet	2015-01-25	12
2	Ryan	2015-01-25	6
3	Milan	2015-01-25	9
2	Milan	2015-01-26	12
3	Milan	2015-01-26	12

`SELECT emp.name, SUM(working_hours)
FROM tabelle
GROUP BY emp.name`

emp_name	Total working hours
Rjeet	36
Ryan	28
Milan	57
Ruchi	12

HAVING
VS WHERE

Selektion mit logischen und/oder Verknüpfungen

Beispiel : Liste aller Teile der Teileart E und H

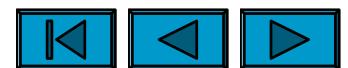
Achtung UND/ODER:

```
SELECT *
FROM TEILE
WHERE TeileArt="E" OR TeileArt="H"
ORDER BY TeileArt, Bezeichnung;
```

Operatoren in der WHERE-Klausel

Vergleichsoperatoren	< , > , >= , <= , = , <>
Intervaloperatoren	[NOT] BETWEEN „A“ AND „B“
Enthaltenopertor	[NOT] IN (“A”, “B”, “C”)
Auswahloperator	ALL , ANY , SOME
Ähnlichkeitsoperator	[NOT] LIKE „*Begriff*“ (Manche DBMS % statt *)
Existenzopertor	EXISTS
Nulloperator	IS [NOT] NULL

Internet-Tipp: z.B. <https://luo-darmstadt.de/sqltutorial/>
W3school –SQL-Kurs



Having vs. Where

Where bezieht sich auf die Selektion von Zeilen

Having bezieht sich auf die Selektion von Gruppen

Fazit: Prüfen, worauf sich ein Kriterium bezieht.

Aufgabe 3:

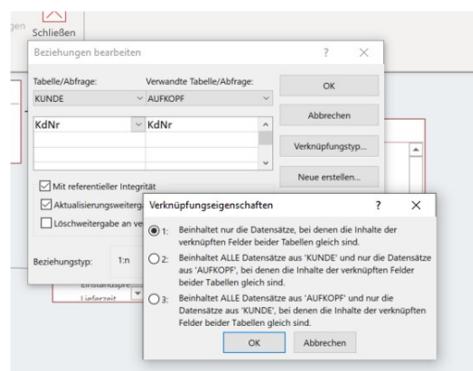
Verknüpfungseigenschaften bei Abfragen

ANSICHT - VERKNÜPFUNGSEIGENSCHAFTEN

oder

Doppelklick auf Verknüpfungsline

Weiter mit Verknüpfungstyp



2. 25.04.2021
Burger

Variante 1:

Exklusionsverknüpfung

Datensätze aus zwei Tabellen, deren Werte in den verknüpften Feldern identisch sind



AufNr	ArtNr	ArtBez
A001	IN311524	CPU Intel Cora 8
A001	LX004520	Optra SC 1275, Farblaserdrucker
A001	SON0055	Multiscan 500sx 19" Trinitron
A002	Ily01824	Ilyama ProLite E43000S
A002	LX004520	Optra SC 1275, Farblaserdrucker
A003	EIZ02390	EIZO T566
A003	HP045510	HP Laser Jet
A003	LX000200	Lexmark Color Jet
A003	SON0055	Multiscan 500sx 19" Trinitron

Alle Auftragspositionen inkl. der Bezeichnung

3. 25.04.2021
Burger

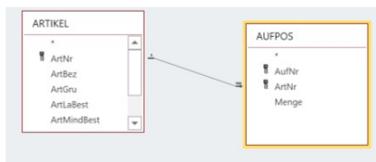
Bei diesen Abfragen werden ausschließlich (Exklusion) die innerhalb zweier Tabellen vorhandenen Zeilen mit entsprechenden Kriterien (Inner Join) berücksichtigt. Zumeist stellt dieses Kriterium die Übereinstimmung des Schlüsselwertes dar: Tab1.Feld1 = Tab2.Feld2 (Equi-Join).

Im Gegensatz dazu übernehmen die folgenden Outer-Joins jeweils aus einer Tabelle Datensätze auch ohne Entsprechung in der anderen Tabelle. Mit left bzw. right wird deutlich gemacht, ob aus der ersten oder zweiten Tabelle Datensätze einbezogen (Inklusion) werden sollen, die keine Entsprechung haben (vgl. Varianten 2 & 3).

Variante 2: SQL: ... From Tab1 **left join** Tab2

Links-Inklusionsverknüpfung

Alle Datensätze der Mastertabelle, auch ohne identische Werte im verknüpften Feld der Detailtabelle



	ArtNr	AufNr
BUN0030		
EI02390	A003	
FUF0011		
HP045510	A003	
HP054981		
IB612200		
Ily01824	A002	

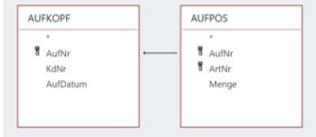
Alle Artikel, ohne Aufträge oder mit auch mit den Aufträgen verknüpft

4.25.04.2021
Burger

Variante 3: ... From Tab1 **right join** Tab2

Rechts-Inklusionsverknüpfung

Alle Datensätze der Detailtabelle, auch ohne identische Werte im verknüpftem Feld der Mastertabelle



	AufNr	ArtNr
A001	IN311524	
A001	IX004520	
A001	SON0055	
A002	Ily01824	
A002	IX004520	
A003	EI02390	
A003	HP045510	
A003	IX000200	
A003	SON0055	
	LX004520	

Alle Auftragspositionen (hier Attribut ArtNr) und falls vorhanden die dazugehörigen Kopfdaten! HIER liegt somit ein Fehler vor, da es in diesem Fall keine Position ohne dazugehörigen Kopf geben darf.

5.25.04.2021
Burger

Aufgabe: Überlegen Sie für die beiden Beziehungen, welche Varianten einen Sinn ergeben? Und wenn ja, welchen?

	Kunde-Aufkopf	Artikel-Artlief
Inner-Join (Exklusionsv.) <i>nur Datensätze die in beiden existieren z.B. Kunde in Kunden und Aufkopf</i>	Alle Aufträge inkl. Kundeninfos	Alle Artikel mit lizenzierter Informationen (von Artikel)
Left-outer-Join <i>alle links + die in rechts vorhanden</i>	Alle Kunden mit und ohne Aufträge	Nur Artikel mehrere lizenziert -> zeigt mehrere
Right-outer-Join <i>alle rechts + die in links vorhanden</i>	Gleich inner Join (Referentielle Integrität aktiviert + Muss-Feld)	

Self- Join

Auf den ersten Blick wenig sinnvoll. Dennoch gibt es zahlreiche Fälle, welche dieses Konstrukt notwendig machen.

Beispiel:

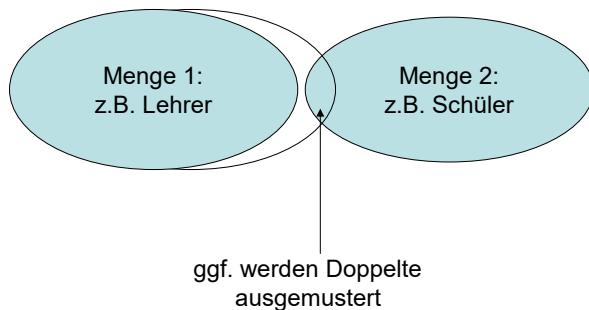
Versuchen Sie eine Abfrage zu erstellen, die Ihnen zu einer beliebig vom Bearbeiter einzugebenden Artikelbezeichnung alle Alternativen dieser Artikelgruppe zur Verfügung stellt.
(Zeit: max. 10 min)

Distinct

```
SELECT "8.ArtNr, 3.ArtBetr, 1.ArtGrup
FROM Artikel 1, Artikel 3
WHERE 1.ArtBetr = "... " / Like    """&[Suchbegriff]&"""
AND 3.ArtGrup = 1.ArtGrup;
```

UNION

- Vereinigung zweier Mengen.



Praktische Bedeutung, z.B. bei Geschäftspartnern, vgl. auch folgende Aufgabe
(Zur Vertiefung: z.B. https://glossar.hs-augsburg.de/Mengenoperatoren_in_SQL)

Bsp.:

Zu einer Infoveranstaltung in Stuttgart sollen sowohl Kunden als auch Lieferanten eingeladen werden. Erstellen Sie eine Abfrage, die sie mit der Serienbrieffunktion einer Textverarbeitungssoftware nutzen können.

Teillösung für Kunden:

```
SELECT KdNr AS Schlüssel, KdName AS Name
FROM Kunde
WHERE Kdort = "Stuttgart"
```

Union

Teillösung für Lieferer:

```
SELECT LNR, LName
FROM Lieferer
WHERE LOrt = "Stuttgart";
```

Subqueries (Unterabfragen)

Bei Operationen wie **IN**, **ANY**, **ALL** und **EXISTS** können Unterabfragen verwendet werden. Eine Unterabfrage ist eine SELECT-Anweisung innerhalb einer SELECT-Anweisung.

Beispiel 1: Zeige Name, PLZ und Straße aller Kunden, die gestern einen Auftrag erteilt haben.

Q1

```
SELECT KdNr, KdPlz, KdStrasse  
FROM Kunde  
WHERE KdNr IN
```

Q2

```
(SELECT KdNr  
FROM AufKopf  
WHERE AufDatum = DATE() - 1;
```

Vgl. IN-Operator



Subqueries (Unterabfragen)

Beispiel 2:

Gibt die Namen aller Spieler an, die mindestens eine Strafe erhalten haben.

```
SELECT      NAME  
FROM        SPIELER  
WHERE       EXISTS
```



Q1

```
SELECT      *  
FROM        STRAFEN  
WHERE       SPIELER.SPIELERNR =  
                      SPIELERNR
```



Q2

Typisch für
EXISTS

Ein EXISTS-Prädikat wird als wahr erkannt, wenn in einer Suchbedingung eine Unterabfrage mindestens eine Zeile selektiert. Daher ist EXISTS i.d.R. korreliert zu verwenden, d.h. Q1 und Q2 mit WHERE PS=FS verknüpft

Subqueries (Unterabfragen)

Beispiel 3: Welchen Preis muss ein Artikel unterschreiten bei

ALL: 110,00

ANY: 354,20

Q1
SELECT *
FROM Artikel
WHERE VKPreis < ALL|ANY

Q2
(SELECT VKPreis
FROM Artikel
WHERE ArtGru="CPU")

VKPreis
330,00 €
354,20 €
110,00 €

Weitere SQL-Befehle (vgl. FS)

4 ARBEITEN MIT DER DATENBANK

4 Arbeiten mit der Datenbank

4.1 Einfügen von Zeilen in die Tabellen

Nachdem der Prozeß der Erzeugung der Datenbank und ihrer Tabellen abgeschlossen ist, kann die Datenbank zum ersten Mal mit Daten gefüllt werden. Das Einfügen von Zeilen (Datensätzen) in die Datenbank muß unter Zuhilfenahme von SQL zeilenweise geschehen. Die allgemeine Form des Einfügekommandos lautet:

```
INSERT INTO Tabellename
  [(Spaltenname,...)][VALUES (Konstante,...) | Select-Anweisung];
```

INSERT :

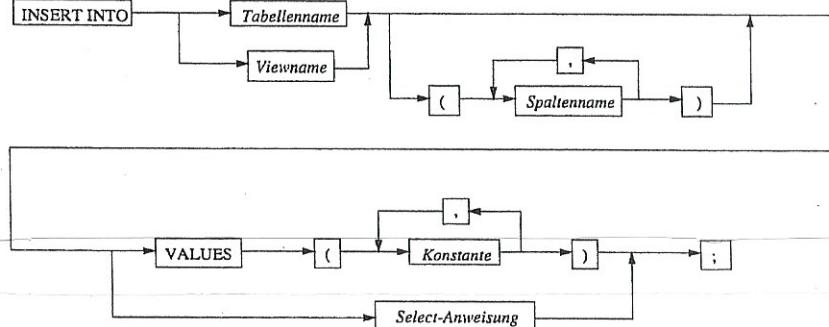
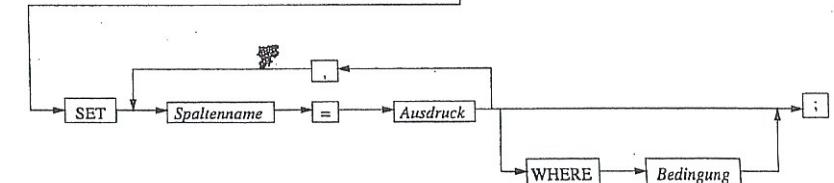
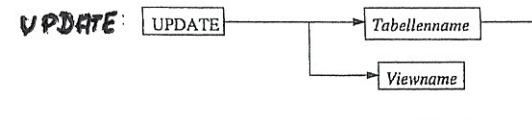


Abbildung 4: Syntaxdiagramm der INSERT-Anweisung

Das INSERT-Kommando kann auf sehr unterschiedliche Arten benutzt werden, wie die allgemeine Form zeigt. Die einfachste Form besteht in der Variante, bei der einfach für alle Spalten konstante Werte eingegeben werden. Sollen nur bestimmte Spalten in einer Tabelle mittels des INSERT-Kommandos mit Werten versehen werden, kann man diese Spalten selektieren. Die nicht im Kommando aufgeführten Datenfelder erhalten den Wert NULL. Es ist auch möglich, den Wert NULL in der Werteliste nach dem Schlüsselwort VALUES explizit zu nennen. Während der Wert NULL zu allen Datentypen kompatibel ist, muß beim Einfügen in eine Tabelle darauf geachtet werden, daß die einzufügenden Werte genau dem für die jeweilige Spalte definierten Datentyp entsprechen. Der Wert NULL darf aber nicht in Spalten eingefügt werden, die bei der Erstellung der Tabelle den Constraint NOT NULL hatten. Weiterhin läßt sich das INSERT-Kommando in Kombination mit der SELECT-Anweisung (s. nächsten Abschnitt) dazu verwenden, Dateninhalte einer Tabelle in eine zweite Tabelle zu kopieren.

13

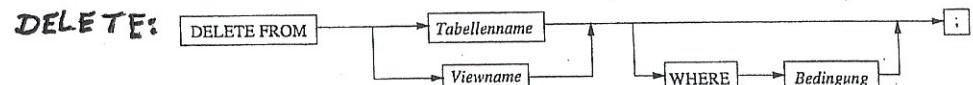
Filtrierung d. Datensätze



Für Tabellename wird der Name der Tabelle eingetragen, in der die Werte von Spalten verändert werden sollen. Für Spaltenname trägt man einen Namen einer Spalte aus der betreffenden Tabelle ein. Es kann in einer UPDATE-Anweisung jeweils nur immer eine Aktualisierung der Werte einer Tabelle vorgenommen werden. Mittels der WHERE-Klausel werden, wie bei der SELECT-Anweisung, die zu ändernden Datensätze selektiert; läßt man die WHERE-Klausel fort, so werden alle Datensätze der gewählten Tabelle geändert. Wie bei der SELECT-Anweisung kann die WHERE-Klausel des UPDATE-Kommandos durch eine Unterabfrage ergänzt werden.

4.3.3 Löschen von Datensätzen

Zum Löschen von Datensätzen aus einzelnen Tabellen der Datenbank wird die DELETE-Anweisung benutzt. Sie wirkt wie die UPDATE-Anweisung jeweils immer nur auf eine Tabelle. Die allgemeine Form lautet:



Vorsicht ist geboten bei der Benutzung von DELETE ohne Angabe einer Auswahlbedingung in der WHERE-Klausel, weil nämlich dann durch die SQL-Anweisung alle Datensätze einer Tabelle gelöscht werden. Manche Datenbanksysteme sind für diesen Fall so benutzerfreundlich konstruiert, daß vor dem tatsächlichen Löschen noch einmal zurückgefragt wird, ob diese Maßnahme ausgeführt werden soll. Wenn der Anwender keine besonderen Vorkehrungen zu einer etwaigen Rekonstruktion der gelöschten Daten getroffen hat, sind die gelöschten Daten unwiderruflich verloren. Wird DELETE ohne eine Unterabfrage benutzt, dann dürfen sich die Spaltenangaben in der Bedingung der WHERE-Klausel nur auf die Tabelle beziehen, aus der gelöscht werden soll; das bedeutet, daß eine Verknüpfung von Tabellen hier nicht erlaubt ist.

DDL (data definition language) – Erstellung von Tabellen – I

Syntax:

```
CREATE TABLE Tabellename
(<Spaltenname> <Datentyp>
[, <Spaltenname> <Datentyp>]
[, ...]);
```

Beispiel:

```
CREATE TABLE KUNDE
(knr String (5),
kundenname String,
seit Date,
umsatz Double);
```



Übung zum Erstellen von Tabellen

Für die folgenden Aufgaben ist mit dem DBMS (Datenbankmanagement-System) Access eine leere Datenbank (SQL.accdb) anzulegen.
Sämtliche Aufgabenstellungen sind in dieser DB zu erfüllen.

Übung:

Erstelle eine Tabelle MIETER mit folgenden Spalten:

Feldname	Datentyp	Feldbeschreibung
mieternr	alpha-nummerisch (4-stellig)	Mieternummer
nachname	Alpha-nummerisch	Nachname des Mieters
vorname	Alpha-nummerisch	Vorname des Mieters
mietpreis	nummerisch	monatlicher Mietpreis
beginn	Datum	Beginn des Mietverhältnisses
plz	Alphanummerisch 5 Stellen	PLZ des Orts der Wohnung

Der Nachname und die PLZ dürfen nicht leer bleiben.



Webprog - Teil

input-types (Vorteile)

↳ z.B. andere Tastaturen

button - führt form action attribute beim klicken aus.

checkbox - ☐ mehrere von Auswahl wählbar

date - datum

email - je nach Browser Validation/aut. Handytastatur .com als extra Knopf

number - nur Zahlen erlaubt

range - slider (min = max =)

radio - Ø nur einen von einer Anzahl wählbar? gleicher "name"

reset - form zurücksetzen

submit - sendet form mit method = post/get ab

password - maskiert/versteckt eingegebene Punkte

text - standard - für längere Texte

Formularelemente

form - umschließt Inhalt | action, target, method

fieldset - Gruppieren von Daten in Form

input - siehe links | type, id, name

label - Text, beziehend auf input | for

textarea - großes Textfeld | name, rows, cols

legend - caption für fieldset

Grid-Layout

```
<div class="wrapper">
```

```
  <div class="box a"> -
```

```
  ;
```

```
.wrapper{
```

display: grid;

gap: -

grid-template-columns: px/fr/repeating

grid-template-rows: px/fr/repeating

grid-template-areas: "A B C"

"D E F"

"G H I"

```
}
```

```
@media (min-width: 3px){
```

.wrapper

- anderes Template

} → für verschiedene Breiten

→ repeating (, -)
wie oft wird pro Zeile/Spalte?

Müssen nicht gleichzeitig sein
nur für gewisse Anwendungen

JavaScript

wofür

responsive + interaktive Elemente auf Webseiten

Events (+Smiley/ColorChoser)

input —>

↳ let x = document.getElementById("id")

x.addEventListener("event", funktion)

funktion: funktion() { ... } → click change

Style veränderbar durch:

x.style. — = —

⚠️ x.innerHTML != x.innerText

↓

z.B. <p> zweiter tatsächlicher Text

PHP

wofür

Erstellung dynamischer Webseiten

Script auf Server (nicht Client) nur

Ergebnis am Client

Get + Post

Get

Daten von bestimmter Quelle anfragen

Query String in URL (hinter angezeigt)

↳ name = Wert

Post

Daten an Server senden um etwas zu erstellen/aktualisieren

Daten im HTTP-Request gespeichert

Smiley

```
<div>
  
  <br>
  <input type="button" value="fröhlich" id="froehlich" disabled="true" />
  <input type="button" value="traurig" id="traurig" />
</div>
<script>
  let froehlich = document.getElementById("froehlich")
  let traurig = document.getElementById("traurig")
  let bild = document.getElementById("smiley")

  froehlich.addEventListener('click', () => {
    traurig.disabled = false;
    froehlich.disabled = true;
    smiley.src = './smileyBilder/smiley2.png'
  });

  traurig.addEventListener('click', () => {
    froehlich.disabled = false;
    traurig.disabled = true;
    smiley.src = './smileyBilder/smiley1.png'
  });
</script>
```

Color-chooser

```
<div>
  <label for="redRange">Rot</label>
  <input type="range" min=0 max=255 step=1 name="rangeInput" value="120" id="redRange" style="width: 80px;" />
  <br>
  <label for="redRange">Grün</label>
  <input type="range" min=0 max=255 step=1 name="rangeInput" value="120" id="greenRange" style="width: 80px;" />
  <br>
  <label for="redRange">Blau</label>
  <input type="range" min=0 max=255 step=1 name="rangeInput" value="120" id="blueRange" style="width: 80px;" />
</div>
<div id="colorBox" style="height: 100px; background-color: #000000;"></div>
<p id="textAusgabe">rgb(120, 120, 120)</p>
</div>
<script>
  let box = document.getElementById("colorBox");
  let red = document.getElementById("redRange");
  let green = document.getElementById("greenRange");
  let blue = document.getElementById("blueRange");
  let textAusgabe = document.getElementById("textAusgabe");

  function changeColor() {
    redValue = red.value;
    greenValue = green.value;
    blueValue = blue.value;

    let aktuelleFarbe = 'rgb(' + redValue + ',' + greenValue + ',' + blueValue + ')';
    box.style.backgroundColor = aktuelleFarbe;
    textAusgabe.innerHTML = '<p>' + aktuelleFarbe + '</p>';
  }

  red.addEventListener('change', changeColor);
  green.addEventListener('change', changeColor);
  blue.addEventListener('change', changeColor);
</script>
```