

Beilagenblatt EF-Informatik 2024

1 Datenbanken

Tabelle: Tabellename (Konvention: Grossbuchstaben), Merkmale/Attribute (Spaltenüberschriften)

Normalformen

1. Normalform Merkmale atomar
2. Normalform 1. NF und Nichtschlüsselmerkmale von jedem Schlüssel voll funktional abhängig.
3. Normalform 2. NF und Nichtschlüsselmerkmale nicht vom Schlüssel transitiv abhängig.

Entitäten-Beziehungsmodell

Jede Entität (Rechteck bei grafischer Darstellung) hat einen Schlüssel und in der Regel zusätzliche Attribute.

Zwischen Entitäten bestehen Beziehungen (Rauten), komplexe Beziehungen zwingend als Tabelle. Assoziationstypen: 1 gegen ein, c kein oder ein, n ein oder mehrere, nc kein, ein, mehrere

SQL

SELECT, FROM, WHERE, ORDER BY (DESC, ASC), GROUP BY, DISTINCT, HAVING, COUNT, SUM, AVG, MIN, MAX

2 Telekommunikation, Webseiten

Schichtenmodell

Schicht Beispiele, Kommentar

Awendungsschicht TCP (Verbindungsauftakt: Dreiweghandshake unter Verwendung von Sequence Number und Acknowledgement Number)

Sicherungsschicht

MAC-Adresse in Hexadezimalzahlen Basis 16: A=10, ..., F=15), Hammingcodeierung; m Datenbits, r Prüfbits, m + r + 1 $\leq 2^r$

RSA

$a \equiv b \pmod{n}$, wenn $a - b$ durch n teilbar ist
 $\mathcal{R}_n(a \cdot b) = \mathcal{R}_n(\mathcal{R}_n(a) \cdot \mathcal{R}_n(b))$
 $c \cdot d = 1 + k(p-1)(q-1) \Rightarrow m^{c \cdot d} \equiv m \pmod{n} (n = pq)$
 $\tilde{m} = \mathcal{R}_n(m^c) \Rightarrow m = \mathcal{R}_n(\tilde{m}^d)$

HTML

```
<!DOCTYPE html>
<html lang="de">
  <head>
    <meta charset="utf-8" />
    <title>Dokumenttitel</title>
  </head>
  <body>
    <h1>Titel</h1>
  </body>
</html>
```

Weitere HTML-Tags: h2, script, p, ol (ordered list), ul (unordered list), li (listelement), div

CSS

Typenselektor: typename, Klasse selektor: classname, ID-Selektor: #idname, Nachbarselектор: A + B Kindselktor: A > B, Nachfahrenselktor: A B: Erstes Kind: A:first-child

JavaScript

DOM-Manipulation: document.getElementsByTagName(tagName).
document.getElementById(classname); document.getElementsByClassName(className);
Events: element.addEventListener("click", myFunction)

3 Programmieren in Java, Algorithmik

Java

Konvention: Klassennamen gross geschrieben, Variablen (z.B. Referenz auf Objekt) klein geschrieben.

Konstruktor: heisst gleich wie Klasse

super(): Zugriff auf Konstruktor der Mutterklasse

Zugriffsmodifikatoren: public, protected (Zugriff in Tochterklasse), private (Zugriff mu innerhalb Klasse)

primitive Datentypen: byte, short, int, long, float, double, boolean

Logische Operatoren: a && b (und), a || b (oder), !a (nicht a)

Type Casting: int myInt = (int) myDouble;

Array: a.length (Länge des Arrays a), a[0] (erstes Element des Arrays)

Algorithmen

selectionsort: $O(n^2)$, Auswahl des höchsten Werts, bei restlichen $n - 1$ Werten analog.

Bubblesort: $O(n^2)$, Aufsteigende Blasen (jeweils Nachbarn vertauschen, wenn in falscher Reihenfolge).

Mergesort: $O(n \log n)$, Teile und Herschle

Dynamische Datenstrukturen

Stack: pop (entfernen oberstes Element, Rückgabe des Werts), push Element zuoberst auf Stack legen.

Queue: poll (Löschen am Anfang), offer (Einfügen am Ende)

LinkedList: Als verketete Liste implementiert

ArrayList: Liste auf Array basierend implementiert

Datenbanken

Tabellen

SCHUELER

S#	Name	Klasse	Freifach
1	Hans	1B	Ma, Ps
2	Fritz	4A	In, Ma
3	Clara	GC	Ps

SCHUELER

S#	Freifach	Name	Klasse
1	Ma	Hans	1B
1	Ps	Hans	1B
2	In	Fritz	4A
2	Ma	Fritz	4A
3	Ps	Clara	GC

SCHUELER

S#	Name	Klasse
1	Hans	1B
2	Fritz	4A
3	Clara	GC

FREIFACH

S#	Freifach
1	Ma
1	Ps
2	In
2	Ma
3	Ps

LEHRER

S#	Klasse	Lehrer
1	1B	Nickel
2	4A	Wolf
3	4A	Wolf

LEHRER

Klasse	Lehrer
1B	Nickel
4A	Wolf

SCHUELER

S#	...
1	
2	
3	

3. Normalf.

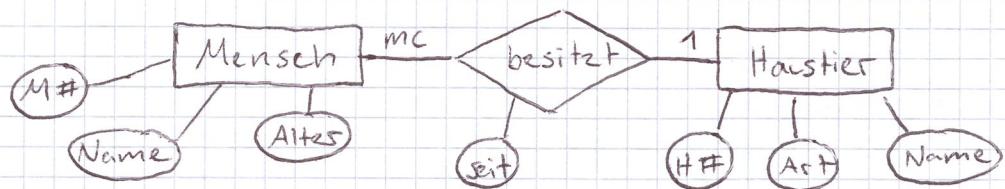
Begriffe:

- voll funktional abhängig alles einzeln, nicht redundant
- transitiv abhängig redundante Informationen
- atomar einmalig

Entitäten - Beziehungsmodell

Entität: ein eindeutig bestimmbarer Objekt

Beziehung: zwischen zwei Entitäten



SQL

SELECT	Merkmale	*	alle Merkmale	DISTINCT	keine Duplikate
FROM	Tabelle				
WHERE	Bedingung				
ORDER BY	sortieren				

'% ... %'

^ Wort + zuvor / danach noch etwas

IS NULL ← leer

FROM verlag v ← Abkürzung

Internet



Protokolle

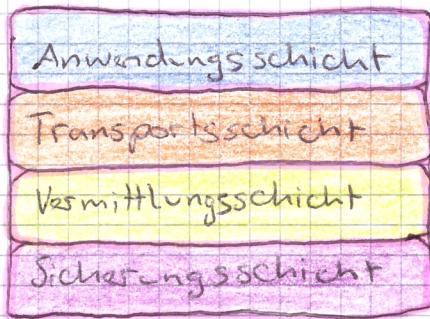
HTTPS	la
POP	E
IMAP	
SMTP	E
DNS:	

Name →

Begriffe:

- Dienst jede Schicht Verfügung
- Protokoll wie etwas
- Schnittstelle zwischen

Internet



- erbringt Dienste (Protokolle)
- DNS
- Verbindungsauflauf & Übertragung
- TCP
 - den richtigen Weg finden
- Übertragungsfehler

Protokolle:

- HTTP(S) laden von Websites
- POP Emails empfangen
- IMAP " "
- SMTP Emails versenden

DNS:

Name \leftrightarrow IP-Adresse

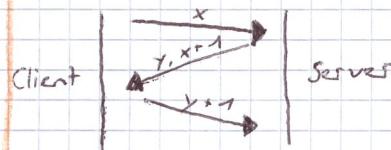
Begriffe:

- Dienst
 - jede Schicht stellt einen zur Verfügung
- Protokoll
 - wie etwas gemacht wird
- Schnittstelle
 - zwischen Schichten

TCP-Segment:

Source Port	Destination Port
Sequence Number	
Acknowledgment N.	
Data	

Verbindungsauflauf:



Übertragung:



IP-Adresse:

Zehnersystem \leftrightarrow Dualsystem

$$130 \quad 1000\ 0010$$

$2^7 + 2^1$

Subnetzmaske:

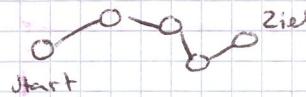
192.168.1.3

255.255.255.0
oder
/24

$$\rightarrow 192.168.1.0$$

Netzwerkanteil

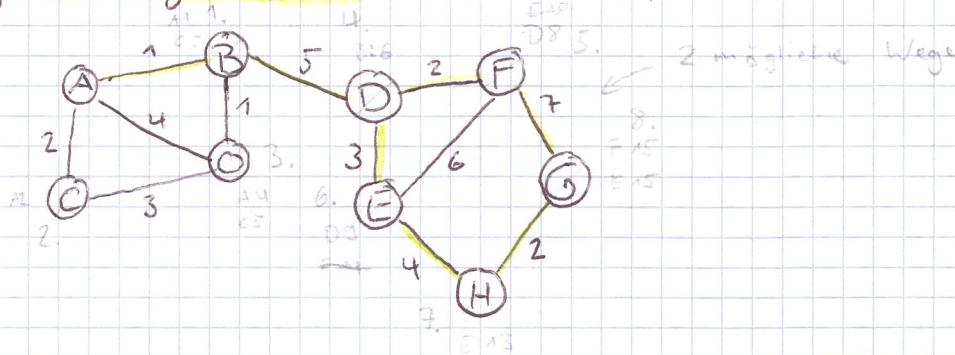
Routing:



Paket macht Zwischenstopps

1	2
0	1
2 ⁰	x
2 ¹	x
2 ²	x
2 ³	x

Dijkstra-Algorithmus: (A nach G)



Mac-Adresse:

00:80:41:AE:FD:7E

↓ zu decimal (1. Stelle $\cdot 16 + 2.$ Stelle $\cdot 1$)

0 : 128 : 65 : 174 : 253 : 126

Broadcast: FF:FF:FF:FF:FF:FF

Hamming code:

$$d(1000\ 1001, 1011\ 0001) = 3$$

$$m + r + 1 \leq 2^r$$

↑ ↑

Länge Prüfbits
Code

RSA - Ver

Authentizität

Integrität

Verschlüsse

Entschlüssel

öffentlic

Symmetrisch

Assymmetrisch

RSA - Verfor

Fehler erkennen:

Zu 6 falsch

	1	2	3	4	5	6	7	8	9	10
	✓	X		X				✓		
0	1	0	0	1	0	0	1	1	0	
2^0		X		X			X			
2^1		X	X			X	X			
2^2			X	X	X	X				
2^3							X	X	X	

RSA-Verfahren

Authentizität: Nachricht ist echt. Nur Besitzer des privaten Schlüssel kann korrekte Signatur erstellen

Integrität: Nachricht unverändert. Mit Hash-Wert und dem öffentlichen Schlüssel kann die Integrität überprüft werden.

Verschlüsseln: $\bar{m} = R_n(m^c) = m^c \bmod n$

Entschlüsseln: $m = R_n(\bar{m}^d) = m^d \bmod n$

öffentlicher Schlüssel: c und n privater Schl.: d und n

Symmetrisch: ein einziger Schlüssel, den beide kennen

Assymmetrisch: zwei untersch. Schlüssel

RSA-Verfahren: 1. 2 grosse Primzahlen p und q

2. $n = p \cdot q$

3. $r = (p-1)(q-1)$

4. $(c, r) = 1$ (teilerfremd)

5. Linearkombination 8 umstellen $d \cdot c = 1 + k \cdot r$

Linearkombination:

1. euklidischer Algorithmus

2. $(a, b) = x \cdot a + y \cdot b$

Bsp.:

$$1. \quad a = 220 \quad b = 175$$

$$220 = 1 \cdot 175 + 45$$

$$175 = 3 \cdot 45 + 40$$

$$45 = 1 \cdot 40 + 5$$

$$40 = 8 \cdot 5 + 0 \quad \text{ggT: } 5$$

$$2. \quad 5 = 45 - 40$$

$$= 45 - (175 - 3 \cdot 45)$$

$$= 4 \cdot 45 - 175$$

$$= 4(220 - 175) - 175$$

$$= 4 \cdot 220 - 5 \cdot 175$$

Webentwickl

HTML: Int

CSS: Lay

Java Script

HTML:

Liste:

Link:

CSS:

• class {

• informat

div > p: f

DOM:



Rechnen mit Resten:

$R_n(m^c)$

$$\text{Bsp.: } R_{143}(2^{103}) = R_{143}(R_{143}(2) \cdot R_{143}(2^2) \cdot \dots)$$

$$\hookrightarrow 2^{103} = 2^{64} \cdot 2^{32} \cdot 2^4 \cdot 2^2 \cdot 2^1$$

$$R_{143}(2) = 2 \quad R_{143}(2^2) = 4 \quad R_{143}(2^4) = 16 \quad R_{143}(2^8) = 113$$

$$R_{143}(2^{16}) = R_{143}(113 \cdot 113) = 42 \quad R_{143}(2^{32}) = 48 \quad R_{143}(2^{64}) = 16$$

$$R_{143}(2 \cdot 4 \cdot 16 \cdot 48 \cdot 16) = R_{143}(R_{143}(4 \cdot 48) \cdot R_{143}(256) \cdot 2)$$

$$= R_{143}(49 \cdot 2 \cdot 113) = R_{143}(49 \cdot 83) = 63$$

Webentwicklung

HTML: Inhalt + Struktur

CSS: Layout

Java Script : Verhalten / Interaktion

HTML:

```
Liste: <ol>
      <li> ... </li>
    </ol>
```

```
<ul>
      <li> ... </li>
    </ul>
```

Link: Google

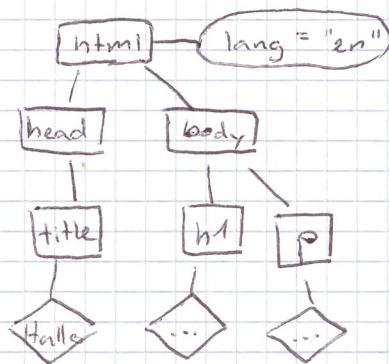
CSS:

• class { color: red; }

• information li { color: blue; } Nachfahren von information

div > p: first-child { color: orange; } p orange, welche direkte Nachfahren von div sind

DOM:



Manipulation: document.get...

, innerHTML = " ";

Event-Handling:

Bubblesort:

```
Bsp.: const b = document.getElementById("btn1");
b.addEventListener("click", berechneMarschzeit);
function berechneMarschzeit() {
    const d = document.getElementById("dist").value;
    const h = document.getElementById("elegain").value;
    const t = d/4 + h/400;
    const p = document.getElementById("marschzeit");
    p.innerHTML = "Marschzeit: " + t + " Stunden";
```

Mergesort: Dm

Algorithmitik

Komplexität: Abhängig von n

Selectionsort: $O(n^2)$

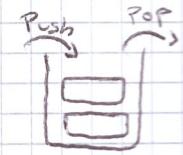
Bubblesort: $O(n^2)$

Mergesort: $O(n \log_2(n))$

```
Selectionsort: for (int i=0; i < a.length; i++) {
    int max = i;
    for (int j = 0; j < a.length; j++) {
        if (a[j].compareTo(a[i]) > 0) {
            max = j;
        }
    }
    int help = a[i];
    a[i] = a[max];
    a[max] = help;
}
```

Dynamische

Stack:



Clear:
public void c

top = null;
size = 0;

}

Queues: →

Offer:
public void off

Node temp

if (size ==

head = te

tail = tem

} else {

tail.setNe

tail = tem

} size++

```

BubbleSort: for (int i=0; i < a.length; i++) {
    int lastToSort = a.length - i;
    for (int j=0; j < lastToSort-1; j++) {
        if (a[j].compareTo(a[j+1]) < 0) {
            int help = a[j];
            a[j] = a[j+1];
            a[j+1] = help;
        }
    }
}

```

Mergesort: Divide and Conquer Rekursives Funktionsaufruf

Dynamische Datenstrukturen

Stack:



Push:

```

public void push(String s) {
    Node temp = new Node(s);
    temp.setNextNode(top);
    top = temp;
    size++;
}

```

Clear:

```

public void clear() {
    top = null;
    size = 0;
}

```

Queues: → □□□→

Offer:

```

public void offer(String s) {
    Node temp = new Node(s);
    if (size == 0) {
        head = temp;
        tail = temp;
    } else {
        tail.setNextNode(temp);
        tail = temp;
    }
    size++;
}

```

Pop:

```

public String pop() {
    String s = top.getContent();
    if (top != null) {
        top = top.getNextNode();
        size--;
    }
    return s;
}

```

Poll:

```

public String poll() {
    if (head != null) {
        String s = head.getContent();
        head = head.getNextNode();
        if (size == 1) tail = null;
        size--;
    }
    return s;
}

```

Linked List: dynamisch, schneller beim Einfügen am Anfang

ArrayList: festgelegte Größe, besserer Zugriff auf einzelne Elemente

Programmieren

Begriffe: Datenfeld - Daten existieren so lange wie das Objekt

Parameter - existieren nur während Ausführung. Werte von Außen.

lokale Variablen - können nur im jeweiligen Block benutzt werden

Integer - ganze Zahlen String - Zeichenfolge Double / Float - Kommaz.

Boolean - nur zwei Werte

Objekt - durch Klasse definiert. Haben Methoden

Klasse - beschreibt bestimmte Art von Objekten

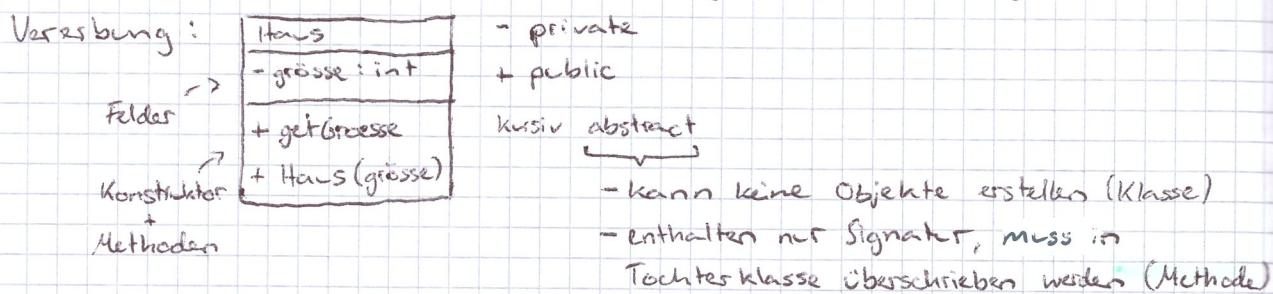
Instanz - individuelles Objekt

Array - "Liste" von Daten

this - zeigt auf Daten außerhalb

Objektorientierte Programmierung:

Vorteile: übersichtlich, einfache Anpassung, mehrfache Nutzung eines Modells



static: Methode gehört der Klasse, keinem Objekt
braucht kein Object

Rekursiv: Methode ruft sich selbst auf

braucht Abbruch-Bedingung

Bsp. int fakultaet (int n)

```
if (n <= 0) {  
    return 1;  
} else {  
    return n * fakultaet (n-1);  
}
```