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) C#
- 4) Unity

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

- **Zahlensysteme**
- **Excel**
- **Programmieren C++, Python inkl. aller Strukturen** (Funktionen, Zeiger, Klassen, Strukturen, Listen, Sortieralgorithmen...)
- **objektorientierte GUI Programmierung** (MS-Visual Studio C#)
- **HTML, CSS, PHP**
- **CMS Framwork (Joomla)**
- **Netzwerke** (Filius + dazu relevante Theorie)
- **Datenbanken inkl. Anbindung PHP/MySQL**

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
- Datenbankanwendungen 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.

C++

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
     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.

Bespiel 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 cpn 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.

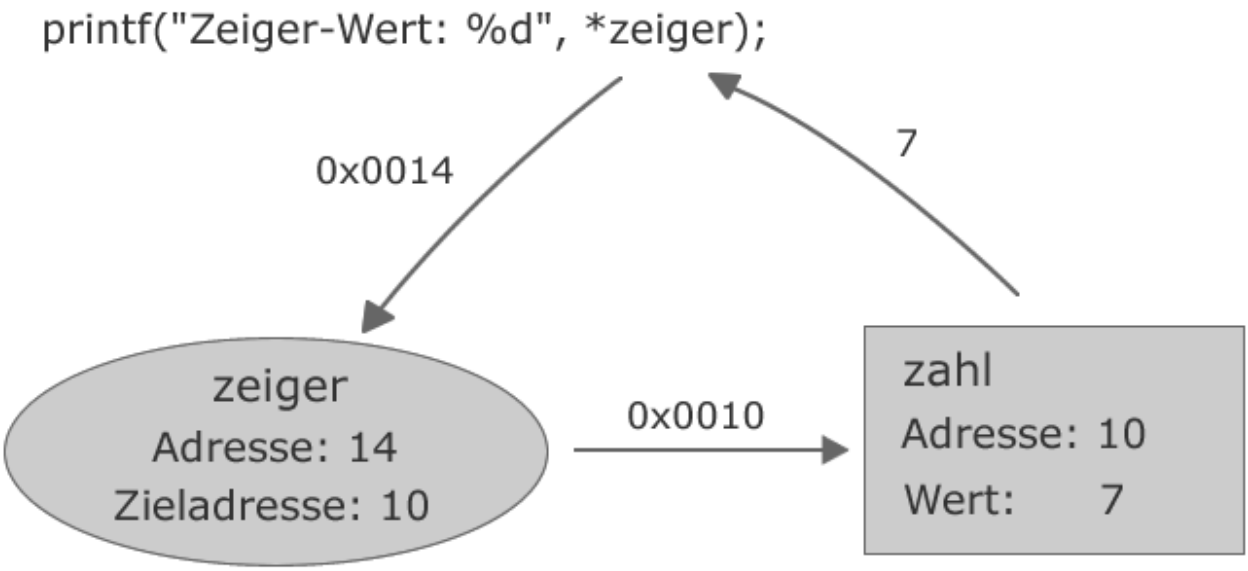
```
int *i,*j,*k; // definiert int* i, int* j, int* k
```

- 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
```

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;
```

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 Bremsscheiben, 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};
```

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.

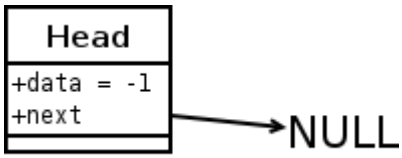
Einfach verkettete Listen zeichnen sich dadurch aus, dass man besonders einfach Elemente einfügen kann, wodurch sie sich besonders gut für Insertion Sort eignen.

Knoten

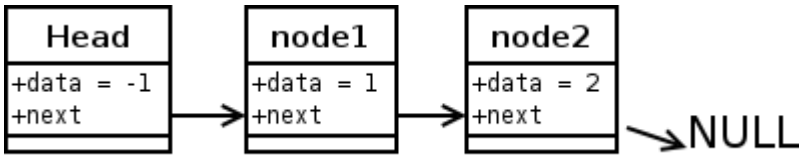
Eine einfach verkettete Liste besteht aus **Knoten, Englisch nodes**, die einen **Zeiger auf das nächste Element** und Daten beinhalten.

```
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.

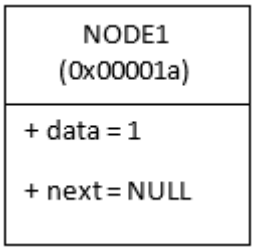
Knoten befüllen & einfügen

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

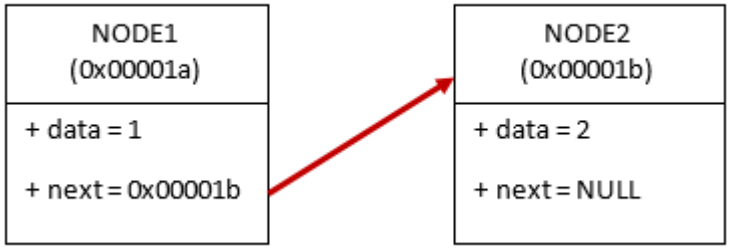
Dazu muss man den next-Zeiger der Liste auf das neue Element setzen, und den next-Zeiger des neuen Element auf den alten Wert des next-Zeigers der Liste:

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

```
        node1=new listnode;


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

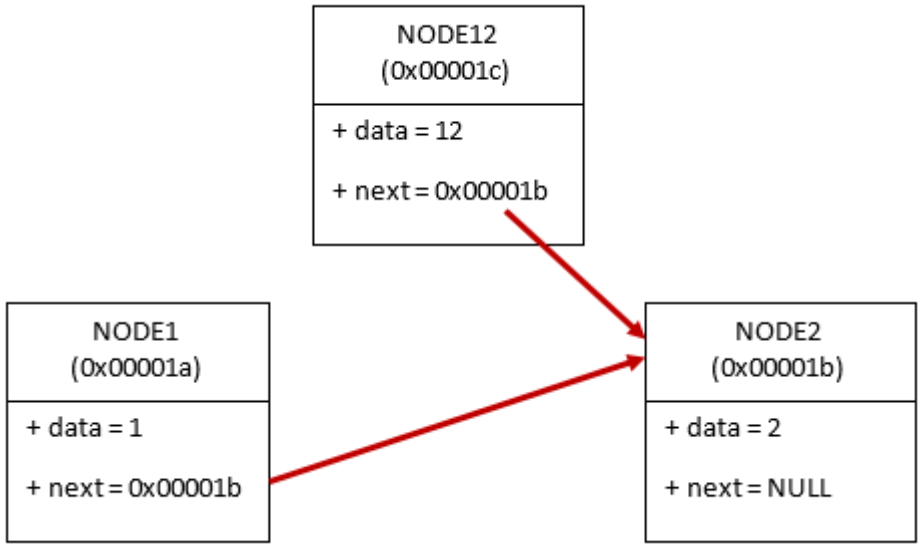



```
    //2. Knoten erstellen
    listnode *node2;
        node2=new listnode;
    node2->data=2;
    node2->next=NULL;
    //1. Knoten zeigt auf 2. Knoten
    node1->next=node2;
```

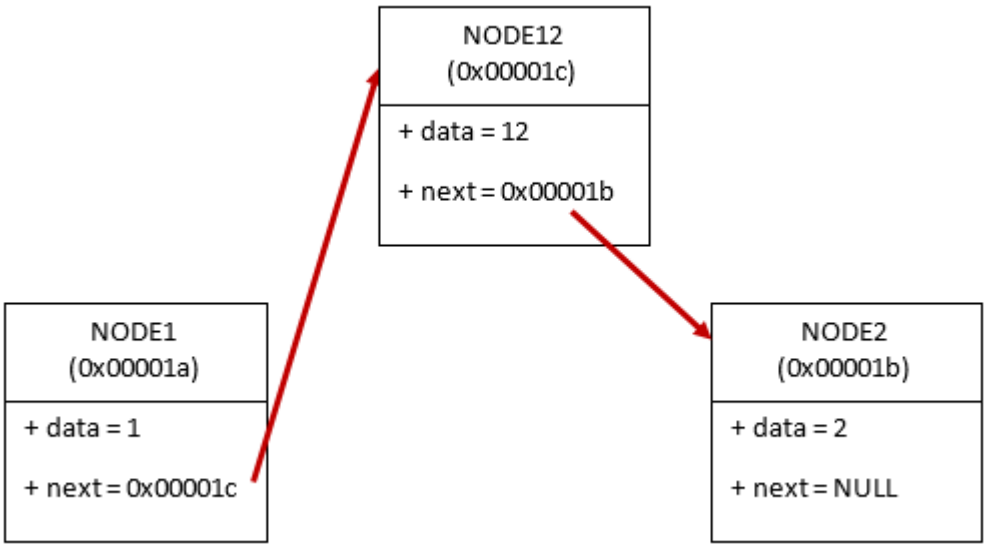



```
    //Knoten 12 zwischen Knoten 1 und 2 einfügen
    listnode *node12;
        node12=new listnode;
    node12->data=12;
    node12->next=node2;
    node1->next=node12;
```

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

```
#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;
   vorneeinhaengen(anzahl);
   cout << "Ausgabe der Liste\n";
   listeausgeben();
   cout << "Die Anzahl aller Elemente betraegt: " << anzahlelemente();

    return 0;


}
```

Doppelt verkettete Listen

Doppelt verkettete Listen (oder doubly linked lists) sind häufig benutze Datenstrukturen und eine Verallgemeinerung der einfach verketteten Listen, die ich hier anhand von Beispielen in der Programmiersprache C++ vorstellen will.

Sie funktioniert in jeder Programmiersprache genauso, die Zeiger oder Referenzen und so etwas wie Klassen oder Structs zur Verfügung stellt.

Wozu?

Doppelt verkettete Listen, oder „double linked lists“ braucht man immer dann, wenn man sich in einer Liste leicht vorwärts und rückwärts bewegen können muss, und wenn man schnell und einfach Elemente der Liste an beliebigen Positionen löschen und neue einfügen muss.

Denn einfach verkettete Listen haben den Nachteil, dass man sie nur in einer Richtung durchlaufen kann.

Darüber hinaus kann man ein referenziertes Listenelement nicht unmittelbar löschen, da man von diesem Element keinen Zugriff auf das Vorgängerelement hat.

Doppelt verkettete Listen umgehen dieses Problem, indem sie in jedem Knoten zusätzlich noch eine Referenz auf den Vorgängerknoten speichern.

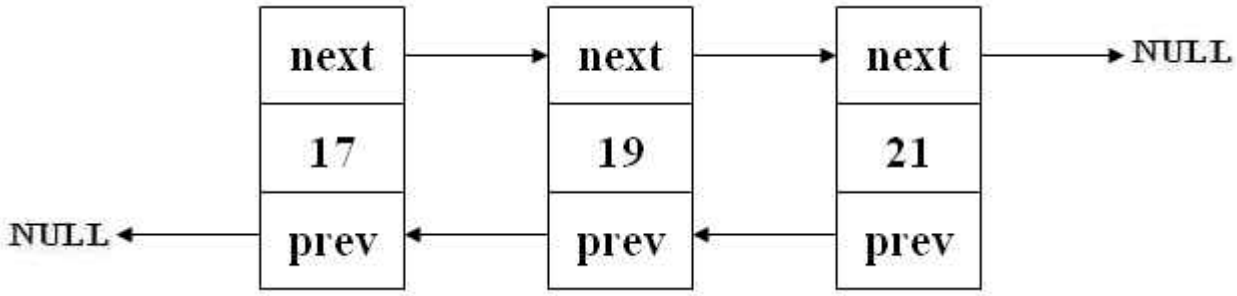
Die Grundidee

Im Gegensatz zu einfach verketteten Listen haben doppelt verkettete Listen in den Knoten eine zusätzliche Instanzvariable für die Referenz auf den Vorgängerknoten

Die Idee einer doppelt verketteten Liste ist es also, für jedes Element zwei Zeiger zu speichern, einen nach links und einen nach rechts:

```
struct DList{
    DList*left;
    DList *right;
    int data;
};
```

Die eigentlichen Daten, die in der Liste gespeichert werden sollen, stehen in data, das hier als int definiert ist. Jeder andere Datentyp, z.B. ein Zeiger auf beliebige andere Strukturen, funktioniert genau so.



Einfügen, Löschen und Ausgeben von Elementen

```
#include <iostream>

using namespace std;

struct DListe{
    DListe *next;
    DListe *prev;
    int zahl;
};


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

    DListe *head=NULL;        //Anlegen des Head-Pointers
    DListe *tail=NULL;        //Anlegen des Tail-Pointers
    DListe *help=NULL;        //Anlegen des Help-Pointers (für das Löschen
und Ausgeben)


    //Einfügen von 10 Elementen in eine doppelt verkettete Liste
    for(int i=0;i<10;i++)
    {
        DListe *elem=new DListe();        //Element erzeugen
        elem->zahl=i;                     //Attribut zahl befüllen
        if(head==NULL && tail==NULL)    //noch kein Element in der Liste
        {
            head=elem;                    //Head zeigt auf das neue Element
            tail=elem;                    //Tail zeigt auf das neue Element
            elem->next=NULL;            //Next zeigt auf NULL
            elem->prev=NULL;            //Prev zeigt auf NULL
        }
        else                            //Elemente sind bereits vorhanden &
Element wird am Ende eingefügt
        {
            tail->next=elem;            //tail->next auf das neue Element
            elem->prev=tail;            //elem->prev zeigt auf das
ursprünglich letzte Element
            elem->next=NULL;            //elem->next zeigt auf NULL
```

```
            tail=elem;                    //tail zeigt auf das neue letzte
Element
        }
    }

    cout << "Geben Sie an, welches Element (0-9) Sie loeschen wollen!" <<
endl;
    int loesche=0;
    cin >> loesche; //zB. Element mit der zahl=2 wird gelöscht

    help=head;                    //help zeigt auf head (Anfang der Liste)
    while(help->zahl!=loesche)    //Wenn help->zahl!=loesche, dann wandert
help in der Liste weiter
    {
        help=help->next;
    }
    //Überprüfe nochmals mit Ausgabe, ob beim richtigen Element (=2)
angekommen?
    cout << help->zahl;
    //Sonderfall 1. Element soll gelöscht werden (zahl=0)
    if(help==head)
    {
        head->next->prev=NULL;
        head=head->next;
        head->prev=NULL;
    }
    else if(help==tail)  //Sonderfall letztes Element soll gelöscht werden
    {
        tail=help->prev;
        help->prev->next=help->next;
    }
    else    //Element ist zwischen 2 anderen Elementen
    {
        help->prev->next=help->next;
        help->next->prev=help->prev;
    }

    //Ausgabe der doppelt verketteten Liste
    cout << endl << endl;
    help=head;
    while(help!=NULL)
    {
        cout << help->zahl << " -> ";
        help=help->next;
    }

    return 0;
}
```

Sortiertes Einfügen

```
      do{
        DListe *elem=new DListe();
        cout << endl << "Geben Sie ein Element ein, das sie einfuegen
wollen!" << endl;
        cin >> elem->zahl;
        elem->next=NULL;
        elem->prev=NULL;
        help=head;

        while((elem->zahl>help->zahl) && (help->next!=NULL))
        {
            help=help->next;
        }

        if(head==help)    //1. Fall => Element wird an erster Stelle
eingefügt
        {
            elem->next=help;
            help->prev=elem;
            head=elem;
        }
        else if(tail==help) //2. Fall => Element wird an letzter Stelle
eingefügt
        {
            elem->prev=help;
            help->next=elem;
            tail=elem;
        }
        else    //3. Fall => Element wird in der Mitte, also zwischen 2
anderen Elemente eingefügt
        {
            elem->next=help;
            help->prev->next=elem;
            elem->prev=help->prev;
            help->prev=elem;
        }

        //Ausgabe der Doppelt verketteten Liste
        ausgabe(head);

        cout << "Wollen Sie noch ein Element einfuegen (j/n)" << endl;
    }while(getch()=='j');
```

2) DATENBANKEN

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

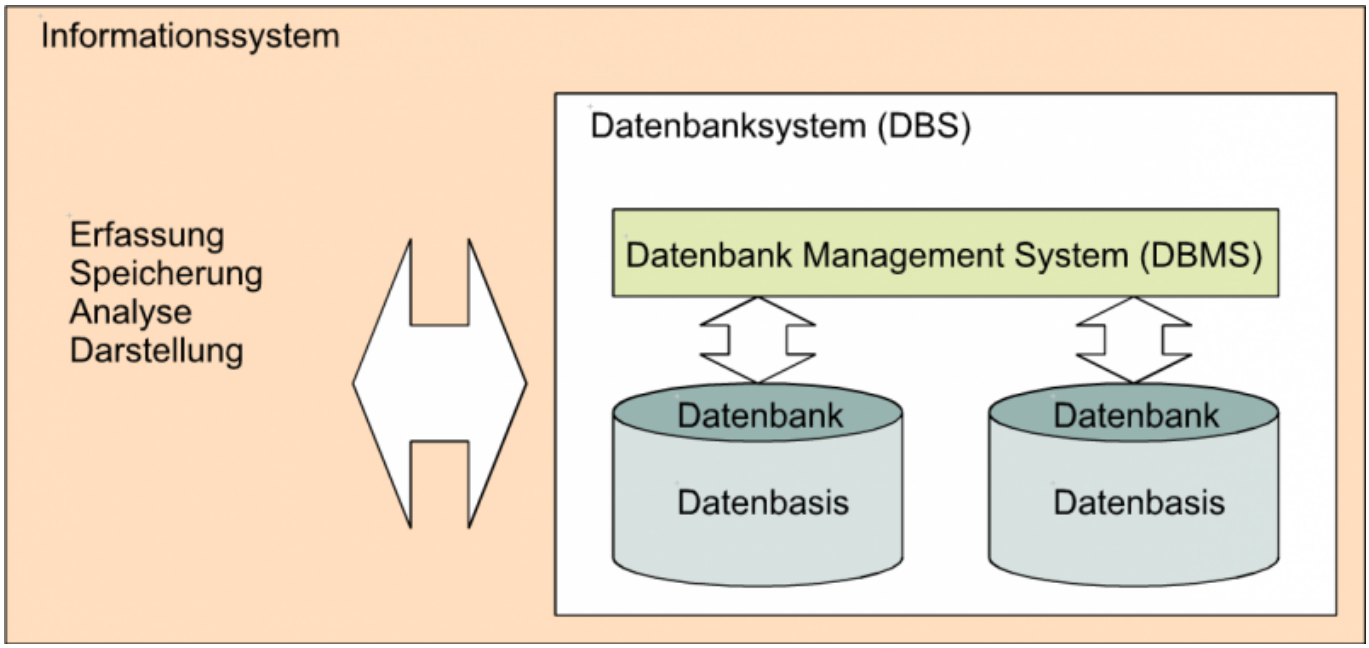
Personalabteilung
Kunden
Gehalt

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 Wideranlauf der gesamte Geldbetrag noch auf dem Konto befindet.

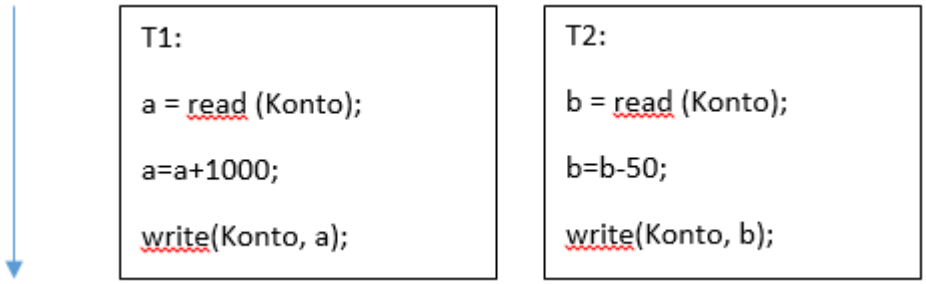
Bsp. zu Dauerhaftigkeit:
Angenommen wir zahlen einen Millionengewinn im Lotto auf unser Konto ein, dann stellt die Dauerhaftigkeit von Transaktionen sicher, dass der Gewinn auch nach einem Systemneustart immer noch auf dem Konto liegt. Für den Wiederanlauf verwenden die meisten DBS ein Log-Protokoll. 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:

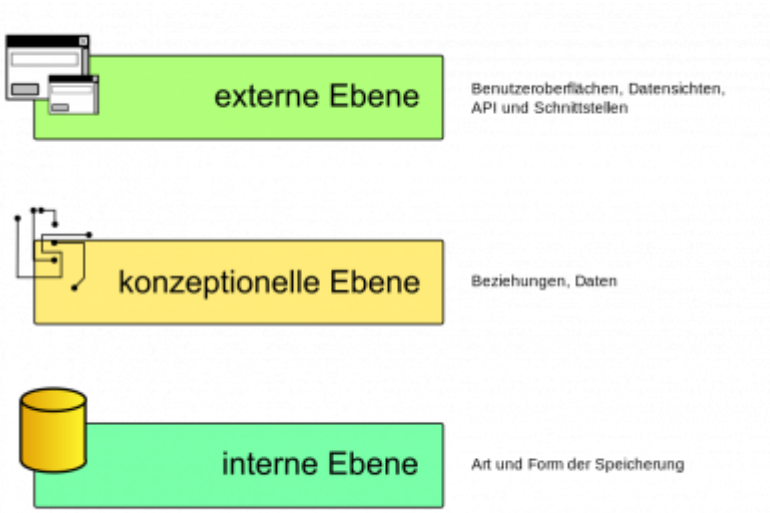
- **Atomicity:** Eine Transaktion stellt eine nicht weiter zerlegbare Einheit dar, mit dem Prinzip: **„ALLES oder NICHTS“**
- **Consistency:** Nach Abschluss der Transaktion liegt wieder ein konsistenter Zustand vor, während der Transaktion sind inkonsistente Zustände erlaubt
- **Isolation:** Nebenläufig 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).

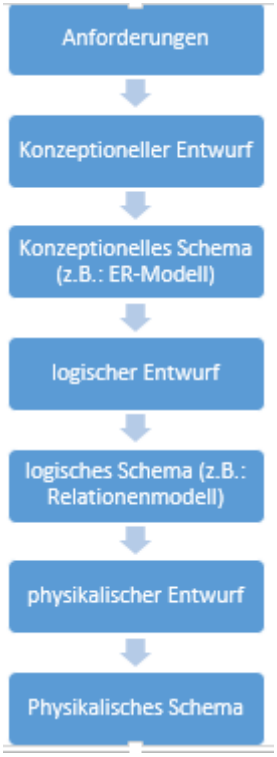
Die Vorteile des Drei-Ebenen-Modells liegen in der

- **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.

2.2) Datenmodellierung

Im Datenbankentwurf werden verschiedene Datenbankmodelle verwendet:



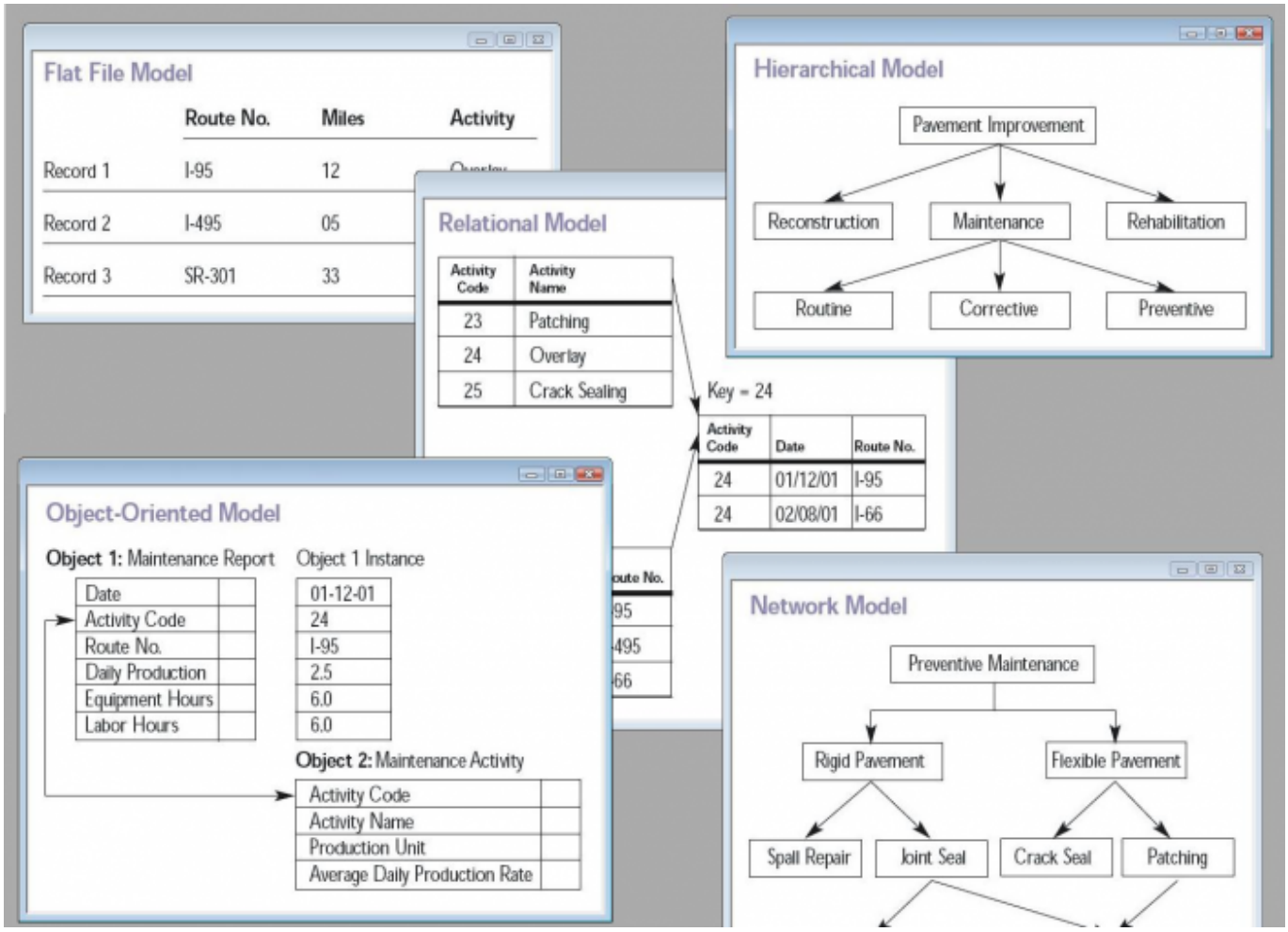
Konzeptionelle Datenmodelle (z.B.: ER-Modell) stehen **problemnahe Modellierungskonzepte** für die ersten Schritte des Datenbankentwurfs zur Verfügung und dienen zur **Kommunikation zwischen Endbenutzern und Datenbankentwicklern**.

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.

2.3) ENTITY-RELATIONSHIP-MODELL

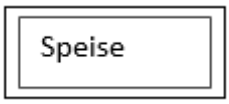
Das ER-Modell wurde im Jahr 1976 von Chen eingeführt. In der Sicht des ER-Modells besteht die Welt aus Entities (Objekttypen) und Relationships (Beziehungen), zwischen den Objekten. Das ER-Modell wurde von Teorey im Jahr 1986 erweitert zum Extended ER-Modell (EER).

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:

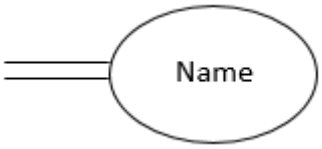
- **identifizierende Attribute = (Schlüssel, Key)** ⇒ sind jene Attribute, die die Objekte einer Entity eindeutig kennzeichnen.

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

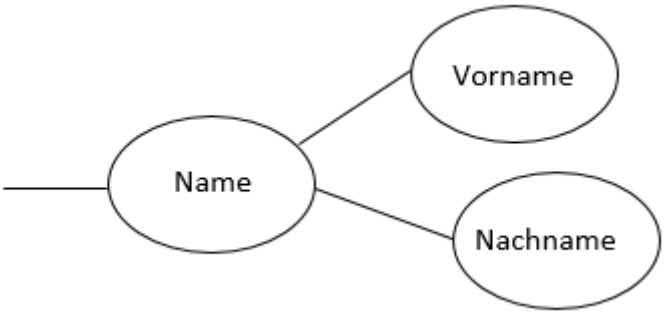
Schlüssel (Keys) können aus **einem oder mehreren Attributen** bestehen. Jede Entität kann mehrere verschiedene Schlüssel haben, wir stellen im ER-Modell aber immer **einen ausgezeichneten Schlüssel als Primärschlüssel (=Primary Key)** dar. Entitäten deren Objekte, nur mit Hilfe von anderen Entitäten 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**



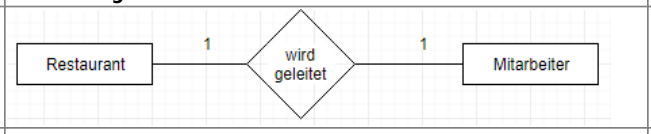
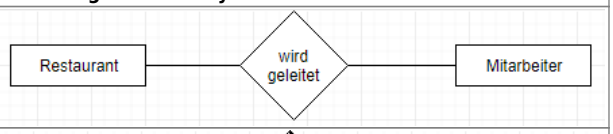
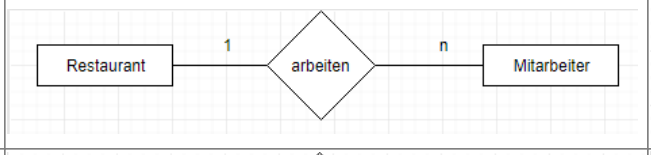
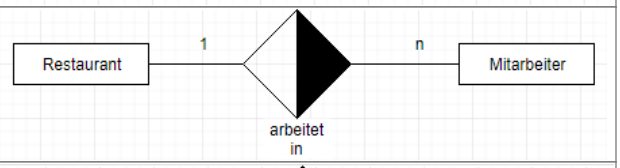
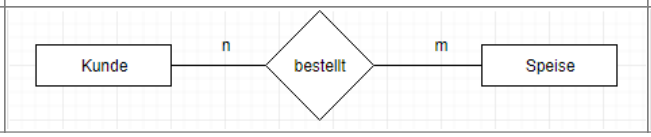
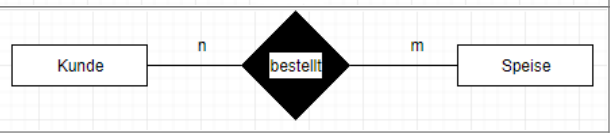
2.3.1.1) Komplexität von Beziehungen

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** ⇒ Jedes Objekt von A gehört genau zu einem Objekt von B und umgekehrt
- **1:n** ⇒ 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** ⇒ Jedes Objekt von A gehört zu einem oder mehreren Objekten von B und umgekehrt.

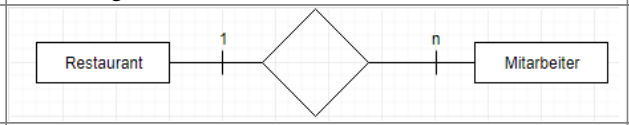
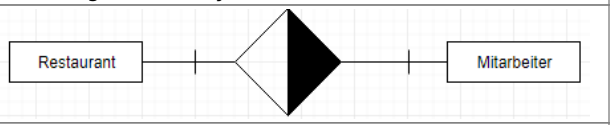
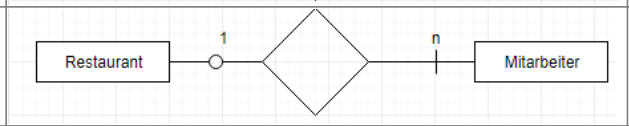
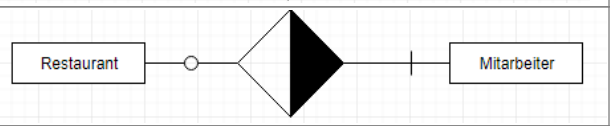
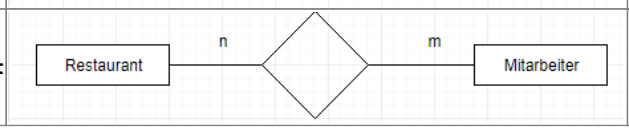
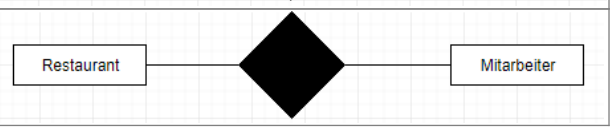
Die Darstellung einer Komplexität von Beziehung nach Chen geschieht durch Anbringung von Ziffern an die jeweiligen Entitäten. Dabei wird nur zwischen den Werten 1 und viele (n,m) unterschieden. Die genaue Anzahl der zugeordneten Objekte, d.h. die Kardinalität wird selten verwendet, da diese sich bei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

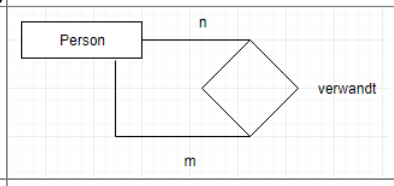
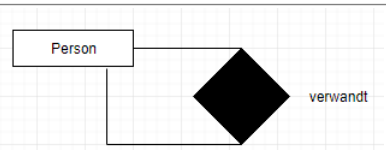
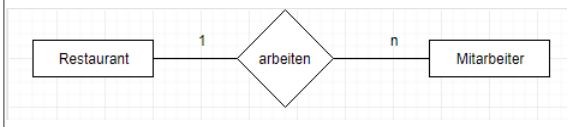
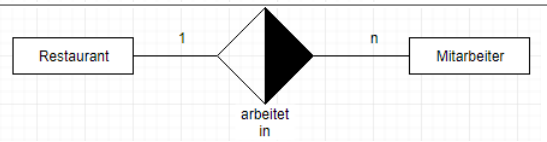
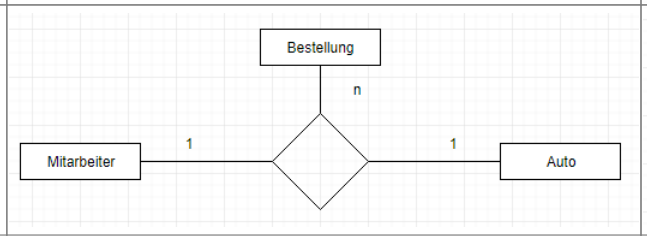
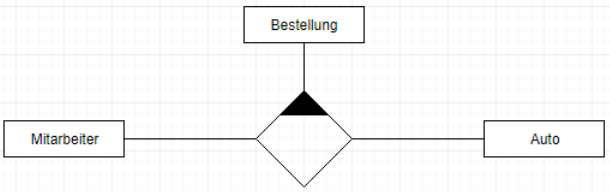
Bsp.:
zwingend ⇒ Ein Restaurant hat hat zumindest einen oder mehrere Mitarbeiter und jeder Mitarbeiter ist genau einem Restaurant zugeordnet.
optional ⇒ Ein Restaurant hat zwingend einen Geschäftsführer. Manche Mitarbeiter sind Geschäftsführer.

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.

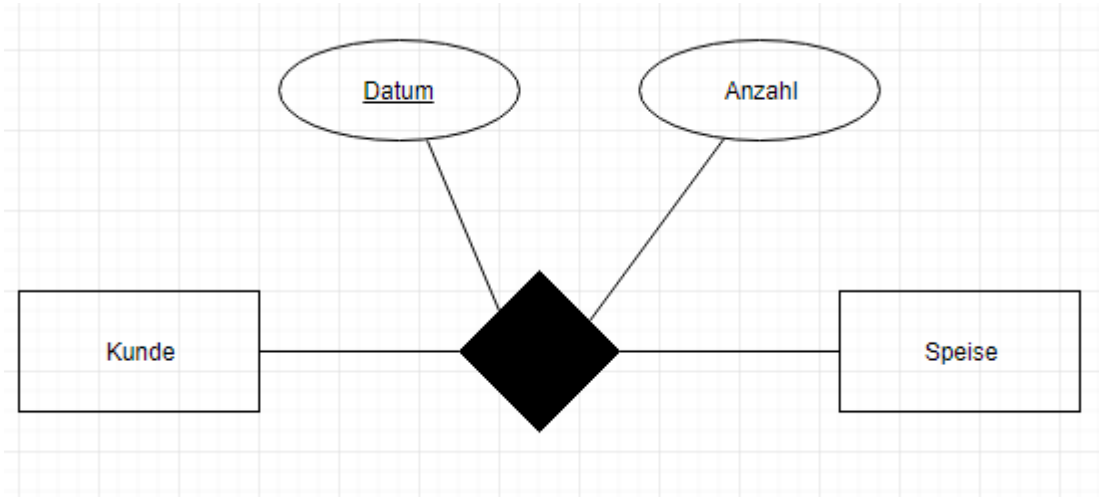
Grad der Beziehung	Darstellung nach Chen	Darstellung nach Teorey
unär		
binär		
ternär		

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 Stellt 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.

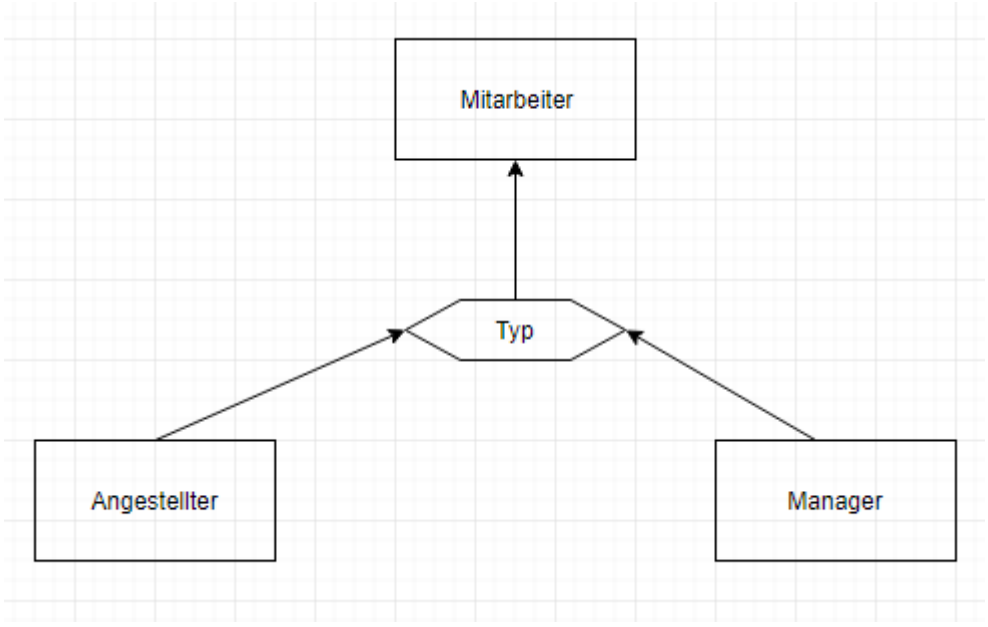
2.3.2) Erweiterte ER-Konstrukte: Die Generalisierung

Bei der Modellierung der Entities Mitarbeiter und Kunden wird man feststellen, dass sie viele gemeinsame Attribute besitzen. Diese können zu einer Enitity 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 SVNR der Mitarbeiter nur bei den speziellen Entities gespeichert.

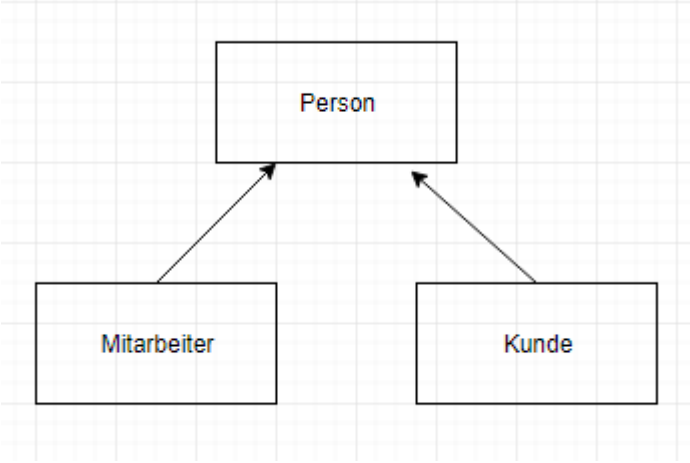
Die Generalisierung gibt an, dass mehrere Entities (Subtyp-Entity) mit bestehenden gemeinsame Attributen zu einer Entity auf einer höheren Ebene (Supertyp-Entity) generalisiert werden können.

Die Disjunktheit der Generalisierbarbeit 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)



2.3.1) Übungen

2.3.1.1) Universität

Ein Institut hat einen 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 identifziert. 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 Realitat ubereinstimmt. 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).

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

Eine **Relation r(R)** auf einem **Relationenschema R** ist eine endliche Menge von Abbildungen $\{t_1,....,t_m\}$ 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:

```
Dom(rnr)   = Menge aller Integer
Dom(name)  = Menge aller Namen
Dom(haube) = {k|0<=k<=4}
```

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

```
t3(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 $t_1(K) != t_2(K)$ 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 („∩“)
- Vereinigung („∪“)
- 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 ∩ s)
- Vereinigung (r ∪ s)
- Differenz (r - s)
- Differenz (s - r)

Lösung:

- Durchschnitt (r ∩ s)

A	**B**	**C**
a1	b2	c1

- Vereinigung (r ∪ 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:

$ σ_{A=a}(r) = \{ t∈r|t(A)=a \} $

Beispiele:
a) $ σ_{A=a1}(r) = ? $

A	B	C
a1	b1	c1
a1	b2	c1

b) $ σ_{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“ („∧“), „oder“ („∨“) und Negationen („¬“).

c) $ σ_{(B=b1)∧¬(A=a1)}(s)=? $

d) $ σ_{(haube > 1) ∧ ¬(typ="österreichisch" ∨ typ="international")}(Restaurant)=? $

Bsp: Relation Restaurant

rnr	name	adresse	haube	typ
1	Schnitzelhaus	Amstetten	1	österreichisch
2	Brauhof	Amstetten	3	international
3	Pizza Hollywood	Amstetten	2	italienisch

Lösung ⇒ 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:

$π_X(r) = \{t(X)|t∈r\} für X ⊆ R $

Beispiele:

e) $ π_{A,B}(r)=? $

f) $ π_{name,adresse,haube}(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 ⋈ s = \{|\{R,S\}| ∃ t_r ∈ r \text{ und } ∃ t_s ∈ 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 ⋈ s

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

Aufgabe:

```
Restaurant  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⋈s $ und $ r'=π_R(q) $, dann gilt $ r'⊆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⋈s=q

A	**B**	**C**
a	b	c

Relation $ π_{AB}(q)=r' $

A	**B**
a	b

2.4.2.4.2 Das Kartesische Produkt

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

Beispiel:
Relation r

A	B
a1	b1
a2	b1

Relation s

C	D
c1	d1
c2	d1
c2	d2

Relation r x s = r ⋈ 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 $R ∩ S=\{\}$. 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 % 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.

Übungen zum Relationenmodell

Übung 1

Relation Restaurant			
rnr	**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		
rnr	**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

Lösung Übung 1 a)

$ σ_{(Adr=Amstetten)}(Restaurant)=? $

Lösung Übung 1 b)

$ σ_{(Adr="Amstetten")∧(haube >= 2)}(Restaurant)=? $

Lösung Übung 1 c)

$ π_{RName}(Restaurant)= ? $

Lösung Übung 1 d)

$ π_{RName}(σ_{(Adr="Amstetten")}(Restaurant))= ? $

Lösung Übung 1 e)

$ Restaurant ⋈ Speise $

Lösung Übung 1 f)

$ π_{RName}(Restaurant ⋈ Speise) $

Lösung Übung 1 g)

$ π_{RName}(σ_{SName="Salat"}(Restaurant ⋈ Speise)) $

Lösung Übung 1 h)

$ π_{RName, SName, Preis}(σ_{SName="Salat"}(Restaurant ⋈ Speise)) $

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			
RNr	**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			
PNr	**PName**	**MinLeist**	**Bereich**
P1	DrawIt	1	Grafik
P2	AskIt	3	Datenbank
P3	WriteIt	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			
ANr	**AName**	**StudAss**	**Gehalt**
A1	Novak	Brunner	3
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			
RNr	**_PNr_**	**Platz**	**Code**
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		
ANr	**_PNr_**	**Stund**
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 WriteIt 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 genaue 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

RNr	StudAss	Leist	Speicher
R1	Huber	1	100
R2	Brunner	3	80
R3	Brunner	3	400
R4	Vogt	2	120
R5	Huber	2	500

Relation Programm

PNr	PName	MinLeist	Bereich
P1	DrawIt	1	Grafik
P2	AskIt	3	Datenbank
P3	WriteIt	1	Text
P4	ConnectIt	2	Internet
P5	PaintIt	2	Grafik
P6	StoreIt	3	Datenbank

Relation Assistent

ANr	AName	StudAss	Gehalt
A1	Novak	Brunner	3
A2	Dvorak	Vogt	1
A3	Husak	Vogt	1
A4	Pfeiffer	Brunner	2

Relation Installation

RNr	_PNr_	Platz	Code
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

Relation Benutzung

ANr	_PNr_	Stund
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

Lösung Übung 2a)

$ π_{AName}(σ_{Stund>5}(Assistent ⋈ Benutzung)) $

Lösung Übung 2b)

$ π_{PName}((Programm)⋈(σ_{StudAss="Huber"}(Rechner))⋈ (Installation)) $

Lösung Übung 2c)

$ π_{PName}((π_{ANr, PNr}(Benutzung) ÷ π_{ANr}(Assistent)) ⋈ Programm)) $

Lösung Übung 2d)

$ π_{RNr, PNr}(Installation) ÷ π_{PNR}(σ_{RNr="R1"}(Installation)) $

Lösung Übung 2e)

$ π_{StudAss}(Rechner) - (π_{StudAss}(σ_{PName="WRITE IT"}(Rechner ⋈ Installation ⋈ Programm))) $

Lösung Übung 2f)

Relation muss dupliziert werden ⇒ AssistentCopy und anschließend mit sich selbst gejoint werden, nachdem die Attribute unterschiedlich heißen muss man sie in eckigen Klammern angeben!

$ σ_{AName=ANameCopy}(π_{AName,ANameCopy}(Assistent_{[StudAss=StudAssCopy]}AssistentCopy)) $

Lösung Übung 2g)

$ π_{StudAss}(Rechner) ∩ π_{StudAss}(Assistent) $

Lösung Übung 2h)

$ σ_{(RNr<RNrCopy)∧(Leist=LeistCopy)}(Rechner × RechnerCopy) $

Lösung Übung 2i)

$ π_{PName}(Programm ⋈ (π_{RNr, PNr}(Installation) ÷ π_{RNr}(Rechner))) $

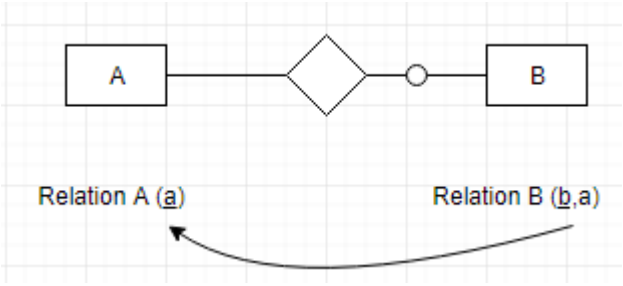
Lösung Übung 2j)

$ π_{RNr, PNr}(σ_{MinLeist}(Programm ⋈ Installation ⋈ Rechner)) $

2.5 Umsetzung des ER-Modells in ein Relationenmodell

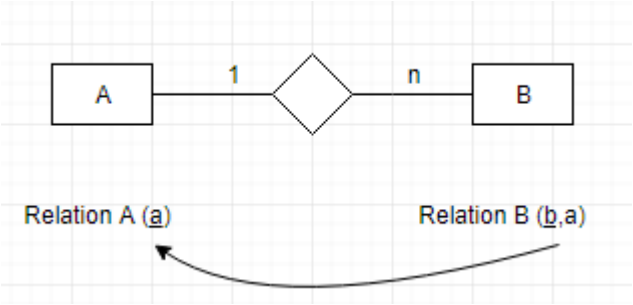
2.5.1 1-zu-1 Beziehung

Bei einer 1-zu-1 Beziehung zwischen Entitäten wird die Beziehung aufgelöst, in dem 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.



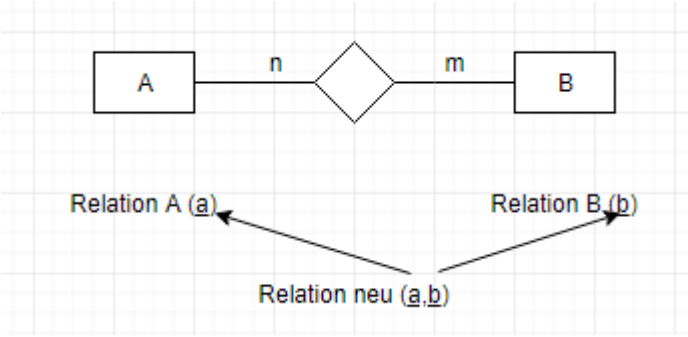
2.5.2 1-zu-n Beziehung

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.



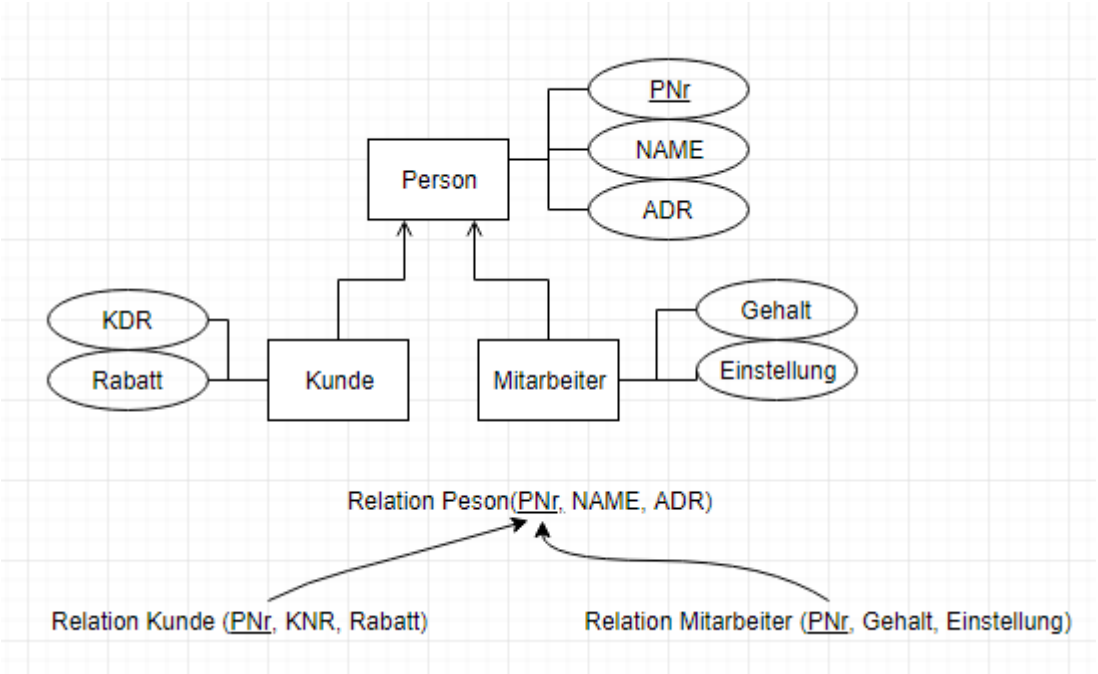
2.5.3 n-zu-m Beziehung

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

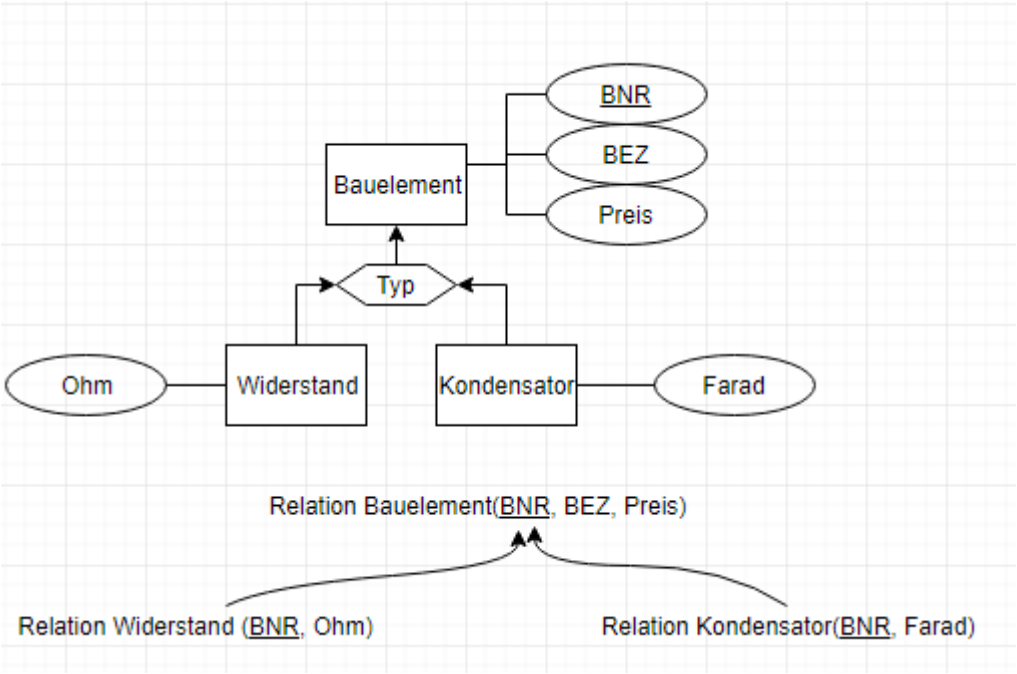


2.5.4 Generalisierung

2.5.4.1 nicht disjunkte Entitäten

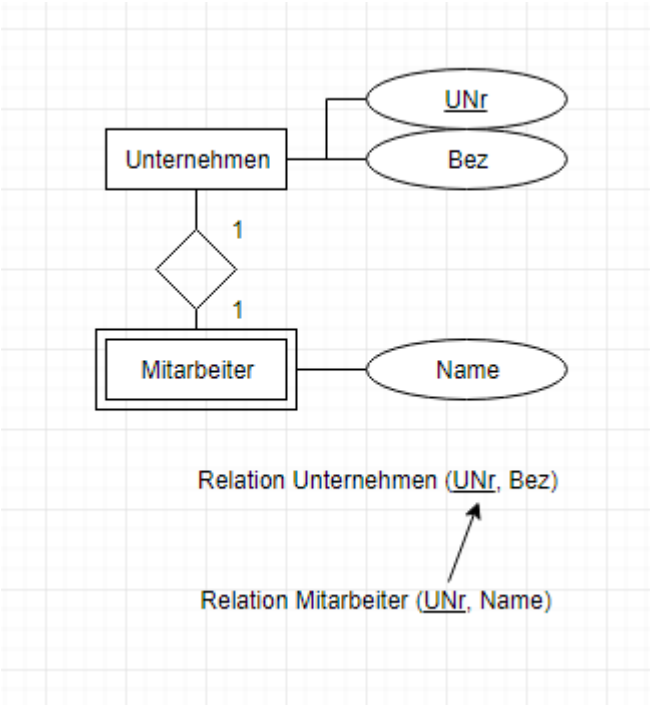


2.5.4.1 disjunkte Entitäten



2.5.5 Weak-Entities

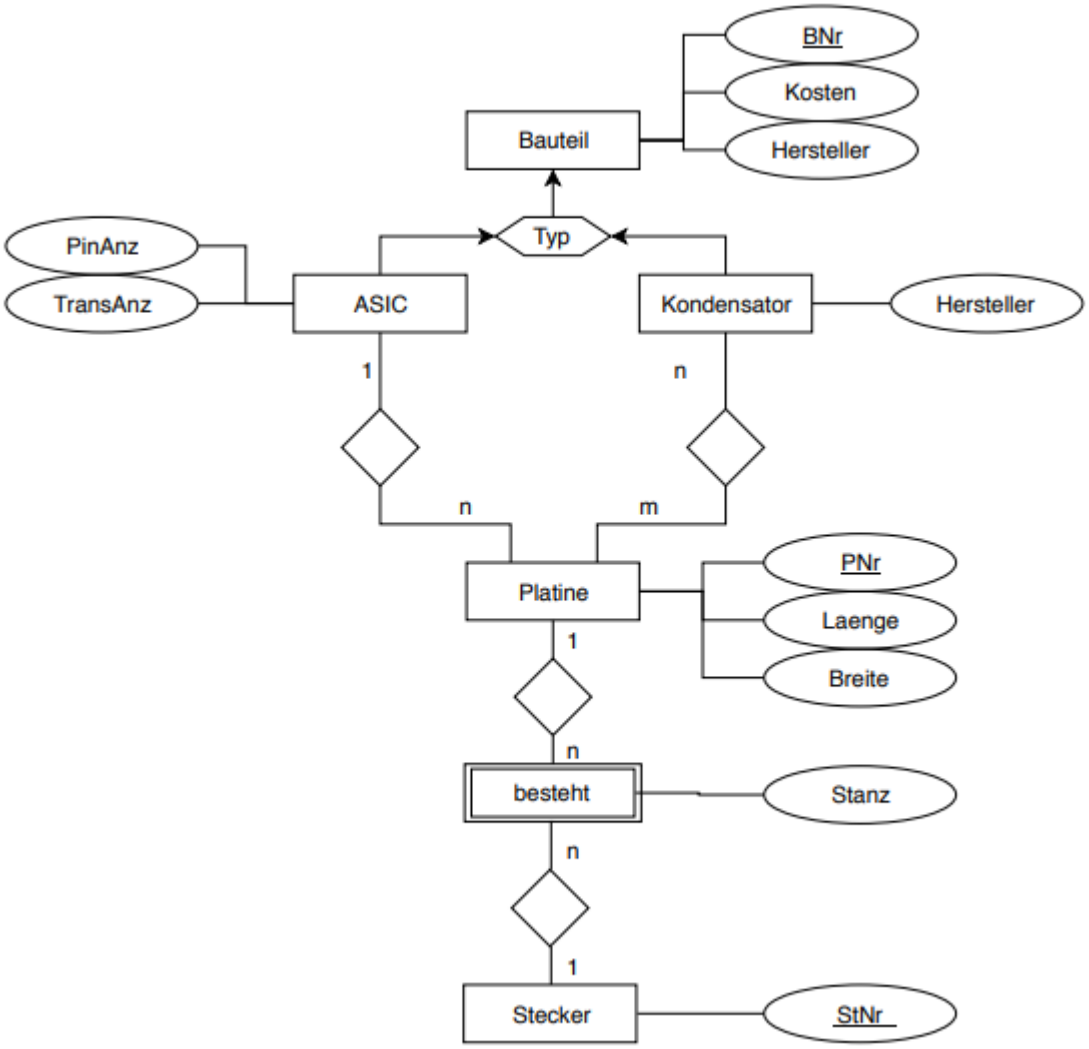
Bei schwachen Entitäten, bei denen die eigenen Attribute nicht ausreichen um ein Tupel eindeutig zu identifizieren, müssen die Schlüsselattribute der damit verbundenen Entitäten zum Schlüssel hinzugenommen werden.



Übungen zur Umsetzung von ER-Modell => Relationenmodell

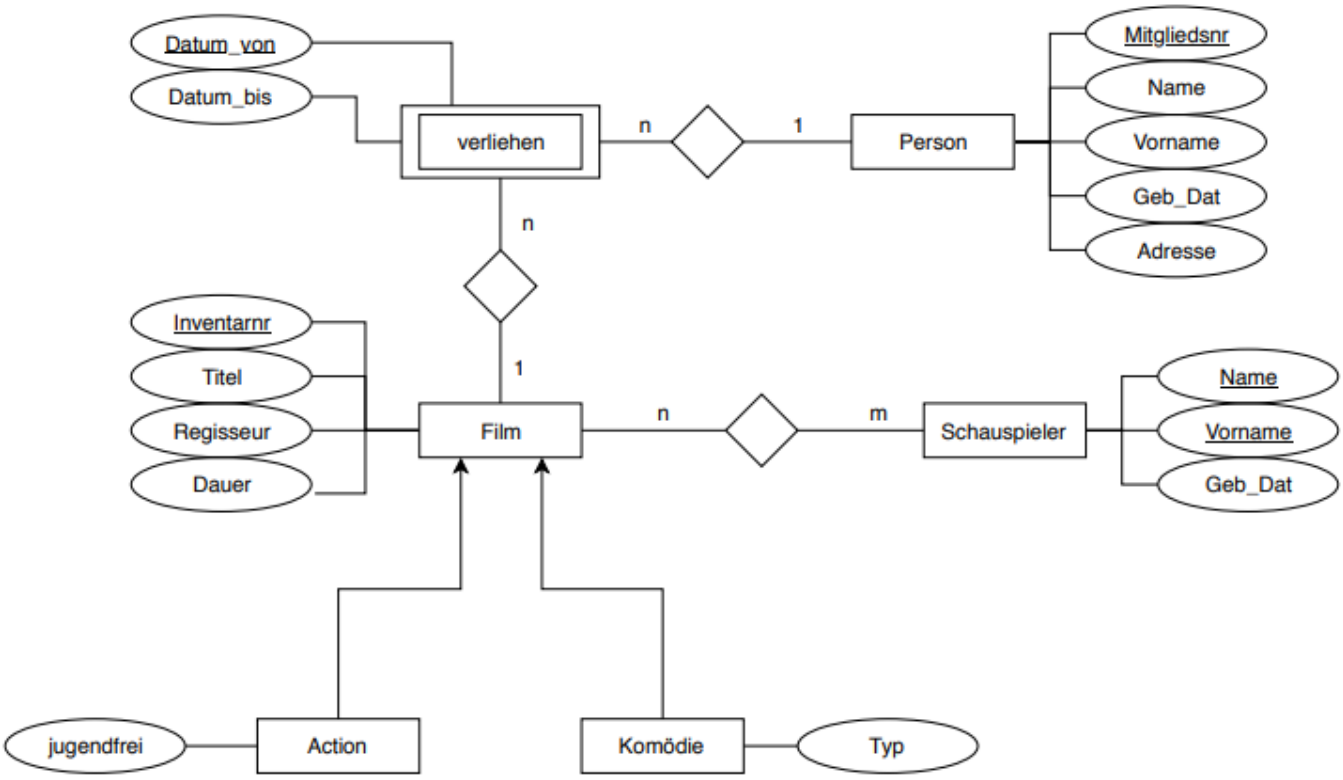
Übung 1

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



Übung 2

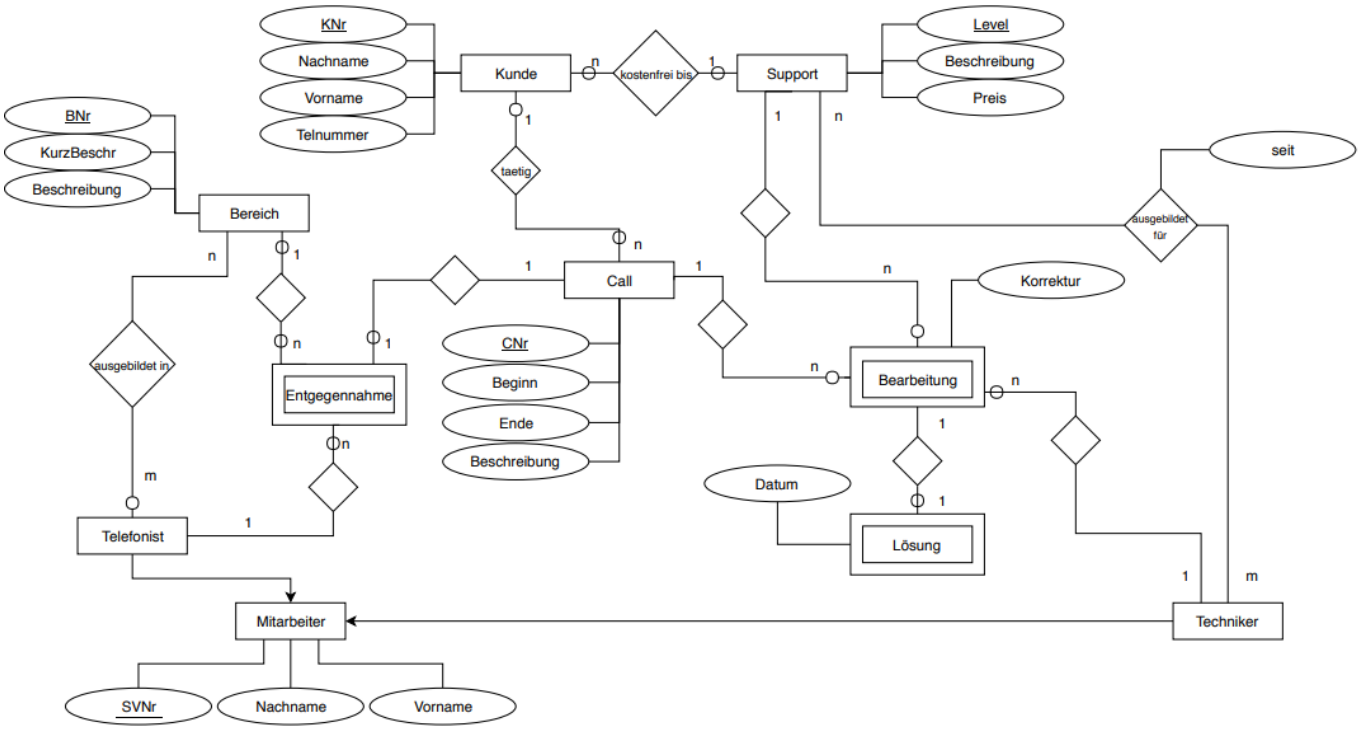
Gegeben ist folgendes ER-Diagramm zur Verwaltung eines Videoverleihs. Entwickeln Sie daraus die Relationen der Datenbank.



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.



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

_CD_ID_	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

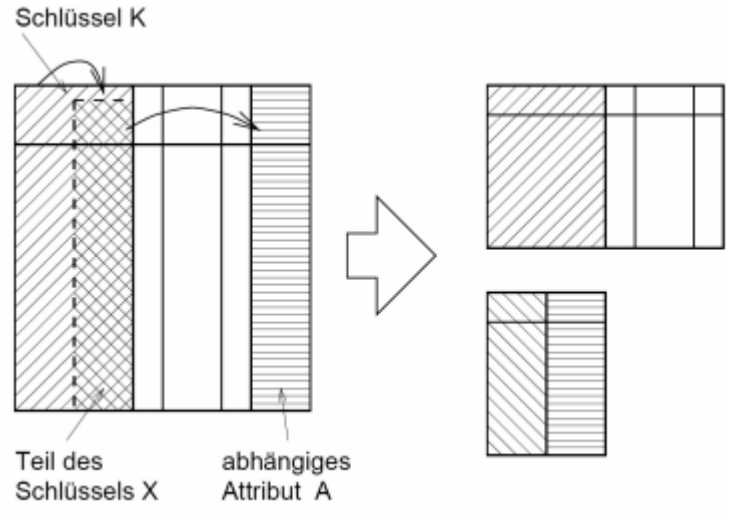
_CD_ID_	Album	Interpret	Titelliste
4711	All That You Can Leave Behind	U2	Beautiful Day
4711	All That You Can Leave Behind	U2	Walk On
4711	All That You Can Leave Behind	U2	Kite
4711	All That You Can Leave Behind	U2	Wild Honey
4712	Tattoo You	Rolling Stones	Start Me Up
4712	Tattoo You	Rolling Stones	Hang Fire
4712	Tattoo 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.

Jedes Tupel muss immer vom gesamten Schlüssel abhängen, sonst ist es nicht in 2. Normalform!



Beispiel 1

Gegeben ist folgende Tabelle:

_CD_ID_	**Album**	**Interpret**	**_Track_**	**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	Tattoo You	Rolling Stones	1	Start Me Up
4712	Tattoo You	Rolling Stones	2	Hang Fire

Problem

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

_CD_ID_	**Album**	**Interpret**	**_Track_**	**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	Tattoo You	Rolling Stones	2	Hang Fire

Lösung 2NF

_CD_ID_	**Album**	**Interpret**
4711	All That You Can Leave Behind	U2
4712	Tattoo You	Rolling Stones

_CD_ID_	**_Track_**	**Titel**
4711	1	Beautiful Day

_CD_ID_	_Track_	Titel
4711	2	Walk On
4711	3	Kite
4712	1	Start Me Up
4712	2	Hang Fire

Beispiel 2

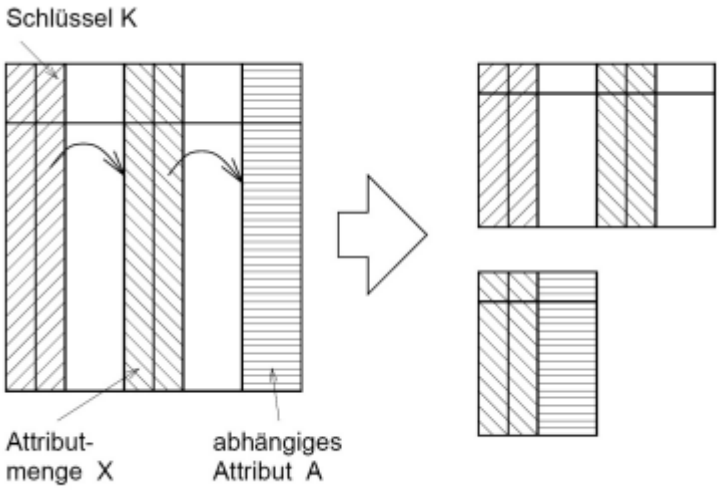
PLZ	Nachname	Ort	Geburtsdatum
3300	Muster	Amstetten	03.05.2003
3300	Maier	Amstetten	05.06.2002
3304	Wimmer	St. Georgen	03.05.2003

Problem: Der Ort hängt nicht vom gesamten Schlüssel ab!

3. Normalform (3NF)

Ein Relationenschema R ist genau dann in 3NF, wenn es die 1NF und die 2NF erfüllt, und wenn kein nicht primes Attribut von einem Schlüssel in R transitiv abhängt.

Transitivitätsregel K→X, X→A : K→A



Beispiel 1

Gegeben sei eine Tabelle mit folgenden Feldern:

_CD_ID_	Album	Interpret	Band-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 **Band-**

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.

_CD_ID_	Album	Interpret	Band-Gründungsjahr
4711	All That You Can Leave Behind	U2	1976
4712	Sticky Fingers	Rolling Stones	**1972**
4713	Tattou You	Rolling Stones	1962

Lösung

_CD_ID_	Album	Interpret
4711	All That You Can Leave Behind	U2
4712	Sticky Fingers	Rolling Stones
4713	Tattou You	Rolling Stones

Interpret	Band-Gründungsjahr
U2	1976
Rolling Stones	1962

Beispiel 2

_ID_	PLZ	Nachname	Ort	Geburtsdatum
1	3300	Muster	Amstetten	03.05.2003
2	3300	Maier	Amstetten	05.06.2002
3	3304	Wimmer	St. Georgen	03.05.2003

Problem: Der Ort ist eigentlich von der PLZ abhängig

Lösung

Zwei Tabellen:

PLZ	Ort
3300	Amstetten
3304	St. Georgen

_ID_	PLZ	Nachname	Geburtsdatum
1	3300	Muster	03.05.2003
2	3300	Maier	05.06.2002
3	3304	Wimmer	03.05.2003

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 Sprachelement von SQL lassen sich in mehrere Kategorien unterteilen, die allerdings im Standard nicht festgeschrieben sind.

Allgemeines

Öffnen der MySQL-Konsole

- Xampp starten mit MySQL
- cmd
- in C:\xampp\mysql\bin wechseln
- mysql -u root

MySQL Befehlsreferenz

- mysql-befehlsreferenz.pdf

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

2.7.1) DDL - Data Definition Language

2.7.1.1) Datentypen

In SQL stehen folgende Datentypen zur Auswahl:

- exakt numerisch
  - INTEGER
    - 4 Bytes
    - Ganzzahlen von 0 bis ~4,3 Mill. oder von -2.147.483.648 bis +2.147.483.647
  - SMALLINT
    - 2 Bytes
    - Ganzzahlen von 0 bis 65.535 oder von -32.768 bis +32.767
  - NUMERIC [(M,D)]
    - M…maximale Anzahl der gezeigten Stellen
    - D…Anzahl der Kommastellen bei einer Dezimalzahl
  - DECIMAL [(M,D)]
- angenähert numerisch
  - DOUBLE
    - 8 Bytes
    - Fließkommazahl
  - REAL
    - Alias für DOUBLE
  - FLOAT
    - 4 Bytes
- Zeichenketten
  - CHAR
    - feste Länge
    - 0-255 Zeichen
  - VARCHAR
    - variable Länge
- Bitketten
  - BLOB (Binary Large Objects)
- Datum und Uhrzeit
  - DATE
    - 3 Bytes
    - Datum im Format 'YYYY-MM-DD'. Wertebereich von 01.01.1000 bis 31.12.9999
  - TIME
    - 3 Bytes
    - Zeit zwischen -838:59:59 und +839:59:59. Ausgabe: 'hh:mm:ss'
  - TIMESTAMP
    - 4 Bytes
    - Zeitstempel. Wertebereich: 1.1.1970 bis 2037.
- Logischer Datentyp
  - BOOLEAN

2.7.1.2) ERZEUGEN von Relationenschemata

Mit dem Befehl „**CREATE TABLE *relationenname* (...)**“ wird ein Relationenschema definiert. Zu jedem Attribut wird ein Typ angegeben. Optional können eine oder mehrere Integritätsbedingungen angegeben werden. Mögliche Integritätsbedingungen sind:

- die Angabe, dass das Attribut einen Primärschlüssel darstellt (**PRIMARY KEY**)
- dass das Attribut keinen Nullwert annehmen darf (**NOT NULL**)
- dass ein Attribut eindeutig sein muss (**UNIQUE**) - diese Bedingung gilt automatisch für Primärschlüsselattribute
- oder eine durch die Klausel „**CHECK (*bedingung*)**“ formulierte Wertebereichseinschränkung

Weiters kann für Attribute durch die Klausel „**DEFAULT *wert***“ ein Vorgabewert angegeben werden.

Beispiel

Es soll eine leere Relation mit dem Namen **Restaurant** erstellt werden. Es wird festgelegt, dass das Attribut **rnr** ein Primärschlüssel ist, dass die Attribute **name** und **adresse** keine Nullwerte annehmen dürfen, und dass **haube** nur die Werte (0-4) annehmen darf. Für den Typ eines Restaurants ist als Defaultwert „österreichisch“ festzulegen.

```
CREATE TABLE restaurant
(
    rnr INT NOT NULL,
    name VARCHAR(100) NOT NULL,
    adresse VARCHAR(100) NOT NULL,
    haube INT CHECK (0<=haube AND haube <=4),
           /*oder CHECK (haube IN (0,1,2,3,4)),*/
           /*oder CHECK (haube BETWEEN 0 AND 4),*/
    typ VARCHAR(100) DEFAULT 'österreichisch',
    CONSTRAINT PK_Restaurant PRIMARY KEY(rnr)
);
```

Eine spezielle Integritätsbedingung (zu Attributen oder zu Relationen) ist die mittels einer **REFERENCES-Klausel** angegebene Fremdschlüsselbedingung **FOREIGN KEY**, die eine Abhängigkeit repräsentiert. Zu jeder Fremdschlüsselbedingung kann durch **„ON UPDATE *action*„** und **„ON DELETE *action*„** angegeben werden, wie auf Verletzungen durch Änderung oder Löschen des referenzierten Schlüsselwertes reagiert werden soll. Mögliche Aktionen sind:

- die Änderung bzw. das Löschen zu verhindern (**NO ACTION**)
- fortzusetzen (**CASCADE**) oder
- den Wert des referenzierenden Attributes auf einen Nullwert (**SET NULL**) bzw. den Defaultwert des Attributs (**SET DEFAULT**) zu setzen

Beispiel

Der nächste SQL-Befehl erzeugt das Relationenschema **Speise**. Die angeführten Integritätsbedingungen legen u.a. fest, dass die Attribute **rnr** und **name** gemeinsam den

Primärschlüssel der Relation **Speise** bilden und dass das Attribut **rnr** der Relation **Speise** ein Fremdschlüssel ist, der sich auf das Schlüsselattribut **rnr** der Relation **Restaurant** bezieht (**Speise[rnr] ⊆ Restaurant[rnr]**). Die Angabe **ON UPDATE CASCADE** legt fest, dass eine Änderung der Nummer eines Restaurants bei den entsprechenden Speisen mitgezogen wird, und die Angabe **ON DELETE NO ACTION** legt fest, dass ein Restaurant nicht gelöscht werden darf, solange noch Speisen für dieses Restaurant vorhanden sind.

```
CREATE TABLE Speise
(
    rnr INTEGER,
    name VARCHAR(150),
    CONSTRAINT PK_Speise PRIMARY KEY (name),
    CONSTRAINT FK_Speise FOREIGN KEY (rnr) REFERENCES restaurant(rnr) ON
UPDATE CASCADE ON DELETE NO ACTION
);
```

2.7.1.3) ÄNDERN von Relationenschemata

Die Änderung der Struktur einer Tabelle wird mit dem Befehl „**ALTER TABLE *relationenname…***“ durchgeführt.

Beispiel - Hinzufügen eines Attributes KALORIEN zur Relation SPEISE

```
ALTER TABLE Speise ADD Kalorien INT;
```

Beispiel - Hinzufügen einer CHECK-Klausel für Kalorien

```
ALTER TABLE Speise ADD CONSTRAINT ck CHECK (Kalorien>=0);
```

Beispiel - Entfernen eines Attributes

```
ALTER TABLE Restaurant DROP Adresse;
```

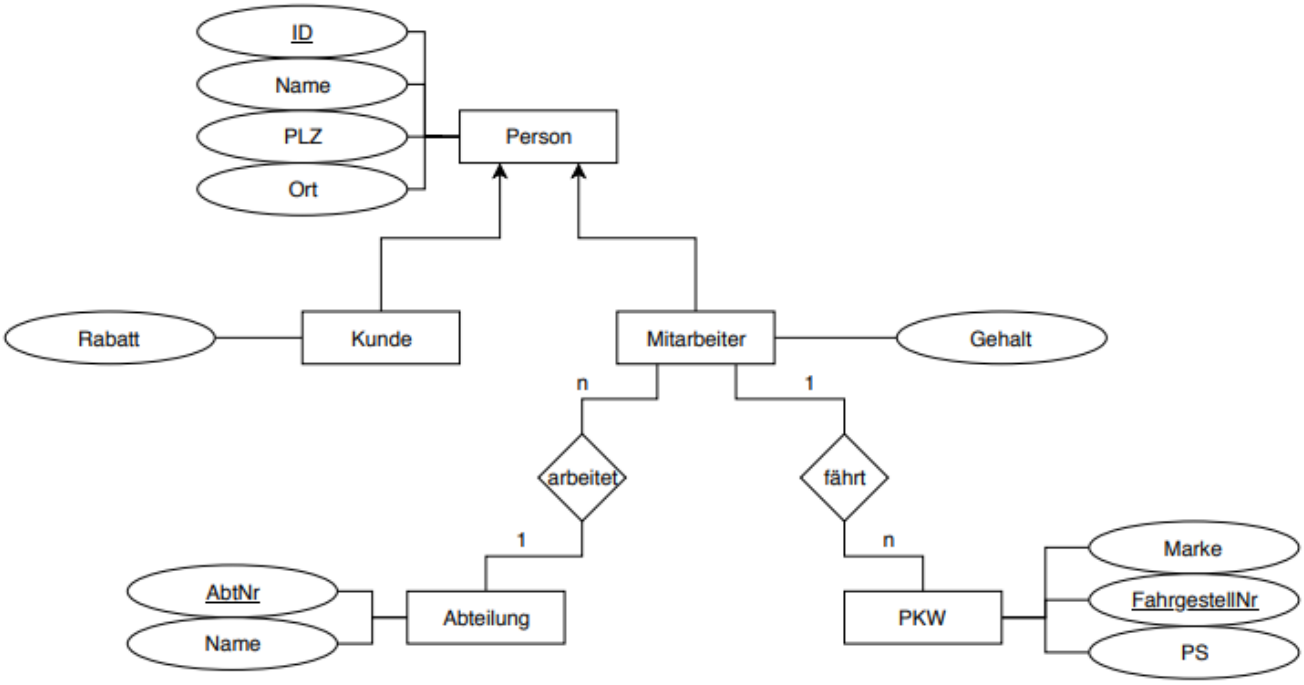
2.7.1.4) ENTFERNEN von Relationenschemata

Mit dem Befehl „**DROP TABLE *relationenname***“ wird eine Relation wieder gelöscht.

```
DROP TABLE Restaurant;
```

2.7.1.5) Übungsbeispiel

Gegeben ist folgendes ER-Modell:



Gesucht sind die SQL-Statements zur Erstellung der Relationen

```
CREATE TABLE Orte (
    Plz INT NOT NULL,
    Ort VARCHAR(100),
    CONSTRAINT PK_Plz PRIMARY KEY(Plz)
);

CREATE TABLE Person (
    ID INT NOT NULL,
    Name VARCHAR(100),
    Plz INT NOT NULL,
    CONSTRAINT FK_Person FOREIGN KEY (Plz) REFERENCES Orte(Plz),
    CONSTRAINT PK_ID PRIMARY KEY (ID)
);

CREATE TABLE Kunde (
    ID INT NOT NULL,
    Rabatte INT NOT NULL,
    CONSTRAINT PK_ID PRIMARY KEY (ID),
    CONSTRAINT FK_KUnde FOREIGN KEY (ID) REFERENCES Person(ID) ON DELETE
CASCADE
);

CREATE TABLE Abteilung(
    AbtNr INT NOT NULL,
    Name VARCHAR(140),
```

```
    CONSTRAINT PK_AbtNr PRIMARY KEY(AbtNr)
);

CREATE TABLE Mitarbeiter(
    ID INT NOT NULL,
    Gehalt INT NOT NULL,
    AbtNr INT NOT NULL,
    CONSTRAINT PK_ID PRIMARY KEY(ID),
    CONSTRAINT FK_Mitarbeiter_AbtNr FOREIGN KEY(AbtNr) REFERENCES
Abteilung(ID),
    CONSTRAINT FK_Mitarbeiter_ID FOREIGN KEY(ID) REFERENCES PERSON(ID)
);

CREATE TABLE PKW(
    Marke VARCHAR(100),
    PS INT NOT NULL,
    FahrgestellNr INT NOT NULL,
    ID INT NOT NULL,
    CONSTRAINT PK_fgsNr PRIMARY KEY (FahrgestellNr),
    CONSTRAINT FK_PKW_ID FOREIGN KEY (ID) REFERENCES Mitarbeiter(ID)
);
```

DDL-Übungen

Aufgabe 1 (am PC)

Gegeben sind folgende Relationen:

Relation Rechner

RNr	StudAss	Leist	Speicher
R1	Huber	1	100
R2	Brunner	3	80
R3	Brunner	3	400
R4	Vogt	2	120
R5	Huber	2	500

Relation Programm

PNr	PName	MinLeist	Bereich
P1	DrawIt	1	Grafik
P2	AskIt	3	Datenbank
P3	WriteIt	1	Text
P4	ConnectIt	2	Internet
P5	PaintIt	2	Grafik
P6	StoreIt	3	Datenbank

Relation Installation

RNr	PNr	Platz	Code
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

Relation Benutzung

ANr	PNr	Stund
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

Relation Assistent

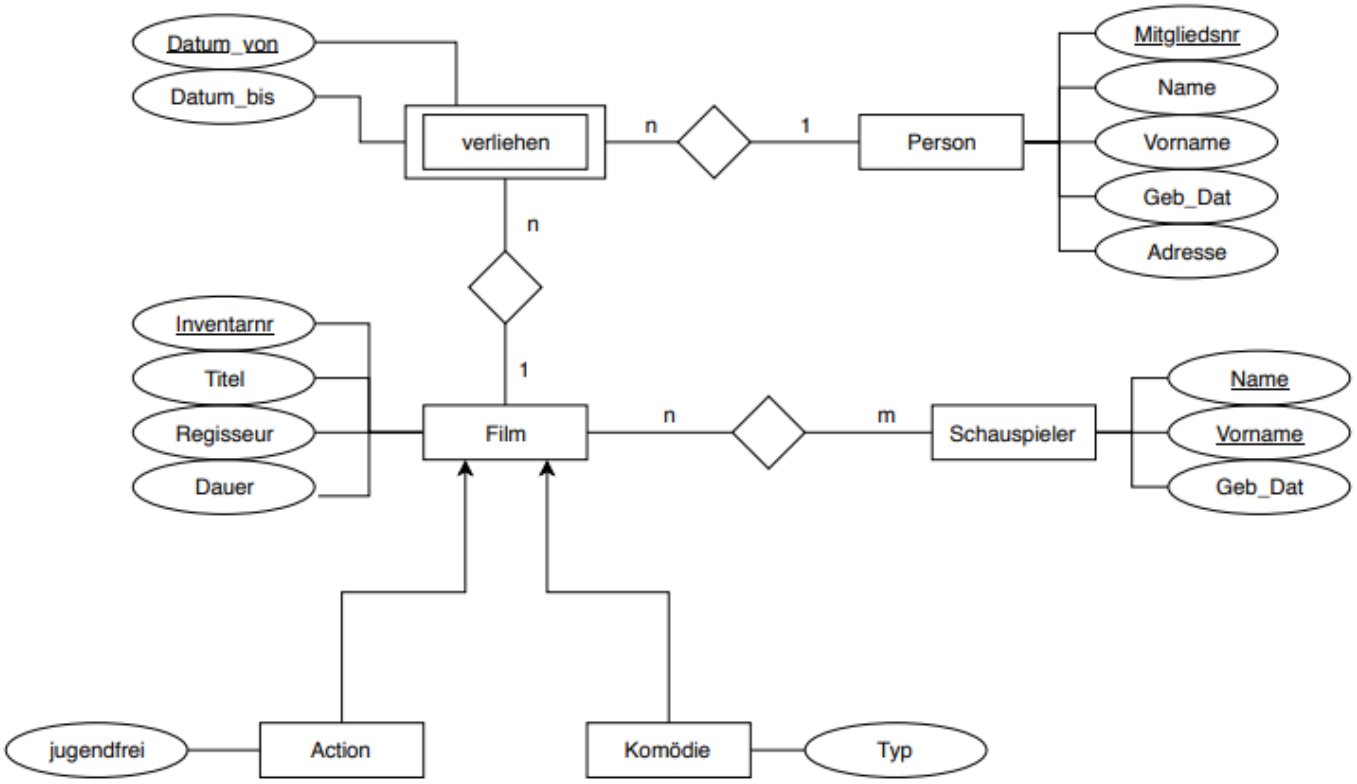
ANr	AName	StudAss	Gehalt
A1	Novak	Brunner	3
A2	Dvorak	Vogt	1
A3	Husak	Vogt	1
A4	Pfeiffer	Brunner	2

Gesucht sind die SQL-Statements zur Erstellung der Relationen! Erstellen sie die Relationen mithilfe der SQL-Statements in einer MySQL-Datenbank namens „Rechnerverwaltung“(XAMPP)! Zum Erstellen der Datenbank verwendet folgenden Befehl:

```
CREATE DATABASE Rechnerverwaltung;
```

Aufgabe 2 (am PC)

Gegeben ist folgendes ER-Modell:



Gesucht sind die SQL-Statements zur Erstellung der dafür notwendigen Relationen! Erstellen sie die Relationen mithilfe der SQL-Statements in einer MySQL-Datenbank namens „Filmverwaltung“(XAMPP)!

Aufgabe 3 (am PC)

Gegeben sind folgende Relationen:

Relation Winzer					
wnr	**name**	**strasse**	**plz**	**ort**	**telefon**
1	Lackner Tinnacher	Steinbach 8	4567	Gamlitz	1234567
2	Weingut Prager	Weissenkirchen 48	3610	Weissenkirchen	1234567
3	Weingut Emmerich Knoll	Unterloiben 10	3601	Unterloiben	12344457
4	Weingut F.X. Pichler	Unterloiben 27	3601	Unterloiben	11122233
5	Weingut Spätlese	Weintalstrasse 23	1136	Wien	
6	Freie Weingarten Wachau	Kremstalstrasse 23	3600	Krems	2304002
7	Stiftskellerei				

Relation Winzer					
wnr	**name**	**strasse**	**plz**	**ort**	**telefon**
8	Weingut Biegler	Wienerstrasse 88	3502	Gumpoldskirchen	54564565

Relation Wein							
nr	**bezeichnung**	**sorte**	**jahrgang**	**preis**	**anzahl**	***wnr***	**position**
1	Riesling Kellerberg	Riesling	1999	28.00	24	4	1
2	Loibenberg	Gr. Veltliner	2000	19.00	36	4	2
3	Ried Kreutles	Gr. Veltliner	2000	19.00	15	3	4
4	Riesling Smaragd	Riesling	2000	21.00	30	2	5
5	Grauburgunder	Grauburgunder	2003	16.00	72	1	6
6	Morillon	Chardonnay	2003	9.00	55	1	7
7	Riesling Federspiel	Riesling	2003	9.90	80	6	3
8	Chardonnay	Chardonnay	2003	9.00	16	8	8

Relation Keller			
knr	**reihe**	**regal**	**fach**
1	1	1	1
2	1	1	2
3	1	1	3
4	1	2	1
5	1	2	2
6	2	1	1
7	2	1	2
8	2	2	1

Relation Protokoll				
pnr	***nr***	**pDatum**	**verwendung**	**anzahl**
1	1	2003-01-12	Geschenk an Herrn Berger	12
2	3	2003-07-10	Eigenbedarf	2
3	1	2003-07-23	Eigenbedarf	4
4	6	2003-08-14	Geschenk (Frau Kunz)	6
5	1	2003-08-27	Glasbruch	1
6	4	2003-11-03	Korkgeruch	1
7	6	2003-11-03	Eigenbedarf	3

Gesucht sind die SQL-Statements zur Erstellung der dafür notwendigen Relationen! Erstellen sie die Relationen mithilfe der SQL-Statements in einer MySQL-Datenbank namens „Winzerverwaltung“(XAMPP)!

2.7.2) DQL - Data Query Language

2.7.2.1) Grundkonstruktion einer SQL-Abfrage

Grundkonstruktion einer SQL-Abfrage ist von der Form

„**SELECT** *attributListe* **FROM** *relation*“

Beispiel

```
SELECT name,adresse FROM restaurant;
```

entspricht
$ π_{name,adresse}(restaurant) $

```
SELECT * FROM restaurant;
/*liefert die gesamte Tabelle (d.h. mit allen Attributen)
```

SQL-Abfrage mit Verknüpfung mehrerer Relationen:

„**SELECT** *attributListe* **FROM** *relationen* **WHERE** *bedingung*“

Eine solche SQL-Abfrage wird ausgewertet, indem zunächst das **kartesische Produkt** der in der **FROM**-Klausel angegebenen Relationen gebildet wird. Daraus werden dann jene Tupel ausgewählt, die die in der **WHERE**-Klausel angegebenen Bedingungen erfüllen. Zudem wird auf die in der **SELECT**-Klausel angegebenen Attribute dieser Tupel projiziert.

Beispiel

Wir wollen wissen welche Speisen in welchem Restaurant angeboten werden.

```
SELECT * FROM Speise, Restaurant WHERE Restaurant.rnr = Speise.rnr;
```

Die Bedingunng in der WHERE-Klausel kann sich aus mehreren, durch die logischen Operatoren AND, OR und NOT verbundenen Teilbedingungen zusammensetzen.

Beispiel

```
    SELECT rnr, name, haube, typ
    FROM restaurant
```

```
    WHERE haube > 1 AND NOT (typ='österreichisch' OR typ='international');
```

In der SELECT-Klausel können auch Ausdrücke, die aus Attributen berechnet werden, stehen. Durch das Schlüsselwort **AS** kann einem so berechneten Ausdruck ein Attributname zugewiesen werden.

Beispiel

Wir wollen alle Paare von Restaurants auswählen, die sich um mehr als zwei Hauben unterscheiden und die Differenz der Hauben ausgeben.

```
    SELECT r1.name, r2.name, (r2.haube-r1.haube) AS differenz
    FROM restaurant r1, restaurant r2
    WHERE r1.haube +2 < r2.haube;
```

In SQL werden Duplikate, d.h. Ergebnistupel mit identischen Werten, nicht automatisch eliminiert. Um Duplikate zu eliminieren, muss der Liste von ausgewählten Attribute das Schlüsselwort **DISTINCT** vorangestellt werden.
Durch die Angabe einer ORDER BY-Klausel mit den Schlüsselworten ASC oder DESC kann die Ausgabe aufsteigend bzw. absteigend nach bestimmten Attributen sortiert werden. Der Defaultwert ist ASC.

Beispiel

Wir wollen eine alphabetisch sortierte Liste aller Speisen von Restaurants mit mehr als einer Haube.

```
    SELECT DISTINCT s.name
    FROM Restaurant r, Speise r
    WHERE (r.rnr = s.rnr) AND (r.haube>1)
    ORDER BY s.name, s.preis;
```

Übungsbeispiele

Gegeben sind folgende Relationen

Person (id, name, plz)
Orte (plz, ort)
Mitarbeiter (id, gehalt, abtnr)
Kunde (id, rabatt)
Abteilung (abtnr, name)
PKW (marke, ps, id, fahrgestellnr)

Aufgaben

Entwickeln Sie SQL-Anweisungen für folgende Abfragen

1. Geben Sie die Namen aller Personen aus Amstetten aus

2. Geben Sie die Namen aller Mitarbeiter aus Amstetten aus
3. Gesucht sind die Namen aller Mitarbeiter die auch Kunden sind
4. Namen aller Mitarbeiter die PKWs mit mehr als 1000 PS fahren
5. Gesucht sind die Namen jener Orte, in denen Mitarbeiter wohnen, die in einer Abteilung „Buchhaltung“ arbeiten

Lösungen

```
1)
SELECT name
FROM Person p, Orte o
WHERE o.ort="Amstetten" AND p.plz=o=plz;


2)
SELECT name
FROM Person p, Orte o, Mitarbeiter m
WHERE m.id=p.id AND o.ort="Amstetten" AND p.plz=o.plz;


3)
SELECT name
FROM Person p, Orte o, Mitarbeiter m
WHERE m.id=p.id AND m.id=k.id;

4)
SELECT name
FROM Person p, Mitarbeiter m, PKW
WHERE p.id=m.id AND PKW.id=p.id AND PKW.ps>100;

5)
SELECT ort
FROM Orte o, Mitarbeiter m, Person p, Abteilung a
WHERE o.plz=p.plz AND p.id=m.id AND m.abtnr=a.abtnr AND p.plz=o.plz;
```

2.7.2.2) Die WHERE-Klausel

Neben den Boolschen Funktionen (AND,OR,NOT) sind folgende weitere Angaben zulässig:

IN-Operator

Anstatt eine Reihe von OR-Verknüpfungen durchzuführen, kann auch der IN-Operator verwendet werden.

Bsp:

```
SELECT * FROM ... WHERE typ IN ('österreichisch', 'international');
```

BETWEEN-Operator

Bsp.:

```
SELECT * FROM ... WHERE preis BETWEEN 100 AND 150;

//ist dasselbe wie

SELECT * FROM ... WHERE preis >= 100 AND preis<=150;
```

IS NULL-Operator

Bsp.: Es sollen alle Personen gesucht werden, die beim Attribut TelNr den NULL-Wert eingetragen haben.

```
SELECT * FROM ... WHERE TelNr IS NULL;
```

LIKE-Operator

Der Like-Operator ist ein Vergleichsoperator, der Datenwerte einer Spalte mit einem vorgegebenen Muster vergleicht.

Allgemeine Form:

AttributName [NOT] LIKE 'muster';

Dabei haben in 'muster' zwei Zeichen eine besondere Bedeutung:

- % .... steht für 0 bis beliebig viele Zeichen
- _ .... steht für genau ein beliebiges Zeichen

Bsp.: Alle Speisen deren Name mit 'M' beginnt

```
SELECT Name FROM Speise WHERE Name LIKE 'M%';
```

Bsp.: Alle Restaurants, welche an 2. Stelle im Namen ein 'o' haben

```
SELECT Name FROM Restaurant WHERE Name LIKE '_o%';
```

2.7.2.3) Mengenoperationen

SQL stellt die Mengenoperationen **UNION** (Vereinigung), **INTERSECT** (Durchschnitt) und **EXCEPT** (Mengendifferenz) für typkompatible Relationen (d.h. Relationen mit der gleichen Anzahl von jeweils typkompatiblen Attributen) zur Verfügung.

Beispiel: Wir wollen eine Liste der Namen aller Restaurants und der Namen aller Speisen

```
(SELECT Name FROM Restaurant)
UNION
(SELECT Name FROM Speise)
```

Mit typkompatiblen Relationen sind dabei nicht die Relationen **Restaurant** oder **Speise** gemeint, sondern die Ergebnisse der beiden **SELECT** Statements.

2.7.2.4) Aggregatfunktionen und Gruppierung

Aggregatfunktionen

Es gibt folgende Aggregatfunktionen in SQL:

- MIN
- MAX
- SUM
- AVG
- COUNT

Bsp: Gesucht ist der Preis des teuersten Burgers:

```
SELECT MAX(Preis)
FROM Speise
WHERE Name LIKE '%Burger%';
```

Bsp.: Wie viele Speisen werden im Brauhof angeboten?

```
SELECT COUNT (*)   //COUNT (*) zählt alle Tupel
FROM Restaurant r, Speise s,
WHERE r.rnr = s.rnr AND r.name="Brauhof";
```

Gruppierung

Die Select-Abfrage kann um eine Klause „**GROUP BY** *attributListe*“ und eine Klausel „**HAVING** *bedingung*“ ergänzt werden, um Gruppen von Tupeln zu bilden und auf diese Gruppen Aggregatfunktionen anzuwenden.
Dabei werden diese Tupel nach gemeinsamen Werten in den in der **GROUP BY**-Klausel angeführten

Attributen gruppiert. Von diesen Gruppen werden jene eliminiert, die die in der **HAVING**-Klausel angegebene Bedingung nicht erfüllen.

Anschließend werden etwaige in der **SELECT**-Klausel angegebenen Aggregatfunktionen auf jede der verbliebenen Gruppen angewendet. Diese Aggregatfunktionen (außer **COUNT**) können auch auf arithmetische Ausdrücke über Attributen angewandt werden.

Beispiel

Relation Restaurant			
rnr	**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

Frage: Wie viele Restaurants haben die gleiche Haubenanzahl?

```
SELECT Haube, COUNT (*) AS AnzahlRestaurants
FROM Restaurant
GROUP BY Haube;
```

Ergebnis:

Haube	**AnzahlRestaurants**
0	2
1	2
2	2

Frage: Wie viele Restaurants in Amstetten haben die gleiche Haubenanzahl?

```
SELECT Haube, COUNT (*) AS AnzahlRestaurants
FROM Restaurant
WHERE Adr='Amstetten'
GROUP BY Haube;
```

Ergebnis:

Haube	**AnzahlRestaurants**
0	1
1	1
2	2

Beispiel

Gesucht ist eine SQL-Abfrage, die alle Restaurants (rnr und name) und die Anzahl der jeweils gebotenen Speisen ausgibt:

Relation Restaurant			
rnr	**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		
rnr	**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

```
SELECT r.rnr, r.name, COUNT(*) AS Anzahl
FROM Restaurant r, Speise s
WHERE r.rnr=s.rnr
GROUP BY r.rnr, r.name;
```

1. Schritt: JOIN

1	McDonalds	Amstetten	0	Hamburger	1
1	McDonalds	Amstetten	0	Salat	1,80
2	McDonalds	Ybbs	0	Hamburger	1
2	McDonalds	Ybbs	0	Cheeseburger	1,30
3	Hollywood	Amstetten	1	Hawaii	8,6
4	Casa Venezia	Amstetten	2	Hawaii	9
5	grill.Bar	Amstetten	2	Salat	5

2. Schritt: GROUP BY

1	McDonalds	Amstetten	0	Hamburger	1
1	McDonalds	Amstetten	0	Salat	1,80
2	McDonalds	Ybbs	0	Hamburger	1
2	McDonalds	Ybbs	0	Cheeseburger	1,30
3	Hollywood	Amstetten	1	Hawaii	8,6
4	Casa Venezia	Amstetten	2	Hawaii	9
5	grill.Bar	Amstetten	2	Salat	5

3. Schritt: COUNT (*) AS Anzahl (pro Gruppe)

rnr	RName	Anzahl
1	McDonalds	2
2	McDonalds	2
3	Hollywood	1
4	Casa Venezia	1
5	grill.Bar	1

Beispiel

Gesucht sind alle Restaurants (rnr, name), welche mehr als 5 verschiedene (Tipp: Schlüsselwort DISTINCT) Speisen anbieten!

Relation Restaurant			
rnr	**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		
rnr	**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

Beispiel

Gesucht ist eine SQL-Abfrage, die die rnr und die Anzahl der Haube jener Restaurants angibt, deren teuerste Speise mehr als 8€ kostet. Weiters werden jeweils die Anzahl der Speisen als auch deren Durchschnittspreis ausgegeben.

Relation Restaurant			
rnr	**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		
rnr	**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
4	Hühnerstreifensalat	8,20

Überschrift

DQL - ÜBUNGEN

Aufgabe 1 (am PC)

Gegeben sind folgende Relationen:

Relation Rechner

RNr	StudAss	Leist	Speicher
R1	Huber	1	100
R2	Brunner	3	80
R3	Brunner	3	400
R4	Vogt	2	120
R5	Huber	2	500

Relation Programm

PNr	PName	MinLeist	Bereich
P1	DrawIt	1	Grafik
P2	AskIt	3	Datenbank
P3	WriteIt	1	Text
P4	ConnectIt	2	Internet
P5	PaintIt	2	Grafik
P6	StoreIt	3	Datenbank

Relation Installation

RNr	PNr	Platz	Code
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

Relation Benutzung

ANr	PNr	Stund
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

Relation Assistent

ANr	AName	StudAss	Gehalt
A1	Novak	Brunner	3
A2	Dvorak	Vogt	1
A3	Husak	Vogt	1
A4	Pfeiffer	Brunner	2

Welches Ergebnis liefern die folgenden SQL-Abfragen? Überprüfen Sie ihre DQL-Statements anhand der in 2.7.1 erstellten Datenbank namens **„Rechnerverwaltung“** (Sind Fehler enthalten, so sind diese zu markieren)

a)

```
SELECT I.RNr
FROM Installationen I, Programm P
WHERE I.RNr=P.RNr AND Bereich='Grafik';
```

b)

```
SELECT StudAss, SUM(Gehalt) AS Ges
FROM Assistent
GROUP BY StudAss;
```

c)

```
SELECT StudAss, SUM(Gehalt)
FROM Assistent
GROUP BY StudAss
HAVING SUM(Gehalt)>4;
```

d)

```
SELECT SUM(Speicher)
FROM Rechner
WHERE Leist=3;
```

Aufgabe 2

In einem Weinkeller werden viele Weinflaschen (WEIN) gelagert. Jede Weinflasche wurde von einem Winzer (WINZER) befüllt und kann anhand der Datenbank leicht in den Regalreihen (KELLER) des Kellers gefunden werden. Es kann dabei angenommen werden, dass immer ausreichend Platz in den Regalen des Kellers vorhanden ist. Wird eine Flasche aus dem Keller entfernt, wird ein entsprechender Eintrag (PROTOKOLL) vermerkt. Wenn der Eigentümer des Kellers eine Flasche selbst trinkt, wird im Protokoll als Verwendung 'Eigenbedarf' eingetragen.

Relation Winzer

wnr	name	strasse	plz	ort	telefon
1	Lackner Tinnacher	Steinbach 8	4567	Gamlitz	1234567
2	Weingut Prager	Weissenkirchen 48	3610	Weissenkirchen	1234567
3	Weingut Emmerich Knoll	Unterloiben 10	3601	Unterloiben	12344457
4	Weingut F.X. Pichler	Unterloiben 27	3601	Unterloiben	11122233
5	Weingut Spätlese	Weintalstrasse 23	1136	Wien	
6	Freie Weingarten Wachau	Kremstalstrasse 23	3600	Krems	2304002
7	Stiftskellerei				
8	Weingut Biegler	Wienerstrasse 88	3502	Gumpoldskirchen	54564565

Relation Wein

nr	bezeichnung	sorte	jahrgang	preis	anzahl	wnr	position
1	Riesling Kellerberg	Riesling	1999	28.00	24	4	1
2	Loibenberg	Gr. Veltliner	2000	19.00	36	4	2
3	Ried Kreutles	Gr. Veltliner	2000	19.00	15	3	4

Relation Wein							
nr	**bezeichnung**	**sorte**	**jahrgang**	**preis**	**anzahl**	**wnr**	**position**
4	Riesling Smaragd	Riesling	2000	21.00	30	2	5
5	Grauburgunder	Grauburgunder	2003	16.00	72	1	6
6	Morillon	Chardonnay	2003	9.00	55	1	7
7	Riesling Federspiel	Riesling	2003	9.90	80	6	3
8	Chardonnay	Chardonnay	2003	9.00	16	8	8

Relation Keller			
knr	**reihe**	**regal**	**fach**
1	1	1	1
2	1	1	2
3	1	1	3
4	1	2	1
5	1	2	2
6	2	1	1
7	2	1	2
8	2	2	1

Relation Protokoll				
pnr	**nr**	**pDatum**	**verwendung**	**anzahl**
1	1	2003-01-12	Geschenk an Herrn Berger	12
2	3	2003-07-10	Eigenbedarf	2
3	1	2003-07-23	Eigenbedarf	4
4	6	2003-08-14	Geschenk (Frau Kunz)	6
5	1	2003-08-27	Glasbruch	1
6	4	2003-11-03	Korkgeruch	1
7	6	2003-11-03	Eigenbedarf	3

Folgende SQL-Abfragen sind gesucht! Überprüfen Sie ihre DQL-Statements anhand der in 2.7.1 erstellten Datenbank namens **„Winzerverwaltung“**

a) Geben Sie für die Sorte ’Riesling’ die Namen aller Winzer sowie die Flaschenanzahl aus. Sortieren Sie dabei nach der Flaschenanzahl absteigend.

name	**anzahl**
Freie Weingarten Wachau	80
Weingut Prager	30
Weingut F.X. Pichler	24

b) Ermitteln Sie für jeden Winzer den durchschnittlichen Flaschenpreis und die Gesamtanzahl der Flaschen im Keller. Berücksichtigen Sie dabei nur Winzer, von denen bekannt ist, aus welchem Ort sie kommen. Sortieren Sie die Liste nach dem Preis absteigend.

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 Weingarten Wachau	9.90	80

name	durchschnittspreis	gesamtanzahl
Weingut Biegler	9.00	16

c) Geben Sie eine Liste aller Weinbezeichnungen sowie den Namen des erzeugenden Winzers aus, von denen im Jahr 2003 keine Flasche getrunken worden ist (Verwendung in Tabelle Protokoll ='Eigenbedarf'). Sie brauchen dabei nur Winzer berücksichtigen, von denen mindestens eine Flasche im Keller vorhanden ist.

nr	bezeichnung	name
2	Loibenberg	Weingut F.X. Pichler
4	Riesling Smaragd	Weingut Prager
5	Grauburgunder	Lackner Tinnacher
7	Riesling Federspiel	Freie Weingarten Wachau
8	Chardonnay	Weingut Biegler

d) Suchen Sie die Winzer, von denen der Kellereigentümer die meisten Flaschen getrunken (Verwendung in Tabelle Protokoll = 'Eigenbedarf') hat. Geben Sie jeweils den Namen des Winzers sowie die Gesamtkosten des von diesem Winzer konsumierten Weines aus.

name	anzahl	kosten
Weingut F.X. Pichler	4	112.00

e) Geben Sie fur jeden Winzer aus,

- wie viele gunstige (Preis ≤ 10 Euro, Preisklasse niedrig),
- wie viele im Mittelfeld (10 Euro bis 20 Euro, Preisklasse mittel) und
- wie viele teure (Preis > 20 Euro, Preisklasse gehoben) Weinflaschen im Keller liegen.

name	anzahl	preisklasse
Freie Weingarten Wachau	80	niedrig
Lackner Tinnacher	55	niedrig
Lackner Tinnacher	72	mittel
Weingut Biegler	16	niedrig
Weingut Emmerich Knoll	15	mittel
Weingut F.X. Pichler	24	gehoben
Weingut F.X. Pichler	36	mittel
Weingut Prager	30	gehoben

f) Erstellen Sie eine Liste, die angibt, wie viele Flaschen jedes in der Datenbank gespeicherten Winzers sich im Keller befinden. Sortieren Sie dabei nach der Flaschenanzahl absteigend.

name	anzahl
Lackner Tinnacher	127
Freie Weingarten Wachau	80
Weingut F.X. Pichler	60
Weingut Prager	30
Weingut Biegler	16
Weingut Emmerich Knoll	15
Weingut Spätlese	

name	anzahl
Stiftskellerei	

2.7.3) DML (Data Manipulation Language)

Unsere Tabellen sind angelegt und warten darauf, mit Daten gefüttert bzw. durchsucht zu werden. Lernen Sie im Folgenden, wie das gemacht wird.
Die Data Manipulation Language umfasst insgesamt 3 Arten von Statements:

2.7.3.1) Einfügen von Datensätzen

Bevor Sie mit einer Datenbank arbeiten, sollten zunächst einmal Daten in der Datenbank vorhanden sein.
Um Daten in eine Datenbank einzufügen, benutzen wir den Befehl INSERT. Dessen Aufbau ist recht einfach:

```
INSERT INTO <Tabelle> [(Spalte1, Spalte2,...)] VALUES (<wert1>,
<wert2>,...);
```

Als Beispiel wollen wir einen Datensatz in die Tabelle Administratoren einfügen:

```
INSERT INTO Administratoren (UserID, USER, Passwort) VALUES
(1,'Meister','geheim');
```

Die Zuordnung, welcher Wert in welche Spalte eingefügt wird, geschieht über die jeweiligen Positionen innerhalb der Klammern;
⇒ der erste Wert „Meister“ wird in die erste Spalte „User“ eingefügt.

Wenn wir einen neuen Datensatz anlegen, müssen wir nicht alle Spalten sofort füllen. Die Tabelle Administratoren hat eigentlich drei Spalten. Damit das aber funktioniert und sich das DBMS nicht beschwert, darf die Spalte nicht als **NOT NULL** definiert sein. Oder, wenn sie es ist, muss sie einen Defaultwert aufweisen.

2.7.3.2) Ändern von Datensätzen

Einen Datensatz können wir mit UPDATE ändern - Vorsicht, es ist nicht möglich, damit einen Datensatz einzufügen.

Hier die vollständige Syntax:

```
UPDATE Relation
   SET <Spalte1>=<Wert> [, <Spalte2>=<Wert>, ...]
     [WHERE <Bedingung>]
```

Die WHERE-Bedingung ist optional, Sie haben dieselben Möglichkeiten zum Formulieren wie bei der SELECT-Anweisung. Geben Sie keine Bedingung an, wirkt sich ein Update auf alle aufgezählten Spalten aus.

Wollen wir die Spalte User aller Datensätze auf einen neuen Wert setzen, schreiben wir:

```
UPDATE Administratoren SET Passwort='noch geheimer';
```

Wollen wir nur einen einzelnen Datensatz ändern bzw. nur einige, so müssen wir eine entsprechende WHERE-Klausel formulieren.

```
UPDATE Administratoren
   SET Passwort='ganz geheim'
   WHERE UserID='1';
```

Sie sollten für das Update eines einzelnen Datensatzes unbedingt dessen Primärschlüssel als Bedingung nehmen, wie in diesem Beispiel. Die Bedingung über die Spalte User zu formulieren, würde nur sicher sein, wenn wir beim Einfügen des Datensatzes darauf achten, in dieser Spalte keine Werte doppelt zu haben.

2.7.3.3) Löschen von Datensätzen

Was wir zum Abschluss des Abschnitts über das UPDATE gesagt haben, gilt auch für die DELETE-Anweisung. Benutzen Sie zum Löschen mit DELETE immer eine WHERE-Klausel mit Bezug auf den Primärschlüssel, außer Sie wollen wirklich alle Datensätze einer Tabelle löschen.

Die Syntax zum Löschen eines oder mehrerer Datensätze lautet:

```
DELETE FROM Relation
   [WHERE <Bedingung>]
```

Auch hier ist die WHERE-Klausel optional, wird keine angegeben, werden alle Datensätze gelöscht - die Tabelle ist danach leer.

Um eine Eintrag aus der Administrator-Tabelle zu löschen, schreiben wir:

```
DELETE FROM Administratoren WHERE UserID=1;
```

DML-Übungen

Aufgabe 1

- a) Fügen Sie zur Relation Kunde einige Datensätze ein.(Es soll dabei einen Kunden mit einer ID=1 geben und auch einige, die einen negativen Kontostand haben)
- b) Erstellen Sie eine neue Relation Kunde Mahnung mit den gleichen Attributen wie die Relation Kunde. Fügen Sie anschließend jene Tupel der Relation Kunde in die Relation Kunde Mahnung ein, die einen negativen Kontostand besitzen.
- c) Der Weinhändler möchte jene Kunden, die ihm kein Geld schulden (Kontostand≥0), belohnen und schreibt ihnen einen Betrag von 10 Euro auf ihrem Kontostand gut.
- d) Der Kunde mit der KundenID=1 hat auf die Mahnung des Weinhändlers bezahlt, hat jedoch versichert, dass er nie wieder Kunde bei ihm sein wird. Löschen Sie diesen Kunden aus der Relation Kunde.
- e) Löschen Sie alle Tupel der Relation Kunde Mahnung und danach die gesamte Relation.

Aufgabe 2 (am PC)

Gegeben sind folgende Relationen:

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

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

Relation Installation			
RNr	PNr	Platz	Code
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

Relation Benutzung		
ANr	PNr	Stund
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

Relation Assistent			
ANr	AName	StudAss	Gehalt
A1	Novak	Brunner	3
A2	Dvorak	Vogt	1
A3	Husak	Vogt	1
A4	Pfeiffer	Brunner	2

Befüllen Sie die in 2.7.1) bereits angelegten Relationen in ihrer MySQL-Datenbank namens **„Filmverwaltung“** mithilfe von DML-Statements.

Aufgabe 3 (am PC)

Gegeben sind folgende Relationen:

Relation Winzer					
wnr	**name**	**strasse**	**plz**	**ort**	**telefon**
1	Lackner Tinnacher	Steinbach 8	4567	Gamlitz	1234567
2	Weingut Prager	Weissenkirchen 48	3610	Weissenkirchen	1234567

Relation Winzer					
wnr	**name**	**strasse**	**plz**	**ort**	**telefon**
3	Weingut Emmerich Knoll	Unterloiben 10	3601	Unterloiben	12344457
4	Weingut F.X. Pichler	Unterloiben 27	3601	Unterloiben	11122233
5	Weingut Spätlese	Weintalstrasse 23	1136	Wien	
6	Freie Weingarten Wachau	Kremstalstrasse 23	3600	Krems	2304002
7	Stiftskellerei				
8	Weingut Biegler	Wienerstrasse 88	3502	Gumpoldskirchen	54564565

Relation Wein							
nr	**bezeichnung**	**sorte**	**jahrgang**	**preis**	**anzahl**	**wnr**	**position**
1	Riesling Kellerberg	Riesling	1999	28.00	24	4	1
2	Loibenberg	Gr. Veltliner	2000	19.00	36	4	2
3	Ried Kreutles	Gr. Veltliner	2000	19.00	15	3	4
4	Riesling Smaragd	Riesling	2000	21.00	30	2	5
5	Grauburgunder	Grauburgunder	2003	16.00	72	1	6
6	Morillon	Chardonnay	2003	9.00	55	1	7
7	Riesling Federspiel	Riesling	2003	9.90	80	6	3
8	Chardonnay	Chardonnay	2003	9.00	16	8	8

Relation Keller			
knr	**reihe**	**regal**	**fach**
1	1	1	1
2	1	1	2
3	1	1	3
4	1	2	1
5	1	2	2
6	2	1	1
7	2	1	2
8	2	2	1

Relation Protokoll				
pnr	***nr***	**pDatum**	**verwendung**	**anzahl**
1	1	2003-01-12	Geschenk an Herrn Berger	12
2	3	2003-07-10	Eigenbedarf	2
3	1	2003-07-23	Eigenbedarf	4
4	6	2003-08-14	Geschenk (Frau Kunz)	6
5	1	2003-08-27	Glasbruch	1
6	4	2003-11-03	Korkgeruch	1
7	6	2003-11-03	Eigenbedarf	3

Befüllen Sie die in 2.7.1) bereits angelegten Relationen in ihrer MySQL-Datenbank namens **„Winzerverwaltung“** mithilfe von DML-Statements.

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);


                echo "<table bgcolor=\"#dddddd\" border=\"0\" width=\"600\">
\n";
```

```
        while($row=mysqli_fetch_array($erg))
        {

            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:

Absenden

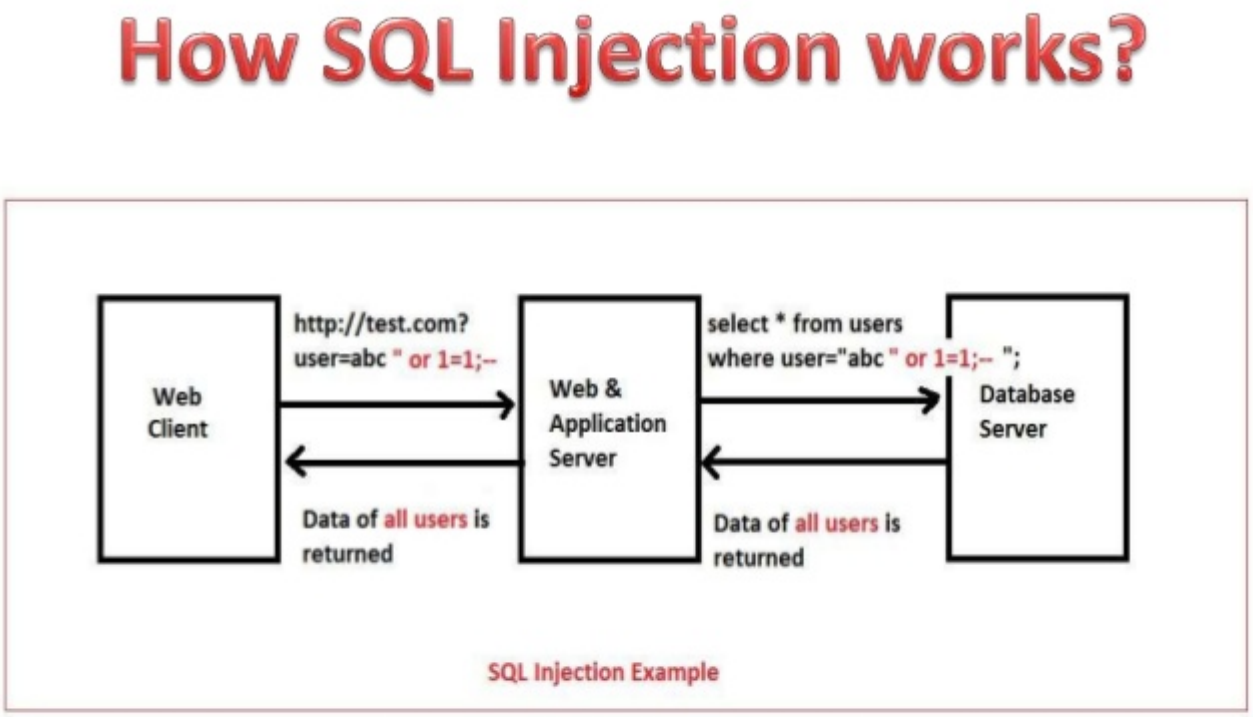
Alle Einträge anzeigen

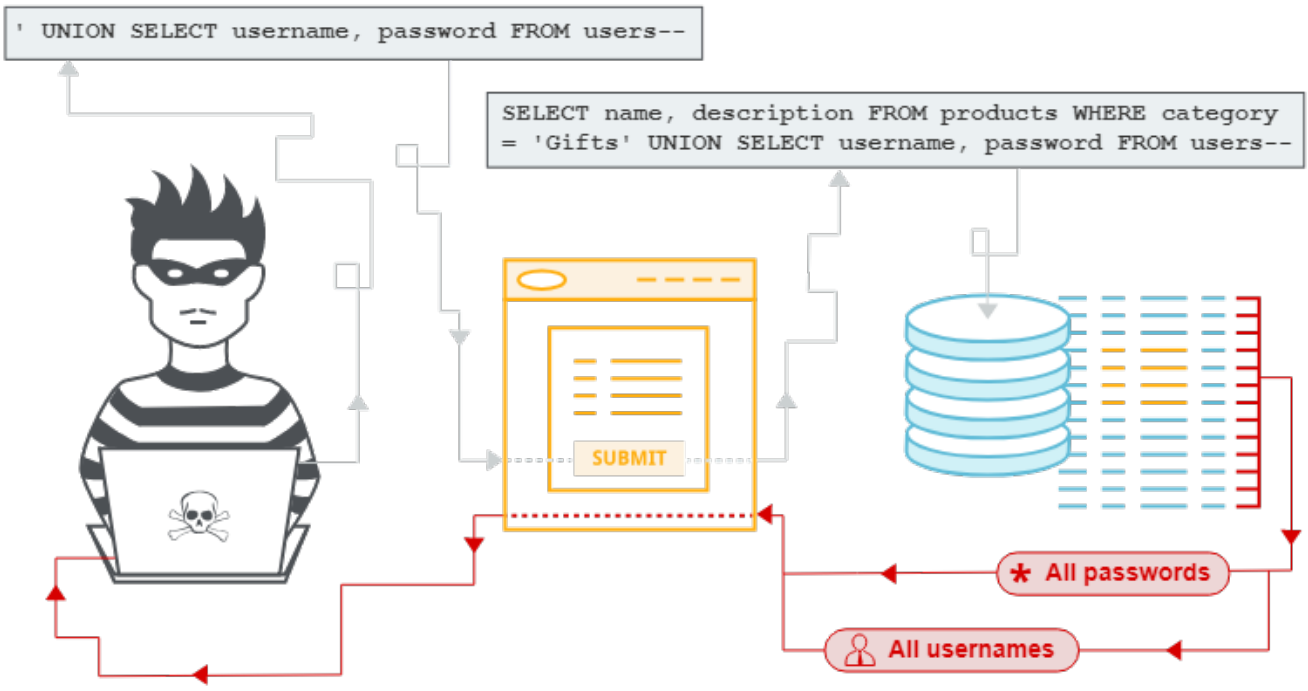
SQL Injection

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 bösartigen Code in die SQL-Statements mithilfe von Formulardaten in Webseiten





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: 105 OR 1=1

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.

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.

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

3) C#

Programmierung mit GUI in C#

GUI

Grafische Benutzeroberfläche oder auch grafische Benutzerschnittstelle (Abk. GUI von englisch graphical user interface) bezeichnet eine Form von Benutzerschnittstelle eines Computers. Sie hat die Aufgabe, Anwendungssoftware auf einem Rechner mittels grafischer Symbole, Steuerelemente oder auch Widgets genannt, bedienbar zu machen. Dies geschieht bei Computern meistens mittels einer Maus als Steuergerät, mit der die grafischen Elemente bedient oder ausgewählt werden, bei Smartphones, Tablets und Kiosksystemen in der Regel durch Berührung eines Sensorbildschirms.

Programmen mit GUI vs. Konsolenanwendungen

Vorteile Konsolenanwendung

- **schneller, schlanker** und benötigen **weniger Arbeitsspeicher**
- **übersichtlicher** und **leichter zu warten**
- **leichter portierbar** – die **GUI-Programmierung funktioniert auf jedem Betriebssystem unterschiedlich**. Bei der Trennung von GUI und Hauptprogramm kann das Hauptprogramm oft ohne tiefgreifende Änderungen für ein anderes System kompiliert werden. So muss jeweils nur eine kleine GUI für jede Plattform von Grund auf neu erstellt werden. Diese kann dann auch besser auf die Eigenheiten der jeweiligen Plattform abgestimmt werden.
- **vielseitiger verwendbar** – die Funktion von Konsolenprogramms kann oft per **Verkettung (Piping) mit anderen Programmen** – auch mit GUI-Programmen – kombiniert werden; Ein- und Ausgaben von Kommandozeilenprogrammen kann durch Ein-/Ausgabeumleitung aus und in Dateien geleitet werden (Batchbetrieb).
- **leicht erweiterbar** ohne **Zusatzkomplexität**

Vorteile GUI-Programme

- **Einfachere, intuitivere Benutzung** des Programms durch sichtbare Objekte
- **Learning-by-doing** Konzept - unerfahrene Benutzer können die Funktionen des Programms erahnen

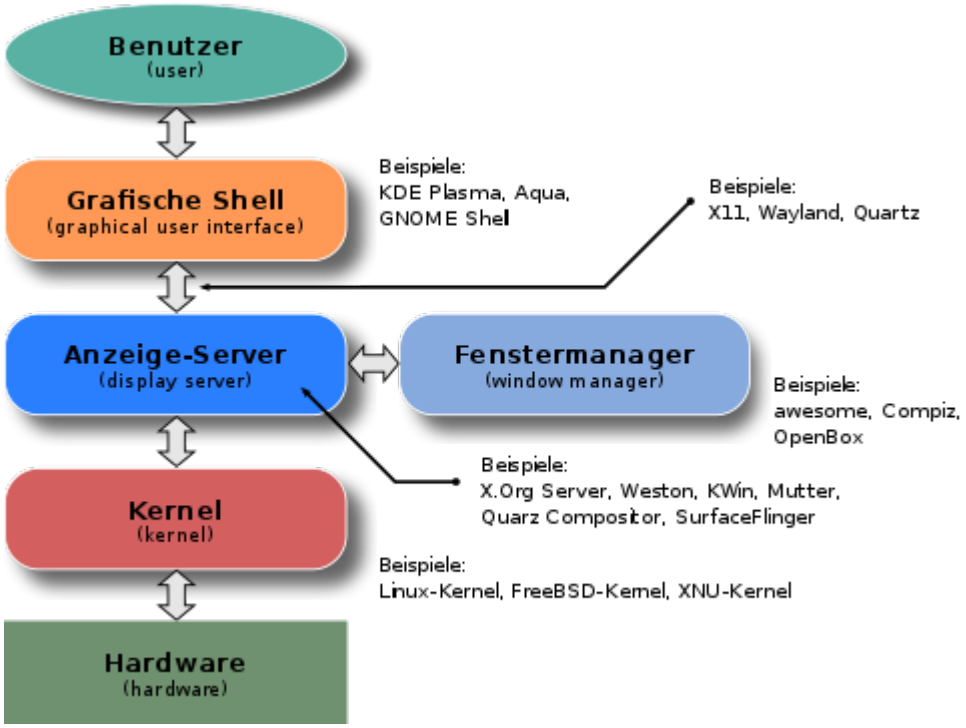
Fenstersystem

Ein Fenstersystem (englisch windowing system) ist der Unterbau einer grafischen Benutzeroberfläche (GUI), deren Hauptaufgabe die Verwaltung von Programmfenstern ist. Im Normalfall ist es Teil einer größeren Desktop-Umgebung.

Aus der Sicht eines Programmierers implementiert das Fenstersystem die grafischen Basisfunktionen, wie das Darstellen von Schriftarten, Zeichnen von Linien, Kurven und Pixelgrafiken, und das Abstrahieren der Grafikhardware (Grafikkarte).

Das Fenstersystem gestattet es dem Anwender mit mehreren Programmen gleichzeitig zu arbeiten, indem jedes Programm „in“ einem oder mehreren eigenen Bereichen des Bildschirms, den Fenstern, ausgeführt wird, die üblicherweise rechteckig sind, mit dem Zeigegerät (Maus) frei bewegt werden können und einander überlappen dürfen.

Einige Fenstersysteme, wie das X Window System in Unix-artigen Umgebungen, haben erweiterte Fähigkeiten wie Netzwerkstransparenz, die es dem Anwender gestatten die grafische Oberfläche einer Anwendung auf einem anderen Computer darzustellen. Das X-Window-System implementiert auch kein festes Aussehen der Umgebung, wodurch die Fenstermanager, GUI-Toolkits und Desktop-Umgebungen volle Freiheit bei der optischen Gestaltung und Handhabung haben.



GUI in C++

C + + bringt von Haus aus kein GUI mit, da es als hardwarenahe plattformunabhängige Sprache erstellt wurde und GUIs stark vom verwendeten Betriebssystem abhängen. Dieser „Mangel“ könnte einige Neueinsteiger vielleicht auf den ersten Blick abschrecken, da sie sich einen Einstieg in C + + oder gar in die gesamte Programmierung mit vielen Buttons, Textfeldern, Statusanzeigen und Menüeintragen erhofft haben.

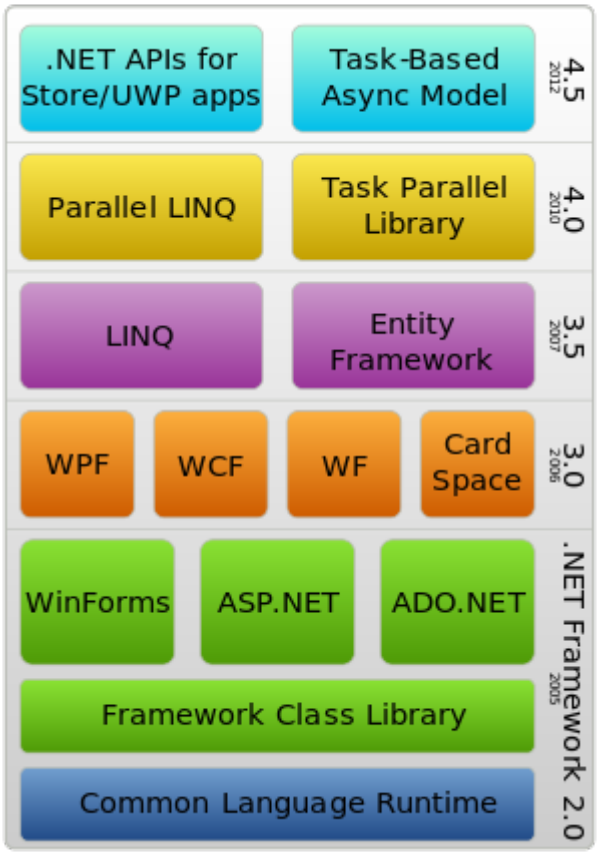
Sobald man mit C + + dann doch ein GUI programmieren will, sollte auf externe Bibliotheken zurückgegriffen werden. Alle GUIs erfordern ein Verständnis der C + + Grundlagen. Nachfolgend sind einige Bibliotheken zur GUI-Programmierung aufgelistet. Viele der Bibliotheken lassen sich auch mit anderen Programmiersprachen einsetzen.

.NET-Framework

.NET Framework (auch mit dotNetFx oder nur mit NetFx abgekürzt) ist ein **Teil von Microsofts Software-Plattform .NET.** Es umfasst eine Sammlung zahlreicher **Werkzeuge und Schnittstellen**, die die **Entwicklung von Anwendungsprogrammen unterstützen und**

vereinfachen sollen, sowie eine Laufzeitumgebung für Programme, die mithilfe dieses Frameworks geschrieben wurden. Da diese Programme hierbei nicht herkömmlich direkt in Instruktionen für einen Prozessor übersetzt werden, braucht ein System für die Ausführung der Programme zusätzliche Software (welche die Laufzeitumgebung des Frameworks bildet). Programme, die mit dem .NET Framework geschrieben wurden, werden in der Regel so ausgeliefert, dass benötigte Komponenten des Frameworks automatisch mitinstalliert werden. Der Benutzer muss sich also nur im Ausnahmefall um das Vorhandensein der Laufzeitumgebung kümmern. Außerdem ist das Framework seit vielen Jahren bereits Bestandteil von Windows Installationen, wenn auch nicht immer in der aktuellsten Version und daher ist diese für manche Zwecke nicht ausreichend.

Ziele des Frameworks, wie auch der Plattform .NET, sind unter anderem **Kompatibilität zwischen verschiedenen Plattformen**, **Zusammenführung bisher inkompatibler Microsoft Technologien** und **Verbesserung veralteter Standards**. Aus Anwendersicht birgt das Framework zwar keine direkten Vorteile, da sich seine Funktionen und Anwendungsbereiche hauptsächlich auf Softwareentwicklung, -betrieb und -wartung beschränken, dennoch ist es eine notwendige Voraussetzung für den Betrieb zahlreicher Anwendungsprogramme.



Qt

Qt ist eine leistungsstarke plattformübergreifende Klassenbibliothek mit Meta-Object Compiler (gennant: moc), die von der norwegischen Firma Trolltech entwickelt wurde. Trolltech wurde 2008 von Nokia übernommen, welches jedoch wirtschaftliche Schwierigkeiten hatte und in der Folge Qt 2011 an das finnische Softwareunternehmen Digia verkaufte.

Die Entwicklung wird inzwischen vom Qt Project vorangetrieben, so dass jeder sich daran beteiligen kann. Die Klassenbibliothek ist unter der GNU General Public License (GPL) und der GNU Lesser

General Public License (LGPL) lizenziert. Es gibt außerdem eine proprietäre Lizenz von Digia, die allerdings lediglich zusätzlichen technischen Support beinhaltet.

Speziell aufgrund der Veränderungen, die mit Qt 5 und C + + Einzug in das Framework halten, ist Qt eine der mächtigsten Bibliotheken für C + +. Ein weiterer Vorteil von Qt ist die vollständig freie Entwicklungsumgebung Qt Creator, die insbesondere Anfängern das Programmieren leichter macht.

VCL - Visual Component Library

Die **Visual Component Library (VCL)** ist ein GUI-Toolkit für Windows-Anwendungen. Sie wurde von **Embarcadero (Rad Studio)**, vormals Borland, Inprise und CodeGear, erstellt. Die VCL kann in den Programmiersprachen Borland Delphi, C + +, C, C# verwendet werden. Sie wird von den meisten Borland-Entwicklungsumgebungen als Komponentensammlung benutzt.

MFC - Microsoft Foundation Classes

Die Microsoft Foundation Classes (MFC) sind eine Sammlung objektorientierter Klassenbibliotheken (GUI-Toolkit), die von Microsoft für die Programmierung von Anwendungen mit grafischen Benutzeroberflächen für Windows mit C + + entwickelt wurden.

WF - Windows Forms

Windows Forms ist ein **GUI-Toolkit des Microsoft .NET Frameworks**. Es ermöglicht die Erstellung grafischer Benutzeroberflächen (GUIs) für Windows in C#. Im Vergleich zu Microsoft Foundation Classes (MFC), die auf der Programmiersprache C + + basiert, ist der Einstieg in die Programmierung mit Windows Forms einfacher. Das Framework basiert nicht auf dem Paradigma Model View Controller (MVC).

Mit .NET Framework 3.0 wurde von Microsoft eine Alternative zu Windows Forms bereitgestellt, die Windows Presentation Foundation, welche eine stärkere Trennung der grafischen Oberfläche vom Programmcode und – unter Zuhilfenahme von XAML, einer XML-basierenden Sprache – ein dynamischeres Layout ermöglicht.

GUI in C#

WPF - Windows Presentation Foundation

Windows Presentation Foundation (kurz WPF), auch bekannt unter dem Codenamen Avalon, ist ein Grafik-Framework und Fenstersystem des .NET Frameworks von Microsoft. Es wird seit Windows Vista mit Windows ausgeliefert und lässt sich auf Windows XP (bis zur Version 4.0) und Windows Server 2003 nachinstallieren. Für das neue Framework .NET Core soll WPF ab der Version 3.0 (für 2019 erwartet) unter Windows zur Verfügung stehen.[2]

WPF ist eine 2006 neu eingeführte Klassenbibliothek, die zur Gestaltung von grafischen Benutzeroberflächen und zur Integration von Multimedia-Komponenten und Animationen dient. Sie

vereint DirectX, Windows Forms, Adobe Flash, HTML und CSS.

WPF stellt ein umfangreiches Modell für den Programmierer bereit. Dabei werden die Präsentation und die Geschäftslogik getrennt, dies wird vor allem durch die Auszeichnungssprache XAML (basierend auf XML) unterstützt. XAML beschreibt Oberflächen-Hierarchien deklarativ als XML-Code. WPF-Anwendungen können sowohl Desktop- als auch Web-Anwendungen sein und benutzen, wenn möglich, Hardwarebeschleunigung. Das Framework versucht, die verschiedenen Bereiche, die für die Präsentation wichtig sind (Benutzerschnittstelle, Zeichnen und Grafiken, Audio und Video, Dokumente, Typographie), zu vereinen.

C#

1) Taschenrechner

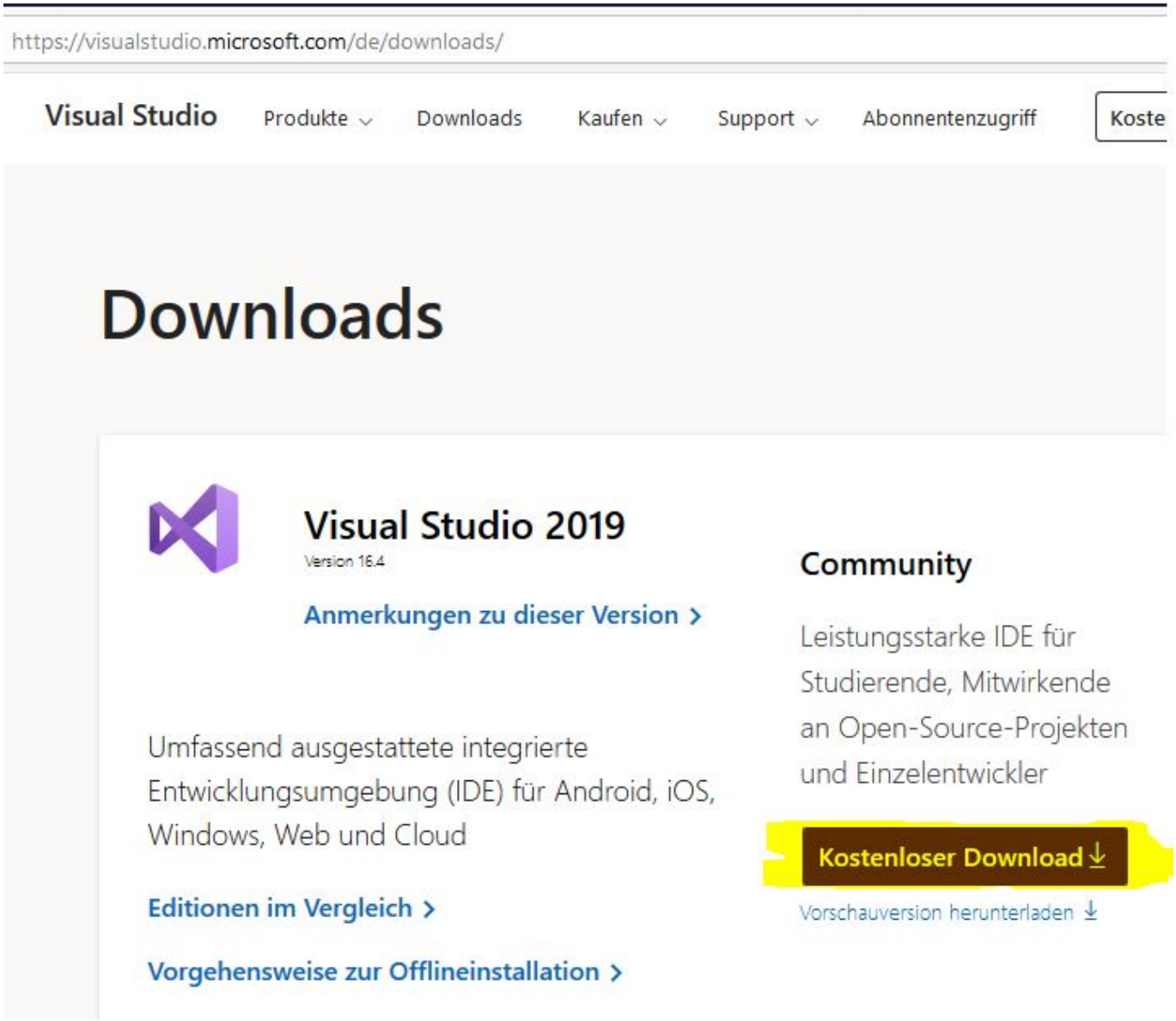
2) Zahlensysteme

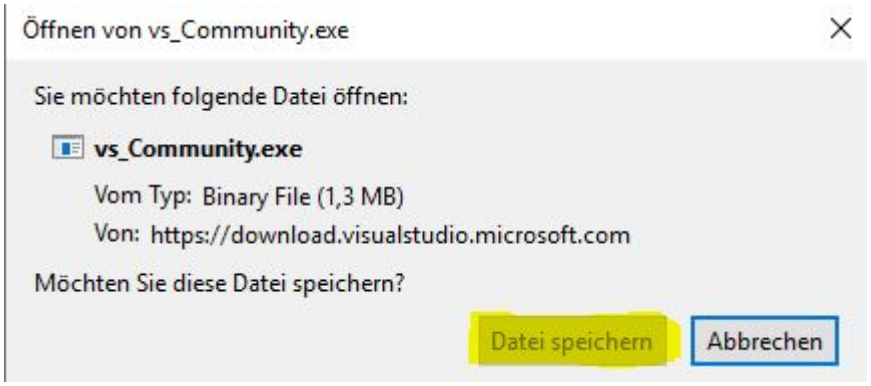
Visual Studio

Download von Visual Studio Community

- https://visualstudio.microsoft.com/de/thank-you-downloading-visual-studio/?sku=Community&rel=16

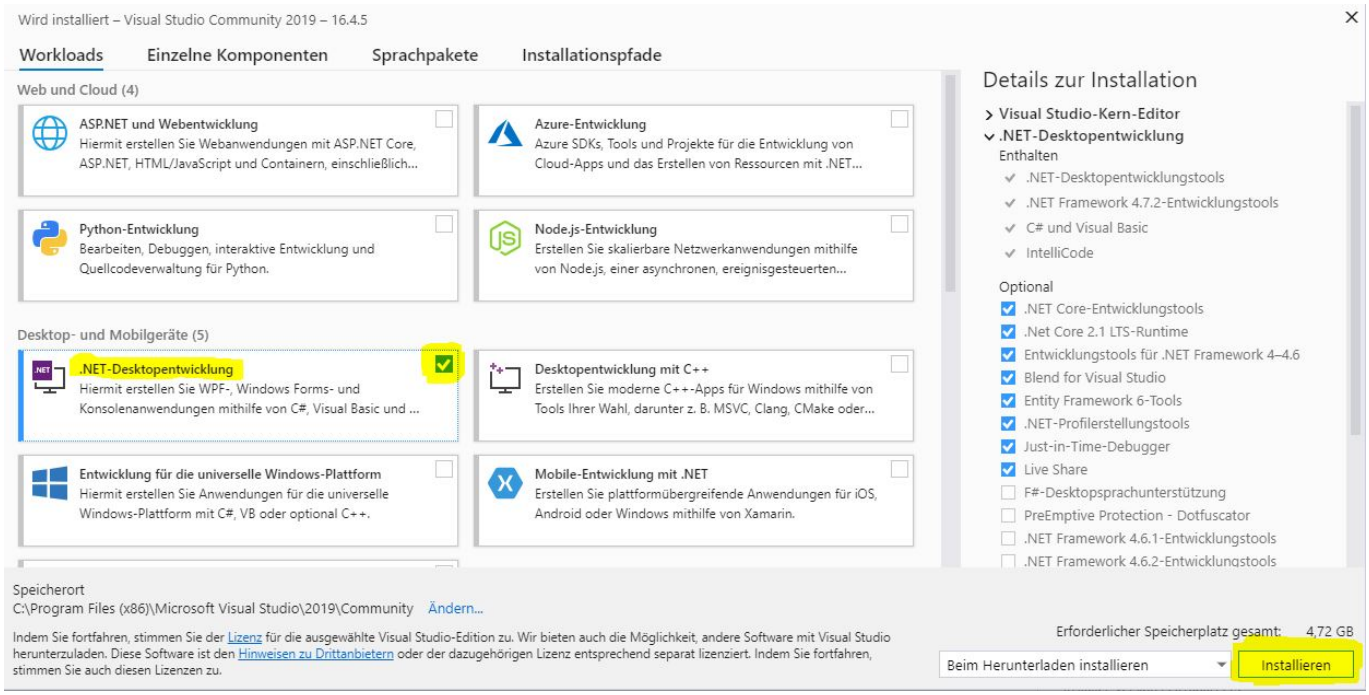
Installation von Visual Studio 2019

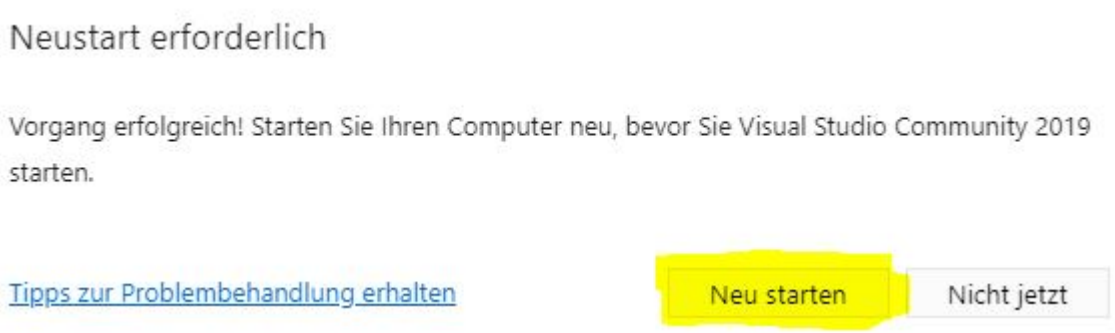


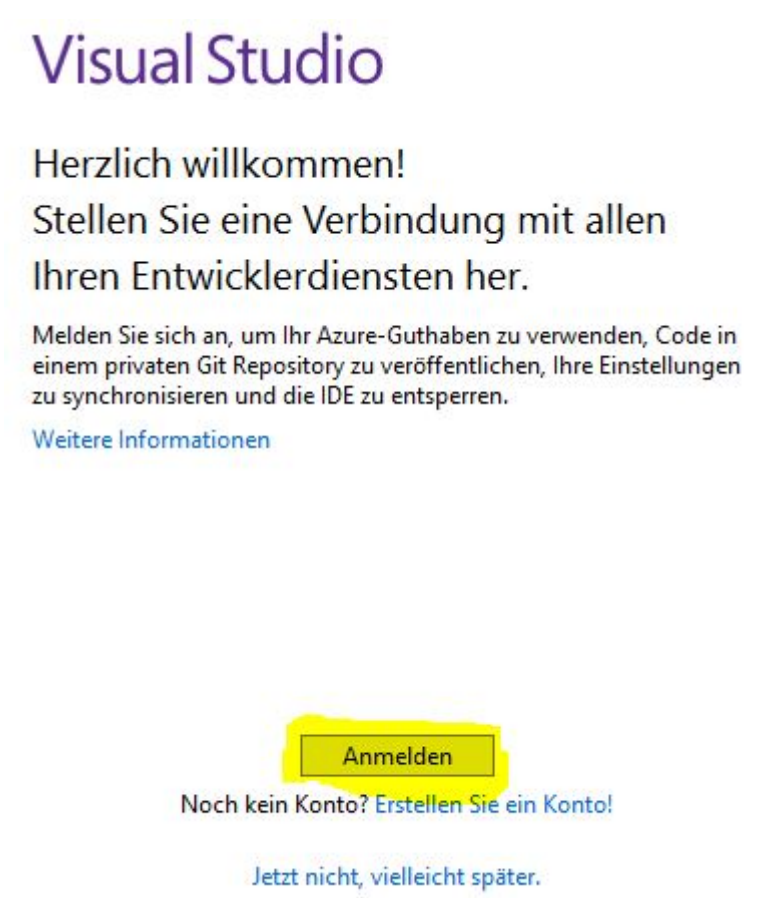

Öffnen von vs_Community.exe
Sie möchten folgende Datei öffnen:
vs_Community.exe
Vom Typ: Binary File (1,3 MB)
Von: https://download.visualstudio.microsoft.com
Möchten Sie diese Datei speichern?
Datei speichern
Abbrechen

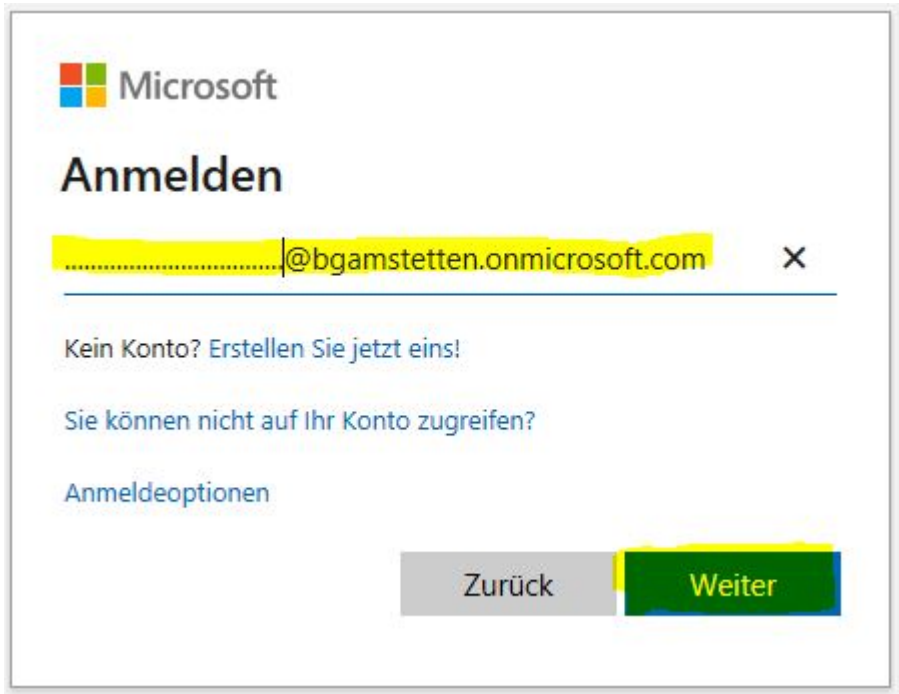

Benutzerkontensteuerung
Möchten Sie zulassen, dass durch diese App Änderungen an Ihrem Gerät vorgenommen werden?
Visual Studio Installer
Verifizierter Herausgeber: Microsoft Corporation
Dateiursprung: Festplatte auf diesem Computer
Weitere Details anzeigen
Ja
Nein


Visual Studio Installer
Bevor Sie beginnen, müssen wir einige Dinge so einrichten, damit Sie Ihre Installation konfigurieren können.
Weitere Informationen zum Datenschutz finden Sie in der Microsoft-Datenschutzerklärung.
Indem Sie fortfahren, erklären Sie sich mit den Microsoft-Software-Lizenzbedingungen einverstanden.
Fortfahren

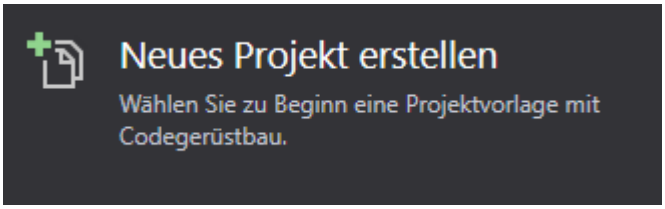

Wird installiert – Visual Studio Community 2019 – 16.4.5
Workloads
Einzelne Komponenten
Sprachpakete
Installationspfade
Web und Cloud (4)
ASP.NET und Webentwicklung
Hiermit erstellen Sie Webanwendungen mit ASP.NET Core, ASP.NET, HTML/JavaScript und Containern, einschließlich...
Azure-Entwicklung
Azure SDKs, Tools und Projekte für die Entwicklung von Cloud-Apps und das Erstellen von Ressourcen mit .NET...
Python-Entwicklung
Bearbeiten, Debuggen, interaktive Entwicklung und Quellcodeverwaltung für Python.
Node.js-Entwicklung
Erstellen Sie skalierbare Netzwerkanwendungen mithilfe von Node.js, einer asynchronen, ereignisgesteuerten...
Desktop- und Mobilgeräte (5)
.NET-Desktopentwicklung
Hiermit erstellen Sie WPF-, Windows Forms- und Konsolenanwendungen mithilfe von C#, Visual Basic und ...
Desktopentwicklung mit C++
Erstellen Sie moderne C++-Apps für Windows mithilfe von Tools Ihrer Wahl, darunter z. B. MSVC, Clang, CMake oder...
Entwicklung für die universelle Windows-Plattform
Hiermit erstellen Sie Anwendungen für die universelle Windows-Plattform mit C#, VB oder optional C++.
Mobile-Entwicklung mit .NET
Erstellen Sie plattformübergreifende Anwendungen für iOS, Android oder Windows mithilfe von Xamarin.
Details zur Installation
Visual Studio-Kern-Editor
.NET-Desktopentwicklung
Enthalten
.NET-Desktopentwicklungstools
.NET Framework 4.7.2-Entwicklungstools
C# und Visual Basic
IntelliCode
Optional
.NET Core-Entwicklungstools
.Net Core 2.1 LTS-Runtime
Entwicklungstools für .NET Framework 4–4.6
Blend for Visual Studio
Entity Framework 6-Tools
.NET-Profilerstellungstools
Just-in-Time-Debugger
Live Share
F#-Desktopsprachunterstützung
PreEmptive Protection - Dotfuscator
.NET Framework 4.6.1-Entwicklungstools
.NET Framework 4.6.2-Entwicklungstools
Speicherort
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community
Ändern...
Indem Sie fortfahren, stimmen Sie der Lizenz für die ausgewählte Visual Studio-Edition zu. Wir bieten auch die Möglichkeit, andere Software mit Visual Studio herunterzuladen. Diese Software ist den Hinweisen zu Drittanbietern oder der dazugehörigen Lizenz entsprechend separat lizenziert. Indem Sie fortfahren, stimmen Sie auch diesen Lizenzen zu.
Erforderlicher Speicherplatz gesamt: 4,72 GB
Beim Herunterladen installieren
Installieren

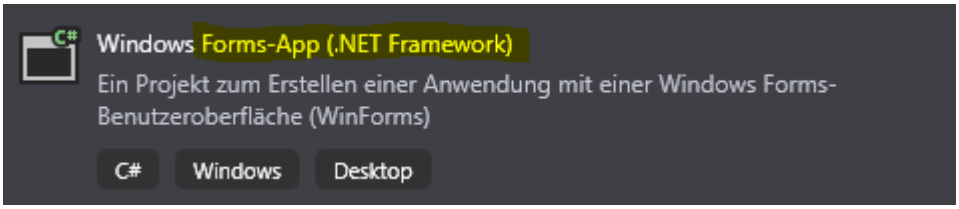

Neustart erforderlich
Vorgang erfolgreich! Starten Sie Ihren Computer neu, bevor Sie Visual Studio Community 2019 starten.
Tipps zur Problembehandlung erhalten
Neu starten
Nicht jetzt


Visual Studio
Herzlich willkommen!
Stellen Sie eine Verbindung mit allen Ihren Entwicklerdiensten her.
Melden Sie sich an, um Ihr Azure-Guthaben zu verwenden, Code in einem privaten Git Repository zu veröffentlichen, Ihre Einstellungen zu synchronisieren und die IDE zu entsperren.
Weitere Informationen
Anmelden
Noch kein Konto? Erstellen Sie ein Konto!
Jetzt nicht, vielleicht später.

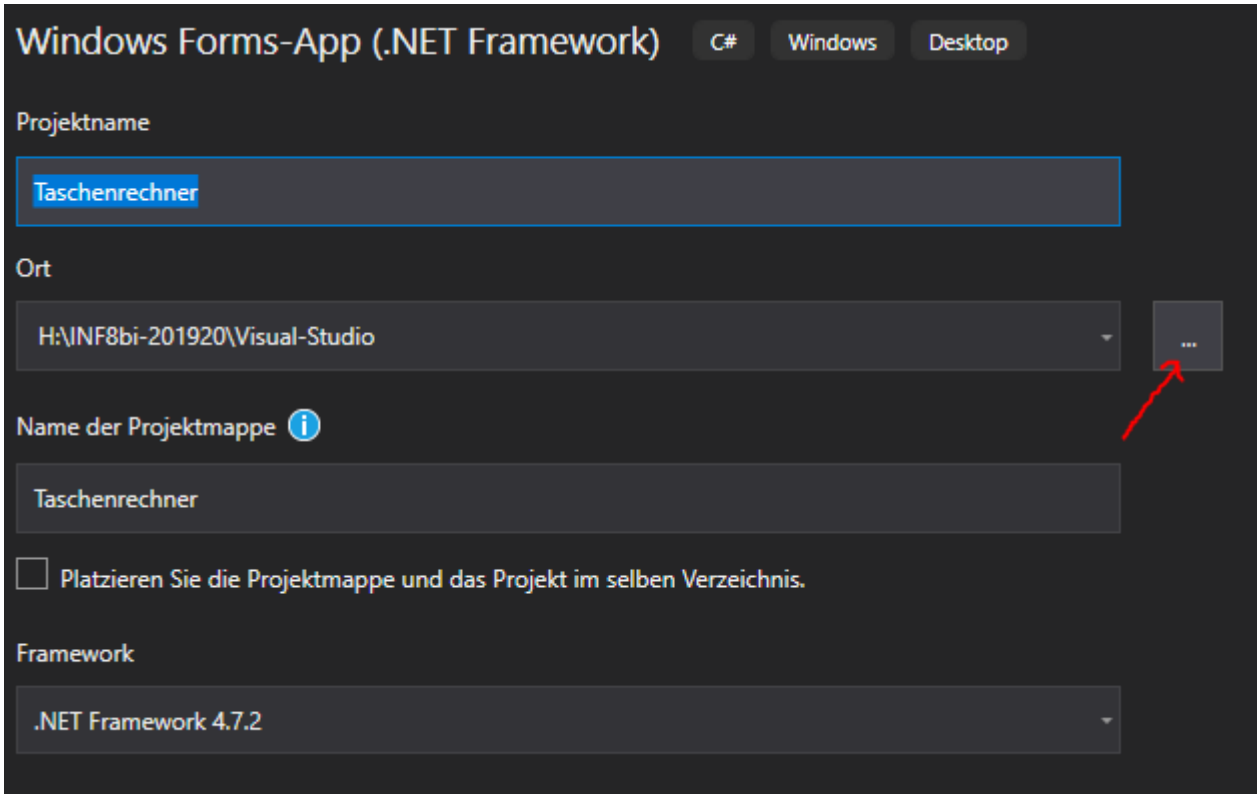

Microsoft
Anmelden
.....@bgamstetten.onmicrosoft.com
Kein Konto? Erstellen Sie jetzt eins!
Sie können nicht auf Ihr Konto zugreifen?
Anmeldeoptionen
Zurück
Weiter

Erstellen eines neuen Projekts

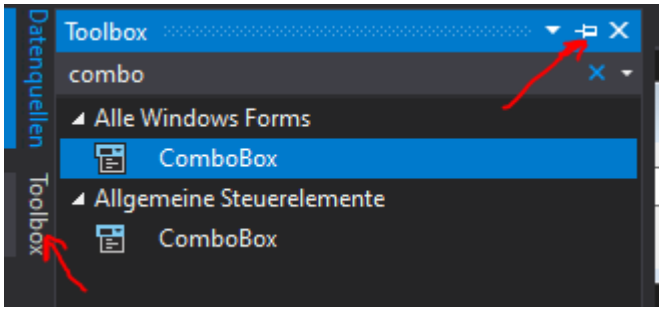




Titel und Speicherort wählen:



Toolbox öffnen und verankern:

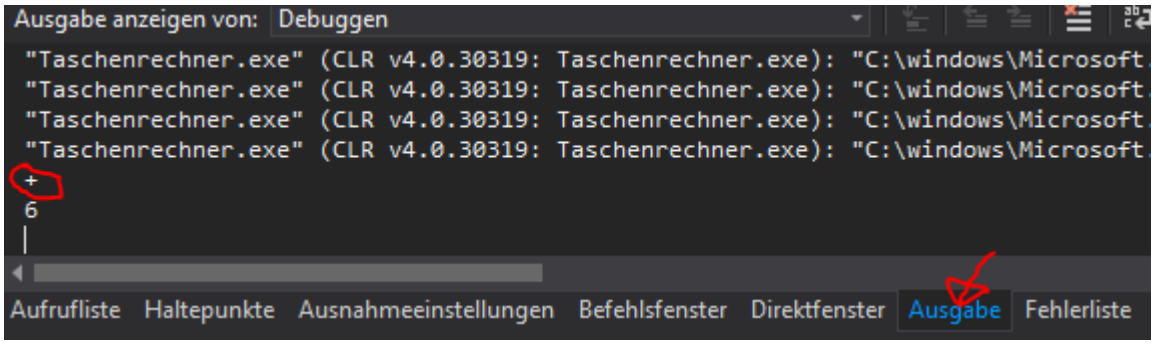


Mittels Umschalt+F7 kann in das Design gewechselt werden.

Code Beispiele

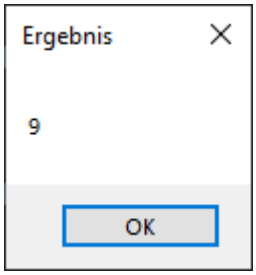
Console.WriteLine

```
Console.WriteLine(comboBox1.SelectedItem);
```



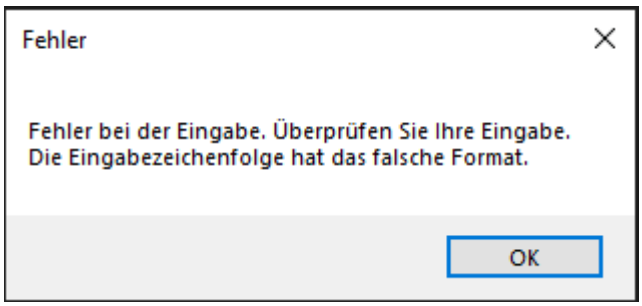
MessageBox

```
MessageBox.Show(erg.ToString(), "Ergebnis");
```



try - catch

```
try
{
    //Hier steht ein Code, wo es zu Fehlern kommen kann
}
catch(Exception ex)
{
    MessageBox.Show("Fehler bei der Eingabe. Überprüfen Sie Ihre Eingabe.\n"
+ ex.Message,"Fehler");
}
```



Sicherheitsabfragen - Eingabe von korrekten Zahlen in Textbox

Man möchte z.B. dass in einer Textbox nur Zahlen eingegeben werden können...

Man verwendet das Ereignis OnKeyPress

```
        private void TextBox1_KeyPress(object sender, KeyPressEventArgs e)
        {
            Console.WriteLine((int)e.KeyChar);
            if(char.IsDigit(e.KeyChar)||(int)e.KeyChar==8||e.KeyChar=='-
'||e.KeyChar==',')
            {
              //MessageBox.Show("Ist Ziffer");
              if(e.KeyChar=='-' && tb_op1.TextLength!=0)       //Verhindert
ein Minus, das nicht an 1. Stelle ist.
                {
                    e.Handled = true;
                }
              if(e.KeyChar==',' &&tb_op1.Text.Contains(','))   //Verhindert
zwei Kommas.
                {
                    e.Handled = true;
                }
            }
            else
            {
                //MessageBox.Show("Keine Ziffer");
                e.Handled = true;  //
            }
        }
```

Zweites Fenster - Neue Form

- In der Projektmappe ein neues Form2 hinzufügen.
- In Form1:

```
        private void button1_Click(object sender, EventArgs e)
        {
```

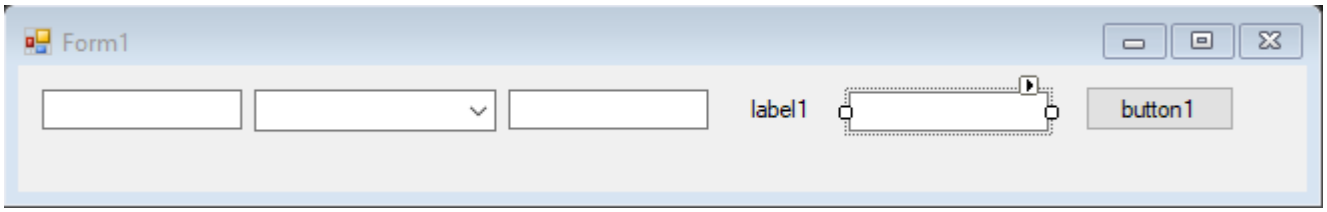
```
            Form2 frm = new Form2();
            frm.ShowDialog();   //Öffnet Fenster exklusiv
        }
```

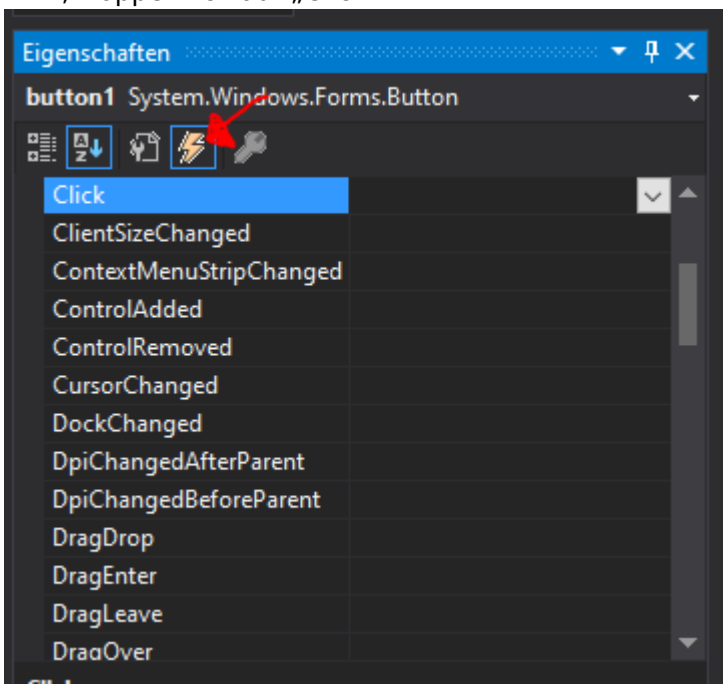
- Fenster in Form2 wieder schließen:

```
        private void button1_Click(object sender, EventArgs e)
        {
            ActiveForm.Close();
        }
```

3.2.1 Projekt Taschenrechner

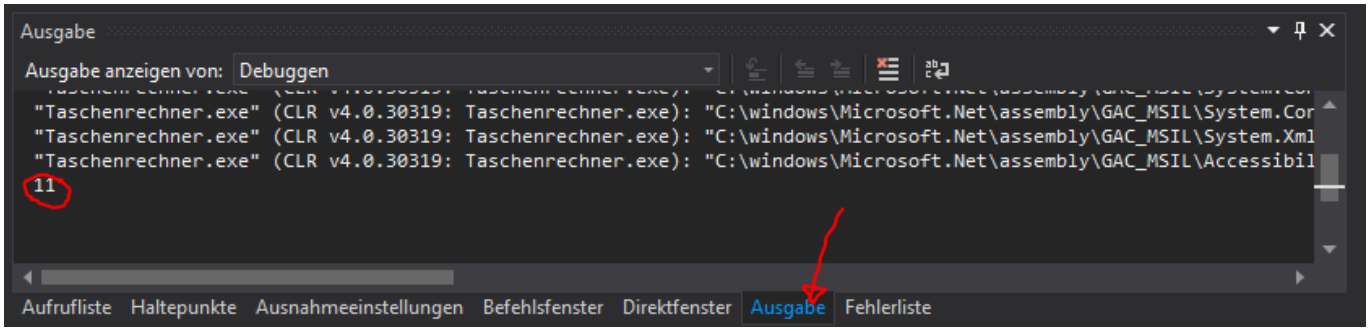
- 3 TextBoxen
- 1 ComboBox
- 1 Label
- 1 Button



- Änderung der Eigenschaften
  - Name, Size, Enable, Text, ...
  - ComboBox: DropDownStyle: DropDownList
- Ereignisse, z.B. Button
  - Blitz, Doppelklick auf „Click“
  - 


```
private void button1_Click(object sender, EventArgs e)
{
   float erg = float.Parse(tb_op1.Text) + float.Parse(tb_op2.Text);
   Console.WriteLine(erg);
}
```

Mit dem Befehl `Console.WriteLine(erg);` kann im Ausgabe-Fenster etwas ausgegeben werden.




```
  private void button1_Click(object sender, EventArgs e)
        {
            Console.WriteLine(comboBox_op.SelectedItem);
            float erg;
            try
            {
                switch (comboBox_op.SelectedItem.ToString()[0])    //wandelt
das Objekt in der ComboBox in einen String um
                {                                                  //und
greift mit [0] auf das erste Zeichen zu.
                    case '+':
                        erg = float.Parse(tb_op1.Text) +
float.Parse(tb_op2.Text);
                        break;
                    case '-':
                        erg = float.Parse(tb_op1.Text) -
float.Parse(tb_op2.Text);
                        break;
                    case '*':
                        erg = float.Parse(tb_op1.Text) *
float.Parse(tb_op2.Text);
                        break;
                    case '/':
                        erg = float.Parse(tb_op1.Text) /
float.Parse(tb_op2.Text);
                        break;
                    case '%':
                        erg = float.Parse(tb_op1.Text) %
float.Parse(tb_op2.Text);
                        break;
                    default: erg = 0; break;
                }
                Console.WriteLine(erg);
                tb_erg.Text = erg.ToString();
                //MessageBox.Show(erg.ToString(), "Ergebnis");
            }
            catch(Exception ex)
            {
                MessageBox.Show("Fehler bei der Eingabe. Überprüfen Sie Ihre
Eingabe.\n" + ex.Message,"Fehler");
            }
```

```
        }
```

ComboBox soll zu Beginn ein bestimmtes Objekt ausgewählt haben:

```
        public Form1()   //wird beim Erstellen der Form aufgerufen
        {
            InitializeComponent();
            comboBox_op.SelectedIndex = 0;    //Erstes Element der ComboBox
wird eingestellt
        }
```

Projekt Umrechner Zahlensysteme

Umrechnung Binär -> Dezimal Variante 2

```
        if(textBox_bin.TextLength>0)
            {
                int dez = 0;
                int ziffer = 0;
                int dezzahl = 0;
                int i = 0;
                string bin = textBox_bin.Text;
                int binzahl = int.Parse(textBox_bin.Text);
                while(binzahl>0)
                {
                    ziffer = binzahl % 10;
                    binzahl = binzahl / 10;
                    dezzahl = dezzahl + ziffer * (int)Math.Pow(2, i);
                    i++;
                }
                Console.Out.WriteLine(dezzahl);
            }
```