

↓
Informatik 8bi Schuljahr 2019/2020 als PDF exportieren

Informatik 8. Klasse - Schuljahr 2019/20

Lehrinhalte

- [Lehrplaninhalte](#)

[Remote-Zugriff auf Schulserver](#)

Kapitel

- 1) Datenstrukturen
- 2) Datenbanken
- 3) PHP & MySQL
- 4) Unity 3D



Leistungsbeurteilung

- **Schularbeiten (SA)**
 - 2x SA (2h und 3h)
- **Mitarbeit (MA)**
 - Aktive Mitarbeit im Unterricht (aMA)
 - Mündliche Stundenwiederholungen (mMA)
 - Schriftliche Stundenwiederholungen (sMA)
- **Praktische Arbeiten (PA)**
 - 1x praktischer Arbeitsauftrag pro Woche
- [Aktueller Leistungsstand](#)

Stoff für die 1. Schularbeit in Informatik - 8BI - 5.12.2019 (2h)

Stoff für die 2. Schularbeit in Informatik - 8BI - 5.3.2019 (3h)

Themengebiete RDP

From:

<http://elearn.bgamstetten.ac.at/wiki/> - Wiki

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920

Last update: **2019/10/10 11:15**



Was wird in der 8. Klasse gemacht?

7. Semester

8. Klasse (3 Stunden, eine 2- oder 3-stündige Schularbeit)

Sicherung der Nachhaltigkeit

- Notwendiges Vorwissens für die Kompetenzbereiche dieses Moduls wiederholen und aktivieren
- Grundlagen für die Kompetenzbereiche dieses Moduls ergänzen und bereitstellen

Gesellschaftliche Aspekte der Informationstechnologie

Berufliche Perspektiven

- Informatikberufe und Einsatzmöglichkeiten der Informatik in verschiedenen Berufsfeldern benennen und einschätzen können.

Verantwortung, Datenschutz und Datensicherheit

- Die Entwicklung der Informatik beschreiben und bewerten können.
- Die Bedeutung von Informatik in der Gesellschaft beschreiben, die Auswirkungen auf die Einzelnen und die Gesellschaft einschätzen und Vor- und Nachteile an konkreten Beispielen abwägen können.
- Maßnahmen und rechtliche Grundlagen im Zusammenhang mit Datensicherheit, Datenschutz und Urheberrecht kennen und anwenden können.

Informatiksysteme - Hardware, Betriebssysteme und Vernetzung

Technische Grundlagen und Funktionsweisen (Hardware)

- Aktualisierungen im Zusammenhang mit der Hardware kennen

Betriebssysteme (Windows, Linux, MacOS, iOS, Android)

- Aktualisierungen im Zusammenhang mit Betriebssystemen kennen

Mensch-Maschine-Schnittstelle

- Maßnahmen für einen barrierefreien zu Zugang Informatik-Systemen angeben können

Algorithmik und Programmierung

Algorithmen und Datenstrukturen

- Algorithmen erklären, entwerfen, darstellen können.
- Datenstruktur Bäume kennen und einsetzen können
- Rekursionen kennen und einsetzen können
- Dynamische Programmierung kennen
- Algorithmen mit Bäumen erstellen können
- Algorithmen mit Rekursionen erstellen können

Programmierung (Objektorientierte visuelle Programmiersprache)

- Algorithmen in einer Programmiersprache implementieren können
- Datenbank Anwendungen programmieren können
- Programme mit Bäume erstellen können
- Rekursive Algorithmen erstellen können

Angewandte Informatik, Datenbanksysteme und Internet

Datenmodelle und Datenbanksysteme

- Einen Webserver konfigurieren können
- Internetdienste (Mail-Server, Web-Server, FTP-Server) in ihrer Funktionsweise verstehen und einsetzen können

Web-Techniken (Content-Management-Systeme)

- Content-Management-Systeme installieren können
- Rechte bei Content-Management-Systemen vergeben können
- Oberfläche bei Content-Management-Systemen einstellen und anpassen können
- Die Funktionsweise durch Plugins und Module erweitern können

8. Semester

8. Klasse (3 Stunden, eine 3- oder 4-stündige Schularbeit)

Sicherung der Nachhaltigkeit

Wiederholen, Vertiefen von Fähigkeiten und Vernetzen von Inhalten, um einen umfassenden Überblick über die Zusammenhänge unterschiedlicher informatischer Teilgebiete zu gewinnen.

Inhalt und Umfang der Klausurarbeit im Prüfungsgebiet Informatik

(1) Im Rahmen der Klausurarbeit im Prüfungsgebiet „Informatik“ ist den Prüfungskandidatinnen und Prüfungskandidaten eine Aufgabenstellung mit drei bis fünf voneinander unabhängigen Aufgaben, die in Teilaufgaben gegliedert sein können, aus unterschiedlichen Kompetenzbereichen – Gesellschaftliche Aspekte der Informationstechnologie, Informatiksysteme, Algorithmik und Programmieren sowie Angewandte Informatik, Datenbank und Internet - mit ausgewogenen Anforderungen schriftlich vorzulegen. Mindestens eine Aufgabe hat anwendungsorientierten Charakter zu haben. Für die Bearbeitung zumindest einer Aufgabe muss Computertechnologie eingesetzt werden. (2) Die Arbeitszeit hat 270 Minuten zu betragen.

From:

<http://elearn.bgamstetten.ac.at/wiki/> - Wiki

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:0_lehrplaninhalte

Last update: **2019/09/06 19:37**



C++

- [1.1\) Zeiger \(Wdhg. 7.Klasse\)](#)
 - [1.1.1\) Zeiger-Übung \(Wdhg. 7.Klasse\)](#)
- [1.2\) Datenstruktur struct](#)
- [1.3\) Einfach verkettete Listen](#)
- [1.4\) Doppeltverkettete Listen](#)
- [1.5\) Binäre Bäume](#)
- [1.6\) Rekursionen](#)
 - [1.6.1\) Rekursion-Übung](#)

From:

<http://elearn.bgamstetten.ac.at/wiki/> - **Wiki**

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:1



Last update: **2019/09/08 11:50**

Zeiger (Pointer)

Zeiger (engl. pointers) sind Variablen, die als Wert die Speicheradresse einer anderen Variable enthalten.

Jede Variable wird in CPP an einer bestimmten Position im Hauptspeicher abgelegt. Diese Position nennt man Speicheradresse (engl. memory address). CPP bietet die Möglichkeit, die Adresse jeder Variable zu ermitteln. Solange eine Variable gültig ist, bleibt sie an ein und derselben Stelle im Speicher.

Am einfachsten vergegenwärtigt man sich dieses Konzept anhand der globalen Variablen. Diese werden außerhalb aller Funktionen und Klassen deklariert und sind überall gültig. Auf sie kann man von jeder Klasse und jeder Funktion aus zugreifen. Über globale Variablen ist bereits zur Kompilierzeit bekannt, wo sie sich innerhalb des Speichers befinden (also kennt das Programm ihre Adresse).

Zeiger sind nichts anderes als normale Variablen. Sie werden deklariert (und definiert), besitzen einen Gültigkeitsbereich, eine Adresse und einen Wert. Dieser Wert, der Inhalt der Zeigervariable, ist aber nicht wie in unseren bisherigen Beispielen eine Zahl, sondern die Adresse einer anderen Variable oder eines Speicherbereichs. Bei der Deklaration einer Zeigervariable wird der Typ der Variable festgelegt, auf den sie verweisen soll.

```
#include <iostream>

int main() {
    int Wert;           // eine int-Variable
    int *pWert;         // eine Zeigervariable, zeigt auf einen int
    int *pZahl;         // ein weiterer "Zeiger auf int"

    Wert = 10;          // Zuweisung eines Wertes an eine int-Variable

    pWert = &Wert;      // Adressoperator '&' liefert die Adresse einer
    Variable            // Variable
    pZahl = pWert;       // pZahl und pWert zeigen jetzt auf dieselbe Variable
```

Beispielhafte Speicherbelegung des Programms im Hauptspeicher:

Datentyp	Variable	Adresse	Wert
int	Wert	0x0001	10
int *	pWert	0x0005	0x0001
int *	pZahl	0x0009	0x0001
...
...

Der Adressoperator & kann auf jede Variable angewandt werden und liefert deren Adresse, die man einer (dem Variablentyp entsprechenden) Zeigervariablen zuweisen kann. Wie im Beispiel gezeigt, können Zeiger gleichen Typs einander zugewiesen werden. Zeiger verschiedenen Typs bedürfen einer Typumwandlung. Die Zeigervariablen pWert und pZahl sind an verschiedenen Stellen im Speicher abgelegt, nur die Inhalte sind gleich.

Wollen Sie auf den Wert zugreifen, der sich hinter der im Zeiger gespeicherten Adresse verbirgt, so verwenden Sie den Dereferenzierungsoperator `*`.

```
*pWert += 5;
*pZahl += 8;

std::cout << "Wert = " << Wert << std::endl;
```

Beispielhafte Speicherbelegung des Programms im Hauptspeicher:

Datentyp	Variable	Adresse	Wert
int	Wert	0x0001	10 -> 15 -> 23
int *	pWert	0x0005	0x0001
int *	pZahl	0x0009	0x0001
...
...

Ausgabe:

```
Wert = 23
```

Man nennt das den Zeiger dereferenzieren. Im Beispiel erhalten Sie die Ausgabe Wert = 23, denn pWert und pZahl verweisen ja beide auf die Variable Wert.

Um es noch einmal hervorzuheben: Zeiger auf Integer (int) sind selbst keine Integer. Den Versuch, einer Zeigervariablen eine Zahl zuzuweisen, beantwortet der Compiler mit einer Fehlermeldung oder mindestens einer Warnung. Hier gibt es nur eine Ausnahme: die Zahl 0 darf jedem beliebigen Zeiger zugewiesen werden. Ein solcher Nullzeiger zeigt nirgendwohin. Der Versuch, ihn zu dereferenzieren, führt zu einem Laufzeitfehler.

Beispiel Zeigerübung

```
int main()
{
    int zahl=10;
    int *z=NULL;
    int **zz=NULL;

    cout<< zahl<<endl;           //Wert wird ausgegeben
    cout<<&zahl<<endl;           //adresse wird ausgegeben
    cout<<&zahl+1<<endl;         //adresse von zahl+1
    cout<< zahl +1<<endl;        //11

    z=&zahl;
```



```

    cout<< *z<< endl;           //10
    cout<< z<< endl;           //Adresse von zahl= Wert von z
    cout<< &z<< endl;          //Adresse von Zeiger z

    zz=&z;                       //Adresse von Zeiger z wird in Zeiger zz

    cout<< *zz<<endl;           //Wert von z= Adresse von zahl //&zahl
//z
    cout<< **zz<<endl;          //Wert von zahl //zahl
    cout<< zz<<endl;            //Adresse von z =Wert von zz //&z
//zz
    cout<< &zz<<endl;           //Adresse von Zeiger zz //&zz

    getch();
    return 0;
}

```

Verschiedene Konventionen bei der Definition von Zeigern

Bei der Definition einer Zeigervariablen muss das Zeichen * nicht unmittelbar auf den Datentyp folgen. Die folgenden vier Definitionen sind gleichwertig:

```

int* i; // Whitespace (z.B. ein Leerzeichen) nach *
int *i; // Whitespace vor *
int * i; // Whitespace vor * und nach *
int*i; // Kein whitespace vor * und nach *

```

Versuchen wir nun, diese vier Definitionen nach demselben Schema wie eine Definition von „gewöhnlichen“ Variablen zu interpretieren. Bei einer solchen Definition bedeutet

T v;

dass eine Variable v definiert wird, die den Datentyp T hat. Für die vier gleichwertigen Definitionen ergeben sich verschiedene Interpretationen, die als Kommentar angegeben sind:

```

int* i; // Definition der Variablen i des Datentyps int*
int *i; // Irreführend: Es wird kein "*i" definiert,
        // obwohl die dereferenzierte Variable *i heißt.
int * i; // Datentyp int oder int* oder was?
int*i; // Datentyp int oder int* oder was?

```

Offensichtlich passen nur die ersten beiden in dieses Schema. Die erste führt dabei zu einer richtigen und die zweite zu einer falschen Interpretation.

Allerdings passt die erste Schreibweise nur bei der Definitionen einer einzelnen Zeigervariablen in dieses Schema, da sich der * bei einer Definition nur auf die Variable unmittelbar rechts vom *

bezieht:

```
int* i,j,k; // definiert int* i, int j, int k
           // und nicht: int* j, int* k
```

Zur Vermeidung solcher Missverständnisse sind zwei Schreibweisen verbreitet:

- C-Programmierer verwenden oft die zweite Schreibweise von oben, obwohl sie nicht in das Schema der Definition von „gewöhnlichen“ Variablen passt. Diese Schreibweise wird auch in Turbo CPP verwendet.

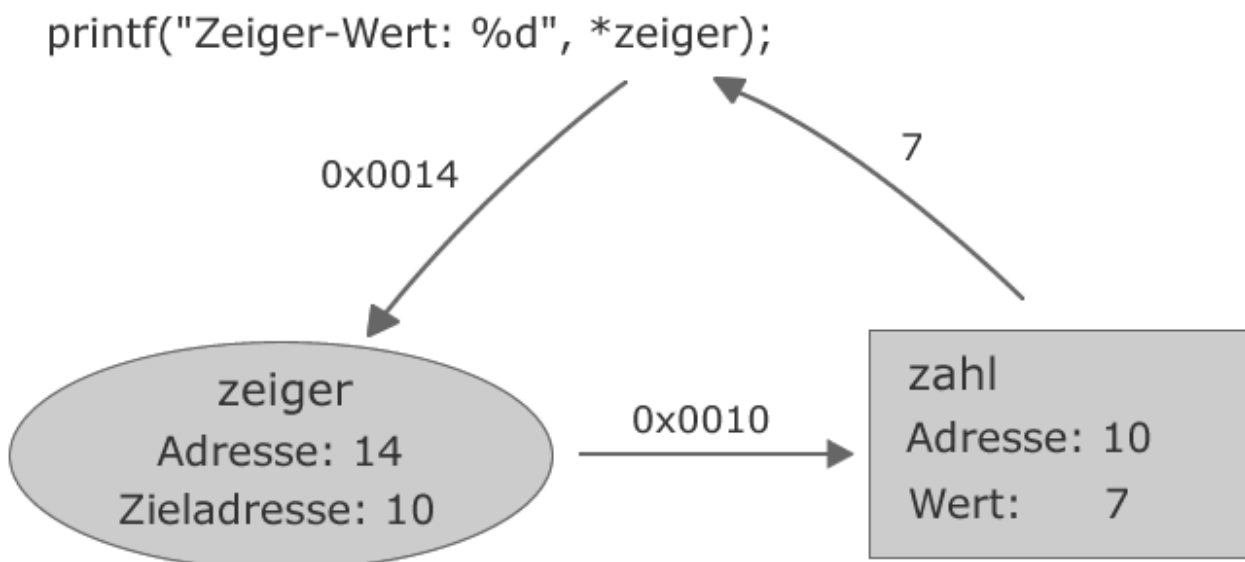
```
int *i,*j,*k; // definiert int* i, int* j, int* k
```

- Bjarne Stroustrup (einer der C bzw. CPP-Entwickler) empfiehlt, auf Mehrfachdefinitionen zu verzichten. Er schreibt den * wie in „int* pi;“ immer unmittelbar nach dem Datentyp.

Ein weiteres Beispiel....

```
int zahl = 7;
int *zeiger;
zeiger = &zahl;
printf("Zeiger-Wert: %d\n", *zeiger);
```

Ein **Zeiger repräsentiert eine Adresse** und nicht wie eine Variable einen Wert. Will man auf den Wert der Adresse zugreifen, auf die ein Zeiger zeigt, muss der Stern * vor den Namen gesetzt werden.



Zeiger auf Zeiger

Zeiger zeigen auf Adressen. Sie können nicht nur auf die Adressen von Variablen, sondern auch auf die Adressen von Zeigern verweisen. Dies erreicht man mit dem doppelten Stern-Operator **.

```
int zahl=7;
```

```
int *zeiger = &zahl;  
int **zeigerAufZeiger = &zeiger;  
  
cout << "Wert von zeigerAufZeiger -> zeiger -> zahl:" << **zeigerAufZeiger;
```

Ausgabe

```
Wert von zeigerAufZeiger -> zeiger -> zahl: 7
```

From:

<http://elearn.bgamstetten.ac.at/wiki/> - Wiki

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:1:1_01

Last update: **2019/09/08 12:24**



Aufgabe 1.1.1

Gib nach jeden Programmierbefehl alle Adressen und Inhalte der einzelnen Variablen aus!

```
int a=2, b=5, *c=&a, *d=&b;

a = *c * *d;
*d -= 3;
b = a * b;
c = d;
b = 7;
a = *c + *d;
```

Aufgabe 1.1.2

Gib nach jeden Programmierbefehl alle Adressen und Inhalte der einzelnen Variablen aus!

```
int a=2, b=5, *c=&a, *d=&b;
int **zz=NULL;

a = *c + *d;
zz=&d;
**zz=*zz-10;
*c *= 3;
b = a * *c;
c = d;
a = 7-*d;
b = *c * *d;
*c = *c + **zz;
```

Aufgabe 1.1.3

```
char *a=NULL, *b=NULL, *c=NULL;
char d;

a = new char;
b = new char;

*a = 'S';

*b = 'T';

c = b;

cout << *a << endl;
cout << *b << endl;
```

```
cout << *c << endl;

d = 'U';

*c = 'G';

b = &d;

cout << d << endl;
cout << *b << endl;

d = 'H';

b = a;

a = c;

c = &d;

*b = 'I';

cout << *c << *b << *a << d;
```

From:

<http://elearn.bgamstetten.ac.at/wiki/> - Wiki

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:1:1_01:1_01_01



Last update: **2019/09/08 12:13**

Struktur - eine zusammengesetzter Datentyp

Mit Arrays können Variablen gleichen Typs zusammengestellt werden. In der realen Welt gehören aber meist Daten unterschiedlichen Typs zusammen. So hat ein Auto einen Markennamen und eine Typbezeichnung, die als Zeichenkette unterzubringen ist. Dagegen eignet sich für Kilometerzahl und Leistung eher der Typ Integer. Für den Preis bietet sich der Typ float an. Bei bestimmten Autohändlern könnte auch double erforderlich sein. Alles zusammen beschreibt ein Auto.

Modell

Vielleicht werden Sie einwerfen, dass ein Auto noch mehr Bestandteile hat. Da gibt es Bremscheiben, Turbolader und Scheibenwischer. Das ist in der Realität richtig. Ein Programm interessiert sich aber immer nur für bestimmte Eigenschaften, die der Programmierer mit dem Kunden zusammen festlegt. Unser Beispiel würde für einen kleinen Autohändler vielleicht schon reichen. Eine Autovermietung interessiert sich vielleicht überhaupt nicht für den Wert des Autos, aber möchte festhalten, ob es für Nichtraucher reserviert ist. Eine Werkstatt dagegen könnte sich tatsächlich für alle Teile interessieren. Ein Programm, das die Verteilung der Firmenfahrzeuge verwaltet, interessiert sich vielleicht nur für das Kennzeichen. Es entsteht also ein Modell eines Autos, das bestimmte Bestandteile enthält und andere vernachlässigt, je nachdem was das Programm benötigt. Bereits in C gab es für solche Zwecke die Struktur, die mehrere Variablen zu einer zusammenfasst. Das Schlüsselwort für die Bezeichnung solch zusammengesetzter Variablen lautet struct. Nach diesem Schlüsselwort folgt der Name des neuen Typen. In dem folgenden geschweiften Klammernblock werden die Bestandteile der neuen Struktur aufgezählt. Diese unterscheiden sich nicht von der bekannten Variablendefinition. Den Abschluss bildet ein Semikolon.

struct

Um ein Auto zu modellieren, wird ein neuer Variablentyp namens TAutoTyp geschaffen, der ein Verbund mehrerer Elemente ist.

```
struct TAutoTyp // Definiere den Typ
{
    char Marke[MaxMarke];
    char Modell[MaxModell];
    long km;
    int kW;
    float Preis;
}; // Hier vergisst man leicht das Semikolon!
```

Syntaxbeschreibung

Das Schlüsselwort struct leitet die Typdefinition ein. Es folgt der Name des neu geschaffenen Typs, hier TAutoTyp. In dem nachfolgenden geschweiften Klammerpaar werden alle Bestandteile der

Struktur nacheinander aufgeführt. Am Ende steht ein Semikolon, das man selbst als erfahrener Programmierer immer wieder einmal vergisst. Variablendefinition Damit haben wir den Datentyp TAutoTyp geschaffen. Er kann in vieler Hinsicht verwendet werden wie der Datentyp int. Sie können beispielsweise eine Variable von diesem Datentyp anlegen. Ja, Sie können sogar ein Array und einen Zeiger von diesem Datentyp definieren.

```
TAutoTyp MeinRostSammler; // Variable anlegen
TAutoTyp Fuhrpark[100];   // Array von Autos
TAutoTyp *ParkhausKarte;  // Zeiger auf ein Auto
```

Elementzugriff

Die Variable MeinRostSammler enthält nun alle Informationen, die in der Deklaration von TAutoTyp festgelegt sind. Um von der Variablen auf die Einzelteile zu kommen, wird an den Variablenname ein Punkt und daran der Name des Bestandteils gehängt.

```
// Auf die Details zugreifen
MeinRostSammler.km = 128000;
MeinRostSammler.kW = 25;
MeinRostSammler.Preis = 25000.00;
```

Zeigerzeichen

Wenn Sie über einen Zeiger auf ein Strukturelement zugreifen wollten, müssten Sie über den Stern referenzieren und dann über den Punkt auf das Element zugreifen. Da aber der Punkt vor dem Stern ausgewertet wird, müssen Sie eine Klammer um den Stern und den Zeigernamen legen.

```
TAutoTyp *ParkhausKarte = 0; // Erst einmal keine Zuordnung
ParkhausKarte = &MeinRostSammler; // Nun zeigt sie auf ein Auto
(*ParkhausKarte).Preis = 12500; // Preis für MeinRostSammler
```

Das mag zwar logisch sein, aber es ist weder elegant noch leicht zu merken. Zum Glück gibt es in C und C++ eine etwas hübschere Variante, über einen Zeiger auf Strukturelemente zuzugreifen. Dazu wird aus Minuszeichen und Größer-Zeichen ein Symbol zusammengesetzt, das an einen Pfeil erinnert.

```
ParkhausKarte->Preis = 12500;
```

L-Value

Strukturen sind L-Values. Sie können also auf der linken Seite einer Zuweisung stehen. Andere Strukturen des gleichen Typs können ihnen zugewiesen werden. Dabei wird die Quellvariable Bit für Bit der Zielvariable zugewiesen. TAutoTyp MeinNaechstesAuto, MeinTraumAuto; MeinNaechstesAuto = MeinTraumAuto;

Trotzdem die beiden Strukturvariablen nach dieser Operation ganz offensichtlich gleich sind, kann

man dies nicht einfach durch eine Anwendung des doppelten Gleichheitszeichen nachprüfen. Sie können bei Strukturen die Typdeklaration und die Variablendefinition zusammenfassen, indem der Name der Variablen direkt nach der geschweiften Klammer eingetragen wird.

```
struct // hier wird kein Typ namentlich festgelegt
{
    char Marke[MaxMarke];
    char Modell[MaxModell];
    long km;
    int kW;
    float Preis;
} MeinErstesAuto, MeinTraumAuto;
```

Hier werden im Beispiel die Variablen MeinErstesAuto und MeinTraumAuto gleich mit ihrer Struktur definiert. Werden auf diese Weise gleich Variablen dieser Struktur gebildet, muss ein Name für den Typ nicht unbedingt angegeben werden. Damit ist dann natürlich keine spätere Erzeugung von Variablen dieses Typs möglich. Initialisierung Auch Strukturen lassen sich initialisieren. Dazu werden wie bei den Arrays geschweifte Klammern verwendet. Auch hier werden die Werte durch Kommata getrennt.

```
TAutoTyp JB = {"Aston Martin", "DB5", 12000, 90, 12.95};
TAutoTyp GWB = {0};
```

From:

<http://elearn.bgamstetten.ac.at/wiki/> - Wiki

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:1:1_02

Last update: **2019/09/08 12:14**



Lineare Listen (=Einfach verkettete Listen)

Einfach **verkettete Listen** oder **linked lists** sind eine **fundamentale Datenstruktur**, die ich hier anhand von Code-Beispielen und Grafiken erklären will.

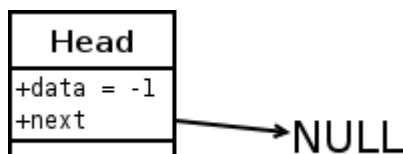
Einfach verkettete Listen zeichnen sich dadurch aus, dass man besonders einfach Elemente einfügen kann, wodurch sie sich besonders gut für Insertion Sort eignen.

Knoten

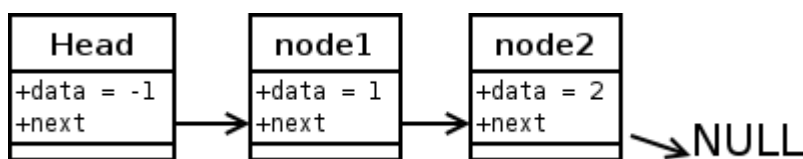
Eine einfach verkettete Liste besteht aus **Knoten**, **Englisch nodes**, die einen **Zeiger auf das nächste Element** und Daten beinhalten.

```
typedef struct listnode{  
    int data;  
    listnode *next;  
};
```

Eine **leere Liste** besteht aus einem Kopf (Head) und nichts sonst:



Wenn man mehrere Elemente einfügt, sieht das so aus:



Eine einfach verkettete Liste mit einem Kopf und zwei Knoten.

Knoten befüllen & einfügen

Wenn man einen Zeiger auf ein Element der Liste hat, ist es einfach, ein Element dahinter einzufügen.

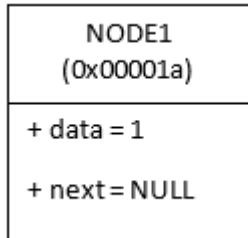
Dazu muss man den next-Zeiger der Liste auf das neue Element setzen, und den next-Zeiger des neuen Element auf den alten Wert des next-Zeigers der Liste:

```
//1. Knoten erstellen  
listnode *node1;
```

```
node1=new listnode;
```

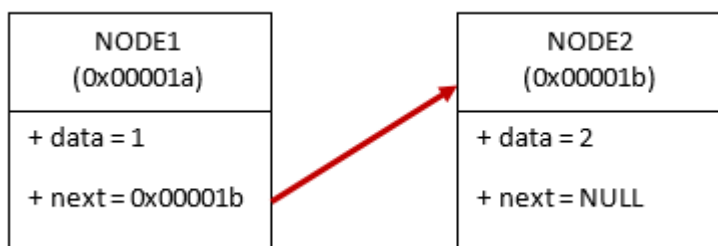
```
//1. Knoten befüllen
```

```
node1->data=1; //oder (*node1).data=1;  
node1->next=NULL;
```



```
//2. Knoten erstellen
```

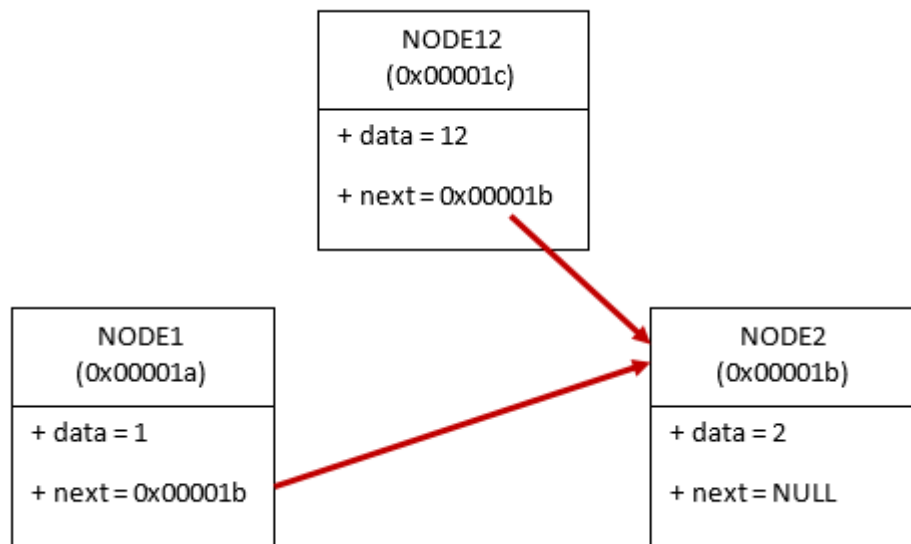
```
listnode *node2;  
node2=new listnode;  
node2->data=2;  
node2->next=NULL;  
//1. Knoten zeigt auf 2. Knoten  
node1->next=node2;
```



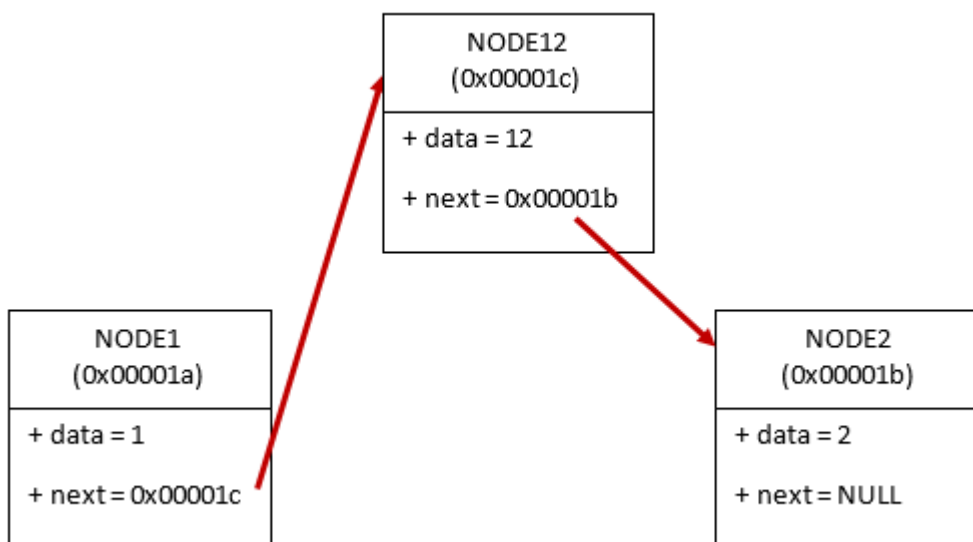
```
//Knoten 12 zwischen Knoten 1 und 2 einfügen
```

```
listnode *node12;  
node12=new listnode;  
node12->data=12;  
node12->next=node2;  
node1->next=node12;
```

Schritt 1



Schritt 2



Automatisiertes Hinzufügen von Knoten

Ein neuer Listenknoten wird durch Aufruf von `new` erzeugt. Dabei muss darauf geachtet werden, dass der Zeiger `next` gleich korrekt gesetzt wird. Wenn Sie nicht sofort den Nachfolger einhängen können, setzen Sie den Zeiger auf `NULL`.

```
#include <iostream>

using namespace std;

typedef struct listnode{
```

```
int data;
listnode *next;

};

int main(int argc, char** argv) {

    //Kopf der Liste
    listnode *head=NULL;

    //10 Knoten hinzufügen
    for(int i=1; i<10; i++)
    {
        listnode *node = new listnode;
        node->data=i;
        node->next=head;
        head=node;
    }

    //Knoten ausgeben
    listnode *help=NULL;
    help=head;

    while(help!=NULL)           //Solange Hilfszeiger nicht auf NULL zeigt
    {
        cout << help->data << endl;
        help=help->next;
    }

    return 0;
}
```

- [1.3.1\) Musterbeispiel](#)

From:

<http://elearn.bgamstetten.ac.at/wiki/> - **Wiki**

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:1:1_03

Last update: **2019/09/12 10:51**



```
#include <iostream>
using namespace std;
/* run this program using the console pauser or add your own getch,
system("pause") or input loop */

typedef struct listnode{

    int data;
    listnode *next;
};

listnode *head=NULL;

void vorneeinhaengen(int n)
{
    for(int i=0;i<n;i++)
    {
        listnode *node = new listnode;    //Anlegen eines neuen Knotens
        cout << "Wert eingeben: ";
        cin >> node->data;                //Daten in den Knoten übernehmen
        node->next=head;                  //Zeiger node->next zeigt auf ersten
        Knoten
        head=node;                        //Listenkopf ist der neue Kneten
    }
}

void listeausgeben()
{
    listnode *h=NULL;                    //Hilfszeiger
    h=head;                             //der auf den Kopf der Liste zeigt.

    while (h!=NULL)                     //Solange Hilfszeiger nicht auf NULL zeig
    {
        cout << h->data << " -> ";
        h=h->next;
    }
}

int anzahlelemente()
{
    int anzahl=0;
    listnode *h=NULL;
    h=head;
    while(h!=NULL)
    {
        anzahl++;
        h=h->next;
    }
    return anzahl;
}
```

```
}  
  
int main(int argc, char** argv) {  
  
    int anzahl=0;  
    cout << "Einfach verkettete Liste\n";  
    cout << "Geben Sie an, wie viele Elemente Sie einhaengen wollen: ";  
    cin >> anzahl;  
    vorneeeinhaengen(anzahl);  
    cout << "Ausgabe der Liste\n";  
    listeausgeben();  
    cout << "Die Anzahl aller Elemente betraegt: " << anzahlelemente();  
  
    return 0;  
  
}
```

From:

<http://elearn.bgamstetten.ac.at/wiki/> - **Wiki**

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:1:1_03:1_03_01



Last update: **2019/09/08 21:13**

2) DATENBANKEN

- 2.1) Allgemeines
- 2.2) Datenmodellierung
- 2.3) Entity Relationship Modell (Konzeptionelles Modell)
 - 2.3.1) Übungen
- 2.4) Relationenmodell (Logisches Modell)
 - 2.4.1) Übungen
- 2.5) Umsetzung ER-Modell --> Relationenmodell
 - 2.5.1) Übungen
- 2.6) Normalformen
- 2.7) SQL (Physisches Modell)
- 2.8) SQL Anbindung mit PHP
 - 2.8.1) Übung
 - SQL-ISLAND GAME - Schaffst du es den Piloten zu befreien?
 - SOLOLEARN - Play with SQL

From:

<http://elearn.bgamstetten.ac.at/wiki/> - **Wiki**

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:2

Last update: **2019/11/11 11:56**



2.1) Allgemeines

2.1.1) Definitionen

2.1.1.1) Datenbanksystem:

Ein Datenbanksystem ist ein computergestütztes System bestehend aus einer Datenbasis zur Beschreibung eines Ausschnitts der realen Welt sowie Programmen zum geregelten Zugriff auf die Datenbasis.

2.1.1.2) Datenbankverwaltungssystem (DBMS-data base managment system)

Ist jener Teil der Software die zwischen den eigentlichen Daten und den Benutzern der Daten liegt und alle Anfragen der Benutzer verarbeitet. Sie stellt jene Einrichtung zur Verfügung die notwendig sind um neue Daten anzulegen, zu löschen, abzufragen und zu verändern

2.1.2) Motivation

Die ersten Computer unterstützten Informationssysteme, wurden in Form von Einzellösungen, d.h. durch einzelne Anwendungsprogramme mit privaten Dateien realisiert. Diese Programme verwendeten unmittelbar das zugrunde liegende Dateisystem auf den jeweiligen Rechner. Gleichartige Daten wurden in separaten Dateien gespeichert, die selbst wieder aus einzelnen Datensätzen bestanden.

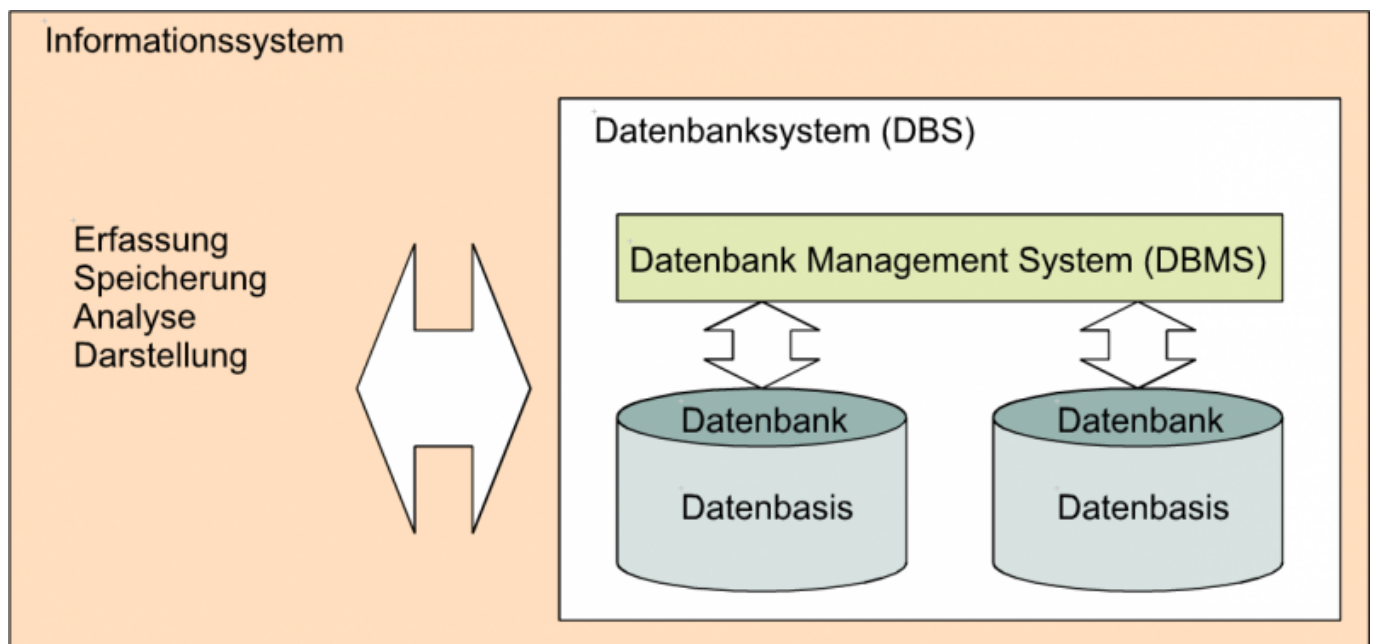
Produktion
Angestellte
Teile
Verkauf
Kunden
Teile
Fakturierung
Kunden
Teile
Personalabteilung
Kunden
Gehalt

Diese Systeme waren schwer wartbar da mehrfach verwendete Daten auch mehrfach gespeichert wurden, deshalb entstand die integrierte Datenverarbeitung bei der Dateien in mehreren Anwendungsprogrammen verwendet werden. Doch auch die separate Abspeicherung von teilweise in Beziehung stehenden Daten würde zu schwerwiegenden Problemen führen:

- **Redundanz** - Dieselben Informationen werden doppelt gespeichert.

- **Inkonsistenz** - Dieselben Informationen werden in unterschiedlichen Versionen gespeichert.
- **Integritätsverletzung** - Die Einhaltung komplexer Intigritätsbedingungen fällt schwer.
- **Verknüpfungseinschränkung** - Logisch verwandte Daten sind schwer zu verknüpfen wenn sie in isolierten Dateien liegen.
- **Mehrbenutzerprobleme** - Gleichzeitiges Editieren der Datei führt zur Anomalie (lost update).
- **Verlust von Daten** - Außer einem kompletten Backup ist kein Recovery-Mechanismus vorhanden.
- **Sicherheitsprobleme** - Abgestufte Zugriffsrechte können nicht implementiert werden.
- **Hohe Entwicklungskosten** - Für jedes Anwendungsprogramm müssen die Fragen zur Dateiverwaltung erneut gelöst werden.

Heute werden Informationssysteme meist mit Hilfe von Datenbanksystemen realisiert.



Einerseits ermöglicht diese Trennung zwischen Anwendungsprogrammen und Daten die sogenannte **physische Datenunabhängigkeit**, d.h. Programme sind von den konkreten Speicher- und Zugriffsmethoden unabhängig. Andererseits stellen Datenbanksysteme ein Datenmodell, also eine Sprache zur Beschreibung von Datenstrukturen, zur Verfügung, die es ermöglicht einzelne Programme auf speziellen logischen Darstellungen der gespeicherten Datenbank arbeiten zu lassen (logische Unabhängigkeit).

2.1.3) Funktionalität von Datenbanksystemen

2.1.3.1) Persistente Datenhaltung

Ein DBMS muss Mechanismen zur Verfügung stellen, die eine **persistente Speicherung von Daten** garantieren, d.h. dass die Daten in der DB über die Ausführungszeit von Programmen hinaus erhalten bleiben. Die Daten werden üblicherweise auf einem Hintergrundspeicher (Sekundärspeicher - HDD) persistent gehalten. Nachdem Programme nur auf Daten im Hauptspeicher (Primärspeicher) direkt zugreifen können werden Ausschnitte der Datenbank zeitweise auch in einem Teil des Hauptspeichers, dem Datenbankpuffer, verwaltet.

Nachdem ein **Plattenzugriff sehr viel länger dauert als ein Hauptspeicherzugriff**, sind spezielle Techniken notwendig um unnötige Plattenzugriffe zu vermeiden.

Spezielle Puffersatzstrategien werden verwendet um bei Platzmangel im Datenbankpuffer zu entscheiden welche Blöcke wieder auf die Platte ausgelagert werden (z.B. **least recently used, least frequently used**). Aus Effizienzgründen gruppieren sogenannten Clustertechniken Datensätze so, dass jene Datensätze, auf die oft gemeinsam zugegriffen wird, physisch benachbart gespeichert werden. Weiters werden verschiedene Indextechniken verwendet um Daten auf einem Hintergrundspeicher rasch zu finden.

2.1.3.2) Recovery

DBMS unterstützen Änderungen in der Datenbank durch Transaktionen. Eine Transaktion ist eine Folge von Aktionen (Lese- und Schreibzugriffe auf Daten in der DB), die eine Datenbank von einem konsistenten Zustand in einen anderen konsistenten Zustand überführt. Die Recoveryeinheit eines **DBMS gewährleistet die Atomarität und die Dauerhaftigkeit (Persistenz) von Transaktionen** trotz eventuell bei der Transaktion aufgetretenen Hard- oder Softwarefehlern.

Atomarität bedeutet, dass **entweder alle Aktionen** einer Transaktion ausgeführt werden **oder keine**.

Dauerhaftigkeit bedeutet dass **alle Effekte** einer einmal erfolgten Transaktion **trotz aufgetretenen Fehlern erhalten bleiben**.

Bsp. zu Atomarität: Überweisung eines Geldbetrages von einem Konto auf ein Sparbuch

- Kontostand lesen
- Kontostand schreiben
- Sparbuch lesen
- Sparbuch schreiben

Angenommen während der Überweisung tritt nach der Abbuchung aber noch vor der Aufbuchung ein Systemabsturz ein, so möchten die Kunden davon ausgehen können, dass sich nach einem Wiederanlauf der gesamte Geldbetrag noch auf dem Konto befindet.

Bsp. zu Dauerhaftigkeit:

Angenommen wir zahlen einen Millionengewinn im Lotto auf unser Konto ein, dann stellt die Dauerhaftigkeit von Transaktionen sicher, dass der Gewinn auch nach einem Systemneustart immer noch auf dem Konto liegt. Für den Wiederanlauf verwenden die meisten DBS ein Log-Protokoll. In diesem Log-Protokoll werden der Start, das Ende und der Abbruch von Transaktionen verzeichnet, sowie die von Transaktionen durchgeführten Modifikationen von Datensätzen (Einfügen, Löschen und Ändern von Datensätzen). Zu jeder Änderung wird der alte Datensatz (before image) und der neue Datensatz (after image) im Log-Protokoll verzeichnet. Beim Wiederanlauf werden alle nicht beendeten Transaktionen unter Verwendung der before-images zurückgesetzt und alle bereits erfolgreich abgeschlossenen Transaktionen nachgeholt.

2.1.3.3) Concurrency Control

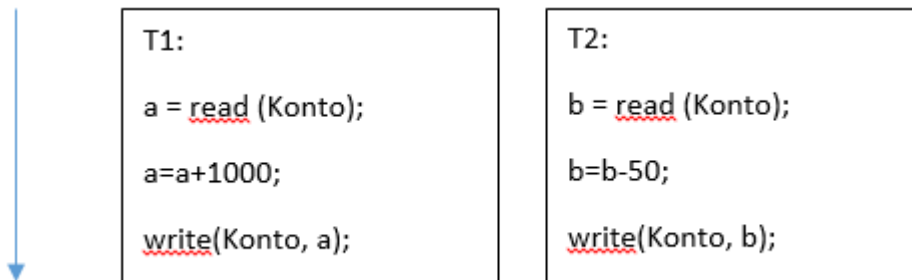
Die Concurrency Control - Einheit eines DBMS ermöglicht mehreren Benutzern eine Datenbank gemeinsam zur selben Zeit zu nutzen ohne ihre Konsistenz zu gefährden. Das traditionell verwendete **Korrektheitskriterium für parallele (oder verzahnte) Ausführung von Transaktionen im**

Mehrbenutzerbetrieb ist die Serialisierbarkeit.

Die Serialisierbarkeit garantiert folgende Eigenschaft:

Das Ergebnis der beliebigen Parallelausführung mehrerer Transaktionen entspricht dem Ergebnis irgendeiner Hintereinander-Ausführung dieser Transaktion.

Bsp.: Angenommen wir wollen unsere Telefonrechnung (50€) bezahlen. Wir gehen zur Bank, wo die Abbuchung vom Konto durchgeführt wird. Diese Abbuchung wird nun gleichzeitig mit der Gehaltsbuchung (1000€) auf das Konto durchgeführt.



Angenommen der aktuelle Kontostand beträgt 2000€. Nachdem beide Transaktionen abgeschlossen wurden, ist der Kontostand auf 1950€. Eine Hintereinanderausführung hätte aber 2950€ ergeben, d.h. diese verzahnte Ausführung ist nicht serialisierbar und daher nicht korrekt.

Um Serialisierbarkeit von Transaktionen zu gewährleisten verwenden die meisten DBMS Sperrverfahren.

Dabei legt eine Transaktion auf Datenobjekte die sie schreiben oder lesen soll, eine **Sperre**. Besitzt eine Transaktion auf einem Datenobjekt eine Sperre und fordert eine andere Transaktion für dieses Datenobjekt ebenfalls eine Sperre an, so wird diese Sperre nur dann gewährt, wenn die neu angeforderte Sperre mit der bereits bestehenden Sperre verträglich ist. Ist sie es nicht, so muss die neue Transaktion **auf die Freigabe der bestehenden Sperre warten**.

Meist werden 2 Typen von Sperren verwendet:

- Geteilte Lese-Sperren
- Exklusive Schreib-Sperren

Lesesperren verschiedener Transaktionen für dasselbe Datenobjekt sind miteinander verträglich, eine Schreibsperre ist mit keiner Sperre anderer Transaktionen verträglich.

Das Sperren von Datenobjekten ist alleine jedoch nicht ausreichend um die Serialisierbarkeit paralleler Transaktionen zu gewährleisten. Es muss darüber hinaus ein **Sperrprotokoll** eingehalten werden. Das am meisten gebräuchliche Sperrprotokoll ist das **2-Phasen-Sperrverfahren**. Eine Transaktion erfüllt das 2-Phasen-Sperrverfahren, wenn es nach der 1. Freigabe einer Sperre keine neue Sperre mehr anfordert. Weiters garantiert die Concurrency Control-Einheit eines DBMS die Isolation von Transaktionen. **Isolation** bedeutet, dass **Effekte nach einer Transaktion erst nach ihrem erfolgreichem Abschluss für andere Transaktionen sichtbar** werden.

2.1.3.4) ACID - Atomicity Consistency Isolation Durability (zu d. Dt. AKID - Atomarität, Konstistenz, Isolation, Dauerhaftigkeit)

Die Eigenschaften des Transaktionskonzeptes werden unter der Abkürzung **ACID** zusammengefasst:

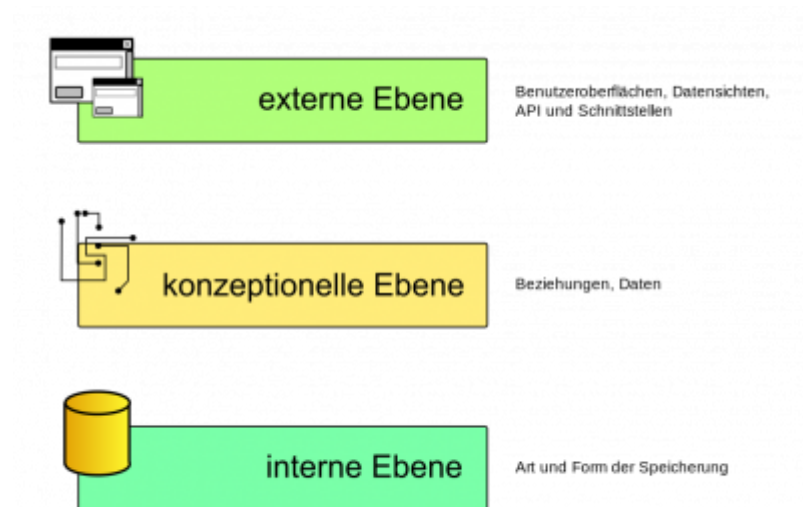
- **Atomicity:** Eine Transaktion stellt eine nicht weiter zerlegbare Einheit dar, mit dem Prinzip: „**ALLES oder NICHTS**“
- **Consistency:** Nach Abschluss der Transaktion liegt wieder ein konsistenter Zustand vor, während der Transaktion sind inkonsistente Zustände erlaubt
- **Isolation:** Nebenläufig ausgeführte Transaktionen dürfen sich nicht beeinflussen, d.h. jede Transaktion hat den Effekt, den sie verursacht hätte, als ob sie allein im System gewesen wäre.
- **Durability:** Die Wirkung einer erfolgreich abgeschlossenen Transaktion bleibt dauerhaft in der Datenbank, auch nach einem späteren Systemfehler.

2.1.3.5) Datenschutz

DBMS bieten die Möglichkeit für einzelne Benutzer oder Benutzergruppen den Zugriff auf Ausschnitte der Datenbank zu beschränken. Dabei kann hinsichtlich der Art des Zugriffs zwischen Lese-, Änderungs-, Einfüge- und Löschzugriffe unterschieden werden.

2.1.4) Architektur eines Datenbanksystems

Moderne Datenbanksysteme unterstützen alle die **ANSI-SPARC-Architektur**:



Die Architektur wurde 1975 vom Standards Planning and Requirements Committee (SPARC) des American National Standards Institute (ANSI) entwickelt und hat das Ziel, den Benutzer einer Datenbank vor nachteiligen Auswirkungen von Änderungen in der Datenbankstruktur zu schützen.

Die drei Ebenen sind:

- **Die externe Ebene**, die den Benutzern und Anwendungen individuelle Benutzersichten bereitstellt. Beispiele: Formulare, Masken-Layouts, Listen, Schnittstellen.
- **Die konzeptionelle Ebene**, in der beschrieben wird, welche Daten in der Datenbank gespeichert sind, sowie deren Beziehungen zueinander. Designziel ist hier eine vollständige und redundanzfreie Darstellung aller zu speichernden Informationen. Hier findet die Normalisierung des relationalen Datenbankschemas statt.

- **Die interne Ebene (auch physische Ebene)**, die die physische Sicht der Datenbank im Computer darstellt. In ihr wird beschrieben, wie und wo die Daten in der Datenbank gespeichert werden. Designziel ist hier ein effizienter Zugriff auf die gespeicherten Informationen. Das wird meistens nur durch eine bewusst in Kauf genommene Redundanz erreicht (z. B. im Index werden die gleichen Daten gespeichert, die auch schon in der Tabelle gespeichert sind).

Die Vorteile des Drei-Ebenen-Modells liegen in der

- **physischen Datenunabhängigkeit**, da die interne von der konzeptionellen und externen Ebene getrennt ist. Physische Änderungen, z. B. des Speichermediums oder des Datenbankprodukts, wirken sich nicht auf die konzeptionelle oder externe Ebene aus.
- **logischen Datenunabhängigkeit**, da die konzeptionelle und die externe Ebene getrennt sind. Dies bedeutet, dass Änderungen an der Datenbankstruktur (konzeptionelle Ebene) keine Auswirkungen auf die externe Ebene, also die Masken-Layouts, Listen und Schnittstellen haben.

Allgemein kann also von einer **höheren Robustheit gegenüber Änderungen** gesprochen werden.

From:

<http://elearn.bgamstetten.ac.at/wiki/> - Wiki

Permanent link:

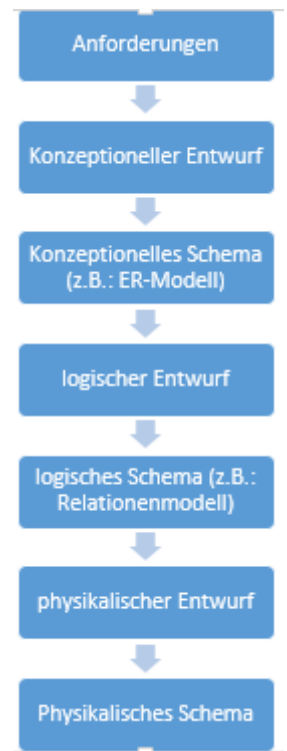
http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:2:2_01

Last update: **2019/11/13 22:56**



2.2) Datenmodellierung

Im Datenbankentwurf werden verschiedene Datenbankmodelle verwendet:



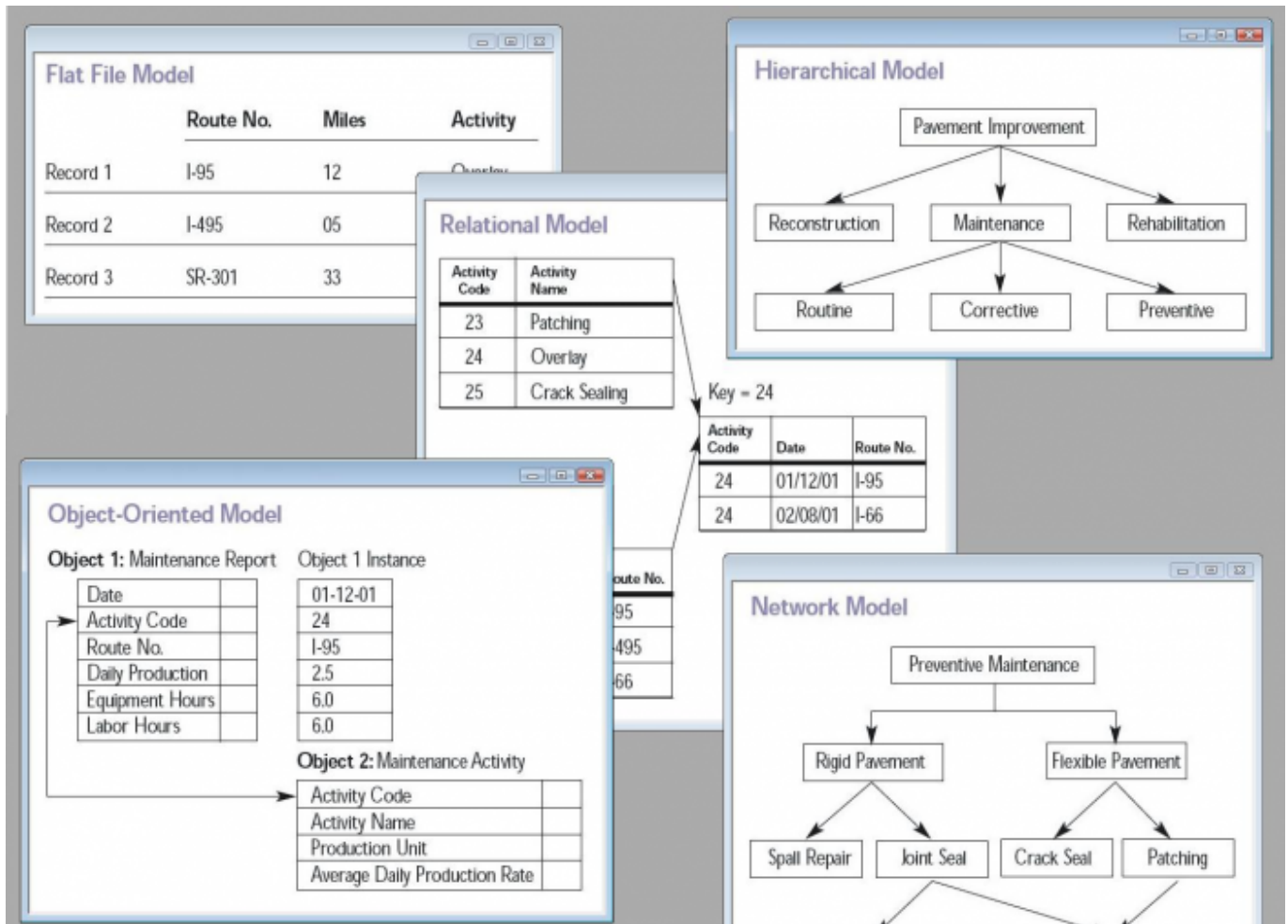
Konzeptionelle Datenmodelle (z.B.: ER-Modell) stehen **problemnahe Modellierungskonzepte** für die ersten Schritte des Datenbankentwurfs zur Verfügung und dienen zur **Kommunikation zwischen Endbenutzern und Datenbankentwicklern**.

Physische Modelle stellen **maschinennahe Konzepte** zur Verfügung und dienen zur **Beschreibung der Organisation von Daten in Dateien** sowie zur Beschreibung von **Zugriffsstrukturen** die ein rasches Einfügen, Suchen und Ändern von Daten ermöglichen.

Logische Datenmodelle dienen der **Überbrückung zwischen konzeptionellen und physischen Datenmodellen**. Sie werden oft auch als Implementierungsmodelle bezeichnet, das logische Datenmodell steht dem Entwickler zur Definition eines Datenbankschemas zur Verfügung und wird weitgehend automatisch vom DBMS in ein physisches Datenbankschema übersetzt.

Zur Formulierung des logischen Schemas stehen je nach zugrunde liegendem DBS folgende Möglichkeiten zur Wahl:

- Hierarchisches Modell
- Netzwerkmodell
- Relationales Modell
- Objektorientiertes Modell



- Anforderungen: Z.B. Banker erklärt, welche Daten (Kunden, etc.) benötigt werden,
- Konzeptioneller Entwurf/Konzeptionelles Schema: z.B. ER-Modell
- Logischer Entwurf: Ist konzeptioneller Entwurf überhaupt umsetzbar?
- Physischer Entwurf: Wo wirklich die Datenbank erstellt wird.

Ziel: Kunde soll Datenbank verstehen, nur Kunde weiß, wie das Unternehmen aufgebaut ist und welche Daten wie verarbeitet werden. Die Fehlerquote soll minimiert werden.

From:

<http://elearn.bgamstetten.ac.at/wiki/> - Wiki

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:2:2_02



Last update: **2019/11/14 10:00**

2.3) ENTITY-RELATIONSHIP-MODELL

Das ER-Modell wurde im Jahr 1976 von Chen eingeführt. In der Sicht des ER-Modells besteht die Welt aus Entities (Objekttypen) und Relationships (Beziehungen), zwischen den Objekten. Das ER-Modell wurde von Teorey im Jahr 1986 erweitert zum Extended ER-Modell (EER).

2.3.1) Grundlegende Objekte

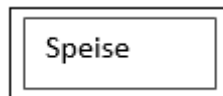
Objekte bzw. Entity-Instanzen repräsentieren unterscheidbare Objekte des Problembereichs. Objekte mit den selben Eigenschaften werden zu Entities zusammen gefasst.

- **Entity**



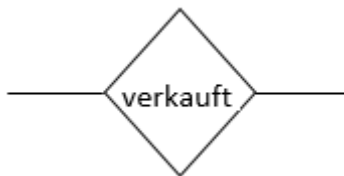
- Jede Entität muss eindeutig sein, z.B. SVNr von Person, FIN von Auto, ...

- **Weak Entity**



- Im Unterschied zur Entity ist die Weak Entity nicht eindeutig. Eine Weak Entity ist alleine nicht „überlebensfähig“. Z.B.: Die Speise braucht eine Zuordnung zum Restaurant.

Beziehungen zwischen Objekten haben keine physische oder konzeptionelle Existenz sondern ergeben sich aufgrund der vorhandenen Objekte.



Attribute sind Eigenschaften der Entities. Es wird unterschieden zwischen:

- **identifizierende Attribute = (Schlüssel, Key)** ⇒ sind jene Attribute, die die Objekte einer Entity eindeutig kennzeichnen.



- **beschreibende Attribute** \Rightarrow sind jene Attribute, die ein Objekt nicht eindeutig kennzeichnen.



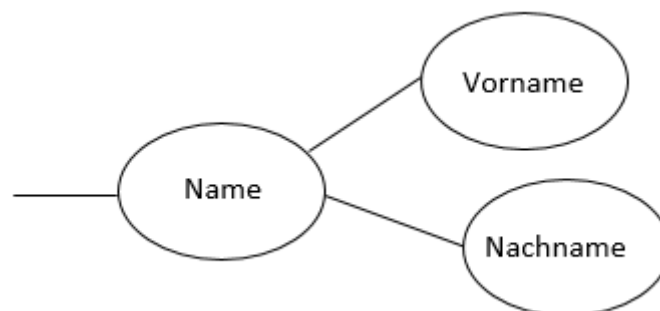
Schlüssel (Keys) können aus **einem oder mehreren Attributen** bestehen. Jede Entität kann mehrere verschiedene Schlüssel haben, wir stellen im ER-Modell aber immer **einen ausgezeichneten Schlüssel als Primärschlüssel (=Primary Key)** dar. Entitäten deren Objekte, nur mit Hilfe von anderen Entitäten identifiziert werden können, nennt man Weak Entities. Sie haben keine Attribute, die alleine den Schlüssel bilden können, obwohl sie dazu beisteuern können.

Weiters können Attribute mehrwertig sein oder komplexe Werte annehmen.

- **mehrwertige Attribute**



- **komplexe Attribute**



2.3.1.1) Komplexität von Beziehungen

Die Komplexität einer Beziehung zwischen Entities beschreibt mit vielen Instanzen, der jeweils anderen Entität jede Instanz einer Entität in Beziehung stehen kann. Bei einer Beziehung zwischen 2 Entitäten A und B gibt es folgende Möglichkeiten:

- **1:1** \Rightarrow Jedes Objekt von A gehört genau zu einem Objekt von B und umgekehrt
- **1:n** \Rightarrow Jedes Objekt von A gehört zu einem oder mehreren Objekten von B, aber jedes Objekt von B gehört genau zu einem Objekt von A.
- **n:m** \Rightarrow Jedes Objekt von A gehört zu einem oder mehreren Objekten von B und umgekehrt.

Die Darstellung einer Komplexität von Beziehung nach Chen geschieht durch Anbringung von Ziffern an die jeweiligen Entitäten. Dabei wird nur zwischen den Werten 1 und viele (n,m) unterschieden. Die genaue Anzahl der zugeordneten Objekte, d.h. die Kardinalität wird selten verwendet, da diese sich bei jeder Instanz einer Beziehung unterscheiden kann. Teorey verwendet nicht mehr die Werte neben

den Entitäten sondern färbt die Hälften der Beziehungen ein, die zu den n-Entitäten gedreht sind.

Konzept	Darstellung nach Chen	Darstellung nach Teorey
1:1		
1:n		
n:m		

2.3.1.2) Existenz einer Entität in einer Beziehung

Die Komplexität einer Beziehung gibt an mit wie vielen Instanzen der anderen Entität in Beziehung stehen kann. Dabei haben wir immer von einer oder von vielen Instanzen gesprochen. Das Konzept der Existenz einer Entität gibt uns die Möglichkeit auszudrücken, ob es immer mindestens eine Instanz geben muss oder eine Instanz geben kann.

Konzept	Darstellung nach Chen	Darstellung nach Teorey
zwingend		
optional		
unbekannt		

Bsp.:

zwingend \Rightarrow Ein Restaurant hat hat zumindest einen oder mehrere Mitarbeiter und jeder Mitarbeiter ist genau einem Restaurant zugeordnet.

optional \Rightarrow Ein Restaurant hat zwingend einen Geschäftsführer. Manche Mitarbeiter sind Geschäftsführer.

2.3.1.3) Grad einer Beziehung

Der **Grad einer Beziehung** ist die **Anzahl der Entitäten**, die in der **Beziehung miteinander verbunden** sind. **Unäre und binäre Beziehungen** kommen in der realen Welt **am häufigsten** vor.

Unäre Beziehungen (Beziehung einer Entität mit sich selbst) heißen auch binär rekursive Beziehungen.

Ternäre Beziehungen (Beziehungen zwischen 3 Entitäten) werden benötigt wenn binäre Beziehungen einen Sachverhalt nicht ausreichend beschreiben. Kann jedoch eine ternäre Beziehung mit 2 oder 3 binären Beziehungen ausgedrückt werden, so ist diese Darstellung vorzuziehen.

Grad der Beziehung	Darstellung nach Chen	Darstellung nach Teorey
unär		
binär		
ternär		

Bsp.:

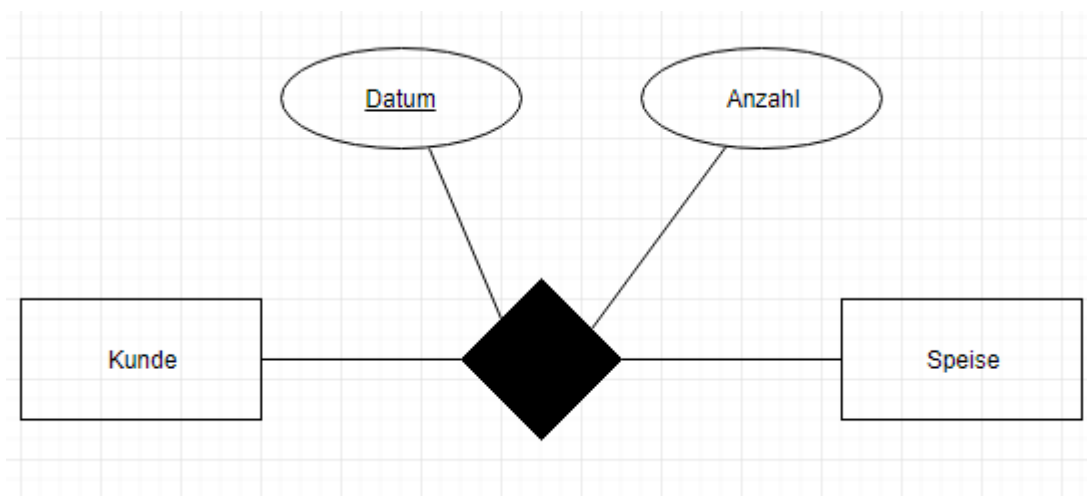
Für eine konkrete Pizzabestellung wird für die Lieferfirma ein Auto und ein Mitarbeiter abgestellt. Daher folgende ternäre Beziehung:

1. Ein Mitarbeiter verwendet für eine Bestellung ein Auto
2. Ein Auto wird bei einer Bestellung von einem Mitarbeiter gefahren.
3. Ein Mitarbeiter kann mit einem Auto mehrere Bestellungen erledigen

2.3.1.4) Attribute einer Beziehung

Attribute können wie wir bisher gesehen haben zu Entitäten hinzugefügt werden. Es gibt aber auch die Möglichkeit Attribute zu Beziehungen hinzuzufügen.

So könnte man die Attribute Anzahl und Datum zu Beziehung zwischen Kunde und Speise schreiben. Dadurch trägt man den Umstand Rechnung, dass ein Kunde eine Speise mehrmals bestellen kann. Würde man Attribut Anzahl zum Kunden hinzufügen würde man ein mehrwertiges Attribut erhalten und zudem die Information verlieren, welche Speise der Kunde zu welchem Datum bestellt hat.



Attribute werden üblicherweise nur zu n:m Beziehungen hinzugefügt, da im Falle von 1:1 oder 1:n zumindest auf einer der beiden Seiten der Beziehung ein einziges Objekt steht und damit die Mehrdeutigkeit kein Problem darstellt.

Wenn Attribute zu n:m Beziehungen geschrieben werden, so muss man überlegen, ob man nicht an Stelle von n:m Beziehungen eine Weak-Entität modelliert.

Denn jede n:m Beziehung entspricht einer Weak-Entität die durch eine 1:n Beziehung mit den beiden anderen Entitäten verbunden ist.

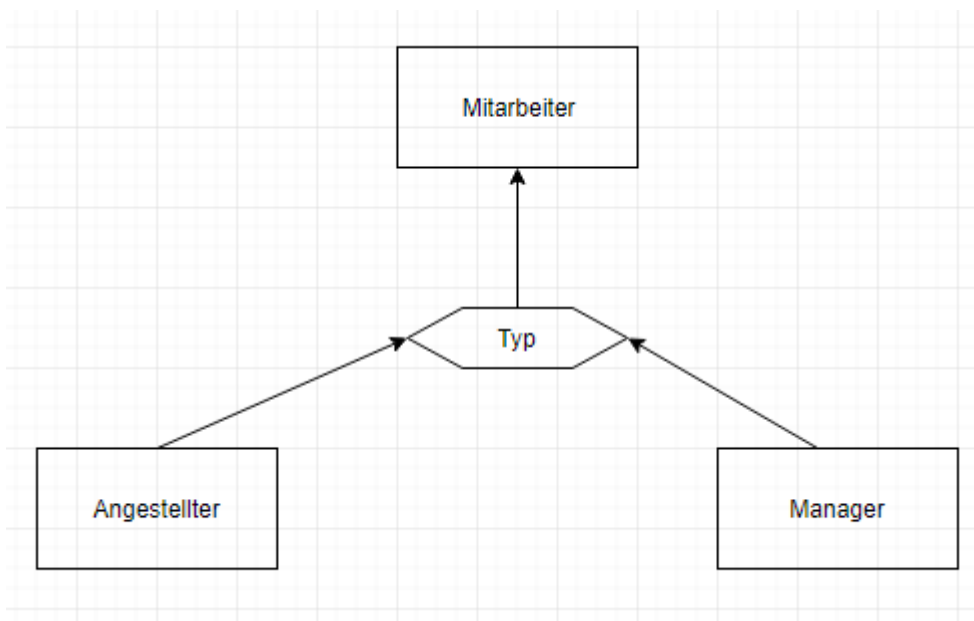
2.3.2) Erweiterte ER-Konstrukte: Die Generalisierung

Bei der Modellierung der Entities Mitarbeiter und Kunden wird man feststellen, dass sie viele gemeinsame Attribute besitzen. Diese können zu einer Entity Person verallgemeinert werden. Mitarbeiter und Kunden werden jeweils mit einer 1:1 Beziehung zu Person verbunden. Die Attribute der Person werden entlang der Hierarchie vererbt und spezielle Attribute wie die SVN-R der Mitarbeiter nur bei den speziellen Entities gespeichert.

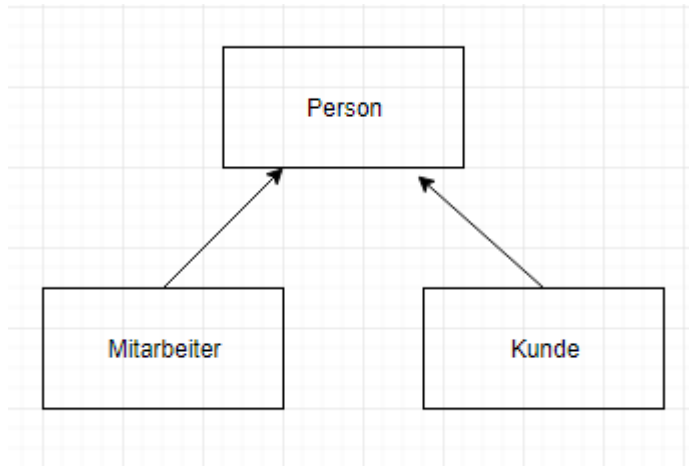
Die Generalisierung gibt an, dass mehrere Entities (Subtyp-Entity) mit bestehenden gemeinsamen Attributen zu einer Entity auf einer höheren Ebene (Supertyp-Entity) generalisiert werden können.

Die Disjunktheit der Generalisierbarkeit beschreibt ob die einzelnen Subtyp-Entities **disjunkt** oder **überlappend** sind.

- **disjunkte Spezialisierung:** Entität kann zu maximal einem Untertyp gehören (Mitarbeiter ist entweder Angestellter oder Manager)



- **überlappende Spezialisierung:** Entität kann zu einem oder mehreren Untertypen gehören (Person kann ein Mitarbeiter oder/und eine Kunde sein)



From:

<http://elearn.bgamstetten.ac.at/wiki/> - **Wiki**

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:2:2_03



Last update: **2019/11/14 10:50**

2.3.1) Übungen

2.3.1.1) Universität

Ein Institut hat eine eindeutige Nummer, einen Namen und eine Adresse. Ein Lektor identifiziert sich anhand seiner Sozialversicherungsnummer, und hat einen Namen. Er ist genau einem Institut zugeordnet, ein Institut kann keine oder mehrere Lektoren haben. Weiters gibt es Lehrveranstaltungen, wobei diese eine eindeutige Nummer haben und einen Titel. Ein Lektor kann mehrere Lehrveranstaltungen leiten, eine Lehrveranstaltung kann von mehreren Lektoren geleitet werden, hat aber mindestens einen Leiter.

2.3.1.2) Familienstammbaum

Jemand will seine Vorfahren (und sonstige Verwandte) in einer Datenbank speichern. Jede Person wird durch ihren Vornamen, Geburtstag und Geburtsort identifiziert. Des Weiteren wird ihr Zuname gespeichert. Bei verstorbenen Personen wird auch der Sterbetag und -ort in der Datenbank verwaltet. Bei den Personen wird zwischen Männern und Frauen unterschieden. Für jede Person wird auch der Vater und die Mutter gespeichert, falls diese bekannt sind.

Personen können andere Personen adoptieren. Dabei wird der Tag der Adoption gespeichert. Des Weiteren sollen alle Ehen mit dem Tag der Hochzeit und, falls die Ehe schief gehen sollte, dem Tag der Scheidung vermerkt werden. Paare können nur einmal heiraten.

2.3.1.3) Friedhof

Für einen Friedhof wird zur Arbeitserleichterung der Verwaltung eine Datenbank erstellt. Unterstreichen Sie je Relation einen Schlüssel. Verwenden Sie nur die vorgegebenen Attributnamen. (Diese sind nur bei ihrer jeweils ersten Erwähnung angeführt.)

Auf dem Friedhof gibt es viele Gräber. Jedes Grab hat eine eindeutige Nummer (GNR), eine Lagebeschreibung (LAGE), einen Besitzer (BESITZER) und eine maximale Sarganzahl (MAXSARG). In einem Grab können sich mehrere Verstorbene befinden. Särge haben eine eindeutige Bestellnummer (BNR) und einen Hersteller (HERSTELLER).

Weiters gibt es Verstorbene, von denen die eindeutige Totenscheinnummer (TNR), das Sterbedatum (SDATUM), das Geburtsdatum (GDATUM), der Vorname (VORNAME) und der Nachname (NACHNAME) bekannt sind. Es ist auch bekannt, in welchem Sarg der Verstorbene liegt. Man kann die Position (POSITION) von Verstorbenen relativ zu einem anderen Verstorbenen im selben Grab angeben (z. B. kann ein Verstorbener rechts, links, unter . . . einem anderen Verstorbenen liegen).

Es gibt 3 Friedhofsgärtner mit eindeutiger Sozialversicherungsnummer (SVNR), und Vor- und Nachnamen (VORNAME, NACHNAME), die für die Betreuung der Gräber zuständig sind. Es ist bekannt, welcher von den Gärtnern für ein bestimmtes Grab verantwortlich ist.

Man kann bei der Friedhofsgärtnerei verschiedene Dienstleistungen bestellen (z. B. Pflanzen setzen, Kerzen zu Allerheiligen, . . .). Jede Dienstleistung wird durch eine Nummer (DNR), eine Beschreibung

(BESCHREIBUNG) und einen Preis (PREIS) beschrieben. Bestellungen beziehen sich immer auf Gräber und Dienstleistungen, es wird das Datum (DATUM) angegeben und die Person (PERSON), die die anfallende Rechnung bezahlt. Für die Statistik wird mitprotokolliert, wann (DATUM) welcher Gärtner welche Dienstleistung bei einem Grab durchgeführt hat und wie viele Stunden (STUNDEN) er dafür gebraucht hat.

2.3.1.4) Politik

Um den Überblick über das Kommen und Gehen der politischen Akteure zu behalten bittet Sie ein Freund um eine Datenbank. Zeichnen Sie aufgrund der vorliegenden Informationen ein ER-Diagramm. Achtung!! Beachten Sie, dass der unten beschriebene Sachverhalt stark vereinfacht ist und nicht notwendigerweise mit der Realität übereinstimmt. Modellieren Sie bitte auf jeden Fall den angegebenen Sachverhalt!

Zu jeder Person wird ihr Vorname (VNAME), Nachname (NNAME) sowie eine besondere Eigenschaft (EIGENSCHAFT) gespeichert. Es kann keine zwei Personen mit dem selben Namen (gleicher Vorname und gleicher Nachname) geben.

Parteien besitzen eine eindeutige Farbe (FARBE), und darüber hinaus ein (nicht notwendiger Weise eindeutiges) Kürzel (KRZL).

Jede Legislaturperiode ist durch ihren Beginn (VON) gemeinsam mit ihrem Ende (BIS) identifizierbar. Jeder Aufgabenbereich der Regierung hat eine eindeutige Bezeichnung (BEZ). Außerdem gibt es eine Beschreibung (BESCHREIBUNG) zu jedem Aufgabenbereich.

Es wird vermerkt welche Person in welcher Legislaturperiode welche Aufgaben übernimmt. Außerdem wird gespeichert welcher Aufgabenbereich in welcher Legislaturperiode in welchem Ministerium angesiedelt war. Ministerien haben einen eindeutigen Namen (NAME) und ein Budget für Werbung (WBUDGET). In jedem Ministerium muss mindestens ein Aufgabenbereich angesiedelt sein (in irgendeiner Legislaturperiode). Außerdem gibt es in jeder Legislaturperiode mindestens drei Aufgabenbereiche.

Es soll außerdem vermerkt werden wie viele Stimmen jede Partei in den verschiedenen Legislaturperioden hatte.

Jeder Parteieintritt erhält eine innerhalb der entsprechenden Partei eindeutige Nummer (NR), es wird das Datum des Eintritts (DATUM) gespeichert, sowie welche Person eingetreten ist (bei jedem Parteieintritt tritt genau eine Person einer Partei bei).

From:

<http://elearn.bgamstetten.ac.at/wiki/> - Wiki

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:2:2_03_01



Last update: **2019/11/11 11:57**

2.4 Relationenmodell

Das Relationenmodell wurde 1970 von Codd entwickelt. Es **repräsentiert** die **Daten einer Datenbank als eine Menge von Relationen**. Eine **Relation** können wir uns dabei **als Tabelle vorstellen**, der **Zeilen Objekte oder Beziehungen zwischen Objekten beschreiben**. Der **Tabellenkopf** heißt **Relationenschema**, die einzelnen **Spalten** werden als **Attribute** bezeichnet, die einzelnen **Zeilen als Tupel**.

Restaurant

ID	Name	PLZ
R1	Mc	3300
R2	Hw	3300
R3	Kb	3300

„Restaurant“ ist ein Relationenschema. ID, Name, PLZ ist Attribut, R1/Mc/3300 etc. sind Tupel

2.4.1 Formalisierung

Ein **Relationenschema R** ist eine **endliche Menge von Attributnamen** $\{A_1, A_2, \dots, A_n\}$. Zu jedem Attributnamen A_i gibt es eine Menge D_i , $1 \leq i \leq n$, den **Wertebereich (=domain)** von A_i , der auch mit $\text{Dom}(A_i)$ bezeichnet wird.

Eine **Relation r(R)** auf einem **Relationenschema R** ist eine endliche Menge von Abbildungen $\{t_1, \dots, t_m\}$ von R nach D. Die Abbildungen werden **Tupel** genannt.

Bsp.: Relationenschemata für RESTAURANT und SPEISE

Restaurant = {rnr, name, adresse, haube, typ}
 Speise = {name, preis, rnr}

rnr	name	adresse	haube	typ
1
2
3	„Green Cottage“	„Kettenbrückengasse 3, 1050 Wien“	2	„chinesisch“
4

Bsp.: Domänen für das Relationenschemata RESTAURANT:

$\text{Dom}(\text{rnr})$ = Menge aller Integer
 $\text{Dom}(\text{name})$ = Menge aller Namen
 $\text{Dom}(\text{haube})$ = $\{k \mid 0 \leq k \leq 4\}$

Bsp.: Die Tabelle Restaurant hat mehrere Tupel. Eines von ihnen ist t3 mit:

$t3(\text{rnr}) = 3$


```
t3(name)      = "Green Cottage"
t3(haube,typ) = 2, "chinesisch"
t3           = 3, "Green Cottage", "Kettenbrückengasse 3, 1050 Wien", 2,
"chinesisch"
```

Ein Schlüssel einer Relation $r(R)$ ist eine Teilmenge K von R , sodass für zwei verschiedene Tupel t_1 und t_2 aus $r(R)$ immer $t_1(K) \neq t_2(K)$ gilt und keine echte Teilmenge K' von K diese Eigenschaft hat.

Im Allgemeinen wird dabei ein Schlüssel als Primärschlüssel ausgezeichnet und in den Relationenschemata durch Unterstreichen der Schlüsselattribute gekennzeichnet.

Bsp.:

```
Restaurant = {__rnr__, name, adresse, haube, typ}
Speise     = {__name__, preis, rnr}
```

2.4.2 Operationen auf Relationen

Um Operationen auf Relationen durchführen zu können, wurde die relationale Algebra eingeführt.

2.4.2.1 Mengenoperationen

Zu den Mengenoperationen gehören:

- Durchschnitt („ \cap “)
- Vereinigung („ \cup “)
- Differenz („ $-$ “)

von Relationen, die über der gleichen Attributmenge mit derselben Anordnung (identische Reihenfolge der Attribute) definiert sind:

Bsp.: Es sind 2 Relationen r und s gegeben:

r

A	B	C
a1	b1	c1
a1	b2	c1
a2	b1	c2

s

A	B	C
a1	b2	c1
a2	b2	c1
a2	b2	c2

Gesucht sind:

- Durchschnitt ($r \cap s$)
- Vereinigung ($r \cup s$)
- Differenz ($r - s$)
- Differenz ($s - r$)

Lösung:

- Durchschnitt ($r \cap s$)

A	B	C
a1	b2	c1

- Vereinigung ($r \cup s$)

A	B	C
a1	b1	c1
a1	b2	c1
a2	b1	c2
a2	b2	c1
a2	b2	c2

- Differenz ($r - s$)

A	B	C
a1	b1	c1
a2	b1	c2

- Differenz ($s - r$)

A	B	C
a2	b2	c1
a2	b2	c2

2.4.2.2 Die Selektion (" σ ")

Bei der Selektion werden Zeilen ausgewählt, die einem bestimmten Kriterium entsprechen:

$$\sigma_{\{A=a\}}(r) = \{t \in r \mid t(A)=a\}$$

Beispiele:

a) $\sigma_{\{A=a1\}}(r) = ?$

A	B	C
a1	b1	c1
a1	b2	c1

b) $\sigma_{\{A=a1\}}(s) = ?$

A	B	C
a1	b2	c1

Noch allgemeiner wird die Selektion, wenn wir erlauben, wohldefinierte Operatoren auf den Attributwerten auszuführen, z.B.: arithmetische Operationen auf Zahlenwerte und logische Verknüpfungen von Attributen durch „und“ („ \wedge “), „oder“ („ \vee “) und Negationen („ \neg “).

c) $\sigma_{\{(B=b1) \wedge \neg(A=a1)\}}(s) = ?$

d) $\sigma_{\{(haube > 1) \wedge \neg(typ=\text{österreichisch} \vee typ=\text{international})\}}(\text{Restaurant}) = ?$

Bsp: Relation Restaurant

nr	name	adresse	haube	typ
1	Schnitzelhaus	Amstetten	1	österreichisch
2	Brauhaus	Amstetten	3	international
3	Pizza Hollywood	Amstetten	2	italienisch

Lösung \Rightarrow Folgender Tupel:

3	Pizza Hollywood	Amstetten	2	italienisch
---	-----------------	-----------	---	-------------

2.4.2.3 Die Projektion (" π ")

Bei der Projektion werden gewisse Spalten einer Tabelle ausgewählt. Man projiziert nach einer Teilmenge der Attribute:

$$\pi_X(r) = \{t(X) \mid t \in r\} \text{ für } X \subseteq R$$

Beispiele:

e) $\pi_{\{A,B\}}(r) = ?$

f) $\pi_{\{\text{name, adresse, haube}\}}(\text{Restaurant}) = ??$

name	adresse	haube
Schnitzelhaus	Amstetten	1
Brauhof	Amstetten	3
Pizza Hollywood	Amstetten	2

2.4.2.4 Der Verbund

2.4.2.4.1 Der natürliche Verbund

Der Verbundoperator verknüpft zwei Relationen über ihre gemeinsamen Attribute:

$r \bowtie s = \{ \{ \{ R, S \} \mid \exists t_r \in r \text{ und } \exists t_s \in s : t_r = t(R) \text{ und } t_s = t(S) \} \}$

Der Verbundoperator ist kommutativ.

Beispiel:

Relation r

A	B
a1	b1
a2	b1
a3	b2

Relation s

B	C
b2	c1
b2	c2
b1	c3
b3	c4

Relation $r \bowtie s$

A	B	C
a1	b1	c3
a2	b1	c3
a3	b2	c1
a3	b2	c2

Aufgabe:

Restaurant \bowtie Speise

Lösung:

rnr	name	adresse	haube	typ	preis
-----	------	---------	-------	-----	-------

Erklärung:

Der natürliche Verbund verknüpft beide Relationen über die gemeinsamen Attribute und gibt daher jene Tupel aus, bei denen sowohl der Name **name** des Restaurants und der Name **name** der Speise als auch die Restaurantnummer **rnr** des Restaurants und jene der Speise gleich sind. In unserem Fall ist das die leere Menge!

Projektionseigenschaften des Verbundoperators

Seien R und S zwei Relationenschema, $q = r \bowtie s$ und $r' = \pi_R(q)$, dann gilt $r' \subseteq r$. Joinen wir also eine Relation r mit einer anderen und projizieren dann nach den ursprünglichen Attributen von r, so können unter Umständen Tupel verloren gehen.

Beispiel:

Relation r

A	B
a	b
a	b'

Relation s

B	C
b	c

Relation $r \bowtie s = q$

A	B	C
a	b	c

Relation $\pi_{\{AB\}}(q) = r'$

A	B
a	b

2.4.2.4.2 Das Kartesische Produkt

Falls $R \cap S = \{\}$, die beiden Relationenschemata also kein gemeinsames Produkt haben, so liefert die Verknüpfung $r \bowtie s$ das Kartesische Produkt, geschrieben als $r \times s$.

Beispiel:

Relation r

A	B
a1	b1
a2	b1

Relation s

C	D
c1	d1
c2	d1
c2	d2

Relation $r \times s = r \bowtie s$

A	B	C	D
a1	b1	c1	d1
a1	b1	c2	d1
a1	b1	c2	d2
a2	b1	c1	d1
a2	b1	c2	d1
a2	b1	c2	d2

Wie wir oben gesehen haben, ist das Kartesische Produkt nur für den Fall definiert, dass $R \cap S = \{\}$. Möchte man das Kartesische Produkt von Relationen bilden, die gemeinsame Attribute haben, so müssen diese in einer der Relationen umbenannt werden. Ist etwa $R = \{A, B, C\}$ und $S = \{A, B, D\}$, so benennen wir die Attribute von S um, so dass $S = \{A', B', D\}$ oder kennzeichnen sie durch Voranstellen des Relationennamens, also $S = \{S.A, S.B, D\}$.

2.4.2.4.3 Weitere Verbundarten

Weitere Verbundarten sind:

- der Gleichverbund (equi-join)
- der Theta-Verbund (theta-join)
- der Semi-Verbund (semi-join)
- der Äußere Verbund (outer-join)

2.4.2.5 Division

Möchte man die Relation r durch s dividieren, so muss die Attributmenge von s eine Teilmenge der Attributmenge von r sein. Das Ergebnis hat die Differenz der Attributmengen als Attribute und wählt jene Tupel aus r aus, die eingeschränkt auf die Differenz der Attribute R-S für alle Tupel aus s denselben Wert haben.

Beispiel:
Relation R

A	B	C	D
a	b	c	d
a	b	e	f
b	c	e	f
e	d	c	d
a	b	d	e
e	d	e	f
a	d	e	f

Relation S

C	D
c	d
e	f

Relation $R \div S$

A	B
a	b
e	d

Sowohl bei $|a|b|$ als auch bei $|e|d|$ kommen beide Tupel der Relation S, nämlich $|c|d|$ und $|e|f|$ vor.

From:

<http://elearn.bgamstetten.ac.at/wiki/> - Wiki

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:2:2_04

Last update: **2019/11/21 10:02**



Übungen zum Relationenmodell

Übung 1

Relation Restaurant			
rrnr	RName	Adr	Haube
1	McDonalds	Amstetten	0
2	McDonalds	Ybbs	0
3	Hollywood	Amstetten	1
4	Casa Venezia	Amstetten	2
5	grill.Bar	Amstetten	2
6	Schinakel	Grein	1

Relation Speise		
rrnr	SName	Preis
1	Hamburger	1
2	Hamburger	1
2	Cheeseburger	1,30
3	Hawaii	8,6
4	Hawaii	9
5	Salat	5
1	Salat	1,80

Abfragen in relationaler Algebra:

- a) alle Restaurants, die in Amstetten zu finden sind
- b) alle Restaurants von Amstetten mit mindestens 2 Hauben
- c) Namen aller Restaurants
- d) Namen aller Restaurants die in Amstetten zu finden sind
- e) alle Restaurants, die auch Speisen anbieten
- f) Namen der Restaurants, die Speisen anbieten
- g) Namen aller Restaurants, die einen Salat anbieten
- h) Namen und Preise aller Speisen samt Name des Restaurants, von Restaurants, die einen Salat anbieten

Übung 2

Gegeben sind einige Relationen und Abfragen. Formulieren Sie die Abfragen mittels Relationaler Algebra und berechnen Sie auch das Ergebnis der Abfragen.

Die Relation **Rechner** beschreibt die Rechner eines Institutes. **RNr** ist eine eindeutige Bezeichnung für den Rechner, **StudAss** ist der eindeutige Name des Studienassistenten, der den Rechner (und die darauf installierten Programme) wartet, **Speicher** gibt die Größe der Festplatte des Rechners an und **Leist** ist ein Maß für die Leistungsfähigkeit des Rechners, wobei 1 die schlechteste und 5 die beste Leistung ist.

Relation Rechner			
<u>RNr</u>	StudAss	Leist	Speicher
R1	Huber	1	100
R2	Brunner	3	80
R3	Brunner	3	400
R4	Vogt	2	120
R5	Huber	2	500

In **Programm** sind die Programme, die das Institut besitzt gespeichert. **PNr** ist eine eindeutige Nummer des Programms, **PName** ist sein Name, **Bereich** gibt an, um was für eine Art von Programm es sich dabei handelt und **MinLeist** bezeichnet die Leistungsfähigkeit, die ein Rechner mindestens besitzen muss, damit das Programm auf ihm laufen kann. Hat ein Programm also die **MinLeist** 4, so kann man ihn nur in einem Rechner mit **Leist** 4 oder 5 einsetzen, nicht aber einem mit **Leist** 1, 2 oder 3.

Relation Programm			
<u>PNr</u>	PName	MinLeist	Bereich
P1	DrawIt	1	Grafik
P2	AskIt	3	Datenbank
P3	Writelt	1	Text
P4	ConnectIt	2	Internet
P5	PaintIt	2	Grafik
P6	StoreIt	3	Datenbank

Die Relation **Assistent** beschreibt die Assistenten, die an dem Institut arbeiten. (Diese sind von den Studienassistenten verschieden) Dabei ist **ANr** eine eindeutige Nummer für den Assistenten, **AName** ist dessen Name, **StudAss** ist der eindeutige Name des Studienassistenten, der den Assistenten bei seiner Tätigkeit unterstützt. **Gehalt** bezeichnet die Gehaltsstufe des Assistenten.

Relation Assistent			
<u>ANr</u>	AName	StudAss	Gehalt
A1	Novak	Brunner	3

Relation Assistent			
<u>ANr</u>	<u>AName</u>	<u>StudAss</u>	<u>Gehalt</u>
A2	Dvorak	Vogt	1
A3	Husak	Vogt	1
A4	Pfeiffer	Brunner	2

In der Relation **Installation** ist verzeichnet, welche Programme auf welchen Rechnern installiert sind. **RNr** und **PNr** geben den entsprechenden Rechner und das Programm an, **Platz** gibt die Größe der Installation auf der Festplatte an und **Code** ist ein Code, den nur die Studienassistenten verstehen.

Relation Installation			
<u>RNr</u>	<u>PNr</u>	<u>Platz</u>	<u>Code</u>
R1	P1	500	X
R1	P3	300	z
R2	P6	300	X
R2	P2	200	pp
R3	P1	400	c
R3	P2	100	tt
R3	P3	500	c
R3	P4	200	pp
R3	P5	200	z
R3	P6	100	t
R4	P5	1000	T
R5	P1	200	p
R5	P5	100	ccc

In der Relation **Benutzung** wird vermerkt, wie lange die Assistenten die verschiedenen Programme benutzen. **ANr** verweist auf den Assistenten, **PNr** auf das Programm und **Stund** gibt an wieviele Stunden am Tag das Programm vom Assistenten pro Tag höchstens benötigt wird.

Relation Benutzung		
<u>ANr</u>	<u>PNr</u>	<u>Stund</u>
A1	P1	5
A1	P2	3
A2	P1	6
A2	P4	2
A2	P5	5
A3	P1	7
A3	P3	3
A4	P1	1
A4	P4	4

Abfragen:

- a) Geben Sie den Namen der Assistenten aus, die ein Programm möglicherweise länger als 5 Stunden benutzen.
- b) Wie heißen die Programme, die von Huber gewartet werden?
- c) Wie heißen die Programme, die von allen Assistenten benutzt werden?
- d) Auf welchem Rechner sind dieselben Programme installiert, wie auf dem Rechner R1?
- e) Von welchen Studienassistenten wird das Programm Writelt nicht gewartet?
- f) Welche Paare von Assistenten werden vom gleichem Studienassistenten betreut? (Dabei soll jedes Paar nur einmal ausgegeben werden)
- g) Welche Studienassistenten betreuen sowohl Assistenten, als auch Rechner?
- h) Welche Paare von Rechner haben dieselbe RLeistung?
- i) Welche Programme (gesucht sind die Namen) sind auf allen Rechnern installiert?
- j) Welche Programme laufen auf einem Rechner, der genau die minimale Leistungsfähigkeit für das Programm besitzt? (Gesucht sind die Paare aus Rechnernummer und Programmnummer)

Hier noch einmal alle Relationen auf einen Blick:

Relation Rechner				Relation Programm				Relation Assistent				Relation Installation				Relation Benutzung		
<u>RNr</u>	StudAss	Leist	Speicher	<u>PNr</u>	PName	MinLeist	Bereich	<u>ANr</u>	AName	StudAss	Gehalt	<u>RNr</u>	<u>PNr</u>	Platz	Code	<u>ANr</u>	<u>PNr</u>	Stund
R1	Huber	1	100	P1	Drawlt	1	Grafik	A1	Novak	Brunner	3	R1	P1	500	X	A1	P1	5
R2	Brunner	3	80	P2	Asklt	3	Datenbank	A2	Dvorak	Vogt	1	R1	P3	300	z	A1	P2	3
R3	Brunner	3	400	P3	Writelt	1	Text	A3	Husak	Vogt	1	R2	P6	300	X	A2	P1	6
R4	Vogt	2	120	P4	Connectlt	2	Internet	A4	Pfeiffer	Brunner	2	R2	P2	200	pp	A2	P4	2
R5	Huber	2	500	P5	Paintlt	2	Grafik					R3	P1	400	c	A2	P5	5
				P6	Storelt	3	Datenbank					R3	P2	100	tt	A3	P1	7
												R3	P3	500	c	A3	P3	3
												R3	P4	200	pp	A4	P1	1
												R3	P5	200	z	A4	P4	4
												R3	P6	100	t			
												R4	P5	1000	T			
												R5	P1	200	p			
												R5	P5	100	ccc			

From:

<http://elearn.bgamstetten.ac.at/wiki/> - **Wiki**

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:2:2_04_01

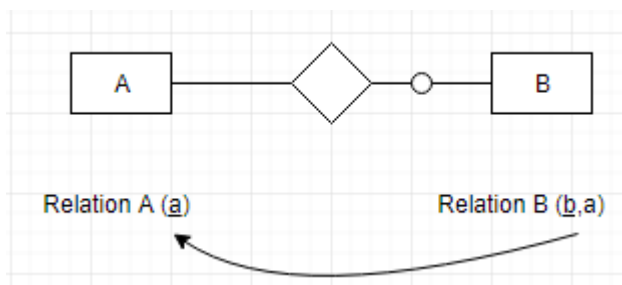


Last update: **2019/11/27 19:40**

2.5 Umsetzung des ER-Modells in ein Relationenmodell

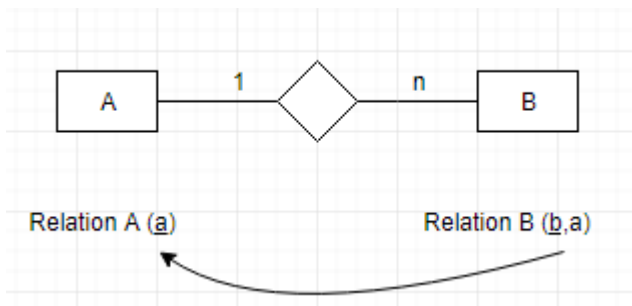
2.5.1 1-zu-1 Beziehung

Bei einer 1-zu-1 Beziehung zwischen Entitäten wird die Beziehung aufgelöst, indem der Schlüssel der einen Entität zur zweiten Entität hinzukommt. Welche Richtung hier verwendet wird, ist dem Designer überlassen. Wenn allerdings, eine optionale Beziehung besteht, wird der Schlüssel auf der optionalen Seite gespeichert.



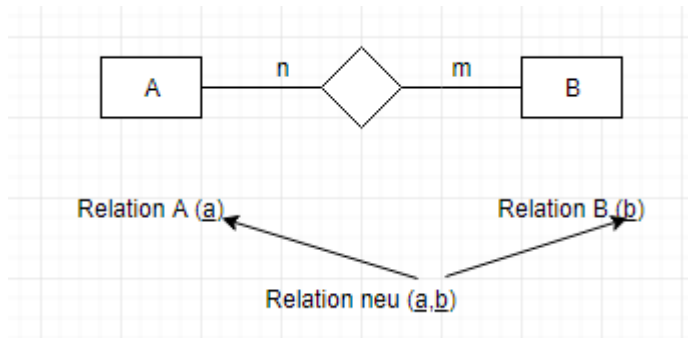
2.5.2 1-zu-n Beziehung

Im Falle einer 1-zu-n Beziehung schreiben wir den Schlüssel der 1-Seite in die Relation die der Entität auf der n-Seite entspricht.



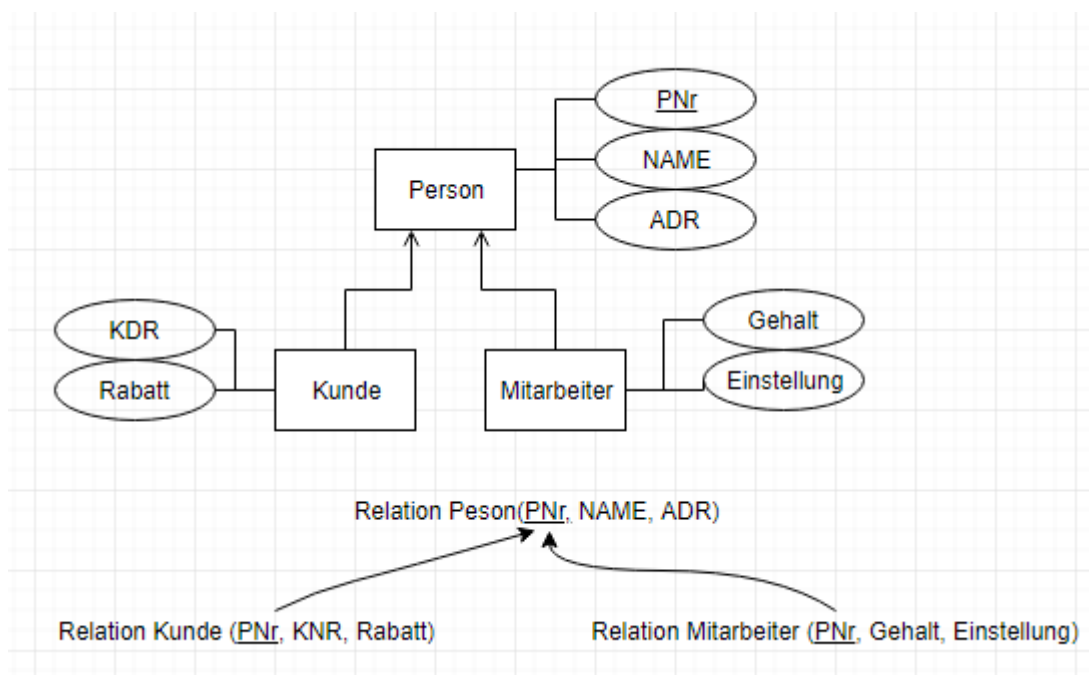
2.5.3 n-zu-m Beziehung

Bei einer n-zu-m Beziehung führen wir eine neue Relation ein, die die Schlüssel beider Entitäten als Schlüssel besitzt.

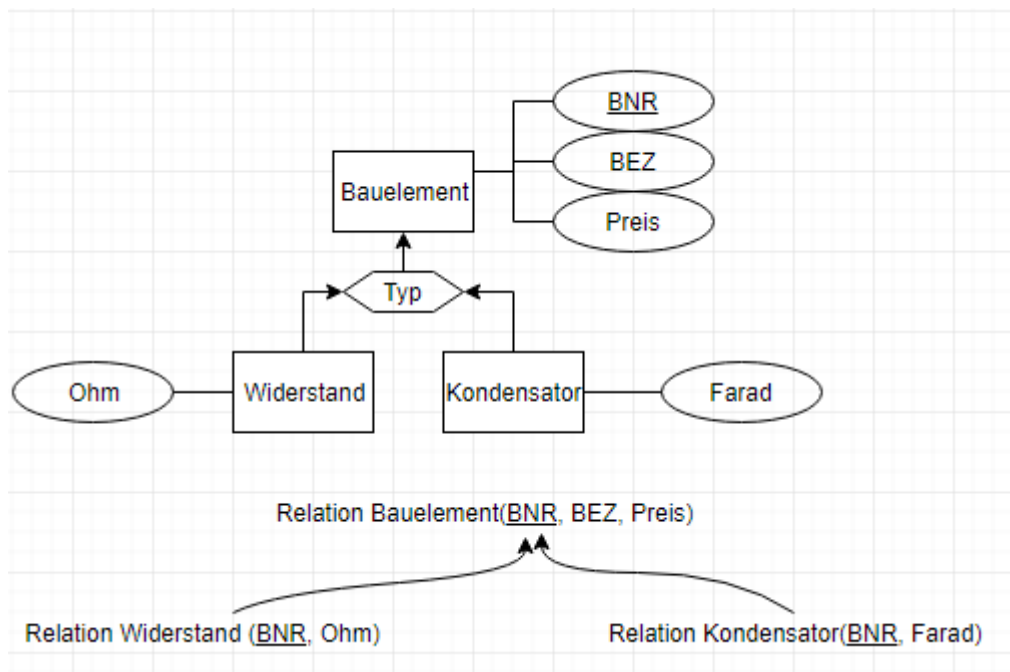


2.5.4 Generalisierung

2.5.4.1 nicht disjunkte Entitäten

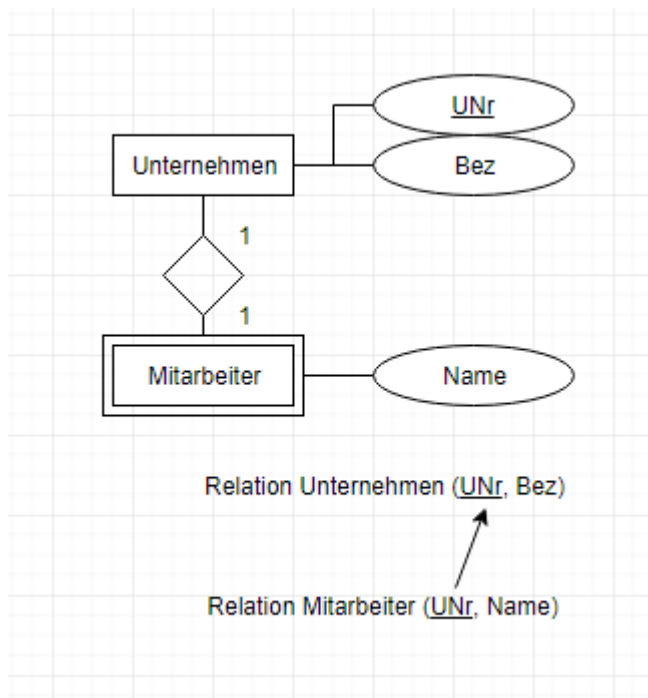


2.5.4.1 disjunkte Entitäten



2.5.5 Weak-Entities

Bei schwachen Entitäten, bei denen die eigenen Attribute nicht ausreichen um ein Tupel eindeutig zu identifizieren, müssen die Schlüsselattribute der damit verbundenen Entitäten zum Schlüssel hinzugenommen werden.



From:

<http://elearn.bgamstetten.ac.at/wiki/> - **Wiki**

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:2:2_05

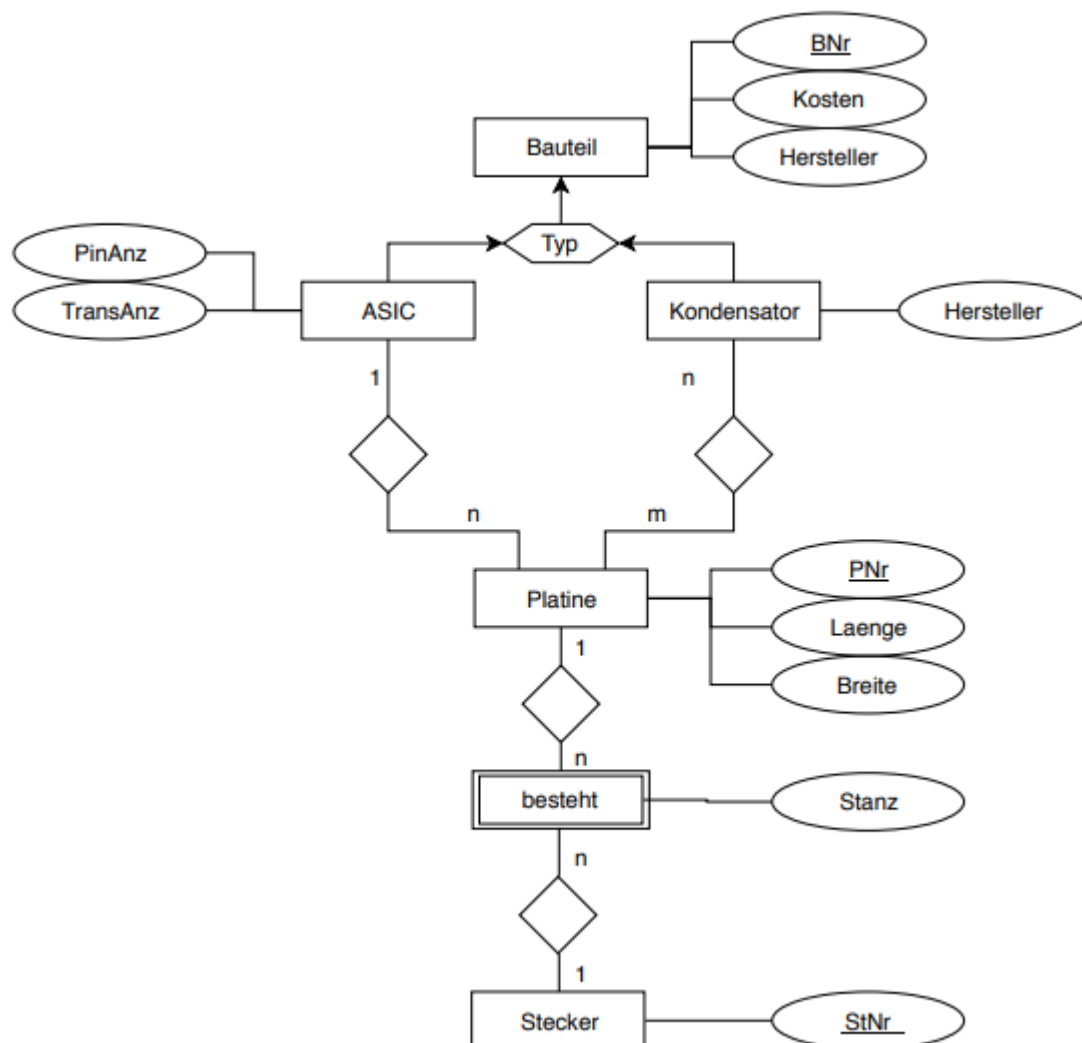


Last update: **2019/11/28 10:26**

Übungen zur Umsetzung von ER-Modell => Relationenmodell

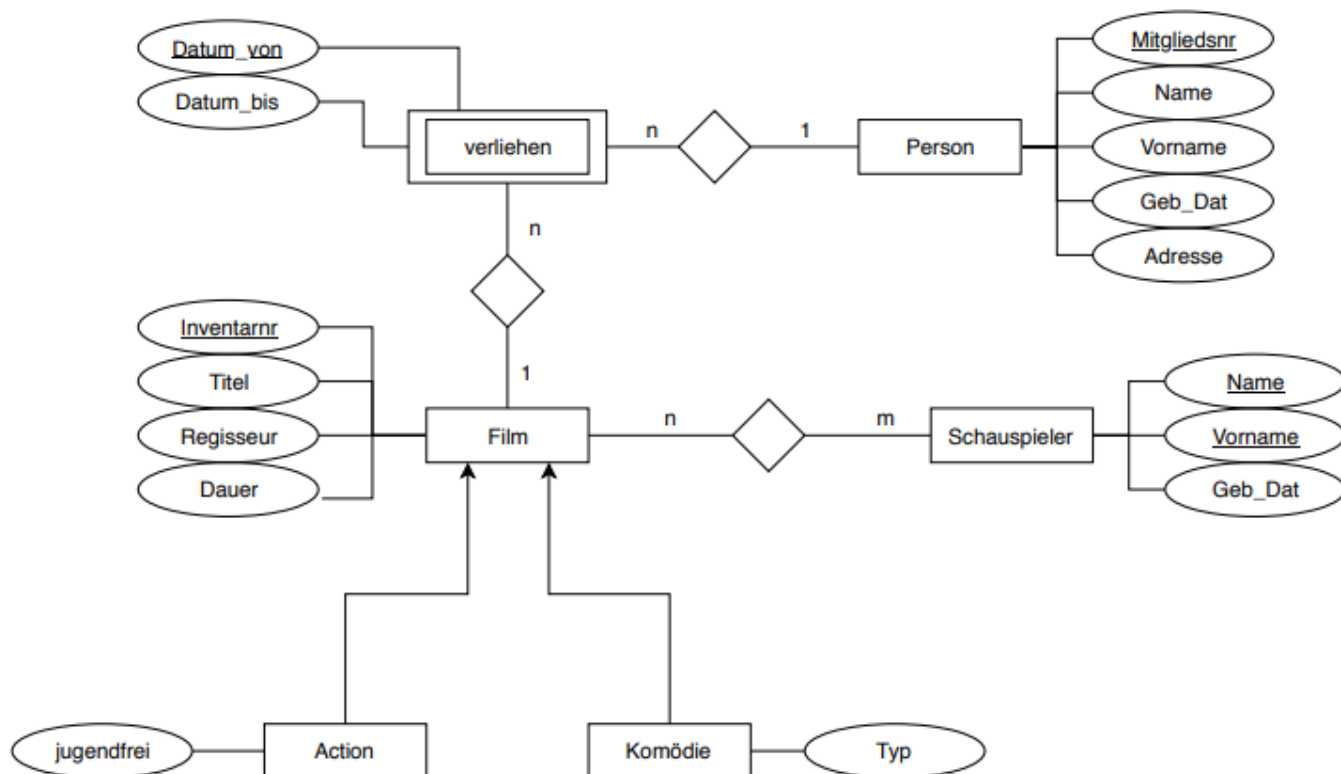
Übung 1

Gegeben ist folgendes ER-Diagramm zur Verwaltung von Elektronikbauteilen. Entwickeln Sie daraus die Relationen der Datenbank.



Übung 2

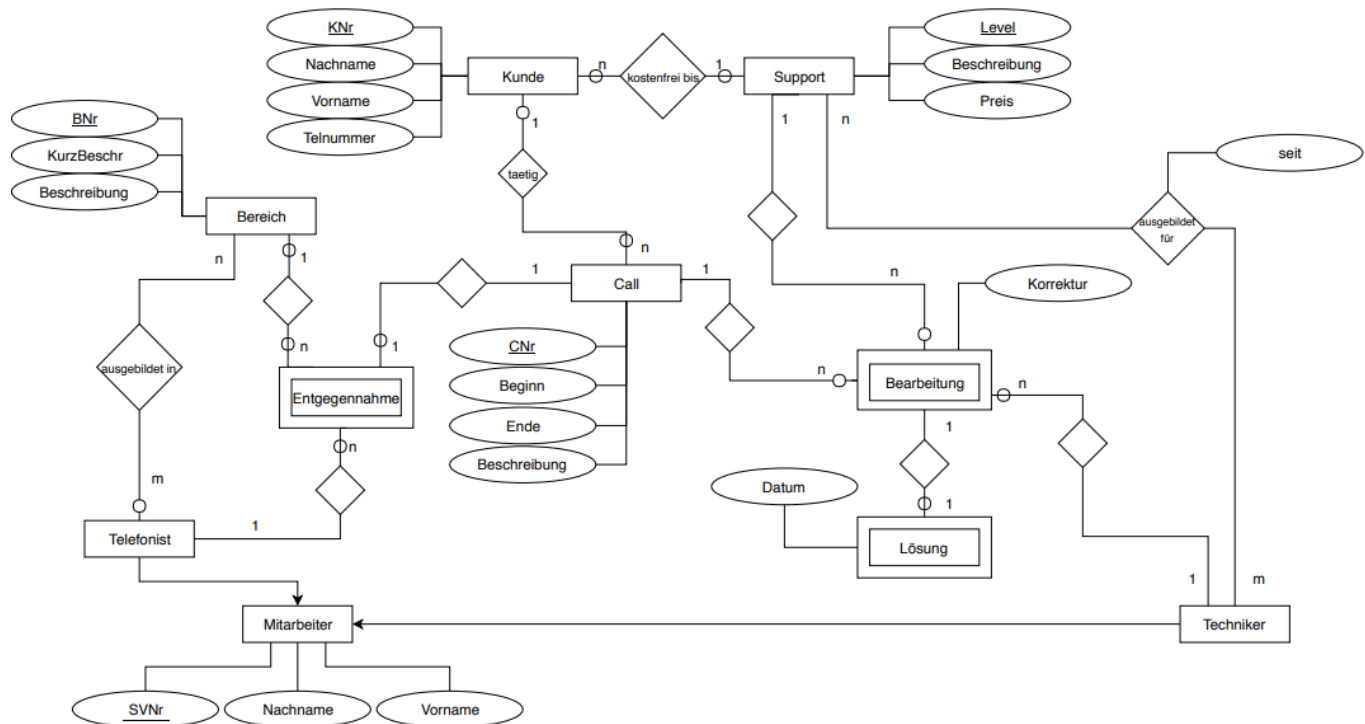
Gegeben ist folgendes ER-Diagramm zur Verwaltung eines Videoverleihs. Entwickeln Sie daraus die Relationen der Datenbank.



Übung 3

Gegeben ist folgendes ER-Diagramm zur Verwaltung eines Videoverleihs. Entwickeln Sie daraus die Relationen der Datenbank.

Das Call Center System erlaubt es bestehenden oder zukünftigen Kunden sich über ein menügesteuertes Interface (Tastenkombinationen am Telefon, Handy, etc.) im Support Center einzuwählen. Das Menü ist dabei in mehrere Bereiche (Produktauskunft, Rechnungsinfo, Technische Probleme, etc.) gegliedert. Eingehende Calls werden entweder von Telefonisten entgegengenommen, welche in allgemeinen Bereichen ausgebildet sind, oder von Technikern bearbeitet, sofern es sich um technische Anfragen handelt. Technische Anfragen werden im Allgemeinen nur für bestehende Kunden bearbeitet. Die mit den Anfragen betrauten Techniker sind für bestimmte Support Levels ausgebildet. Support Levels sind kostenpflichtig, aber Stammkunden bzw. mehr zahlende Kunden können bestimmte Levels gratis in Anspruch nehmen. Technische Probleme können von verschiedenen Technikern auf verschiedene Levels bearbeitet und einer Lösung zugeführt werden.



From:

<http://elearn.bgamstetten.ac.at/wiki/> - Wiki

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:2:2_05_01



Last update: **2019/11/17 17:08**

2.6) Normalformen

Die Kriterien eines guten Datenbankentwurfs sind einerseits möglichst geringe **Redundanz** und andererseits von **Einfügen, Lösch- und Änderungsanomalien** abhängig. Um Redundanzen und Anomalien zu vermeiden, führen wir den Begriff **Normalform** ein, der uns hilft unerwünschte Eigenschaften der Datenbank zu vermeiden.

1. Normalform (1NF)

Ein Relationenschema R ist in 1NF wenn der Wertebereich aller Attribute von R atomar sind. D.h. wenn keine mehrwertigen Attribute vorkommen.

Beispiel - CD Lieder

<u>CD_ID</u>	Name	Titelliste
4711	All That You Can Leave Behind (U2)	Beautiful Day, Walk On, Kite, Wild Honey
4712	Tattou You (Rolling Stones)	Start Me Up, Hang Fire, Slave

- Das Feld **Name** beinhaltet Album und Interpret
- Das Feld **Titelliste** enthält eine Aufzählung aller Titel

Fehler/Problem

- Zur Sortierung nach Interpret muss das Feld **Name** in Album und Interpret aufgeteilt werden
- Den Titel können (mit einfachen Mitteln) nur alle gleichzeitig als Titelliste oder gar nicht dargestellt werden

Lösung 1NF

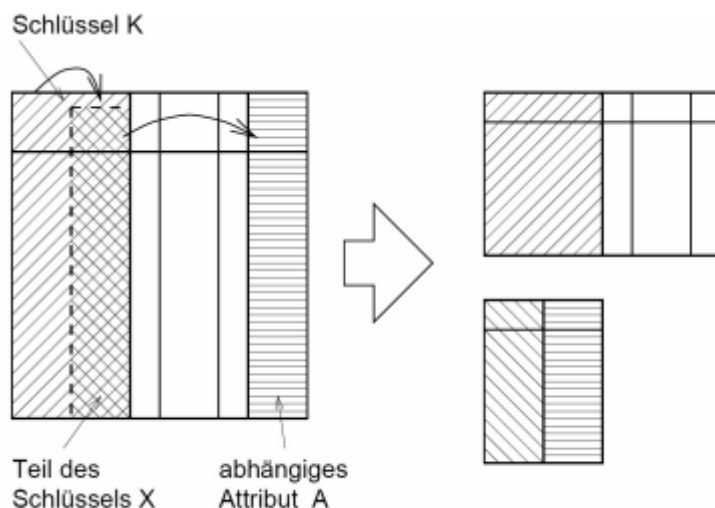
<u>CD_ID</u>	Album	Interpret	Titelliste
4711	All That You Can Leave Behind	U2	Beautiful Day
4711	All That You Can Leave Behind	U2	Walk On
4711	All That You Can Leave Behind	U2	Kite
4711	All That You Can Leave Behind	U2	Wild Honey
4712	Tattou You	Rolling Stones	Start Me Up
4712	Tattou You	Rolling Stones	Hang Fire
4712	Tattou You	Rolling Stones	Slave

2. Normalform (2NF)

Ein Attribut heißt Prim in einer Relation, wenn es mindestens in einem Schlüssel von R enthalten ist bzw. nicht prim, wenn es in keinem Schlüssel enthalten ist.

Ein Relationenschema R ist in zweiter Normalform, wenn es in 1NF und jedes nicht prime Attribut voll funktional von jedem Schlüssel von R abhängig ist, d.h. nur vom ganzen

Schlüssel und nicht nur von einem Teil abhängig ist.



Beispiel

Gegeben ist folgende Tabelle:

<u>CD_ID</u>	Album	Interpret	<u>Track</u>	Titel
4711	All That You Can Leave Behind	U2	1	Beautiful Day
4711	All That You Can Leave Behind	U2	2	Walk On
4711	All That You Can Leave Behind	U2	3	Kite
4712	Tattou You	Rolling Stones	1	Start Me Up
4712	Tattou You	Rolling Stones	2	Hang Fire

Problem

z.B.: Durch ein Update des Albums kommt es zu einer Dateninkonsistenz

<u>CD_ID</u>	Album	Interpret	<u>Track</u>	Titel
4711	All That You Can Leave Behind	U2	1	Beautiful Day
4711	All That You Can Leave Behind	U2	2	Walk On
4711	All That You Can Leave Behind	U2	3	Kite
4712	Sticky Fingers	Rolling Stones	1	Start Me Up
4712	Tattou You	Rolling Stones	2	Hang Fire

Lösung 2NF

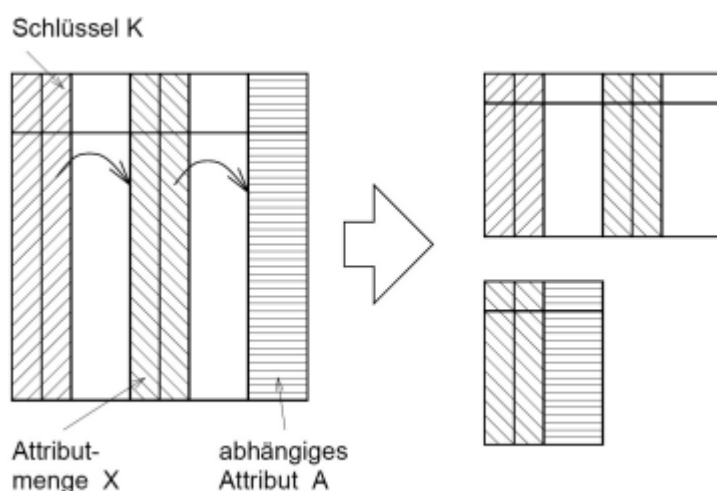
<u>CD_ID</u>	Album	Interpret
4711	All That You Can Leave Behind	U2
4712	Tattou You	Rolling Stones
<u>CD_ID</u>	<u>Track</u>	Titel
4711	1	All That You Can Leave Behind
4711	2	Walk On
4711	3	Kite

<u>CD_ID</u>	<u>Track</u>	Titel
4712	1	Start Me Up
4712	2	Hang Fire

3. Normalform (3NF)

Ein Relationenschema R ist in 1NF ist genau dann in 3NF, wenn kein nicht primes Attribut von einem Schlüssel in R transitiv abhängt.

Transitivitätsregel $K \rightarrow X, X \rightarrow A : K \rightarrow A$



Beispiel

Gegeben sei eine Tabelle mit folgenden Feldern:

<u>CD_ID</u>	Album	Interpret	Gründungsjahr
4711	All That You Can Leave Behind	U2	1976
4712	Sticky Fingers	Rolling Stones	1962
4713	Tattou You	Rolling Stones	1962

Offensichtlich lässt sich der **Interpret** einer CD aus der **CD_ID** bestimmen. Das **Gründungsjahr** der Band hängt dagegen vom **Interpret** und damit nur **!!transitiv!!** von der **CD_ID** ab.

Problem

Auch hier besteht eine Datenredundanz, wodurch Inkonsistenzen beim Ändern auftreten können.

<u>CD_ID</u>	Album	Interpret	Gründungsjahr
4711	All That You Can Leave Behind	U2	1976
4712	Sticky Fingers	Rolling Stones	1972
4713	Tattou You	Rolling Stones	1962

Lösung

<u>CD_ID</u>	Album	Interpret
4711	All That You Can Leave Behind	U2
4712	Sticky Fingers	Rolling Stones
4713	Tattou You	Rolling Stones
<u>Interpret</u>	Gründungsjahr	
U2	1976	
Rolling Stones	1962	

From:

<http://elearn.bgamstetten.ac.at/wiki/> - **Wiki**

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:2:2_06

Last update: **2019/11/18 13:42**



2.7) SQL - Structured Query Language

SQL (**Structured Query Language**) hat sich als Standardabfragesprache für relationale Datenbanken etabliert. SQL stellt die Schnittstelle zwischen der relationalen Datenbank und dem Anwendungsprogramm dar. Es lassen sich damit alle Operationen der Relationenalgebra aus Kapitel 4 realisieren. Die Sprachelemente von SQL lassen sich in mehrere Kategorien unterteilen, die allerdings im Standard nicht festgeschrieben sind.

Allgemeines

Erstellen von Datenbanken

```
CREATE DATABASE DBName ;
```

Anzeigen von Datenbanken

```
SHOW DATABASES ;
```

Selektieren von Datenbanken

```
USE DBName ;
```

Anzeigen von Tabellen in einer Datenbank

```
SHOW TABLES ;
```

Anzeigen von Spalten in einer Tabelle

```
SHOW COLUMNS FROM tabellenname ;
```

Löschen von Datenbanken

```
DROP DATABASE DBName ;
```

2.7.1) DDL (Data Definition Language)

- Anweisungen zur Anlage und Verwaltung von Datenbankschemata
- Anweisungen zur Definition von Relationen einschließlich der Konsistenzbedingungen
- Anweisungen zur Anlage von Datensichten (Views)

- [2.7.1\) DDL](#)
 - [2.7.1.1\) DDL - Übungen](#)

2.7.2) DQL (Data Query Language)

- Abfrage von Daten
- [2.7.2\) Data Query Language](#)
 - [2.7.2.1\) DQL- Übungen](#)

2.7.3) DML (Data Manipulation Language)

- Eingabe von Daten in eine vorhandene Tabelle
- Änderung von Daten in einer Tabelle
- Löschung von Daten in einer Tabelle
- [2.7.3\) DML](#)
 - [2.7.3.1\) DML - Übungen](#)

2.7.4) DCL (Data Control Language)

- Anlegen von Benutzern
- Vergabe von Zugriffsrechten

From:

<http://elearn.bgamstetten.ac.at/wiki/> - Wiki

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:2:2_07

Last update: **2019/11/17 17:08**



2.8) SQL-Datenbankzugriff mit PHP

Um auf eine MySQL-Datenbank zuzugreifen, muss zuerst eine Verbindung (***mysqli_connect***) hergestellt werden. Dazu ist der **Host**, auf dem der MySQL-Server läuft, der **Benutzername** sowie das **Kennwort** anzugeben. Anschließend muss noch eine Datenbank auf dem DBS ausgewählt werden (***mysqli_select_db***). Danach können die SQL-Anweisungen folgen (***mysqli_query***). Im Falle einer SQL-Abfrage liefert diese Funktion alle Tupel zurück, welche der Reihe nach mit ***mysqli_fetch_array*** abgearbeitet werden können.

Im Folgenden soll als Beispiel ein Gästebuch realisiert werden. Dazu wurde eine Tabelle **meldung** erstellt, welche die einzelnen Nachrichten aufnehmen soll:

```
CREATE TABLE meldung (  
  id INT NOT NULL AUTO_INCREMENT,  
  datum DATETIME DEFAULT NULL,  
  name VARCHAR(200) DEFAULT NULL,  
  eintrag TEXT,  
  CONSTRAINT PK_id PRIMARY KEY (id)  
);
```

Zusätzlich zu den Daten **datum**, **name** und **eintrag** wurde ein Attribut **id** eingeführt, welches als Schlüssel dient.

Das folgende Skript **show.php** liest alle Einträge der Datenbank aus und gibt sie in Tabellenform aus:

```
<?php  
$DBHost = "127.0.0.1";  
$DBUser = "root";  
$DBPasswd = "";  
$DBName = "db";  
  
//Verbindung zu DB-Server herstellen  
$con=mysqli_connect($DBHost, $DBUser, $DBPasswd, $DBName)  
    OR die("Konnte DB-Server nicht erreichen!");  
mysqli_select_db($con,$DBName);  
?  
<html>  
  <head>  
    <title>Alle Meldungen</title>  
  </head>  
  <body>  
    <?php  
      $erg=mysqli_query($con, "SELECT datum, name, eintrag FROM meldung  
ORDER BY datum DESC");  
      echo mysqli_error($con);  
  
      while($row=mysqli_fetch_array($erg))  
      {  
        echo "<table bgcolor=\"#dddddd\" border=\"0\" width=\"600\">
```

```

\n";
        echo "<tr><td>Datum: </td><td>".$row["datum"]."</td></tr> \n";
        echo "<tr><td>Name:</td><td>". $row ["name"]. "</td></tr> \n";
        echo "<tr><td valign=\"top\">Eintrag:</td> \n";
        echo "<td>".nl2br(htmlentities($row["eintrag"]))."</td></tr>
\n";
        echo "</table>\n<br>\n";
    }
?>
<hr><a href="insert.php">neuen Eintrag hinzufügen</a>
</body>
</html>

```

Datum: 2018-11-15 20:19:20

Name: Prof. Andreas Lahmer

Eintrag: Hier sehen sie die SQL-Anbindung mittels PHP in Form eines Gästebuchs!

[neuen Eintrag hinzufügen](#)

Um neue Einträge für das Gästebuch zu erstellen, existiert ein weiteres Skript insert.php:

```

<?php
$DBHost = "127.0.0.1";
$DBUser = "root";
$DBPasswd = "";
$DBName = "db";

//Verbindung zu DB-Server herstellen
$con=mysqli_connect($DBHost, $DBUser, $DBPasswd)
    OR die("Konnte DB-Server nicht erreichen!");
mysqli_select_db($con,$DBName);
?>
<html>
<head>
    <title>Neuer Eintrag in unser Gästebuch</title>
</head>
<body>
    <?php

        if(isset($_GET["submit"]))
        {
            $submit = $_GET["submit"];
            $name = $_GET["name"];
            $eintrag = $_GET["eintrag"];

            //Der Submit-Button wurde gedrückt --> die Werte müssen
            überprüft werden
            //und bei Gültigkeit in die DB eingefügt werden

```

```

        $DatenOK=1;           //wir gehen prinzipiell von der
Gültigkeit der Daten aus
        $error="";           //es gab noch keine Fehlermeldung bis hier
hier

        if($name=="")        //Kein Name eingegeben
        {
            $DatenOK=0;
            $error.="Es muss ein Name eingegeben werden!<br>\n";
        }
        if($eintrag=="")      //Kein Kommentar eingegeben
        {
            $DatenOK=0;
            $error.="Ein Eintrag ohne Kommentar macht nicht viel
Sinn!<br>\n";
        }
        if($DatenOK)          //Daten OK -> also in DB eintragen
        {
            $timestamp=date("Y-m-d h:i:s",
time());
            mysqli_query($con,"INSERT INTO eintraege (datum, name,
eintrag) VALUES (\"$timestamp\", \"$name\", \"$eintrag\" );");
            echo mysqli_error($con);
            echo "<b>Daten wurden eingetragen.<b>";
        }
        else
        {
            echo "<h2>Fehler: </h2>\n"; //Fehlermeldung
            echo $error;
        }
    }

    //Formular
?>

    <form action="insert.php" method="GET">
        Name: <input type="text" name="name" size="30" maxlength="200"
value=""><br>
        Text: <br><textarea rows="10" cols="50" wrap="virtual"
name="eintrag"></textarea>
        <br><input type="submit" name="submit" value="Absenden">
    </form>
    <a href="show.php"> Alle Einträge anzeigen</a>
</body>
</html>

```

Name:

Text:

[Alle Einträge anzeigen](#)

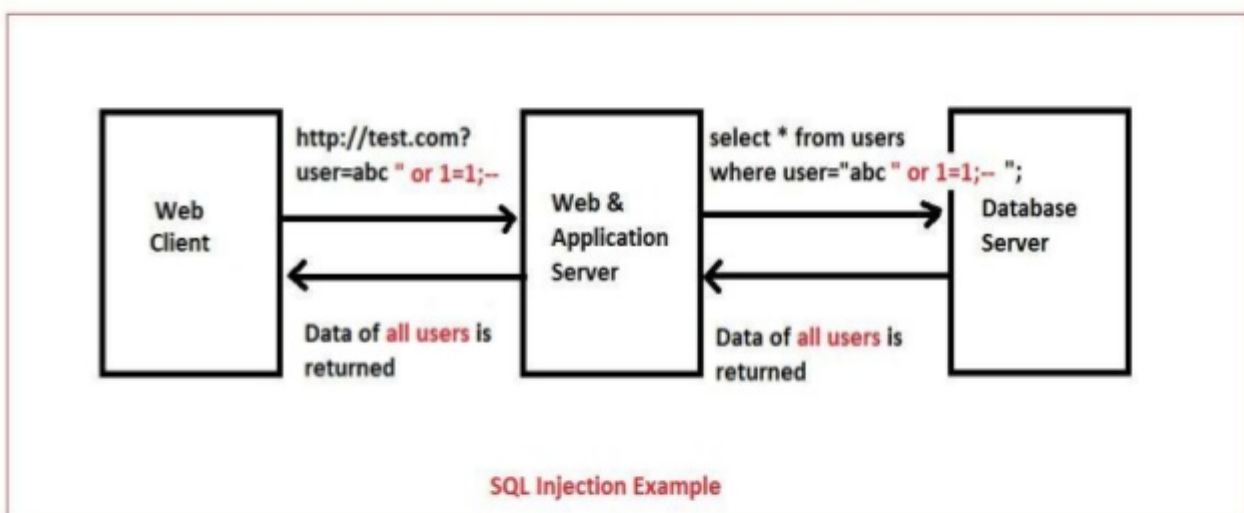
SQL Injection

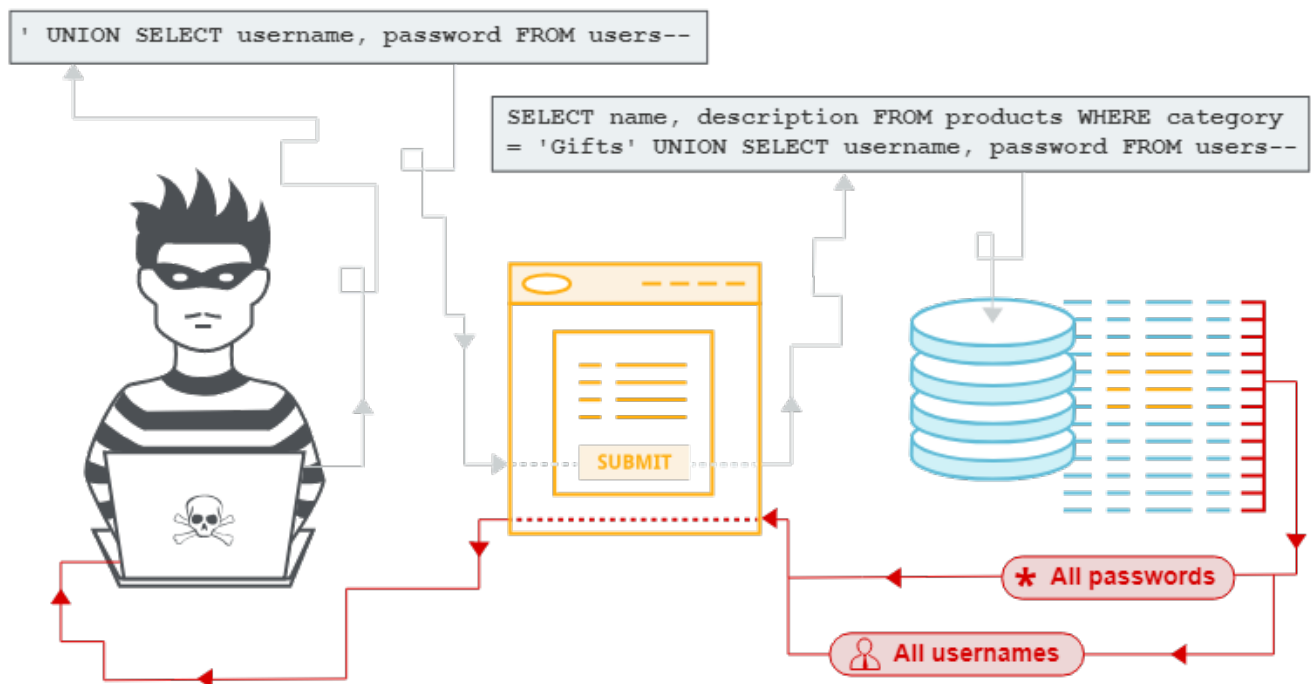
SQL injection ist eine Einschleusung von Code der die Datenbank möglicherweise zerstört

SQL injection ist eine von den meist genutzten Hacking Methoden

SQL injection bezeichnet das Einschleusen von böartigen Code in die SQL-Statements mithilfe von Formulardaten in Webseiten

How SQL Injection works?





SQL Injection basierend auf $1=1$ ist immer wahr (true)

Nehmen wir an, es soll innerhalb einer Webseite die UserId angegeben werden, um die jeweiligen Informationen des Users auszugeben.

Ist diese Eingabe nicht beschränkt, so kann der Benutzer eingeben was er will. Füllt er diese Eingabe „intelligent“ aus, so kann er die ursprüngliche Funktion des SQL-Statements vollkommen verändern.
z.B.:

UserId:

Dadurch wird das SQL-Statement wie folgt verändert:

```
SELECT * FROM Users WHERE UserId = 105 OR 1=1;
```

Somit wird dieses SQL-Statement alle Zeilen der Tabelle Users zurückgeben, da ja $OR\ 1=1$ immer erfüllt ist.

Das wäre besonders heikel, falls die Users Tabelle Benutzernamen & Passwörter beinhaltet. Dann würde das manipulierte SQL-Statement dasselbe verursachen wie folgendes SQL-Statement:

```
SELECT UserID, Name, Passowrd FROM Users WHERE UserId=105 OR 1=1
```

Durch diese Manipulation würde der Hacker Zugriff zu allen Benutzernamen und Passwörter in der Datenbank erlangen, durch einfaches hinzufügen von $OR\ 1=1$.

From:

<http://elearn.bgamstetten.ac.at/wiki/> - **Wiki**

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:2:2_08



Last update: **2019/11/18 13:41**

Aufgabe 1

Erstellen Sie eine php-Seite welche dem Benutzer ein Formular (siehe Abbildung) präsentiert. In diesem Formular kann der Benutzer auswählen, welche der 6 Aufgaben (a bis f) aus der [DQL-Übung](#) angezeigt werden sollen.

Auswahl der Abfrage

☐ SQL:

```
select * from protokoll
```

- ☐ Abfrage 1
☒ Abfrage 2
☐ Abfrage 3
☐ Abfrage 4
☐ Abfrage 5
☐ Abfrage 6

Abschicken

Ergebnis der SQL-Abfrage

name	durchschnittspreis	gesamtanzahl
Weingut F.X.Pichler	23.50	60
Weingut Prager	21.00	30
Weingut Emmerich Knoll	19.00	15
Lackner Tinnacher	12.50	127
Freie Weingärten Wachau	9.90	80
Weingut Biegler	9.00	16

From:

<http://elearn.bgamstetten.ac.at/wiki/> - **Wiki**

Permanent link:

http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201920:2:2_08:2_08_01



Last update: **2019/11/17 17:09**