Informatik 8bi Schuljahr 2018/2019 als PDF exportieren

Informatik 8. Klasse - Schuljahr 2018/19

Lehrinhalte

- Lehrplaninhalte

Remote-Zugriff auf Schulserver

Kapitel

- 1) PHP
- 2) Datenbanken
- 3) C++ Datenstrukturen

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 – 05.12.2018 (2h)

von 1.0 bis 2.7.1.2

Stoff für die 2. Schularbeit in Informatik – 8BI – 13.03.2019 (3h)

ab Kapitel 2.7

Themengebiete Matura

Themenpool Informatik Matura 2018/19

Maturathemen

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.

1) PHP

1.1) Was ist PHP?

1.2) Grundlegende Sprachelemente

1.3) Variablen und Operatoren

1.4) Interaktive Webseiten

1.5) Anführungszeichen

1.6) Kontrollstrukturen

1.7) Schleifen

1.8) Felder

1.9) Formulare

1.10) Externe Dateien

1.11) Datum und Zeit

1.12) Funktionen

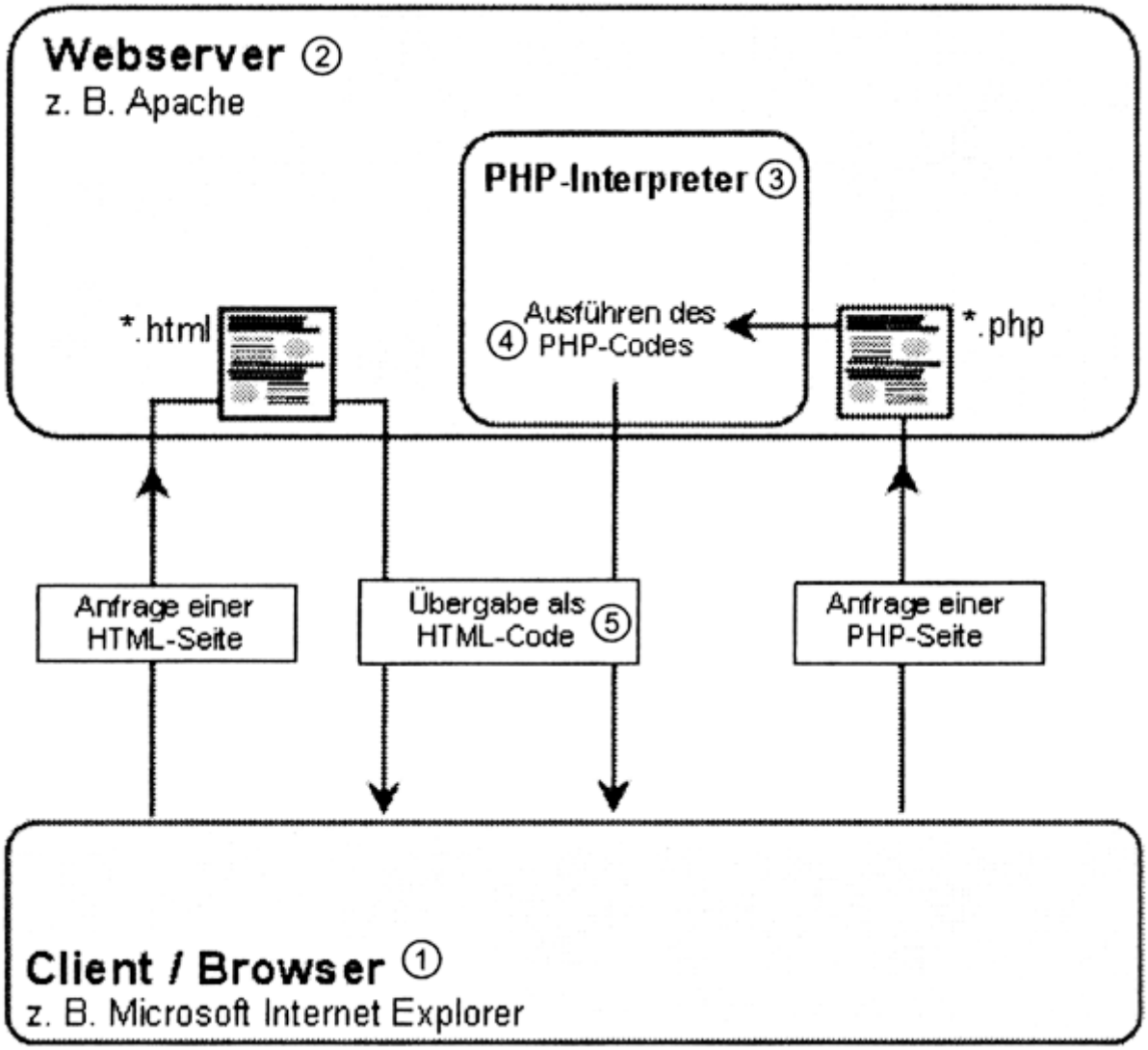
1.1.1) Was ist PHP?

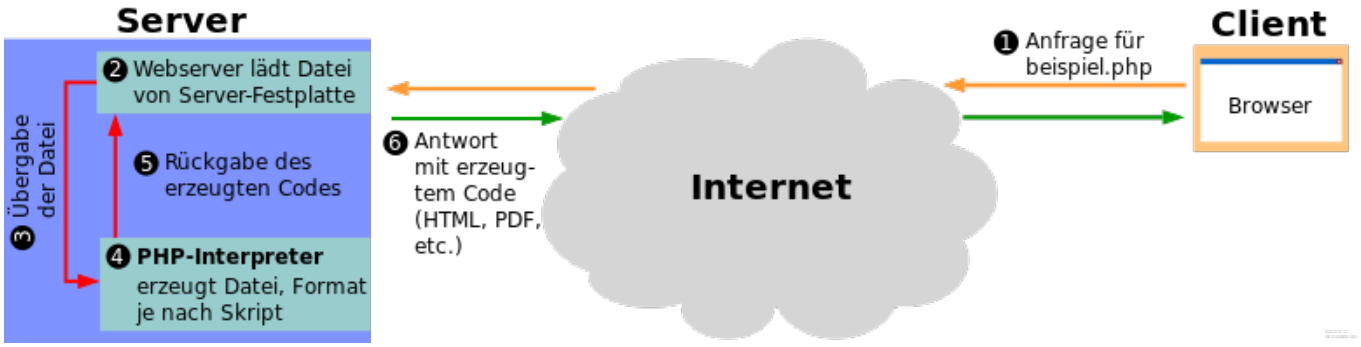
PHP (= PHP Hypertext Preprocessor) ... **Skriptsprache** für Internet, wird am **Webserver ausgeführt**.

PHP, die Kurzform von Personal HomePage oder auch PHP Hypertext Preprocessor, ist eine Skriptsprache, die speziell für den **Einsatz im Internet** entwickelt wurde. Die auszuführenden Anweisungen können **direkt in den HTML-Code integriert** werden.

Bei einer mit **HTML erstellten Webseite ist keine Dynamik** möglich. Mithilfe von **PHP** können auf einer Webseite **Interaktionen** eingebaut oder die Seite in irgendeiner Form, beispielsweise durch **Reaktionen auf Eingaben**, angepasst werden.

Öffnet der Betrachter in seinem Browser (1) eine solche Webseite über einen **PHP-fähigen Webserver** (2), werden die Anweisungen von PHP interpretiert (3), ausgeführt (4) und das Ergebnis als HTML-Code zurückgesendet (5). Ein klarer Vorteil dabei ist, dass der PHP-Quellcode dem Betrachter nicht zugänglich ist. Er sieht im Browser nur den zurückgelieferten HTML-Code.

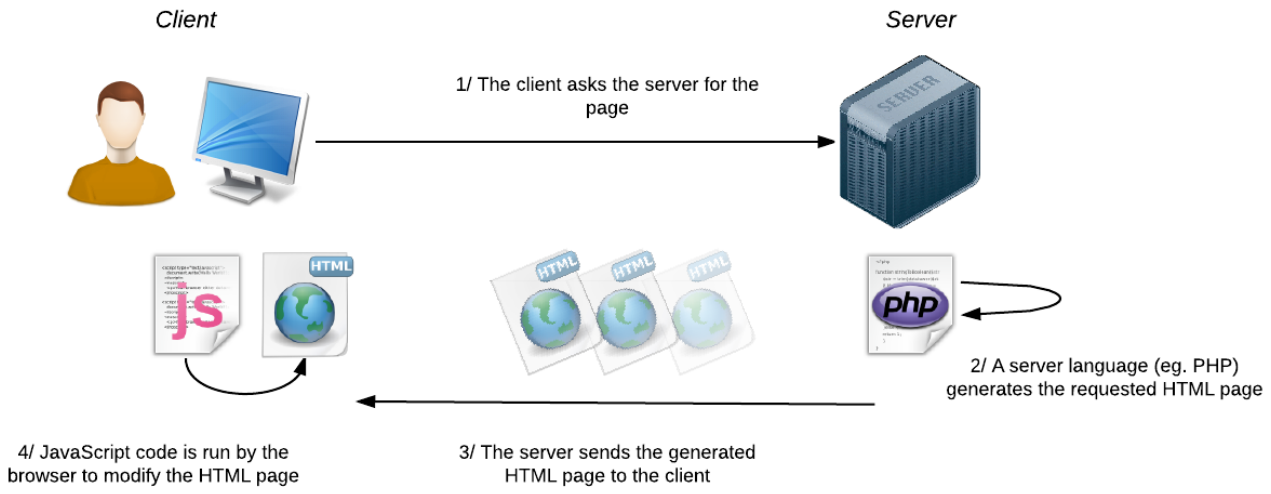


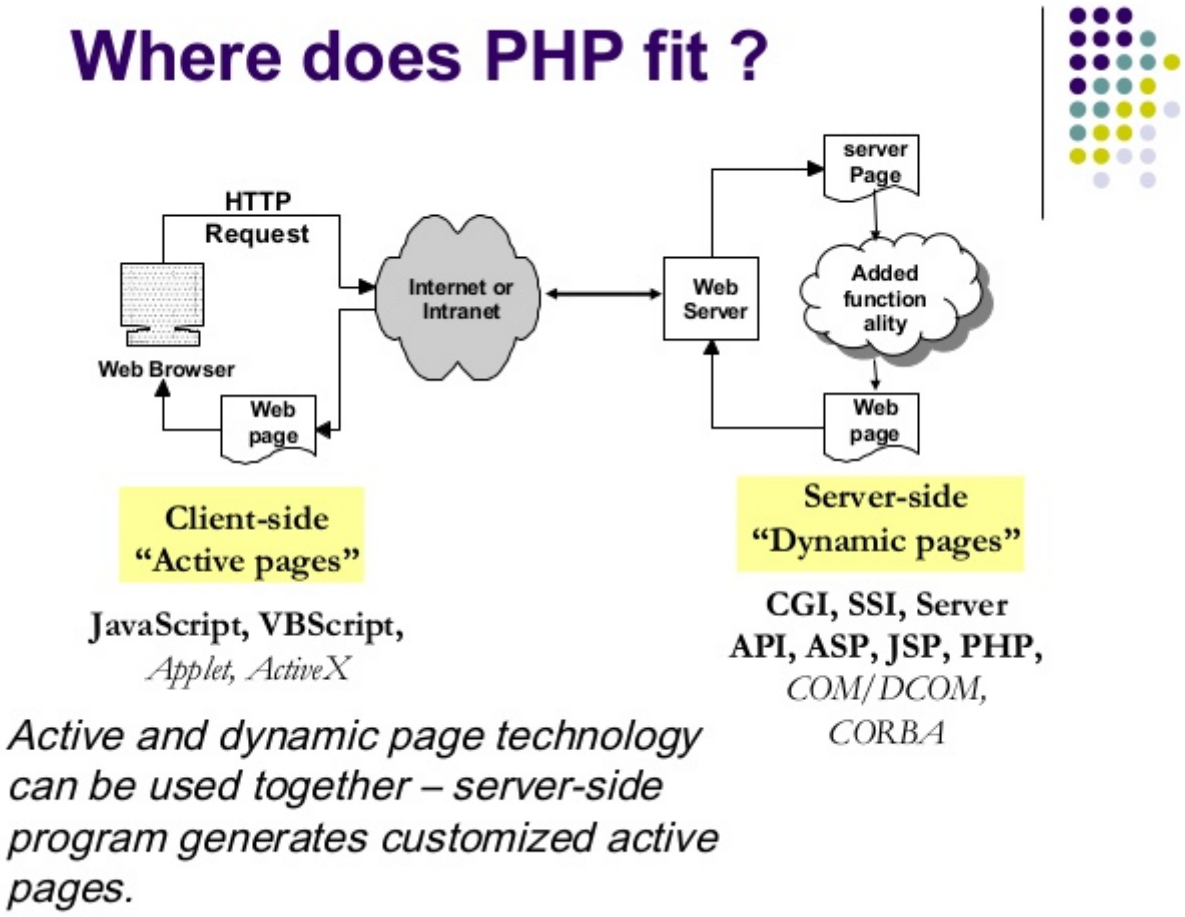


Um dynamische Webseiten zu erstellen und später auf Datenbanken zugreifen zu können, benötigt man neben PHP einen Webserver und einen Datenbankserver. Unter Windows kann man einen lokalen Webserver installieren. Als Entwicklungssoftware hat sich XAMPP sehr bewährt.

Unterschied zu Javascript

Im Gegensatz zu Javascript benötigt PHP einen Webserver, der eine Interaktion mit dem Benutzer ermöglicht und somit eine Webseite dynamisch erscheinen lässt.





Download von XAMPP

- http://www.apachefriends.org/de/xampp.html
- localhost greift auf den Ordner **…**/`htdocs` zu. In diesen Ordner müssen die html- bzw. php-Dateien gespeichert werden.

1.1.2) How to use PHP

PHP-Befehle einfügen

PHP-Code wird direkt in den HTML-Code eingebettet. PHP-Blöcke können vollständig im Dokumentkopf <head> oder im Dokumentrumpf <body> stehen. Weiterhin hat man die Möglichkeit, innerhalb einer Datei mehrmals zwischen PHP- und HTML-Code zu wechseln.

Dateinamenerweiterungen mit PHP-Code

Damit der Webserver erkennt, dass es sich um eine Datei mit einem PHP-Aufruf handelt, werden die Dokumente mit der Dateinamenerweiterung `.php` oder `.php5` gespeichert.

XML-Schreibweise

Der auszuführende PHP-Code wird zwischen dem öffnenden Tag <?php und dem schließenden Tag ?> geschrieben

```
<?php
  hier steht ein (oder mehrere) PHP-Befehl(e)
?>
```

Diese Schreibweise ist am gebräuchlichsten. Alternativ dazu kann man auch einen Skript-Tag verwenden. Diese Schreibweise entspricht im Wesentlichen der JavaScript- bzw. CSS-Angabe:

```
<script language="php">
  hier steht ein (oder mehrere) PHP-Befehl(e)
</script>
```

PHP-Anweisungen voneinander trennen

Jede PHP-Anweisung wird duch einen Strichpunkt beendet.

```
<?php
  PHP-Code;
  PHP-Code;
?>
```

Es empfiehlt sich, nach einer Anweisung in einer neuen Zeile weiterzuschreiben.

1.2.1) Kommentare einfügen

- Kommentare werden nicht vom Interpreter ausgeführt.
- Kommentare dienen zur näheren Beschreibung des Quellcodes

Syntax und Bedeutung der Kommentare

```
<html>
<body>
<?php
   echo "Für den Anfang: "; //einzeiliger Kommentar am Ende eines PHP-Codes
   // einzeiliger KOmmentar

   /*
     mehrzeilige Kommentare sind ebenfalls möglich,
     um den Quellcode ausführlicher zu beschreiben
   */
   echo "Hallo Welt!";
?>
</body>
</html>
```

1.2.2) Daten im Browser ausgeben

Befehl ''echo''

Da PHP eine eingebettete Skriptsprache ist, die vom Server ausgeführt wird, bleibt der Quellcode im Gegensatz zu HTML für den Client unsichtbar. Zeichenketten und Daten werden über den Befehl `echo` auf dem Bildschirm ausgegeben.

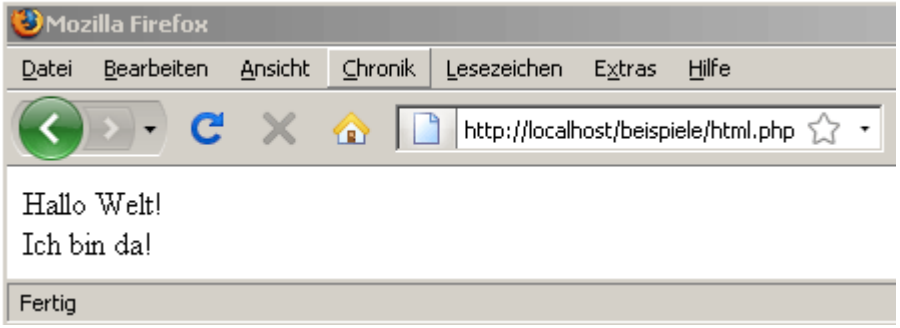
```
  echo "Hallo Welt";
```

Umgang mit HTML-Syntax bei der Ausgabe

Um Zeichenketten anzuzeigen, werden diese direkt an den Browser weitergegeben. Daher kann man innerhalb des Befehls `echo` auch jeden HTML-Tag (z.B. `<br>`) angeben. Dieser wird dann vom Browser als normaler HTML-Befehl interpretiert.

```
<html>
<body>
<?php
  echo "Hallo Welt! <br>"; //Der HTML-Tag <br> wird vom Browser korrekt interpretiert.
  echo "Ich bin da!";
?>
</body>
</html>
```

Screenshot Browser



Im Screenshot wird der HTML-Tag
 richtig interpretiert. Der Browser fügt eine Leerzeile ein. Der Kommentar wird natürlich nicht angezeigt!

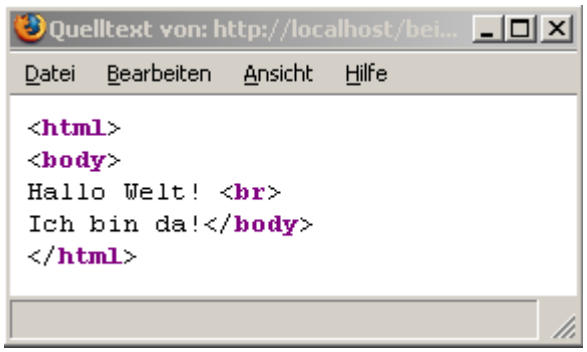
Quelltext HTML

Wie man sieht, wird der PHP-Kommentar auch im HTML-Quelltext nicht angezeigt. Weiters erscheint der übersetzte HTML-Code in einer Zeile.

Der HTML-Quellcode sollte ebenfalls wie der PHP-Quellcode übersichtlich erscheinen. Mittels \n kann man in PHP erwirken, dass im HTML-Quelltext eine Leerzeilen eingefügt wird. \n hat keinen Einfluss auf das Erscheinungsbild im Browser.

```
<html>
<body>
<?php
  echo "Hallo Welt! <br>\n"; // \n bewirkt eine Leerzeile im HTML-Quellcode!
  echo "Ich bin da!";
?>
</body>
</html>
```

Das Ergebnis im HTML-Quelltext sieht wie folgt aus:



Sonderzeichen ausgeben

Wenn man Hochkommas oder einen Backslash ausgeben möchte, setzt man vor den Zeichen einen Backslash:

```
<?php
   echo "<p>Der \"Erfinder\" von PHP war Rasmus Lerdorf. </p>";
   echo "Das Programm wird installiert im Verzeichnis C:\\Programme";
?>
```

Zeilenumbruch im Quelltext

Möchte man im Quelltext einen Zeilenumbruch einfügen, verwendet man \n

```
<?php
   echo "Das ist ein Text<br>\n";
   echo "Das ist noch ein Text";
?>
```

1.2.3) Aufgabe 1

Schreibe ein einfaches PHP-Skript, z.B. eine einfache Webseite mit Informationen über dich (Name, Wohnort, Hobbies, ...)

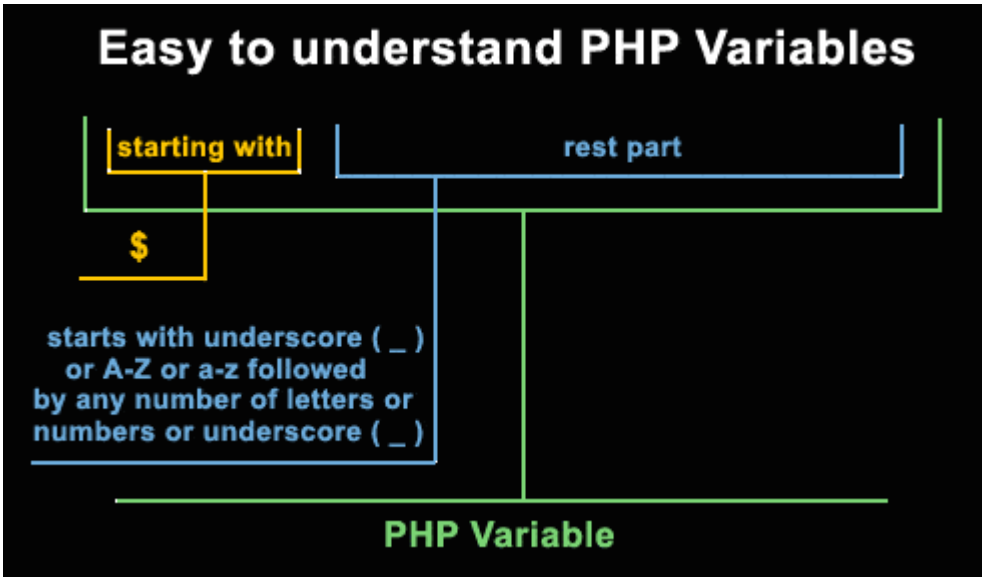
Folgende Elemente sollen dabei vorkommen (eingebunden in PHP-Code):

- Kommentare
- Sonderzeichen, wie z.B. Anführungsstriche
- Farbformatierungen
- Listen
- Tabellen
- ...

1.3.1) Variablen und Operatoren

Variablen

Variablen müssen im Gegensatz zu vielen Programmiersprachen nicht extra deklariert werden. Sie werden mit einem vorangestellten Dollar- Zeichen gekennzeichnet und beim ersten Aufruf definiert.



Folgende Datentypen werden von PHP unterstützt:

Datentyp	**Bezeichnung**	**Beispiel**
Ganze Zahlen	**integer**	1911 oder -1911
Dezimalzahl	**double**	19.11 oder -19.11
Zeichenketten	**string**	„Kette von Zeichen“ oder 'Kette von Zeichen'
Felder (ein- oder mehrdimensional)	**array**	(„Frankfurt“, „Berlin“, „Zürich“)
Objekte	**object**	Verweis auf eine bestimmte Variable

Namensgebung bei Variablen

Bei der Vergabe von Namen für die Variablen gibt es folgende Regeln. Der Name einer Variablen

- muss mit dem Dollarzeichen beginnen und darf kein Leerzeichen enthalten
- darf nur aus Buchstaben und Ziffern bestehen, aber keine Umlaute oder „ß“ enthalten
- muss als zweites Zeichen einen Buchstaben enthalten
- darf keine Sonderzeichen außer dem Unterstrich „_“ enthalten
- kann Groß- oder Kleinbuchstaben enthalten, wobei zwischen Groß- und Kleinschreibung unterschieden wird (`$PrimZahl` ist nicht gleich `$primzahl`)
- darf nicht identisch sein mit einem so genannten reservierten Wort (`and, break, case, class, continue, default, do, else, elseif, empty, endfor, endif`)

Variablen und Operatoren für Zahlen

Numerische Datentypen

Die numerischen Datentypen werden in Ganzzahl-Datentypen `integer` und Fließkommazahl-Datentypen `double` unterteilt.

Beispiel: preis.php

```
<?php
$preis_apfel = 2.50;
$menge = 4;
$gesamtpreis = $preis_apfel * $menge;
echo $gesamtpreis;
?>
```

Arithmetische Operatoren

Operator	Name	Bedeutung
+	Addition	$a + $b ergibt die Summe
-	Subtraktion	$a - $b ergibt die Differenz
*	Multiplikation	$a * $b ergibt das Produkt
/	Division	$a / $b ergibt den Quotienten
%	Modulo	$a % $b ist der Rest der ganzzahligen Division von $a / $b
+ +	Prä-, bzw. Postinkrement	$a++ (++$a) erhöht die Variable a um 1 nach (vor) der weiteren Verwendung
- -	Prä-, bzw. Postdekrement	$a- - (- -$a) verringert die Variable a um 1 nach (vor) der weiteren Verwendung

Beispiel: berechnung.php

```
<?php
  $preis_apfel = 2.50;
  $menge1 = 4;
  $menge2 = 10;
  $menge3 = 15;
  $gesamtpreis = $preis_apfel * ($menge1 + $menge2 + $menge3);
  echo $gesamtpreis;
?>
```

Variablen und Operatoren für Zeichenketten

Zeichenkettenoperator

Mittels Zeichenkettenoperator „.“ können mehrere Zeichenketten miteinander verknüpft werden:

```php
<?php
  $a = "Hallo ";
  $b = "Welt!";
  $c = $a.$b;
  echo $c;
?>
```

Ergebnis: Hallo Welt!

Typumwandlung

Der Datentyp einer Variablen wird durch die erste Wertzuweisung bestimmt.

Durch eine vorangestellte Typendeklaration kann dieser Datentyp geändert werden.

Typumwandlung in PHP funktioniert oft wie in C. Der Name des gewünschten Typs wird vor der umzuwandelnden Variablen in Klammern gesetzt, dies wird auch als Cast-Operation bezeichnet.

```php
<?php
      echo (int)$z;
      echo "<br>";
      var_dump($z);   //Gibt die Variable mit zugehörigen Datentyp aus!
      echo "<br>";
      echo $z;
?>
```

Der obige PHP-Code ergibt folgende Ausgabe:

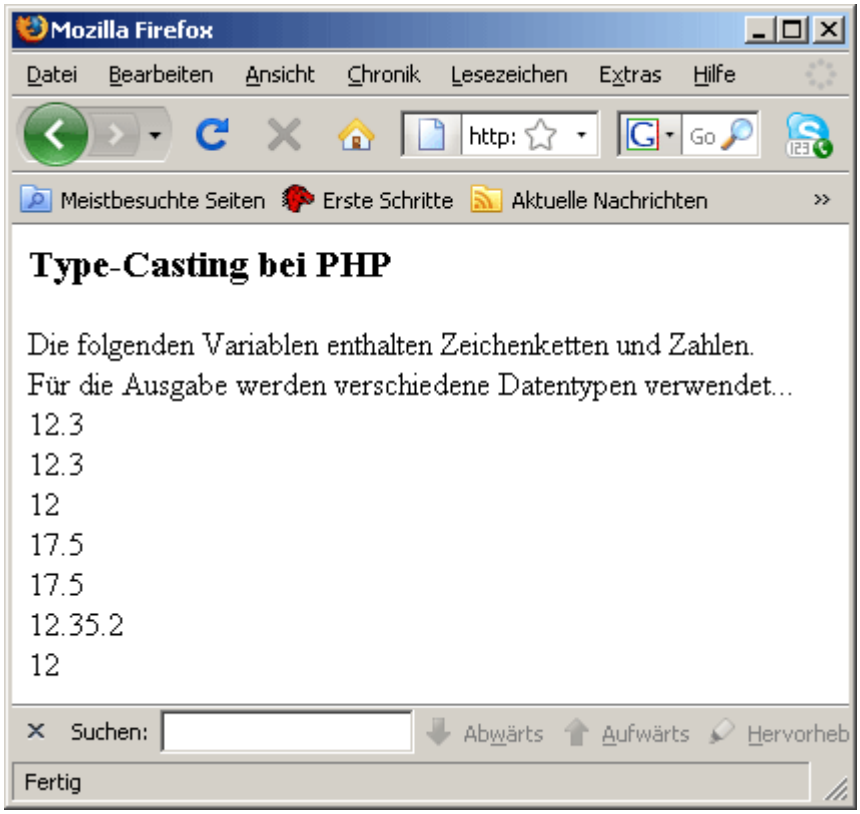
```
10
float(10.5)
10.5
```

Folgende Umwandlungen sind möglich:

- (int), (integer) – nach integer
- (bool), (boolean) – nach boolean
- (float), (double), (real) – nach float
- (string) – nach string
- (array) – nach array
- (object) – Wandlung zum Objekt

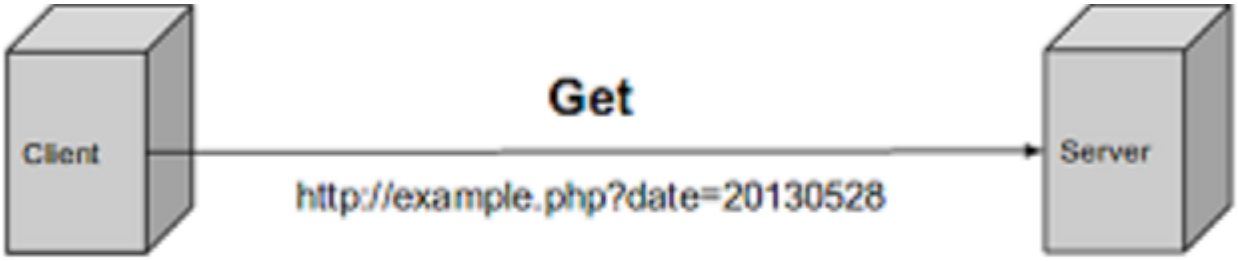
Anstatt eine Variable in einen String umzuwandeln, können Sie die Variable auch in doppelte Anführungszeichen einschließen.

Beachten Sie, dass Tabulatoren und Leerzeichen innerhalb der Klammern erlaubt sind. Deshalb sind die folgenden Beispiele identisch:



1.4.1) Formulare

Mit PHP kann man **interaktive Webseiten** erstellen, bei denen **Benutzereingaben aus Formularen durch PHP ausgewertet** werden. Die **Darstellung der Formulare** wird mit **HTML-Code** umgesetzt. Die Daten aus dem Formular werden dabei **per HTTP** mit den **Methoden POST oder GET** an das **auswertende PHP-Programm** übermittelt.



Methoden der Datenübertragung

POST

- Angabe von `method=„POST“` im einleitenden HTML-Tag `<form>`
- Formulardaten werden direkt an die Webadresse versendet.
- Daten können nicht in der Adresszeile des Browsers manipuliert werden. Optionen sind nur über das Formular möglich.
- Daten werde nicht im Logfile des Servers gespeichert.
- Formulardaten sind nicht im Verlauf des Browsers sichtbar.
- Längere Texte oder Daten können in Formularen übermittelt werden, da keine Beschränkung der Größe besteht.

GET

- Angabe von `method=„GET“` im einleitenden HTML-Tag `<form>`
- Formulardaten werden in der URL übermittelt und werden dann durch den Server in einer speziellen Umgebungsvariablen zwischengespeichert.

- Daten sind in der Adresszeile des Browsers änderbar, ohne das Formular erneut ausfüllen zu müssen.
- Der Aufruf des Skripts mit Angabe der Daten kann als Favorit abgelegt werden.
- Daten sind auf ca. 2KB beschränkt.

Da bei der Methode GET die Daten über die URL an den Server weitergegeben werden, ist es möglich, die Daten zu manipulieren. Die sicherere und am häufigsten genutzte Methode ist daher die Methode POST.

Formulare auswerten mit PHP

Formulardaten eingeben

Zur Eingabe von Formulardaten, genügt eine HTML-Datei, in der ein Formular integriert ist:

formular_eingabe.html

formular_eingabe.html

```
<html>
<body>
<p>Bitte füllen Sie die nachfolgenden Eingabefelder aus:</p>
<form action="formular_auswertung.php" method="POST">
<pre>
<p>
Vorname:  <input type="Text" name="vorname"><br>
Nachname: <input type="Text" name="nachname"><br>
Wohnort:  <input type="Text" name="ort"><br>
</p>
<input type="Submit" value="Abschicken">
<input type="Reset" value="Zurücksetzen">
</pre>
</form>
</body>
</html>
```

Formulardaten übertragen

Klickt man auf den Button „Abschicken“, wird das in `action` angegebene Skript (`formular_auswertung.php`) gestartet.

Formulardaten auswerten

Die mit der Methode POST übermittelten Daten wird mit dem Befehl $_POST[„Parameter“] angesprochen. Der Name des Eingabefeldes im Formular (z.B. name=„vorname“) wird automatisch Element der Variablen $_POST (z.B. $_POST[„vorname“]) Der Eintrag des Eingabefeldes im Formular wird bei der Auswertung zum Wert der Variablen, z.B. $_POST[„vorname“] = „Max“ .

formular_auswertung.php

formular_auswertung.php

```
<?php
  echo "<p>Folgende Daten wurden übermittelt:</p>\n";
  echo "Vorname: " .$_POST["vorname"] ."<br>\n";
  echo "Nachname: " .$_POST["nachname"] ."<br>\n";
  echo "Wohnort: " .$_POST["ort"] ."<br>\n";
?>
```

Formular und Auswertung in einer Datei

Das Formular und die Auswertung können in einer Datei zusammengefasst werden. Mit einer if-Abfrage kann überprüft werden, ob der Submit-Button gedrückt wurde oder nicht.

pizzabestellung.php

pizzabestellung.php

```
Pizzabstellung<br>
Pizza Margarita zum günstigen Preis von 5.50 Euro bestellen! <br>
<pre>
<form action="pizzabestellung.php" method="POST">
  Name: <input type="Text" name="name" />
  Lieferadresse: <input type="Text" name="adresse" />
  Anzahl der Pizzen: <input type="Text" name="anzahl" /><br><br>
  <input type="Submit" name="schicken" value="Bestellung
abschicken"><br>
  <input type="reset" value="Zurücksetzen">
</form>
</pre>
<hr>
<?php
    if ($_POST["schicken"]=="Bestellung abschicken")
    {
    echo "<i>Für folgende Bestellung wird gedankt: </i><br><br>";
    echo "<b>Name:</b> ".$_POST["name"]."<br>";
    echo "<b>Lieferadresse:</b> ".$_POST["adresse"]."<br>";
    if ($_POST["anzahl"]<>0)
```

```
        {
    echo "Menge beträgt <b>".$_POST["anzahl"]." Stück</b> zum Preis von
<b>".$_POST["anzahl"]*5.50." Euro</b>";
    }
    }

?>
```

Überprüfen des mitgeschickten Arrays mittels POST

```
print_r($_POST);
var_dump($_POST);
```

`print_r()` Zeigt den Inhalt der Variablen in lesbarer Form an. `var_dump()` geht einen Schritt weiter, es können auch, durch Beistriche getrennt, mehrere Variablen angegeben werden. Die Informationen sind detaillierter. Sie enthalten auch Typ und Länge der Variablen.
`var_dump(Variable 1[, Variable 2,…]);` Das gesamte mittels POST übermittelte Array wird ausgegeben.

Beispiel Newsletter

1.4.2) Aufgabe 2

Erstelle eine kleine Website für eine Apfelbestellung:

- bioladen.html
  - Eingabe von Name, Adresse, Menge (in kg) 1. Sorte Golden Delicious (1,20 pro kg), 2. Sorte: Idared (1,50 pro kg)
- bioladen-auswertung.php
  - Ausgabe von Name, Adresse, bestellter Menge und Gesamtpreis
  - ab insg. 10 kg bestellter Menge: keine Versandkosten, sonst 5,00 Euro.

1.5.1) Anführungszeichen

Doppelte oder einfache Anführungszeichen

```
$foo = 'Heute';

echo "$foo ist ein schöner Tag!"; // Ausgabe: Heute ist ein schöner Tag!
echo '$foo ist ein schöner Tag!'; // Ausgabe: $foo ist ein schöner Tag!
```

Wenn ein PHP-Programmierer einen Link aus Variablen erstellt, kann man häufig folgenden Code sehen:

```
echo '<a href="' . $link . '" id="' . $id .'" class="' . $class . '">' .
$linktext . '</a>';
```

oder

```
?>
<a href="<?php echo $link; ?>" id="<?php echo $id; ?>"
   class="<?php echo $class; ?>"<?php echo $linktext; ?></a>
<?php
```

oder

```
echo "<a href=\"$link\" id=\"$id\" class=\"$class\">$linktext</a>";
```

Abgesehen vom persönlichen Stil des Programmierers haben die Code-Beispiele eines gemeinsam: sie sind schlecht zu lesen. „Klar!“, werden jetzt die Programmierer sagen, „In HTML müssen die Attribute in doppelten Anführungszeichen stehen und in PHP muss man eine der oben gezeigten Methoden verwenden“. Sicher? Müssen Attribute wirklich in doppelten Anführungszeichen stehen? Die einfache Antwort ist: Nein! Ein Blick in die Spezifikation bringt Folgendes an den Tag:

[…]Standardmäßig verlangt SGML, dass alle Attributwerte entweder von doppelten Anführungszeichen (ASCII dezimal 34) oder einfachen Anführungszeichen (ASCII dezimal 39) begrenzt werden. Einfache Anführungszeichen können im Attributwert enthalten sein, wenn der Wert durch doppelte Anführungszeichen begrenzt ist und umgekehrt.[…]

Doppelte oder einfache Anführungszeichen. Das macht die Sache doch viel übersichtlicher:

```
echo "<a href='$link' id='$id' class='$class'>$linktext</a>";
```

```
echo 'Micro$oft'; // ergibt: Micro$oft
echo "Micro$oft"; // ergibt: Micro + Inhalt der Variable $oft
echo "Micro\$oft"; // ergibt: Micro$oft

echo 'c:\temp'; // ergibt: c:\temp
```

```php
echo "c:\temp"; // ergibt: c: + Tabulator + emp
echo "c:\\temp"; // ergibt: c:\temp
echo 'c:\\temp'; // ergibt: c:\temp

echo 'Kein Hochkomma: \x27'; // ergibt: Kein Hochkomma: \x27
echo " Ein Hochkomma: \x27"; // ergibt: Ein Hochkomma: '
echo ' Ein Hochkomma: \''; // ergibt: Ein Hochkomma: '

echo "<input name='foo' value='$bar'>"; // gültiges HTML
```

1.6.1) Vergleichsoperatoren

Eine Bedingung ist eine Möglichkeit, den Ablauf eines Skripts durch Entscheidungen zu beeinflussen. In einer Bedingung werden Ausdrücke verglichen. Das Ergebnis kann entweder „Ja“ (TRUE) oder „Nein“ (FALSE) sein.

Operator	Name	Bedeutung
==	Gleichheit	$a == $b ergibt TRUE, wenn $a und $b gleich sind.
===	Identisch	$a === $b ergibt TRUE, wenn $a und $b gleich und vom selben Datentyp sind.
!=	Ungleichheit	$a != $b ergibt TRUE, wenn $a und $b ungleich sind.
<	Kleiner	$a < $b ergibt TRUE, wenn $a kleiner $b ist.
>	Größer	$a > $b ergibt TRUE, wenn $a größer $b ist.
<=	Kleiner gleich	$a <= $b ergibt TRUE, wenn $a kleiner oder gleich $b ist.
>=	Größer gleich	$a >= $b ergibt TRUE, wenn $a größer oder gleich $b ist.

Verknüfung von Bedingungen

Operator	Name	Bedeutung
and bzw. &&	UND	$a and $b ergibt 1, wenn $a und $b 1 sind, ansonsten wird 0 zurückgegeben
or bzw. ||	ODER	$a or $b ergibt 1, wenn mindestens eine der beiden Variablen 1 ist.
xor	ausschließendes ODER	$a xor $b ergibt 1, wenn entweder $a oder $b 1 sind, aber nie beide gleichzeitig
!	NICHT	!$a ergibt die Umkehrung des Wahrheitswertes

1.6.2) Die einfache if-Anweisung

Syntax

```
if (Bedingung)
{
  Anweisungen;
}
```

- Die Bedingung steht in runden Klammern.
- Die Anweisungen stehen in geschwungenen Klammern.
- Liefert die Bedingung TRUE zurück, werden die Anweisungen ausgeführt, ist die Bedingung FALSE, werden die Anweisungen ignoriert
- Wird nur eine Anweisung ausgeführt, können die geschwungenen Klammern entfallen.

Beispiel: if1.php

```
<?php
  $menge = 7;
  if ($menge > 5)
     echo "Sie haben mehr als 5 Kilo bestellt.";
?>
```

- Der Variable $menge wird der Wert 7 zugewiesen. Da $menge größer als 5 ist, wird die Anweisung ausgeführt.

Beispiel: if2.php

```
<?php
  $menge = 7;
  if ($menge > 5)
  {
    echo "Sie haben mehr als 5 Kilo bestellt.<br>\n";
    echo "Der Versand ist deswegen kostenfrei.";
  }
?>
```

Die if-Anweisung mit else Zweig

Syntax

```
if (Bedingung)
{
  Anweisungsblock1;
}
```

```
else
{
  Anweisungsblock2;
}
```

- Falls die Bedingung erfüllt ist, wird `Anweisungsblock1` ausgeführt, sonst `Anweisungsblock2`.

Beispiel: ifelse.php

```
<?php
$menge=4;
if($menge > 5)
{
   echo "Sie haben mehr als 5 Kilo bestellt.<br>\n";
   echo "Der Versand ist deswegen kostenfrei.";
}
else
{
  echo "Sie haben $menge Kilo bestellt.<br>\n";
  echo "Der Versand kostet pauschal 7,00 EUR";
}

?>
```

Verschachtelte if-Anweisungen

Syntax

```
if (Bedingung1)
{
    Anweisungsblock1;
    if (Bedingung2)
    {
        Anweisungsblock2;
    }
    else
    {
        Anweisungsblock3;
    }
}
else
{
   Anweisungsblock4;
}
```

1.6.3) Fallauswahl mit switch-Anweisung

Wenn eine Variable mit verschiedenen Werten verglichen werden soll, kann die Programmierung mit einer `if`-Anweisung sehr aufwändig sein. Als Alternative dazu kann man in diesem Fall die `switch`-Anweisung einsetzen:

Syntax

```
switch ($variable)
{
   case Wert1:
      Anweisungsblock1;
      break;
   case Wert2:
      Anweisungsblock2;
      break;
   default:
      Anweisungsblock3;
}
```

- Stimmt der Wert der Variable mit einem der angeführten Auswahlwerte überein, dann wird der Anweisungsblock unmittelbar bis zur Anweisung `break` ausgeführt. Die restlichen Blöcke werden nicht ausgeführt.
- Ist kein `break` am Ende eines `case`-Teils angegeben, so werden auch alle nachfolgenden `case`-Blöcke ausgeführt, bis eine `break`-Anweisung erfolgt.
- Stimmt der Wert der Variablen mit keinem der angegebene Werte überein, wird der Anweisungsblock nach der `default`-Anweisung durchgeführt.

Bsp: switch-case.php

```
<?php
$sorte = "Gala";
   echo "Der Preis für 1kg $sorte beträgt <br>\n";
   switch($sorte) //Preiszuordnung abhängig von Sorte
   {
    case "Jonagold":
         echo "1,50 EUR.";
         break;
    case "Delicious":
         echo "1,60 EUR.";
         break;
    case "Gala":
         echo "1,65 EUR.";
         break;
    case "Elstar":
         echo "2,00 EUR.";
         break;
```

```
    default:
        echo "Diese Sorte haben wir leider nicht im Angebot.";
?>
```

1.6.4) Aufgaben 3-7

Aufgabe 3

Apfelbestellung - Versandkosten

Erweitere die Aufgabe 2 folgendermaßen:

- Wenn mehr als 10 kg Äpfel bestellt werden, ist der Versand kostenfrei, ansonsten werden 7.00 Euro verrechnet.
- Ausgabe:
  - Rückmeldung, ob Versandkosten zu bezahlen sind oder nicht.
  - Gesamtpreis.

Aufgabe 4

Minimum - Maximum

- In einem Formular (zahlen_eingabe.html) sollen drei Zahlen eingegeben werden (mittels drei Input-Feldern)
- Als Rückmeldung (minmax_ausgabe.php) wird das Maximum sowie das Minimum der drei Zahlen ausgegeben.

Aufgabe 5

Kennwortabfrage mit if

Viele Webseiten arbeiten mit einer Anmeldung des Benutzers. Um Zugang zu weiteren Teilen der Webseite zu bekommen, muss sich der Benutzer einem „Loginverfahren“ unterziehen.

- Erstelle hierfür ein Formular (`kennwort.html`), in welchem der Benutzer ein Kennwort eingeben kann. Verwende dazu ein `<input>`-Feld, bei dem das Kennwort bei der Eingabe nicht sichtbar ist.
- Nach Klick auf den Button „Abschicken“ soll eine Rückmeldung (`kennwort_auswertung.php`) erfolgen, ob das Passwort richtig eingegeben wurde.
- Baue das Formular so aus, dass vor dem Passwort auch der Nickname eingegeben werden muss. Eine positive Rückmeldung (z.B. „Zutritt erfolgreich…“) wird nur dann gegeben, wenn Nickname **und** Passwort korrekt eingegeben wurden.

Aufgabe 6

Verschachtelte if-Anweisungen

Baue die Aufgabe 3 folgendermaßen aus:

- Wenn mehr als 20 kg Äpfel bestellt werden, gibt es zur Bestellung zusätzlich ein kleines Präsent.
- Ausgabe:
  - Rückmeldung, ob Versandkosten zu bezahlen sind oder nicht und ob es ein Präsent gibt.
  - Gesamtpreis.

Aufgabe 7

switch-Anweisung

Erweitere die Aufgabe 5 folgendermaßen aus:

- Bei der Bestellung soll die bestellte Apfelart in einem `<input>`-Feld eingegeben werden.

Apfelsorte	Preis
Jonagold	1,50 EUR
Delicious	1,60 EUR
Gala	1,65 EUR
Elstar	2,00 EUR

- Ausgabe
  - Welche Äpfelsorte bestellt wurde (inkl. Preisangabe), bei falscher Eingabe soll eine entsprechende Meldung erfolgen und kein Gesamtpreis berechnet werden.
  - Gesamtpreis je nach bestellter Sorte.

1.7.1) for-Schleife

In der for-Schleife wird genau angegeben, wie oft die Schlaufe durchlaufen werden soll.

Syntax

```
for (Initialisierung; Bedingung; Operation)
{
   Anweisungsblock;
}
```

Beispiel: Ausgabe der ersten n ungeraden Zahlen und die Summe dieser Folge

```
<?php
     echo "<h2>Ausgabe der ersten n ungeraden Zahlen und die Summe dieser
Folge</h2>\n";
     $n=10;
     $summe=0;
     for ($i=1;$i<=$n;$i++)
     {
        $ung=$i*2-1;
        $summe=$summe+$ung;
        echo $i.". ungerade Zahl: ".$ung."<br>\n";
     }
     echo "Die Summe der ersten $n ungeraden Zahlen beträgt: ".$summe;
?>
```

Beispiele für Operatoren in for-Schleifen

Bedingung in der for-Schleife	Werte von $i
for ($i=1; $i<=5, $i++)	1, 2, 3, 4, 5
for ($i=1; $i<5, $i++)	1, 2, 3, 4
for ($i=15; $i>=10, $i--)	15, 14, 13, 12, 11, 10
for ($i=15; $i>10, $i--)	15, 14, 13, 12, 11
for ($i=0; $i<=100, $i=$i+10)	10, 20, 30, 40, 50, 60, 70, 80, 90, 100
for ($i=1; $i<=12, $i=$i+1.2)	1, 2.4, 3.6, 4.8, 7.2, 8.4, 9.6, 10.8, 12

Geschachtelte for-Schleifen

Mit geschachtelten for-Schleifen können zweidimensionale Strukturen abgearbeitet werden.

Beispiel: Erstellen einer Tabelle mit 3 Zeilen und 4 Spalten mittels zweier for-Schleifen

```php
<?php
   echo "<table border='1'>\n";
   for ($i=1;$i<=3;$i++)
   {
       echo "<tr>\n";
       for ($j=1;$j<=4;$j++)
       {
           echo "<td> $i. Zeile, $j. Spalte </td>\n";
       }
       echo "</tr>\n";
   }
   echo "</table>\n";
?>
```

1.7.4) Foreach-Schleife

Die foreach-Schleife ermöglicht es, auf einfache Weise ein Array zu durchlaufen. foreach funktioniert nur in Verbindung mit Arrays. Wenn Sie versuchen, foreach mit einer Variablen eines anderen Datentyps oder einer nicht initialisierten Variablen zu benutzen, gibt PHP einen Fehler aus.

Definition

```
foreach (array_expression as $value) {
  Anweisung/en;
}
```

Mithilfe von foreach wird das Durchlaufen eines Arrays wesentlich vereinfacht. Gegenüber der while-Schleife mit list und each ist die foreach-Schleife syntaktisch deutlich im Vorteil.

Beispiel while

```
<?php
$zahlen = array (10, 20, 30, 40);
while (list(, $value) = each ($zahlen)) {
  echo "Wert: $value<br>\n";
}
?>
```

Beispiel foreach

```
<?php
foreach ($zahlen as $value) {
  echo "Wert: $value<br>\n";
}
?>
```

Das Array wird von Anfang bis Ende durchlaufen und bei jedem Schleifendurchlauf wird das aktuelle Element der Variablen array_expression zugewiesen. Jedes Arrayelement kann wiederum ein Array sein, dann könnte mit einer verschachtelten foreach-Schleife auch dieses Array ausgewertet werden.

Beispiel – mit assoziativem Array

```
$personen = array("Matthias","Caroline","Gülten");

foreach ($personen as $person) {
  echo "Name: $person<br>\n";
}
```

```
// Ausgabe:
// Name: Matthias
// Name: Caroline
// Name: Gülten
```

Typischerweise ist es erlaubt, einzelne Elemente eines solchen Arrays mit unterschiedlichen Datentypen zu belegen. Das können auch weitere Arrays sein.

Erweiterte Syntax

Sollten Sie Arrays mit Schlüssel-Wert-Paaren bauen, kann foreach mit einer erweiterten Syntax diese Paare direkt auslesen. Die grundlegende Syntax lautet:

Definition

```
foreach (array as $key => $value) {
  Anweisung/en;
}
```

Hier wird der Operator ⇒ eingesetzt, der schon bei der Konstruktion des Arrays verwendet wurde.

Beispiel

```
<?php
$person = array("Vorname" => "Caroline",
      "Nachname" => "Kannengiesser",
      "Alter" => 25,
      "Ort" => Berlin);
foreach ($person as $key => $val) {
  echo "Feld $key hat den Wert: $val<br>";
}
?>
```

Ausgabe

```
Feld Vorname hat den Wert: Caroline
Feld Nachname hat den Wert: Kannengiesser
Feld Alter hat den Wert: 25
Feld Ort hat den Wert: Berlin
```

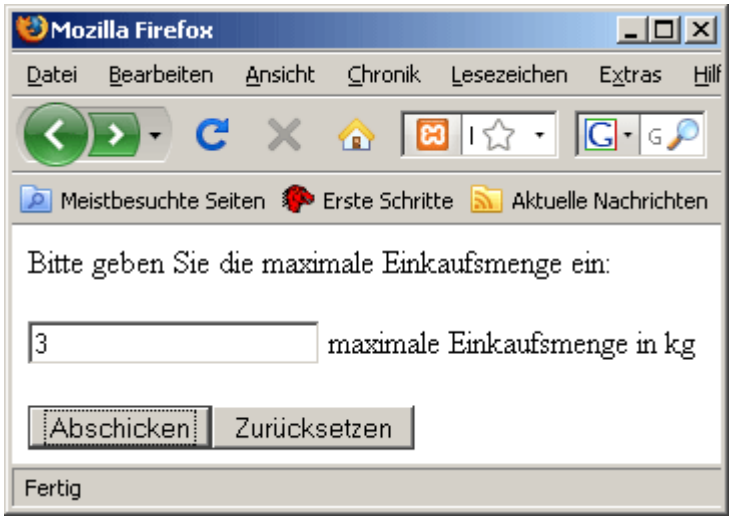
1.7.2) while-Schleife

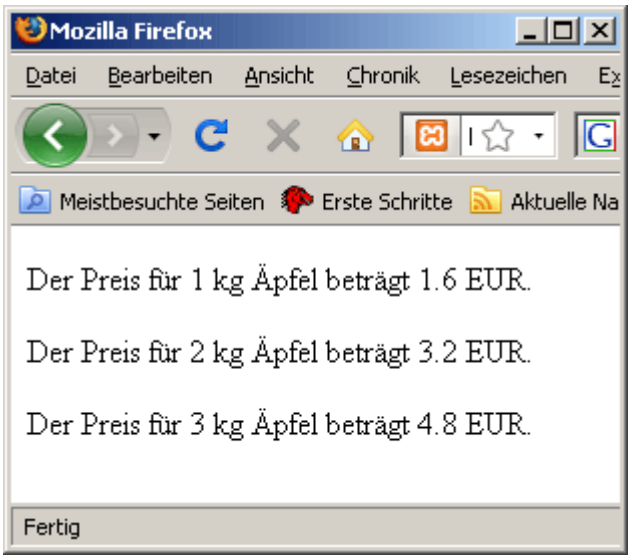
Syntax

```
while(Bedingung)
{
   Anweisungsblock;
}
```

Bsp: while.php

```
<?php
$menge = 1;
$preis = 1.6;
while ($menge <= $_POST["maxmenge"])
{
   $gesamtpreis = $preis * $menge;
   echo "<p>Der Preis für $menge kg Äpfel beträgt $gesamtpreis EUR.</p>\n";
   $menge++;
}
?>
```





1.7.3) do-while-Schleife

Die do-while Schleife unterscheidet sich von der while-Schleife, dadurch, dass die Abbruchbedingung nicht am Anfang, sondern am Ende jedes Schleifendurchlaufs geprüft wird. Die Anweisung wird somit zumindest einmal durchlaufen.

Syntax

```
do
{
   Anweisungsblock;
}
while (Bedingung);
```

Bsp: dowhile.php

```
<?php
   $i=1;
   do
   {
     echo "<br>".$i*$i;
     $i++;
   }
   while ($i<=10);
?>
```

1.7.5) Aufgabe 8 bis 11

Aufgabe 8

Erstelle ein PHP-Skript, welches eine Folge von n Zahlen erzeugt, in der jede Zahl den doppelten Wert der vorhergehenden hat.

Aufgabe 9

Erstelle ein PHP-Skript, das das „Ein-mal-eins“ für die Faktoren zwischen 11 und 20 ausgibt („großes Ein-mal-eins“). Die einzelnen Rechnungen sollen dabei in einer Tabelle aufscheinen.

Aufgabe 10

Schreibe ein PHP-Skript, das die Fibonacci-Zahlen bis zu einer eingegebenen Höchstgrenze erzeugt und ausgibt.

Eine Fibonacci-Zahl wird als Summe der beiden vorhergehenden Fibonacci-Zahlen gebildet. Die erste und die zweite Fibonacci-Zahl sind gleich 1.

Beispiel: 1, 1, 2, 3, 5, 8, 13, ...

Aufgabe 11

Teiler einer Zahl

Erstelle ein HTML- und PHP-Skript, welches die Teiler von Zahlen ermittelt. Weiters soll zusätzlich bestimmt werden, ob es sich um eine Primzahl (Anzahl der Teiler: 2) handelt

a) Version 1: Es werden die Teiler der eingegebenen Zahl ermittelt:

Ausgabe z.B.: 27 hat die Teiler 1, 3, 9, 27

b) Version 2: Es werden die Teiler sämtlicher Zahlen bis zur eingegebenen Zahl ermittelt:

Ausgabe z.B.:

...

23 hat die Teiler 1, 23 (Primzahl)

24 hat die Teiler 1, 2, 3, 4, 6, 8, 12, 24

25 hat die Teiler 1, 5, 25

26 hat die Teiler 1, 2, 13, 26

27 hat die Teiler 1, 3, 9, 27

1.8.1) Was sind Felder?

Felder, auch Arrays oder Feldvariablen genannt, sind in der Lage, mehrere Variablen unter einem gemeinsamen Bezeichner zu speichern. Beispielsweise erhalten Sie in einer Berechnung 20 Ergebnisse. Die Ausgabe der 20 Werte soll erst erfolgen, wenn alle Berechnungen abgeschlossen wurden. Sie könnten 20 Variablen (`$ergebnis1, $ergebnis2, $ergebnis3, $ergebnis4`, ...) definieren, um alle Ergebnisse der Berechnungen zu speichern. Einfacher und schneller kann die Aufgabe gelöst werden, wenn die Ergebnisse der Berechnung in einem Feld abgelegt werden. Der Zugriff auf die einzelnen Ergebnisse erfolgt jeweils über ihre Position innerhalb des Feldes.

Felder in PHP

Felder unterscheiden sich in der Art, wie auf die Werte innerhalb des Feldes zugegriffen werden kann, und in ihrem Aufbau.

Auf Werte in Feldern zugreifen:

- Im numerisch indizierten Feld werden die einzelnen Werte (Value) innerhalb des Feldes über eine laufende Nummer, auch **Index oder Schlüssel** genannt, angesprochen. Der **Schlüssel** ist eine **laufende Nummer**, die automatisiert vergeben wird und bei **Null beginnt**.

```php
$myArray = array();

$myArray[0] = "Susi";
$myArray[1] = "Maxi";

//oder

$myArray = array("Susi","Maxi");
```

Schlüssel & Inhalt des Arrays $myArray

key (Ort)	value (PLZ)
0	Susi
1	Maxi

- Im assoziativen Feld werden die einzelnen Werte (Value) innerhalb des Feldes über einen eindeutigen Schlüssel, Key genannt, angesprochen.

```
$myArray = array();
$myArray['key'] = "value";


//oder

$staedte = array('Guetersloh' => 98000,
            'Bielefeld' => 326715);

echo $staedte["Guetersloh"]; //gibt 98000 aus
```

Schlüssel & Inhalt des Arrays $staedte

key (Ort)	value (PLZ)
Guetersloh	98000
Bielefeld	326715

Aufbau von Feldern

Felder können ein- oder mehrdimensional sein:

- In einem eindimensionalen Feld können Sie beispielsweise eine Liste von Städten darstellen.
- Wenn Sie z. B. eine Tabelle mit verschiedenen Feldern abbilden möchten, können Sie dies mit einem zweidimensionalen Feld tun.
- Sie haben auch die Möglichkeit, Felder mit mehr als zwei Dimensionen zu nutzen.

Eigenschaften von Feldern

- In PHP müssen die Elemente von Feldern nicht vom selben Datentyp sein.
- Felder-Indizes bei numerisch indizierten Feldern beginnen mit 0. Das erste Element hat dementsprechend den Index 0, das zweite Element hat den Index 1, das dritte Element hat den Index 2 und so weiter. Der Index eines Feldes mit n Elementen reicht somit von bis $n-1$. Der Index eines Feldes mit vier Elementen reicht somit von 0 - 3.
- Beim Anlegen eines Feldes muss nicht angegeben werden, wie viele Elemente in diesem Feld gespeichert werden. Sie können auch nach der Erstellung dem Feld zusätzliche Elemente hinzufügen.

1.8.2) Indizierte Felder

Indizierte Felder erstellen

Ein indiziertes Feld anlegen

PHP bietet Ihnen eine einfache und schnelle Möglichkeit, Felder mit verschiedenen Werten zu füllen. Dies wird über die array-Anweisung realisiert.

Syntax und Bedeutung der array() -Anweisung

```
$Feldvariable = array(Wertl,Wert2,Wert3,...) ;
```

- Die `array()`-Anweisung beginnt mit dem reservierten Wort array.
- Der Variablen `$Feldvariable` werden die einzelnen Werte übergeben, indem sie durch Kommata voneinander getrennt angegeben werden.
- Die Indizierung der einzelnen Elemente erfolgt automatisch in der Reihenfolge der Angabe.

Beispiel

```
$staedte = array("Frankfurt","Berlin","Bern");
```

Es befinden sich drei Werte in dem Feld `$staedte`; Frankfurt, Berlin und Bern. Die Reihenfolge, in der die Werte zugewiesen worden sind, bestimmt die Indizes der Werte.

Index	0	1	2
Wert	Frankfurt	Berlin	Bern

Auf indizierte Felder zugreifen

Um auf einzelne Elemente in einem Feld zugreifen zu können, muss ihnen eine eindeutige Kennung zugewiesen werden. Bei numerisch indizierten Feldern entspricht der Kennung der automatisch erzeugte fortlaufende Index.

Auf einen bestimmten Wert eines Feldes greifen Sie zu, indem Sie zusätzlich zum Namen der Feldvariablen in den eckigen Klammern den Indexwert angeben.

```
$staedte[Index];
```

Beispiel

```
$stadt = $staedte[2] ;
```

Der Variablen `$stadt` wird der dritte Wert des Feldes `$staedte`, die Zeichenkette Bern, zugewiesen.

Mit indizierten Feldern arbeiten

Indizierte Felder ändern

Um einen Wert innerhalb einer Feldvariablen zu ändern, geben Sie bei der Wertzuweisung bei der Feldvariablen den entsprechenden Index des zu ändernden Wertes an und weisen Sie den neuen Wert zu.

```
$staedte[l] = "München";
```

In dem bisherigen Beispiel würde somit die Zeichenkette Berlin durch München ersetzt werden.

Indizierte Felder erweitern

Soll ein Feld um einen weiteren Wert am Ende des Feldes ergänzt werden, erzeugt PHP den entsprechenden Index automatisch.

Syntax und Bedeutung

```
$Feldvariable[] = Wert Zuweisung;
```

- Um einem Feld einen weiteren Wert (oder mehrere Werte) hinzuzufügen, brauchen Sie bei der Wertzuweisung keinen Index anzugeben. Das Feld wird am Ende um den Wert (bzw. die Werte) ergänzt und der Index automatisch erhöht.
- Sollte das Feld noch nicht bestehen, erstellt PHP automatisch ein Feld, sobald Sie der Variablen einen Wert zugewiesen haben.

Beispiel

```
$staedte = array("Frankfurt","Berlin","Bern");
$staedte[] = "Graz";
$staedte[] = "Rom";
```

ergibt:

Index	0	1	2	3	4
Wert	Frankfurt	Berlin	Bern	Graz	Rom

Anzahl der Feldelemente ermitteln

Der Befehl `COUNT($bereich);` liefert die Anzahl der Feldelemente des Arrays `$bereich`.

1.8.3) Assoziative Felder

Assoziative Felder erstellen

Ein assoziatives Feld anlegen

Eine weitere Möglichkeit, Feldvariablen anzulegen, sind die assoziativen Felder. Bei ihnen werden für den Zugriff keine fortlaufenden Indizes benutzt, sondern Schlüssel, über deren Wert auf die einzelnen Werte zugegriffen werden kann, so genannte Schlüsselwörter.

Syntax der `array()`-Anweisung bei assoziativen Feldern

Auch die assoziativen Felder lassen sich über die `array()`-Funktion füllen.

```
$Feldvariable = array (Schlüssel1 => Wertl,Schlüssel2 => Wert2, ...) ;
```

Beispiel:

```
$hauptstaedte = array("Schweiz" => "Bern", "Frankreich" => "Paris");
```

- Die array ()-Anweisung beginnt mit dem reservierten Wort array.
- Der Variablen werden die einzelnen Wertpaare, bestehend aus Schlüsselwort (Key) und Wert (Value), übergeben, indem sie durch Kommata voneinander getrennt angegeben werden.
- Die Indizierung der einzelnen Werte erfolgt über die Angabe eines Schlüssels in Form einer beliebigen, eindeutigen Zeichenkette oder eindeutiger Integerwerte. Die Zuweisung erfolgt über ⇒.

Auf assoziative Felder zugreifen

Beim Zugriff auf einen Wert des Feldes kann der Schlüsselwert direkt angegeben oder als Variable übergeben werden:

	Ergebnis
`$stadt = $hauptstaedte["Schweiz"];`	Variable `$stadt` hat den Wert „Bern“

oder:

	Ergebnis
`$k = "Schweiz"; $stadt = $hauptstaedte[$k];`	Variable `$stadt` hat den Wert „Bern“

Beispiel: arr_assoz.php Hier wurde die Zuweisung der Städtenamen mit der `array()`-Anweisung realisiert.

```
<html>
<body>
<?php
$hauptstaedte = array("Schweiz" => "Bern",
                      "Frankreich" => "Paris",
                      "Deutschland" => "Berlin",
                      "Österreich" => "Wien");
$k = "Österreich";
echo "<p>Hauptstadt von $k: " . $hauptstaedte[$k] ."</p>";
echo "Die Hauptstadt von $k ist " .$hauptstaedte["Österreich"];
?>
</body>
</html>
```

- Das assoziative Feld wird über die `array()`-Funktion und die Zuweisung der Schlüssel sowie Feldinhalte angelegt.
- Der Variablen $k wird die Zeichenkette Österreich zugewiesen.
- Mithilfe der Variablen $k wird das entsprechende Element aus dem Feld ausgelesen und angezeigt.
- Hier wird das Element durch die direkte Angabe des Schlüssels herausgefiltert. Die Bezeichnung des Schlüssels wird als Zeichenkette in den eckigen Klammern angegeben.

Mit assoziativen Feldern arbeiten

Daten an Variablen übergeben

Mit einer `foreach()`-Schleife sind Sie in der Lage, die Werte der Feldelemente einzeln auszulesen. Dabei wird jedes Element in einer neuen Variable zwischengespeichert.

```
foreach($Feld as $wert) Anweisung;
foreach($Feld as $index => $wert) Anweisung;
```

- Die Funktion `foreach()` erwartet die Angabe des assoziativen Feldes, dessen Elemente durchlaufen werden sollen.
- In der ersten Variante werden mithilfe von `foreach` bei jedem Schleifendurchlauf der Variablen ($wert) nacheinander die Werte der Arrayelemente zugewiesen.
- Mithilfe der zweiten Variante können Sie zusätzlich den Index des Arrayelements auslesen und der Variablen $index zuweisen, was beispielsweise bei assoziativen Feldern hilfreich ist.

Beispiel: foreach.php
Verschiedene Länder der Welt und deren Hauptstädte werden in einem assoziativen Feld gespeichert. Zur Ausgabe sollen das Land (= Schlüssel) und die dazugehörige Hauptstadt (= Wert) in einer Tabelle angezeigt werden:

```
<?php

 $hauptstaedte = array("Schweiz" => "Bern",
```

```
                        "Frankreich" => "Paris",
                        "Deutschland" => "Berlin");
 $hauptstaedte["Polen"] = "Warschau";
 $hauptstaedte["Italien"] = "Rom";
 $hauptstaedte["Österreich"] = "Wien";

 echo "<table border>";
 echo "<tr><td><b>Land</b></td>" ;
 echo "<td><b>Hauptstadt</b></td></tr>" ;
 foreach($hauptstaedte as $land=>$stadt)
 {echo "<tr><td>$land</td><td align='left'>$stadt</td></tr>" ;
 }
 echo "</table>";

?>
```

- Das assoziative Feld $hauptstaedte wird über die `array`-Anweisung gefüllt.
- Dem assoziativen Feld $hauptstaedte werden weitere Elemente hinzugefügt.
- Der Tabellenkopf wird erstellt.
- Mithilfe einer foreach-Schleife können Sie auf jedes Element des Feldes zugreifen. Die Schleife wird so lange durchgeführt, bis alle Elemente des Feldes angesprochen wurden. In jedem Schleifendurchlauf wird der Variablen `$land` der jeweilige Schlüsselwert und der Variablen `$stadt` der Wert des Feldes an der durch das Schlüsselwort festgelegten Position zugeordnet.
- Die Werte werden in die Tabelle eintragen und über den Ausgabebefehl echo am Bildschirm ausgegeben.

1.8.4) Mehrdimensionale Felder

Mehrdimensionale indizierte Felder erstellen

Grundlagen zu mehrdimensionalen Feldern

Im folgenden Beispiel sehen Sie mehrere Angaben (Vorname, Nationalität und Alter), die jeweils einen Eintrag in einer Feldvariablen bilden. Damit ist dieses Feld ein mehrdimensionales Feld. Bei jeder der Angaben sind Mehrfachwerte möglich. Mehrdimensionale Felder sind verschachtelte Felder, man spricht hier von äußeren und inneren Feldern.

Mehrdimensionale Felder können sowohl numerisch indiziert als auch assoziativ sein. Eine Mischform aus numerisch indizierten und assoziativen Feldern ist ebenfalls möglich. Im Beispiel bietet sich durch mögliche Mehrfachwerte z. B. ein indiziertes Feld an, das jeden Satz an Angaben mit einem eindeutigen numerischen Index versieht.

Index 1	Index 2		
	0	1	2
0	Oliver	spanisch	37 Jahre
1	Maria	deutsch	23 Jahre
2	Oliver	englisch	46 Jahre

In dieser Tabelle wird bereits die Arbeitsweise mehrdimensionaler Felder ersichtlich. Um beispielsweise an die Informationen von Maria zu gelangen, suchen Sie die Zeile mit der Angabe Maria und lesen die einzelnen Werte dieser Zeile aus: `Maria, deutsch, 23 Jahre`.

Mit mehrdimensionalen indizierten Feldern arbeiten

Ein mehrdimensionales indiziertes Feld hat statt *eines* Indexes - in Abhängigkeit von der Anzahl der Verschachtelungen - *mehrere* Indizes.

Syntax eines mehrdimensionalen indizierten Feldes

Die bereits erwähnten Informationen sollen im Folgenden als mehrdimensionales Feld realisiert werden.

Auf eine bestimmte Angabe in einem mehrdimensionalen indizierten Feld zugreifen

Ausschnitt aus der Beispieldatei„mehrdimensional.php“:

```
$person = array(array("Oliver", "spanisch", "37    Jahre"),
           array("Maria",    "deutsch",  "23 Jahre"),
```

```
            array("Oliver",    "englisch", "46    Jahre"));
$vornamel  = $person[l][0]    ;
$nationalitaetl    - $person[l][1]    ;
$alterl    = $person[l][2]    ;
```

Ergebnis:

```
$vorname1       = "Maria";
$nationalitaet1 = "deutsch";
$alter1         = "23 Jahre".
```

Ein mehrdimensionales indiziertes Feld erweitern

```
$person[3][0] = "Johanna";
$person[3][1] = "schwedisch";
$person[3][2] = "19 Jahre";
```

oder alternativ:

```
$person[] = array("Johanna", "schwedisch", "19 Jahre");
```

Mit mehrdimensionalen assoziativen Feldern arbeiten

Syntax eines mehrdimensionalen assoziativen Feldes

Die assoziativen Felder besitzen zum Ansprechen der einzelnen Elemente einen sogenannten Schlüssel. Um ein mehrdimensionales Feld anzulegen, wird jedem Schlüssel ein weiteres Feld mit Schlüssel-Wert-Paaren übergeben (Sonderfall zweidimensional). Verwenden Sie aus Gründen der Übersichtlichkeit und Nachvollziehbarkeit der Programmierung nicht mehr als drei Dimensionen bzw. Verschachtelungen.

Beispiel: mehrdimensional.php

```
$land = array("Spanien"  => array("Hauptstadt"    =>    "Madrid",
                                  "Sprache"  =>    "Spanisch",
                                  "Waehrung" =>    "Euro",
                                  "Flaeche"  =>    "504645 qkm"),
              "England"  => array("Hauptstadt"   =>    "London",
                                  "Sprache"  =>    "Englisch",
                                  "Waehrung" =>    "Pfund Sterling",
                                  "Flaeche"  =>    "130395 qkm"),
              "Portugal" => array("Hauptstadt"   =>    "Lissabon",
                                  "Sprache"  =>    "Portugiesisch",
                                  "Waehrung" =>    "Euro",
                                  "Flaeche"  =>    "92345 qkm"));
```

Hiermit ist jeder Satz an Eintragungen mit entsprechenden Länderinformationen über einen Schlüssel mit aussagekräftiger Bezeichnung - `Spanien, England` oder `Portugal` - direkt ansprechbar.

Auf eine bestimmte Angabe in einem mehrdimensionalen assoziativen Feld zugreifen

Ausschnitt aus Beispieldatei„ mehrdimensional.php“

```
$hauptstadt2  = $land["Portugal"]["Hauptstadt"];
$sprache2     = $land["Portugal"]["Sprache"];
$waehrung2    = $land["Portugal"]["Waehrung"];
$flaeche2     = $land["Portugal"]["Flaeche"];
```

Ergebnis:

```
$hauptstadt2  = "Lissabon";
$sprache2     = "Portugiesisch";
$waehrung2    = "Euro";
$flaeche2     = "92345 qkm";
```

Ein mehrdimensionales assoziatives Feld erweitern

```
$land["Ungarn"]["Hauptstadt"]    = "Budapest";
$land["Ungarn"]["Sprache"]   = "Ungarisch";
$land["Ungarn"]["Waehrung"]  = "Forint";
$land["Ungarn"]["Flaeche"]   = "93036 qkm";
```

oder alternativ:

```
$land["Ungarn"]   = array("Hauptstadt" =>    "Budapest",
                          "Sprache"    => "Ungarisch",
                          "Waehrung    => "Forint",
                          "Flaeche"    => "93036 qkm");
```

Daten aus mehrdimensionalen Feldern extrahieren

Bei der Arbeit mit mehrdimensionalen Feldern kommt es häufig vor, dass Sie die kompletten Daten eines inneren Feldes benötigen. Mit einer `foreach`-Schleife können Sie alle Werte einer Feldvariablen auslesen. Durch die Verwendung einer `foreach`-Schleife müssen Sie keinen Index oder Schlüssel kennen. Bei jedem Schleifendurchlauf wird das aktuelle Element in einer von Ihnen angegebenen Variablen zwischengespeichert. Hierbei handelt es sich auch um eine Feldvariable mit Angaben zum aktuellen „Datensatz“.

Syntax:

```
foreach($feld as $wert);
```

Um die einzelnen Elemente auszulesen und jeden Wert verschiedenen Variablen zu übergeben, verwenden Sie innerhalb der Schleife die `list()`-Funktion.

Syntax

```
list($variable1, $variable2, ...) = $wert;
```

- Der Funktion list () wird das entsprechende Element übergeben, das in die angegebenen Variablen aufgeteilt werden soll.
- Die Variablen werden nacheinander, durch Kommata voneinander getrennt, in den runden Klammern der Funktion list () angegeben.
- Die Anzahl und die Position der Variablen richten sich nach der Anzahl der Elemente im Feld.
- Bei der an dieser Stelle verwendeten Variablen $wert handelt es sich um die Variable, die als letztes Argument in der foreach () -Schleife definiert wurde.

Beispiel: array_zdim_list.php

Verschiedene Angaben zu Ländern werden in einem mehrdimensionalen Feld gespeichert. Zur Ausgabe sollen der Name des Landes, die Hauptstadt, die Sprache und die Landeswährung in separaten Variablen abgelegt werden.

```
<body>
<table width="500">
<tr>
<td width="125"><b>Land</b></td>
<td width="125"><b>Hauptstadt</b></td>
<td width="125"><b>Sprache</b></td>
<td width="125"><b>W&auml;hrung</b></td>
</tr>
    <?php
      $staedte = array(
①         "Japan"         => array("Tokio", "Japanisch", "Yen"),
          "Niederlande"   => array("Amsterdam", "Niederländisch", "Euro"),
          "Polen"         => array("Warschau", "Polnisch", "Z&#322;oty"),
          "Indien"        => array("Neu Delhi", "Indisch", "Rupie"),
          "Island"        => array("Reykjavik", "Isländisch", "Krone"),
          "Italien"       => array("Rom", "Italienisch", "Euro"),
          "Frankreich"    => array("Paris", "Französisch", "Euro"),
          "Spanien"       => array("Madrid", "Spanisch", "Euro"),
          "England"       => array("London", "Englisch", "Pfund Sterling")
          );
```

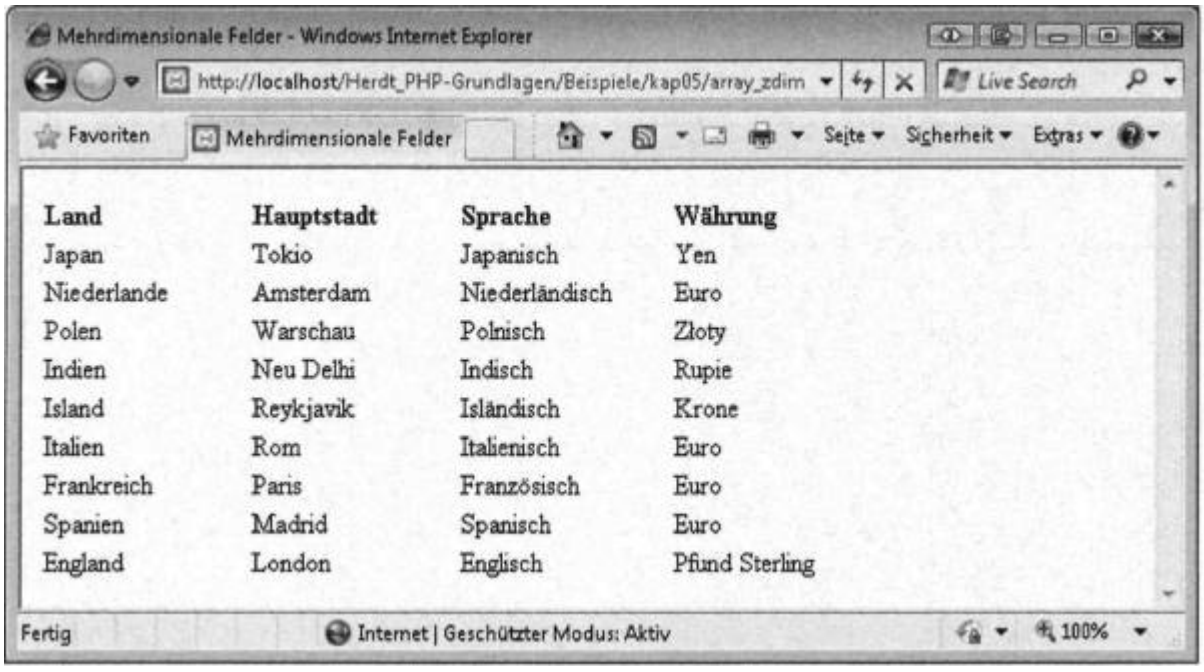
```
②       foreach($staedte as $key=>$ausgabe)
            {
③           list($hauptstadt, $sprache, $waehrung) = $ausgabe;
④           echo "<tr><td>" .$key ."</td>";
            echo "<td>" .$hauptstadt ."</td>";
            echo "<td>" .$sprache ."</td>";
            echo "<td>" .$waehrung ."</td></tr>";
            }
    ?>
</table>
</body>
```

(1) Das assoziative Feld `$staedte` wird neu definiert und mit Werten gefüllt. Die Daten enthalten Informationen zur Hauptstadt eines Landes, zur Sprache und zur im Land verwendeten Währung. Der Name des Landes selbst wird als eindeutiger Schlüssel verwendet.

(2) Mithilfe einer `foreach`-Schleife können Sie auf jedes Element des Feldes zugreifen, indem die Schleife so oft durchgeführt wird, bis auch das letzte Element angesprochen wurde. Da es sich in diesem Beispiel um mehrere Werte pro Element handelt, müssen die zusammenhängenden Werte eines Landes in dem neuen Feld `$ausgabe` zwischengespeichert werden. Im ersten Schleifendurchlauf enthält die Variable `$ausgabe` nur die Werte des zuerst definierten Landes `Japan (Tokio, Japanisch, Yen)`, im zweiten Schleifendurchlauf die Werte der `Niederlande` etc.

(3) Über den Befehl `list()` werden die Daten zum gerade abgerufenen Land (Hauptstadt, Sprache und Währung) in die entsprechenden Variablen `$hauptstadt, $sprache` und `$waehrung` aufgeteilt.

(4) Über den Ausgabebefehl echo werden die Werte der Variablen am Bildschirm ausgegeben. Bei der Variablen `$key` handelt es sich um den Schlüssel (Landesnamen), der in (2) definiert wurde.



Land	Hauptstadt	Sprache	Währung
Japan	Tokio	Japanisch	Yen
Niederlande	Amsterdam	Niederländisch	Euro
Polen	Warschau	Polnisch	Złoty
Indien	Neu Delhi	Indisch	Rupie
Island	Reykjavik	Isländisch	Krone
Italien	Rom	Italienisch	Euro
Frankreich	Paris	Französisch	Euro
Spanien	Madrid	Spanisch	Euro
England	London	Englisch	Pfund Sterling

Beispiel aus SÜ

```php
<?php
// Nicht assoziatives Array
$matrix = array (
            array(1,2,3,4),
            array(5,6,7,8),
            array(9,10,11,12),
            array(13,14,15,16),
            array(17,18,19,20)
          );
// Ausgabe als Zeilen und Spalten
for ($i=0;$i<count($matrix);$i++){
    for ($j=0;$j<count($matrix[$i]);$j++) {echo $matrix[$i][$j].' ';}
    echo '<br>';
}

// Assoziatives Array
$user = array(
          'person'=>array('name'=>'Thomas', 'nickname'=>'Baumi'),
          'lang'
=>array('deutsch','mostviertlerisch','english','spanisch','französisch'),
          'kontakt'=>array('email'=>'thomas.baumgartner@bgamstetten.ac.at',
'web' =>'www.baumi.net')
        );

// Ausgabe
foreach ($user as $eigenschaft => $arrayinhalt) {
  echo $eigenschaft.':';
  foreach ($arrayinhalt as $index =>$wert)
     echo ' '.$index.' ist '.$wert.' | ';
  echo '<br>';
}
?>
```

Verwenden des passenden Feldtyps

Der passende Feldtyp hängt von den Daten ab, die Sie in der Feldvariablen speichern wollen. Die folgende Tabelle soll Ihnen bei der Auswahl des Feldvariablentyps helfen:

Daten	Anzahl an Informationen pro Eintrag	Bevorzugter Feldtyp
Einfache Liste (z. B. Namen, Länder, Automarken etc.)	1	✔ eindimensional/indiziert
Wertepaare (z. B. Länder - Hauptstädte, Artikel - Preis etc.)	2	✔ eindimensional/assoziativ, sofern ein Wert eindeutig ist (= Schlüssel), sonst ✔ mehrdimensional/indiziert oder assoziativ
Mehrere evtl. mehrfach verschachtelte Werte (z. B. Mitarbeiter und ihre Stammdaten, Artikelkategorien - Artikel - Artikeldetails etc.)	> 2	✔ mehrdimensional/indiziert oder assoziativ

1.8.5) Weitere Informationen zu Feldern in PHP

Felder begegnen uns in PHP nicht nur, wenn wir selbst Feldvariablen definieren. PHP arbeitet intern automatisch mit Feldern, und zwar bei folgenden Aktionen:

- Die in einem HTML-Formular übermittelten Daten speichert PHP in einer Feldvariablen.
- Wenn man mit Sessions arbeitet, werden die zur Session gehörigen Daten automatisch in einem Feld abgelegt.
- Bei Abfragen aus Datenbanken befinden sich die Ergebnisse ebenfalls üblicherweise in einer Feldvariablen.

Weitere Feld-Funktionen

PHP kennt annähernd 100 Funktionen für den Umgang mit Feldern. Für nahezu jede Fragestellung liefert PHP eine passende Funktion. Folgende Aufgabengebiete werden abgedeckt:

- Auslesen
- Auswerten (u.a. Dopplungen finden)
- Sortieren
- Suchen
- Teilen und Zusammensetzen
- Verändern (u.a. Füllen, Hinzufügen, Löschen)

Eine Auswahl zusätzlicher wichtiger Feldfunktionen wird kurz in der nachfolgenden Tabelle vorgestellt:

Feldfunktion	**Beschreibung**
`array_flip($feld)`	Im Feld werden Indizes mit Werten vertauscht.
`array_key_exists(wert,$feld)`	Prüfung, ob ein Schlüssel in einem Feld vorhanden ist.
`array_keys($feld)`	Liefert die Indizes des angegebenen Feldes zurück.
`array_merge($feldl,$feld2…)`	Fügt die Elemente mehrerer Felder zu einem Feld zusammen.
`array_push($feld,werte)`	Das Feld wird um den oder die angegebenen Werte am Ende des Feldes erweitert.
`array_search(wert,$feld)`	Das Feld wird nach dem angegebenen Wert durchsucht.
`array_sum($feld)`	Addiert die Werte des Feldes und liefert das Ergebnis zurück.
`array_unique($feld)`	Es wird ein neues Feld erstellt, aus dem doppelte Werte des angegebenen Feldes gelöscht wurden.
`array_values($feld)`	Alle Werte des Feldes werden zurückgeliefert.
`count($feld)`	Gibt die Anzahl der Elemente des Feldes zurück.
`sort($feld)` `rsort($feld)`	Sortiert die Werte des angegebenen Feldes aufsteigend bzw. absteigend.

1.8.6) Aufgaben 12 bis 18

Aufgabe 12 - Indizierter Array

1. Schreibe mittels PHP-Code einen „indizierten Array“, der folgende Elemente auflistet:

- Englisch
- Informatik
- Deutsch
- Geografie
- Bewegung und Sport

2. Gib alle eingegebenen Fächer aus.

3. Schreibe folgenden Satz im PHP-Block und greife auf den Array zu, um ihn zu vervollständigen:
„Mein Lieblingsfach ist [.....] „

Aufgabe 13 - Assoziativer Array

1. Schreibe mittels PHP-Code einen „assoziativen Array“. Beziehe folgende Werte ein:

- Hauptfach - Englisch
- Nebenfach - Geschichte

2. Schreibe die folgenden Sätze im PHP-Block und greife auf den Array zu, um den Satz zu vervollständigen:

[Englisch] ist ein Hauptfach. [Geschichte] ist ein Nebenfach.

Aufgabe 14 - Jahreszeit

Erstelle ein Dokument namens jahreszeit.php. Hier sollen die vier Jahreszeiten in einer Werteliste namens

```
$jahreszeit[0]
```

abgebildet werden können. Nutze dafür die **Kurzform**, indem du alle Werte auf einmal zuweist. Gib den Herbst aus.

Aufgabe 15 - Monatsnamen

Erstelle ein Dokument namens monat.php. Speichere alle Monatsnamen in einem Array namens `$monat`. Nutze dafür die **Langform**, indem du die Monate einzeln zuweist! Gib sodann folgenden

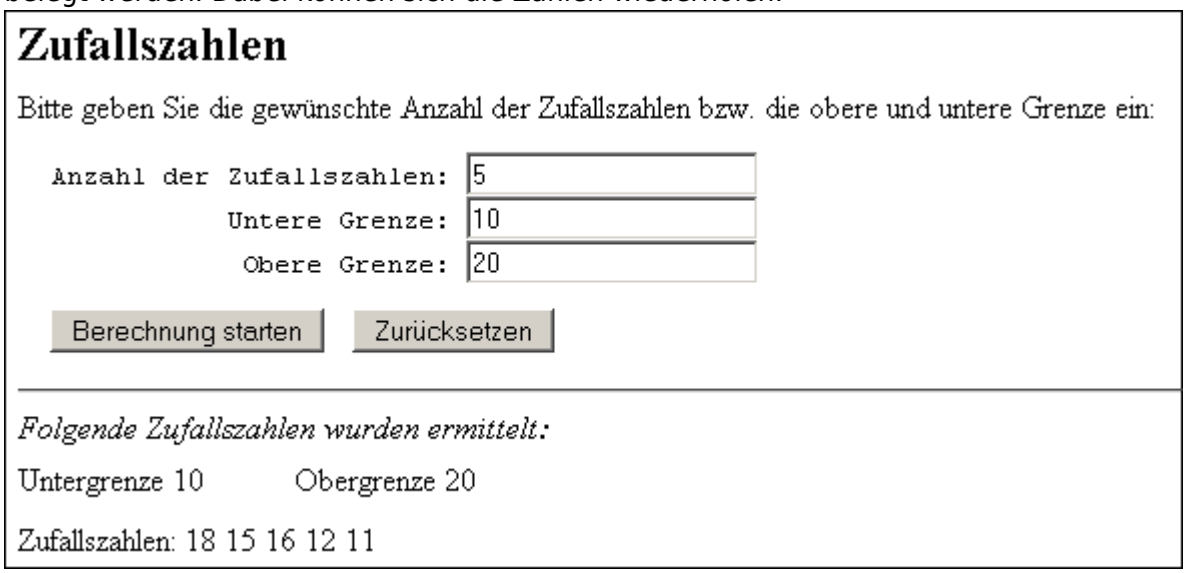
String aus:

„Ein Jahr hat 12* Monate. Der aktuelle Monat ist der Mai.“ (* Zählfunktion einsetzen)

Überlege dabei, welcher Arraytyp sinnvoll ist (indiziert oder assoziativ)!

Aufgabe 16 - Zufallszahlen

Ein Array soll mit einer vorgegebenen Anzahl von Zufallszahlen zwischen den Grenzen ug und og belegt werden. Dabei können sich die Zahlen wiederholen.



Hinweis: Wie man eine Zufallszahl zwischen den Grenzen ug und og zu erhalten kann, zeigt die folgende Zeile:

```
        $zahl[$i]=rand($ug,$og);
```

1. Nach der Generierung der Zufallszahlen soll das Array ausgegeben werden.
2. Der Mittelwert aller Zahlen soll ausgegeben werden. 3. Die Standardabweichung soll ausgegeben werden.

Aufgabe 17 - Primzahlenberechnung mit dem Sieb des Erathostenes

Zur Berechnung aller Primzahlen bis zu einer Obergrenze n gibt es ein Verfahren, das vom griechischem Mahtematiker *Erathostenes* stammt. Es wird das *Sieb des Erathostenes* genannt und basiert auf der Idee, dass die Vielfachen einer Primzahl mit Sicherheit keine Primzahlen sind. Wenn wir z.B. wissen, dass 2 eine Primzahl ist, können wir alle Vielfache von 2 aus der Menge der Primzahlenkandidaten streichen. Die kleinste Zahl der dann verbleibenden Menge ist die nächste Primzahl. Wir eliminieren wiederum alle ihre Vielfache.

Implementieren lässt sich dieser Algorithmus sehr schön mit Hilfe eines Arrays, das zunächst lauter „p“ (Primzahl) in den Feldern stehen hat. Wird eine Zahl gestrichen, ersetzt man das „p“ durch ein „z“

(zusammengesetzte Zahl).

Die Primzahlen sind dann die Indizes jener Felder, in denen „p“ steht.

Aufgabe 18 - Leichtathletik

Erstelle eine neue Datei unter dem Namen `uebung_mehrdim.php` mit einem mehrdimensionales indiziertes Feld mit folgenden Inhalten und gib die Daten anschließend in Tabellenform (mit Überschrift) auf dem Bildschirm aus:

Beginn	Disziplin	Ort	Bemerkung
09:30 Uhr	Diskuswurf	Nebenplatz	Jugendmeisterschaften
10:00 Uhr	5-km-Lauf	Stadion - Laufbahn	Offener Lauf
11:00 Uhr	Halbmarathon	Waldgebiet	Teilnahme ab 18 Jahren
12:00 Uhr	Stabhochsprung	Stadion - Stabhochsprunganlage	Nur Frauen

1.8.7) Fragen zu Feldern

- Erkläre die Begriffe „eindimensionales“ und „mehrdimensionales Feld“ und gib jeweils ein sinnvolles bzw. typisches Beispiel an.
- Erkläre die Begriffe „indiziertes“ und „assoziatives Feld“. Worin besteht der Unterschied.
- Wie kann man bei indizierten bzw. bei assoziativen Felder auf die Feldwerte zugreifen?
- Welche typischen Befehle für den Zugriff auf Feldern sind dir bekannt?

1.9.1) HIDDEN-Feld

Beim Formularfeldtyp **hidden** handelt es sich um ein verstecktes input-Feld, das für den Betrachter nicht sichtbar ist. Dennoch kann in ihm eine Variable gespeichert und auch übergeben werden. Genau darin besteht die Aufgabe eines versteckten Feldes, nämlich im Hintergrund verborgen Daten zu übergeben.

Datei einkauf.html

```
<html>
 <head>
  <title>Datenübergabe mittels HIDDEN-Feld</title>
 </head>
 <body>
  Heute kosten alle Artikel 5,99 Euro
  <form action="kaufen.php" method="POST">
     <input type="Hidden" name="preis" value="5.99">
     Artikel <input type="Text" name="artikel">
     Menge <input type="Text" name="menge">
     <input type="Submit" name="submit" value="Kaufen">
  </form>
  </body>
</html>
```

Datei kaufen.php

```
<html>
 <head>
  <title>Datenübergabe mittels HIDDEN-Feld</title>
 </head>
 <body>
  <?php
    echo "Sie haben den Artikel <b>".$_POST["artikel"] ."</b>".
$_POST["menge"]."
    mal zum Gesamtpreis von " . ($_POST["menge"] * $_POST["preis"]) . " Euro
gekauft.";
  ?>
 </body>
</html>
```

Hidden-Input-Felder stellen eine geeignete Möglichkeit dar, Daten über mehrere Webseiten hinweg zu transportieren. Schließlich muss es ausreichen, wenn der Benutzer beispielsweise seinen Vornamen nur auf der ersten Seite eines fünfseitigen Fragebogens eingibt und wir ihn trotzdem auf der fünften Seite noch immer freundlich beim Vornamen ansprechen können.

Wird die Eingabe auf der ersten Seite mittels eines Texteingabefeldes getätigt, so steht sein Vorname auf der zweiten Seite des Fragebogens in $_GET bzw. $_POST zur Verfügung. Diesen Wert können wir dank PHP in ein Hidden-Input-Feld übernehmen und transportieren den Wert damit sicher zur nächsten Seite, sobald der Benutzer die aktuelle Seite durch Aufruf des Absenden-Buttons verlässt.

Diese Kette kann, solange auf der nächsten Seite ein Formular folgt, beliebig lang sein.

Hier zur Verdeutlichung die entsprechenden Skriptbeispiele:

Erste Seite

```
Name : <input type="Text" name="name">
```

Nutzung des Hidden Input Feldes in den folgenden Seiten

```
<input type="Hidden" name="name" value="<?php echo $_POST["name"]; ?>">
```

Zu beachten ist, dass dieses Beispiel nur funktioniert, wenn keine Anführungszeichen im Namen vorkommen!

1.9.2) Textarea

Bei der Textarea handelt es sich um den großen Bruder des bereits mehrfach erwähnten Textfeldes. Die beiden Varianten unterscheiden sich nur dadurch, dass die Textarea aus mehreren Zeilen bestehen kann und somit für größere Eingaben besser geeignet ist, als ein einzeiliges Texteingabefeld. Im vorliegenden Beispiel wird beim Abschicken des Formulars die Variable `$bemerkung` mit dem Inhalt des Textareafeldes als Wert übergeben. Will man der Textarea einen Wert vorgeben, so muss man diesen Wert zwischen Anfangs- und Endtag des Textfeldes schreiben.

```
<textarea name="bemerkung"></textarea>
```

so sieht es aus:

1.9.3) Checkbox

Checkbox ohne Array und implode()

Wird ein Formular mit Checkboxen abgeschickt, werden in PHP nur die Einträge im Array $_GET bzw. $_POST erzeugt, deren Felder beim Ausfüllen ausgewählt werden. Wenn also nur das 2. und 3. Kästchen angeklickt wird, dann werden nur die Variablen `$_GET["artikel2"]` mit dem Wert Artikel 2 und `$_GET["artikel3"]` mit dem Wert Artikel 3 erzeugt.

Wenn eine oder mehrere Checkboxen standardmäßig den Zustand „Ausgewählt“ besitzen sollen, kann man innerhalb des input-Tags das Attribut checked verwenden.

```
<html>

 <head>
  <title>Checkbox</title>
 </head>
<body>

 <?php
    echo "<b>Sie haben folgende Artikel bestellt:</b><br>";
    echo $_GET["artikel1"] . "<br>";
    echo $_GET["artikel2"] . "<br>";
    echo $_GET["artikel3"] . "<br>";
    echo $_GET["artikel4"] . "<br>";
 ?>

<form action="check.php" method="get">
   <input type="Hidden" name="preis" value="5.99">
   <input type="Checkbox" name="artikel1" value="Artikel 1">Artikel 1<br>
   <input type="Checkbox" name="artikel2" value="Artikel 2">Artikel 2<br>
   <input type="Checkbox" name="artikel3" value="Artikel 3">Artikel 3<br>
   <input type="Checkbox" name="artikel4" value="Artikel 4">Artikel 4<br>
   <input type="Submit" name="submit" value="Kaufen">
</form>
</body>
</html>
```

so sieht es aus:

☐Artikel 1
☐Artikel 2
☐Artikel 3
☐Artikel 4
Kaufen

Checkbox mit Array und implode()

Checkbox lassen sich auch mittels einer Array - Anweisung erstellen. Wobei es bei der Ausgabe wichtig ist, nicht nur ein Element auszugeben, sondern es könnten möglicherweise alle Elemente, oder nur zwei sein.

Dazu dient die implode() - Anweisung

```
implode ( string $glue , arrayname )
Verbindet die Elemente eines Array mittels glue zu einer Zeichenkette
```

Nachdem er nur etwas im Array findet, wenn etwas angeklickt ist, sollte die Ausgabe nur erfolgen, wenn das Array nicht leer ist (eingebaute IF - Anweisung).

```
<html>
 <head>
  <title>Checkbox</title>
 </head>
<body>
<form action="check.php" method="POST">
   <input type="hidden" name="preis" value="5.99">
   <input type="checkbox" name="artikel[]" value="Artikel 1">Artikel 1<br>
   <input type="checkbox" name="artikel[]" value="Artikel 2">Artikel 2<br>
   <input type="checkbox" name="artikel[]" value="Artikel 3">Artikel 3<br>
   <input type="checkbox" name="artikel[]" value="Artikel 4">Artikel 4<br>
   <input type="submit" name="submit" value="Kaufen">
</form>

<?php
   if(isset($_POST["artikel"])){
    $var = $_POST["artikel"];
    $ausgabe = implode(", ", $var);
    echo "<b>Sie haben folgende Artikel bestellt: $ausgabe</b><br>"; }
 ?>
</body>
</html>
```

1.9.4) Radio-Button

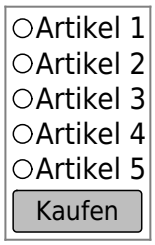
Radio-Buttons ohne Arrays erstellen

Während man bei Checkboxen immer beliebig viele Möglichkeiten auswählen kann, ist man bei Radio-Buttons auf nur eine einzige Antwortmöglichkeit beschränkt. Wie man sehen kann, haben bei Radio-Buttons im Gegensatz zu Checkboxen alle zusammengehörigen Auswahlmöglichkeiten denselben Namen. Daher kann auch nur ein Array-Eintrag erstellt werden, dem dann beim Abschicken der entsprechende Wert des angeklickten Feldes zugewiesen wird.

Ähnlich wie bei Checkboxen kann man hier für einen Radio-Button den Zustand „ausgewählt“ vorgeben, indem man im entsprechenden input-Tag das Attribut checked angibt. Zu beachten ist, dass die Angabe des Attributs nur dann sinnvoll ist, wenn sie sich auf genau einen Radio-Button beschränkt.

```
<html>
 <head>
  <title>Radio-Buttons</title>
 </head>
<body>
<?php
if (isset($_POST["submit"]))
  {
  echo "Sie haben Artikel <b>" . $_POST["artikel"] . "</b> bestellt!";
  }
 ?>
<form action="radiobutton.php" method="POST">
<input type="Hidden" name="preis" value="5.99">
<input type="Radio" name="artikel" value="1" checked>Artikel 1<br>
<input type="Radio" name="artikel" value="2">Artikel 2<br>
<input type="Radio" name="artikel" value="3">Artikel 3<br>
<input type="Radio" name="artikel" value="4">Artikel 4<br>
<input type="Radio" name="artikel" value="5">Artikel 5<br>
<input type="Submit" name="submit" value="Kaufen">
</form>
</body>
</html>
```

so sieht es aus:



Radio-Buttons mit Arrays erstellen

Radio Buttons lassen sich auch mittels einer Array - Anweisung erstellen. Wozu es einem PHP Code im HTML Teil benötigt.
Die Werte der RadioButtons werden in einem Array abgelegt und danch mittels forach ausgegeben. Dabei bekommt der Schüssel, der auch als value-Wert des Radio-Buttens dient, und der Ausgabewert denselben Inhalt. Nämlich jenen der im Array abgespeichert ist.

ACHTUNG!: Übermittelt wird nur der Indexwert, aber nicht der Inhalt des Arrays!

```
<html>
 <head>
  <title>Radio-Buttons</title>
 </head>
<body>
<?php
if (isset($_POST["submit"]))
  { $nummer=$_POST["artikel"]+1;
  echo "Sie haben Artikel <b>" . $nummer . "</b> bestellt!";
  }
?>
<form action="radiobutton.php" method="POST">
<input type="Hidden" name="preis" value="5.99">
<?php
$artikel = array ("Artikel1", "Artikel2", "Artikel3", "Artikel4",
"Artikel5");
 foreach($artikel as $key => $artikelnr){
 echo "<input type='radio' name='artikel' value='$key' />$artikelnr<br>"; }
 ?>
<input type="Submit" name="submit" value="Kaufen">
</form>
</body>
</html>
```

Für komplexere Ausgaben, lässt sich diese auch mit einer **switch-case Anweisung** lösen.

```
<?php
if (isset($_POST["submit"]))
  { echo "Sie haben ";
    switch($_POST["artikel"])
    {case "0":
    echo "Artikel1";
    break;
    case "1":
    echo "Artikel2";
    break;
    case "2":
    echo "Artikel3";
    break;
    case "3":
```

```
    echo "Artikel4";
    break;
    case "4":
    echo "Artikel5";
    break;
    }
    echo " bestellt";
 }
?>
```

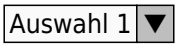
1.9.5) Auswahlfelder

Als Alternative zu den Radiobuttons stehen noch die Select-Auswahlfelder zur Verfügung:

Select-Auswahlfelder:

```
<select name="artikel">
 <option value="1">Auswahl 1</option>
 <option value="2">Auswahl 2</option>
 <option value="3">Auswahl 3</option>
 <option value="4">Auswahl 4</option>
 <option value="5">Auswahl 5</option>
</select>
```

so sieht es aus:



Hier wird analog zu den Radiobuttons nur ein Eintrag im Array $_POST erzeugt, dessen Name durch die Angabe im Element select vorgegeben ist. Der Wert des Eintrags ergibt sich aus der gewählten Option. Ähnlich wie bei einem Radio-Button kann man angeben, dass ein Eintrag einer Auswahlliste standardmäßig ausgewählt sein soll. Dazu muss man beim entsprechenden option-Tag das Attribut selected eintragen. Genau wie bei Radio-Buttons macht die Angabe natürlich nur Sinn, solange sie sich auf einen Eintrag der Liste beschränkt.

1.9.6) Aufgabe 20

Erstelle das nachstehende Formular für die Preisberechnung auf der Website eines Winzerbetriebs:

Weinbestellung

Hier können Sie den Preis für Ihre Weinbestellung einfach berechnen:
☐ Flaschen (1l) Grüner Veltliner Kabinett 2006 (€ 2,20)
☐ Flaschen (0,75 l) Chardonnay 2009 (€ 5,00)
☐ Flaschen (0,75 l) Muskateller 2009 (€ 5,50)
☐ Flaschen (0,75 l) Merlot 2008 (€ 4,50)
○Selbstabholung
○Lieferung innerhalb des Bezirks (zzgl. € 25,00)
Ab einem Gesamtbestellwert (inkl. eventueller Liefergebühr) von insg. 150€ wird ein 4%iger Rabatt gewährt.
☐Ja, ich möchte den original Winzer-Newsletter erhalten!

Die Ausgabe soll auf der selben Seite unterhalb des Submit-Buttons erfolgen und etwa wie folgt aussehen:

Sie haben

- 5 Flaschen Grüner Veltliner (€ 11,00)
- 10 Flaschen Chardonnay (€ 50,00)
- 15 Flaschen Merlot (€ 67,50)

bestellt und wollen die Ware geliefert (€ 25,00) bekommen. Da der Gesamtbestellwert (€ 153,50) mind. € 150,00 beträgt, erhalten Sie einen 4%igen Rabatt und zahlen daher insgesamt nur **€ 147,36**.

Ja, Sie werden in Zukunft den original Winzer-Newsletter erhalten.
Herzlichen Dank für Ihre Bestellung!

1.10.1) Nutzung von externen Dateien

Beispiel für die Nutzung von Daten aus externen Dateien

Wenn Sie Daten, die die Benutzer in ein Formular eingegeben haben, z. B. in einer Excel-Tabelle auswerten oder bearbeiten möchten, können Sie die Daten außerhalb des PHP-Programms in einer externen Datei speichern.

So können Sie beispielsweise auf diese Datei zugreifen, die Daten lesen, aber auch neue Daten hinzufügen bzw. die Daten ersetzen. Beispiele für die sinnvolle Nutzung von externen Dateien sind:

- Dynamische Daten in eine Webseite einbauen: Beispielsweise können Sie einen Hinweis auf das Angebot des Tages in einer externen Textdatei speichern. So brauchen Sie, um die Daten zu verändern, nur die externe Datei zu bearbeiten.
- Eingaben aus einem Formular speichern
- Logdateien schreiben: Um z. B. die Anzahl der Aufrufe einer Seite zu protokollieren, können Sie die Daten in einer eigenen Datei speichern.

Wichtig bei der Bearbeitung von externen Dateien ist die Art des Zugriffs auf die Datei. Hierbei kann PHP unterscheiden zwischen dem Zugriff auf sequenzielle und binäre Dateien.

Sequenzielle Dateien, z. B. Textdateien (`*.txt`) oder Dateien im Tabellenformat (`*.csv`), beinhalten Zeilen unterschiedlicher Länge. Sie werden sowohl beim Schreiben als auch beim Lesen sequenziell (also eine Zeile nach der anderen) bearbeitet.

1.10.2) Dateien öffnen, lesen und schließen

Dateien mit fopen() öffnen

Bevor Sie auf die Daten in Dateien zugreifen können, müssen Sie die entsprechende Datei öffnen. Dazu können Sie die `fopen()`-Anweisung verwenden.

Syntax und Bedeutung der fopen() -Anweisung

```
fopen(Dateiname,Modus);

$datei =  fopen("user.txt","r")
```

- Zusätzlich zur Angabe der zu öffnenden Datei wird der Modus, in dem die Datei geöffnet werden soll, angegeben:

- `r` ... Die Datei wird zum Lesen (r = read) geöffnet; Schreiben ist nicht erlaubt.
- `r+` ... Die Datei wird zum Lesen und Schreiben geöffnet, und der Dateizeiger an den Anfang der Datei gesetzt.
- w ... Öffnet eine Datei zum Schreiben (w = write)
- w+ ... Hier handelt es sich um dieselbe Option wie die Option w, nur wird hier die Datei zum Lesen und Schreiben geöffnet. Der Dateizeiger wird auf den Anfang der Datei gesetzt sowie die Länge der Datei auf 0 Byte.
- a ... Die Datei wird zum Schreiben geöffnet, wobei die neuen Daten die vorhandenen Daten ergänzen (a = append).
- a+ ... Die Datei wird zusätzlich zum Lesen geöffnet.
- b ... Zusätzlich kann der Option Modus der Buchstabe 'b' hinzugefügt werden, der die Behandlung von Binär-Dateien erlaubt.

Dies ist auf Systemen sinnvoll, wenn diese zwischen Binär- und Text-Dateien unterscheiden (wie z. B. Windows).

- Der Rückgabewert der Funktion `fopen()` ist ein Zeiger auf die geöffnete Datei und wird für weitere Zugriffe auf die Datei genutzt. Im Beispiel wird der Dateizeiger in der Variablen `$datei` gespeichert.

Wenn die angegebene Datei nicht gefunden wird, gibt die Funktion fopen Oden Wert FALSE zurück.

Dateien mit fgets() lesen

Mithilfe der Funktion `fgets()` können Sie den Inhalt einer Datei auslesen.

Syntax und Bedeutung der fgets() -Anweisung

```
fgets(Dateizeiger[,Modus]);

$zeile  = fgets($datei);
```

- Im ersten Parameter (Dateizeiger) finden Sie die Angabe zur Datei, aus der gelesen werden soll. Hierbei wird der Dateizeiger im Beispiel `$datei` aus der vorher geöffneten Datei verwendet.
- `fgets()` liest die angegebene Datei zeilenweise aus, wobei Sie optional im Parameter Modus die Länge angeben können. Lassen Sie diesen Parameter weg, ist automatisch eine Länge von maximal 1024 Zeichen für eine zu lesende Zeile festgelegt.
- `fgets()` liest bis zu dem ersten Auftreten eines Zeilenumbruchs bzw. der maximalen Zeilenlänge, dem Ende der Datei (EOF = End of File) oder bis zu der im Parameter Modus ausgegebenen Länge.
- Wenn Sie `fgets()` erneut aufrufen, wird die nächste Zeile gelesen.
- Der Rückgabewert der Funktion fgets() ist eine Zeichenkette. Im Beispiel wird der Rückgabewert in der Variablen `$zeile` gespeichert.

Dateien mit fclose() schließen

Nachdem Sie eine Datei geöffnet und die Daten ausgelesen haben, müssen Sie die Datei wieder schließen, um sie für andere Prozesse oder Benutzer nutzbar zu machen.

Syntax und Bedeutung der fclose()-Anweisung

```
fclose(Dateizeiger);

fclose($datei);
```

- Mit der Funktion `fclose()` schließen Sie die Datei, auf die der Dateizeiger weist. Der Dateizeiger muss gültig, also mit der Funktion `fopen()` geöffnet worden sein.
- Wurde die Datei erfolgreich geschlossen, liefert die Funktion `fclose()` den Wert TRUE zurück, sonst FALSE.

Beispiel zu externe Datei öffnen, lesen und schließen: fgets.php, user.txt

```
<html>
<body>
<?php
 if (file_exists("Gedicht.txt"))
     echo "Datei ist vorhanden!<br><hr>";

 $datei = fopen("Gedicht.txt","r");
 if ($datei)
 {
    echo "<p>Datei Gedicht.txt:</p>";
    while (!feof($datei))
    {
      $zeile = fgets($datei);
     echo $zeile."<br>";
    }
  fclose($datei);
 }
```

```
 else
  echo " Es trat ein Fehler auf! " ;
?>
</body>
</html>
</html>
```

1.10.3) Weitere Möglichkeiten zum Lesen von Dateien

Dateien mit readfile() oder file() lesen

Während die Funktion `fgets()` den Inhalt einer Datei zeilenweise einliest, können Sie mit den Befehlen `readfile()` und `file()` den gesamten Inhalt einer Datei auf einmal wiedergeben.

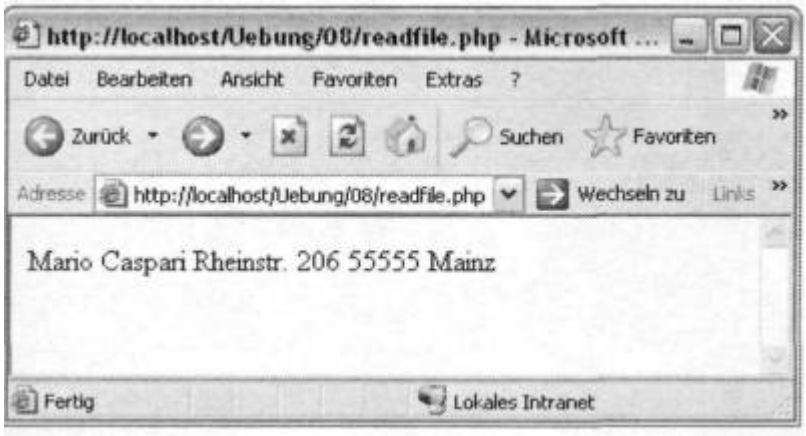
Der Unterschied zwischen den beiden Funktionen `readfile()` und `file()` liegt in der Ausgabe der Daten der Datei:

- `readfile()` liest vollständig den Inhalt der Datei aus und sendet das Ergebnis ohne weitere Bearbeitung direkt an den Browser.
- `file()` hingegen liest den vollständigen Inhalt der Datei zeilenweise in einen eindimensionales array. Hierbei wird jeweils der Inhalt einer Zeile zu einem Element des Feldes.

```
readfile(Datei);

file(Datei);
```

Beispiel readfile.php

```
<html>
<body>
 <h4>Auslesen einer Textdatei</h4>
 <?php readfile("user.txt"); ?>
</body>
</html>
```



Beispiel file.php

```
<html>
<body>
<h4>Auslesen einer Textdatei</h4>
<?php
 $feld = file("user.txt");
```

```
 echo "<p>";
 $i=1;
 foreach($feld as $zeile)
 {echo "Zeile ".$i++.":  ";
  echo $zeile."<br>";
 }
?>
</body>
</html>
```

Inhalt einer Datei in eine Zeichenkette einlesen

Sie können die gesamte Datei in einen String einlesen. Hierbei verbinden Sie die einzelnen ausgelesenen Zeilen mithilfe der Zeichenkettenfunktion `join()`.

Sie können mit der Funktion `join()` ein Feld anhand eines Trennzeichens zu einem String zusammenfügen. Dabei werden die Elemente des Feldes nacheinander an den String angehängt.

```
join(Trennzeichen,Feld);
```

Beispiel join.php

```
<?php
 if (file_exists("user.txt"))
 {$finhalt = join (";",file("user.txt"));
  echo $finhalt;
 }
?>
```

Beispieldatei „join.php“

- Mithilfe der Funktion `file_exists()` wird überprüft, ob die Datei existiert. Die Funktion `file_ exists()`erwartet als einzigen Parameter den Namen einer Datei auf dem Webserver. Dateien anderer Server (URL) können nicht auf ihre Existenz überprüft werden.
- Der Befehl `join()` verbindet die mithilfe der Funktion `file()` ausgelesenen Datei *user.txt* zu einer Zeichenkette. Als Trennzeichen wird hierbei das Semikolon „;“ verwendet.

1.10.4) In Dateien schreiben

Dateien zum Schreiben öffnen

Wollen Sie Daten in eine Datei schreiben, müssen Sie die Datei zuerst öffnen. Beim Öffnen der Datei können Sie bestimmen, ob Sie die bestehenden Daten der Datei überschreiben wollen oder der bestehenden Datei weitere Daten hinzufügen möchten.

(1) Dateien überschreiben

Wenn Sie eine Datei mit der Funktion `fopen()` und dem Modus 'w' für „write“ öffnen, wird die Datei zum Schreiben geöffnet und der Dateizeiger auf den Anfang der Datei verschoben. Gleichzeitig wird die Länge der Datei auf 0 Byte gesetzt. Wenn die Datei nicht existiert, wird sie angelegt.

(2) Daten in Dateien hinzufügen

Um beispielsweise Daten, die in ein Formular eingegeben wurden, fortlaufend in eine Datei zu schreiben, verwenden Sie den Befehl `fopen()` mit dem Modus „a“. Die Datei wird zum Schreiben geöffnet und der Dateizeiger an das Ende der Datei gesetzt, sodass die neuen Daten hinzugefügt werden. Existiert sie nicht, legt PHP sie an.

Daten in Dateien schreiben

Um eine Zeichenkette in eine geöffnete Datei zu schreiben, verwenden Sie die Funktion `fputs()`. Syntax und Bedeutung der fputs-Anweisung

```
fputs (Dateizeiger,Zeichenkette,[Länge])
```

- Mit der Funktion `fputs()` können Sie eine beliebige Zeichenkette in eine Datei schreiben.
- Die entsprechende Datei wird über den mit `fopen()` festgelegten Dateizeiger angesprochen.
- Ist der optionale Parameter Länge angegeben, wird das Schreiben nach der angegebenen Anzahl Bytes beendet. Geben Sie ihn nicht an, wird die gesamte Zeichenkette geschrieben.
- `fputs()` gibt bei Erfolg die Anzahl der geschriebenen Bytes zurück, andernfalls FALSE.

Auf Systemen, die zwischen Binär- und Textdateien (z. B. Windows) unterscheiden, muss die Datei mit der Option 'b' in der `fopen()`-Funktion geöffnet werden.

Nachdem Daten in die Datei geschrieben wurden, muss die Datei geschlossen werden.

Beispiel: Dateidaten ergänzen bestellformular.html, bestellung.php, bestellung_daten.csv

Die Daten, die der Benutzer in das Bestellformular für Äpfel eingibt, sollen in einer `*.csv`-Datei an das Ende der schon vorhandenen Daten gehängt werden.

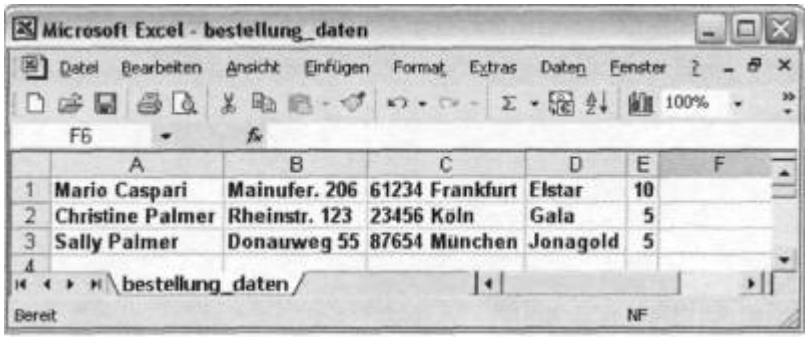

```
<html>
<body>
 <?php
 $datei = fopen("bestellung_daten.csv","a");
 if ($datei == false)
 { echo  "<p>Datei konnte nicht zum Schreiben geöffnet werden</p>";
   exit ;
 }
 $name    = $_POST["name"];
 $strasse = $_POST["strasse"];
 $ort     = $_POST["ort"];
 $sorte   = $_POST["sorte"];
 $menge      = $_POST["menge"];
 fputs ($datei, "$name;$strasse;$ort;$sorte;$menge;n");
 echo "<p>Folgende Angaben wurden gespeichert</p>";
 echo "$name<br>"; echo "$strasse<br>";
 echo "$ort<br>";  echo "$menge kg $sorte<br>";
 fclose($datei);
?>
</body>
</html>
```

Die Datei `bestellungdaten.csv` wird mit `fopen()` und dem Modus 'a' geöffnet, d.h. zum Schreiben geöffnet, und der Dateizeiger wird an das Ende der Datei gesetzt, sodass die neuen Daten hinzugefügt werden.

- Es wird geprüft, ob das Öffnen der Datei erfolgreich war.
- Wenn die Datei nicht erfolgreich geöffnet werden kann, wird mit exit das Programm gestoppt. Mit exit beenden Sie - ohne Möglichkeit der Rückkehr - die Skriptausführung.
- Die Variablen `$name, $strasse` etc. werden mit den Daten, die aus dem Formular übergeben wurden, gefüllt.
- Über die Funktion `fputs()` werden die Werte der Variablen in die geöffnete Datei geschrieben. Hierbei werden die Daten jeweils durch ein Semikolon voneinander getrennt, und am Ende einer

Zeile wird mit-hilfe von „\n“ ein Zeilenvorschub erzeugt, um die neuen Daten in eine eigene Zeile zu schreiben.
- Die geöffnete Datei wird geschlossen.

*.csv können Sie mit Programmen, wie z. B. Microsoft Excel, öffnen und bearbeiten.



1.10.5) Gästebuch

In der hier vorgestellten Variante eines Gästebuches werden nur der Name des Gastes und ein Text in der zugehörigen Textdatei gastbuch.txt gespeichert. Diese Textdatei muss Lese- und Schreibrechte für alle aufweisen.

- Die Funktion `file_exists()` testet, ob die genannte Datei im angegebenen Verzeichnis vorliegt.
- Die Funktion `feof()` testet, ob das Ende der Textdatei erreicht wurde. Der Ausdruck `!feof($fp)` liefert also so lange true, so lange das Dateiende nicht erreicht wurde.
- Die Funktion `fgets()` liest aus der mit der Ressource-ID festgelegten Datei einen so langen Datenstrom, wie das zweite Argument (im Beispiel 1000) angibt.
- Die Funktion `is_writeable()` testet, ob Schreibrechte für die angegebene Datei gewährt wurden.
- Die Funktion `fputs()` schreibt die angegebene Zeichenkette bzw. die Werte der angegebenen Variablen in die Datei, die mit der Ressource-ID identifiziert wurde.

Zusatz-Aufgabe

- Zusätzliche Informationen - etwa über Homepage, Datum, IP-Adresse, der Mitteilung, etc. sollen in das Gästebuch eingetragen werden. Informiere dich bezüglich des Einfügens des aktuellen Datums im Internet.

1.10.6) Beispiel: Besucherzähler

Eine häufige Anwendung in Verbindung mit Dateizugriffen ist die Einbettung eines Counters (Zugriffszählers) in eine Webseite. Ein Counter zählt die Anzahl der Webseiten-Zugriffe. Dabei wird bei jedem Zugriff die Zahl der bisherigen Besucher um eins erhöht. Das heißt, es wird auf eine bestehende Datei zugegriffen und mit ihrem Inhalt gerechnet. Das Ergebnis der Berechnung überschreibt dann den Inhalt der Datei.

Hinweise zum Beispiel:

- In den HTML-Text wird die Funktion `counter()` zur Anzeige der Besucherzahl eingebunden.
- Die Funktion `counter()` wird angelegt.
- Der Name der Datei, in der die Anzahl der Besucher abgelegt werden soll, lautet *counter.txt* und wird der Variablen `$name` zugewiesen. Die Variable `$count` erhält den Ausgangswert 0.
- Die Datei *counter.txt* wird über die Funktion fopen() zum Lesen und Schreiben geöffnet. Der Zugriff auf diese Datei erfolgt über den Dateizeiger `$datei`. Der Dateizeiger wird mit dem Modus `r+` an den Anfang der Datei gesetzt. War der Zugriff auf die Datei nicht erfolgreich, wird das Skript mit einer Meldung verlassen.
- Damit während der Arbeit mit der Datei kein weiterer Zugriff und somit ein gleichzeitiges Überschreiben der Daten möglich ist, wird die Datei exklusiv (LOCK_EX) gesperrt. LOCK_EX steht für eine exklusive, nur schreibende Verriegelung einer Datei (vgl. folgenden Abschnitt).
- Mit `fgets()` werden die ersten 10 Zeichen der Datei ausgelesen. Der Wert 10 wurde gewählt, da hiermit ein Zählen der Besucherzahl mit 10 Stellen (bis 9999999999) gewährleistet ist.
- Damit der neue Wert den alten Wert in der Datei überschreiben kann, wird der Dateizeiger mittels `fseek()` und der Angabe 0 an den Anfang der Datei gesetzt.
- Nach dem Einlesen des Wertes wird er um eins erhöht und am Bildschirm fett formatiert ausgegeben. Über die Funktion `fwrite()` wird der neue Wert `$count` in die Datei geschrieben.
- `fclose()` schließt die Datei *counter.txt* und beendet zugleich die Zugriffssperre, die über `flock()` veranlasst wurde.

Mit dem Befehl `flock()` können Sie eine Datei in der Zeit, in der das Skript in eine Datei schreibt, zusätzlich für weitere Zugriffe sperren.

Syntax und Bedeutung der flock()-Anweisung

```
flock (Dateizeiger,Modus);
```

- `flock()` sperrt den Zugriff auf die Datei, auf die der Parameter Dateizeiger verweist. Der Modus ermöglicht verschiedene Sperrzustände der Datei.
- Darf während Ihres Zugriffs die Datei von anderen nur gelesen werden, setzen Sie den Modus auf LOCK_SH (Shared Lock, Lesezugriff). Soll kein anderer Nutzer zeitgleich die Datei nutzen dürfen, setzen Sie die Option LOCK_EX (Exclusive Lock), `flock()` wartet, bis die Datei wie angegeben benutzt werden kann. Geben Sie locknb (No Block) an, kehrt die Funktion mit dem Wert FALSE zurück, wenn die Datei bereits von einem anderen Programm gesperrt ist. Möchten Sie die Verriegelung wieder freigeben, geben Sie LOCK_UN (Unlock) an.
- Die Funktion liefert bei Erfolg den Wert TRUE zurück bzw. FALSE, wenn ein Fehler auftrat. Ob die Zugriffssicherung funktioniert, erfahren Sie, indem Sie den Rückgabewert überprüfen.
- Die Sperre wird über die Funktion `fclose()` oder am Ende des Skripts automatisch aufgehoben.
- Bei manchen Betriebssystemen ist die Funktion `flock()` auf Prozess-Ebene (Teil einer Applikation) implementiert. Hierbei können Sie sich nicht auf `flock()` verlassen, um Dateien vor dem Zugriff von anderen PHP-Skripten zu schützen. Zudem wird `flock()` nicht von allen Dateisystemen unterstützt und gibt deshalb in solchen Umgebungen immer FALSE zurück.

Den Dateizeiger mit "fseek()" an eine bestimmte Position setzen

Im vorher gezeigten Beispiel ist es notwendig, den Dateizeiger an den Anfang der Datei zu setzen, um den alten Wert des Zählers mit dem neuen Wert überschreiben zu können. Den Dateizeiger können Sie über `fseek()` positionieren.

```
fseek (Dateizeiger,Stelle[,Wie]);
```

- Hierbei geben Sie für den Parameter Stelle die Anzahl in Bytes, bezogen auf den Dateianfang, der durch den Dateizeiger festgelegt wird, an.
- Der optionale Parameter `[,Wie]` legt bestimmte Bezugspunkte für die Ermittlung der Position fest:
  - Der Parameter SEEK_CUR ermöglicht die Verschiebung von der aktuellen Position des Dateizeigers plus Stelle, SEEK_END vom Dateiende und SEEK_SET vom Dateianfang.
- Fehlt der Parameter Wie, ist standardmäßig die Option SEEK_SET gewählt.

1.10.7) Übungsaufgaben

Aufgabe 23: Textdatei einlesen und zeilenweise ausgeben

Gedicht.txt

```
Wer will was Lebendig's erkennen und beschreiben,
Sucht erst den Geist heraus zu treiben,
Dann hat er die Teile in seiner Hand,
Fehlt leider! nur das geistige Band
```

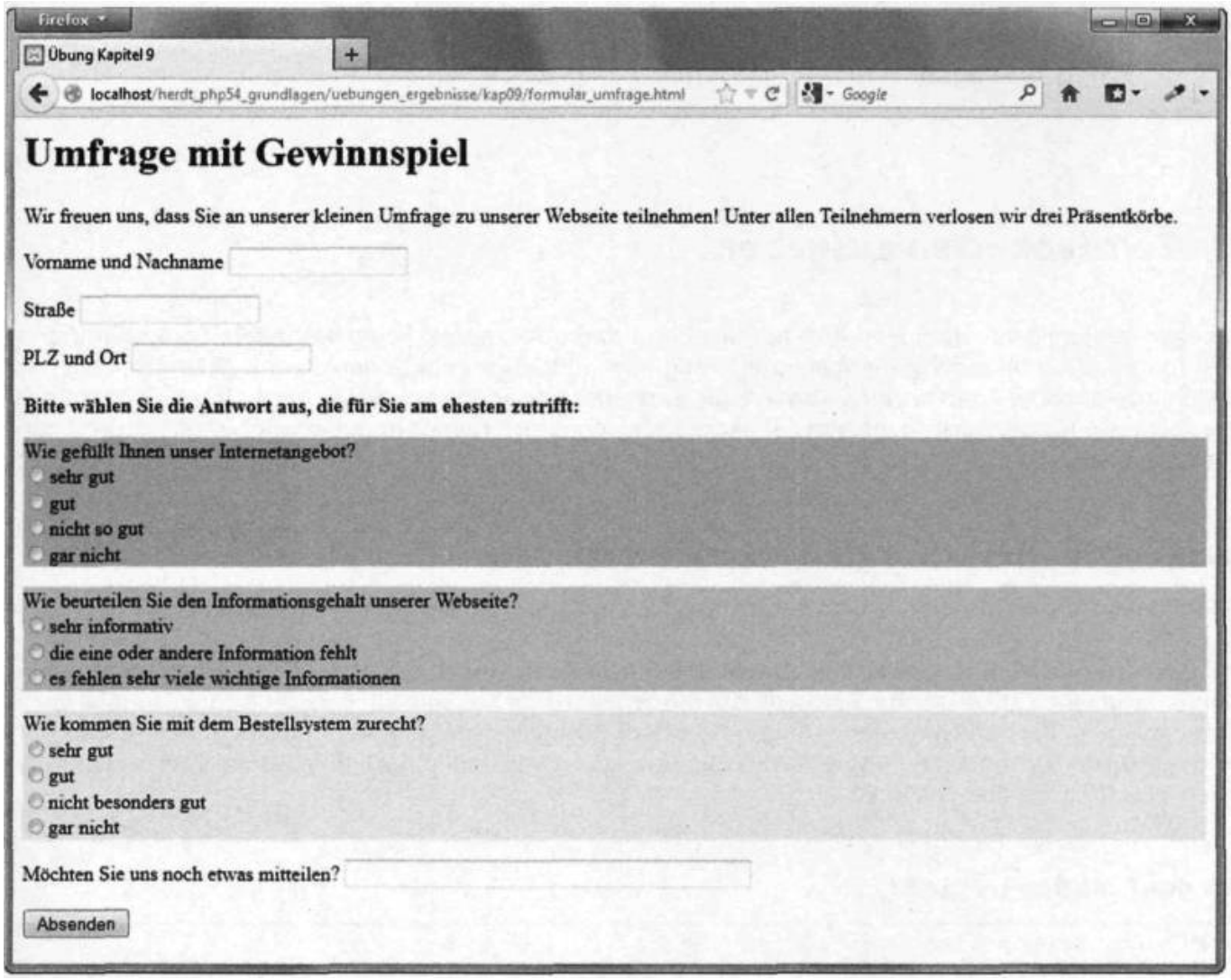
Aufgabe24: Apfel-Bestellungen aufgeben und csv-Datei ausgeben

bestellung_daten.csv

Aufgabe 25: Umfrage mit Gewinnspiel

a) Du möchtest eine Umfrage durchführen. Hierfür erstellst Du ein Formular in das Daten eingegeben werden. Um die Daten auswerten zu können, schreibe ein PHP-Skript, das die Daten an eine Tabelle (*.csv) übergibt und dort speichert.

Das Umfrageformular, das auch das PHP-Script beinhalten soll, speichere unter dem Namen `umfrage.php` und die übergebenen Daten werden in der Datei `umfrage_daten.csv` gespeichert. Hinweis: Die gewählten Optionen werden als Zahlen in die Datendatei geschrieben (1 für 1. Option, 2 für 2. Option, ...).



b) Baue (wie bei Aufgabe 2 aus der Schulübung) die Datei `umfrage.php` zur einem Verwaltungstool aus, dass es ermöglicht, neben der Eingabe von neuen Daten auch die Datei `umfrage_daten.csv` übersichtlich am Schirm auszugeben.

1.11.1) Datum und Zeit

getdate()

- `$datum = getdate()` liefert Informationen zur aktuellen Zeit, zum Wochentag, Monat und vielem mehr. Diese Funktion liefert das Ergebnis in einem assoziativen Feld zurück.
- Der Anweisung `getdate()` kann man den Parameter `Zeitstempel` hinzufügen, mit dem man das auszulesende Datum angibt. Der Parameter `Zeitstempel` erwartet als Wert die Sekundenanzahl seit dem 01.01.1970. Sämtliche Zeitangaben beruhen auf diesem Datum, das auch als Beginn der UNIX-Epoche bezeichnet wird.
- `print_r($datum)` veranschaulicht den Rückgabewert der Variablen `$datum`.

Übung 1

Versuche mittels `$datum = getdate()` folgende Ausgabe des aktuellen Datums durchzuführen:
Stunde: 12
Minute: 36
Sekunde: 20
Tag der Woche: 3 = Wednesday
Tag des Monats: 17
Tag des Jahres: 321
Monat: 11 = November
Jahr: 2010

Lösung

```
<?php
$datum=getdate();
echo "Sekunden ".$datum[seconds] . "<br>";
echo "Minuten ".$datum[minutes] . "<br>";
echo "Stunden ".$datum[hours] . "<br>";
echo "Tag des Monats ".$datum[mday] . "<br>";
echo "Tag der Woche ".$datum[wday] . "<br>";
echo "Monat ".$datum[mon] . "<br>";
echo "Jahr ".$datum[year] . "<br>";
echo "Tag des Jahres ".$datum[yday] . "<br>";
echo "Wochentag ".$datum[weekday] . "<br>";
echo "Monat ".$datum[month] . "<br><br>";
 ?>
```

Datum und Zeit formatieren

Die `date()`-Anweisung erwartet als Parameter die Formatanweisungen, welche Datum-Segmente angezeigt werden sollen.

Bsp: `echo date("l d")." of ". date("F Y h:i:s A");` liefert:
Wednesday 17 of November 2010 12:36:16 PM

Übung 2

Versuche mittels `date()` folgende Ausgabe zu erstellen:
17.11.10 um 12:36:16
17.11.10 = 321. Tag des Jahres 2010
17-11-2010
2010-11-17

Lösung

```
 <?php
echo date("d.m.y"). " um ".date("H:i:s")."<br>";
echo date("d.m.y"). " = ".date("z").". Tag des Jahres ".date("Y")."<br>";
echo date("d-m-Y")."<br>";
echo date("Y-m-d")."<br>";
 ?>
```

Mit date() kann man eine Zeitangabe formatieren oder auswerten. Die Zeitangabe übergeben Sie im Parameter timestamp. Lassen Sie diesen Parameter leer, nimmt die Funktion die aktuelle Zeit. Der Parameter format ist ein String, der festlegt, welche Informationen über die Zeitangabe Sie benötigen. In diesem String sind folgende Platzhalter möglich (*: Ausgabe mit führenden Nullen):

- a - „am“ oder „pm“
- A - „AM“ oder „PM“
- B - Tage bis Jahresende
- d - Tag des Monats *(01 – 31)
- D - Tag der Woche (Wed – 3stellig)
- F - Monatsangabe (December – ganzes Wort)
- g - Stunde im 12-Stunden-Format (1-12)
- G - Stunde im 24-Stunden-Format (0-23)
- h - Stunde im 12-Stunden-Format *(01-12)
- H - Stunde im 24-Stunden-Format *(00-23)
- i - Minuten *(00-59)
- I - (großes i) 1 bei Sommerzeit, 0 bei Winterzeit
- j - Tag des Monats (1-31)
- l - (kleines L) ausgeschriebener Wochentag (Monday)
- L - Schaltjahr = 1, kein Schaltjahr = 0
- m - Monat *(01-12)
- n - Monat (1-12)
- M - Monatsangabe (Feb – 3stellig)
- O - Zeitunterschied gegenüber Greenwich (GMT) in Stunden (z.B.: +0100)
- r - Formatiertes Datum (z.B.: Tue, 6 Jul 2004 22:58:15 +0200)
- s - Sekunden *(00 – 59)
- S - Englische Aufzählung (th für 2(second))
- t - Anzahl der Tage des Monats (28 – 31)
- T - Zeitzoneneinstellung des Rechners (z.B. CEST)

- U - Sekunden seit Beginn der UNIX-Epoche (1.1.1970)
- w - Wochentag (0(Sonntag) bis 6(Samstag))
- W - Wochennummer des Jahres (z.B.: 28)
- Y - Jahreszahl, vierstellig (2001)
- y - Jahreszahl, zweistellig (01)
- z - Tag des Jahres (z.B. 148 (entspricht 29.05.2001))
- Z - Offset der Zeitzone gegenüber GTM (-43200 – 43200) in Minuten

Länder- und Spracheinstellungen ändern

Mit folgendem Code wird die deutsche Sprache für alle Werte eingestellt.

```
<?php
  setlocale(LC_ALL, "german");
?>
```

strftime(Format [,Zeitstempel])

- Die Anweisung erwartet als Parameter die Formatanweisungen
- Ohne die Angabe des zweiten Parameters Zeitstempel liefert die Funktion das aktuelle Datum zurück analog der date()-Funktion.

Zeitfunktionen

time()

- Diese Funktion ist ohne Parameter zu verwenden und gibt die Zeit, die seit dem 1. 1. 1970 um 00:00:00 Uhr vergangen ist, in Sekunden zurück.

strtotime()

- Die Funktion liefert die Anzahl der Sekunden zwischen dem 1.1.1970 und dem angegebenen Datum zurück.

```
<?php
     echo strtotime("07.05.2019 10:00:00");
?>
```

mktime([Stunde[, Minute[, Sekunde[, Monat[, Tag[, Jahr);

- Die Funktion liefert die Anzahl der Sekunden zwischen dem 1. 1. 1970 und dem angegebenen Datum zurück.

```
<?php
  $tag   = 24;
```

```
  $monat = 8;
  $jahr  = 1978;

  $geburt = mktime(0,0,0,$monat,$tag,$jahr);
  $differenz = time() - $geburt;

  echo ((integer)($differenz/86400))." Tage liegen zwischen dem heutigen
Datum, ";
  echo date("j.n.Y")." und dem ".date("j.n.Y", $geburt).".";
?>
```

Datumsangaben überprüfen

checkdate(Monat, Tag, Jahr)

- Als Rückgabewert liefert die Funktion den Wert TRUE, falls das Datum existiert, sonst FALSE.

Zeit mittels microtime

microtime(true) liefert den aktuellen UNIX-Zeitstempel mit Mikrosekunden

```
$zeitstempel=microtime(true);
```

Beispiel:

```
<?php
$time_start = microtime(true);

// Sleep for a while
usleep(100);

$time_end = microtime(true);
$time = $time_end - $time_start;

echo "Did nothing in $time seconds\n";
?>
```

1.11.2) ÜBUNG 26 - GÄSTEBUCH ERWEITERUNG

Erweitere das Gästebuch aus der ÜBUNG 21 dahingehend, dass du nicht nur den Namen und den Text speicherst, sondern auch das Datum des Eintrages.

1.12.1) Funktionen erstellen und aufrufen

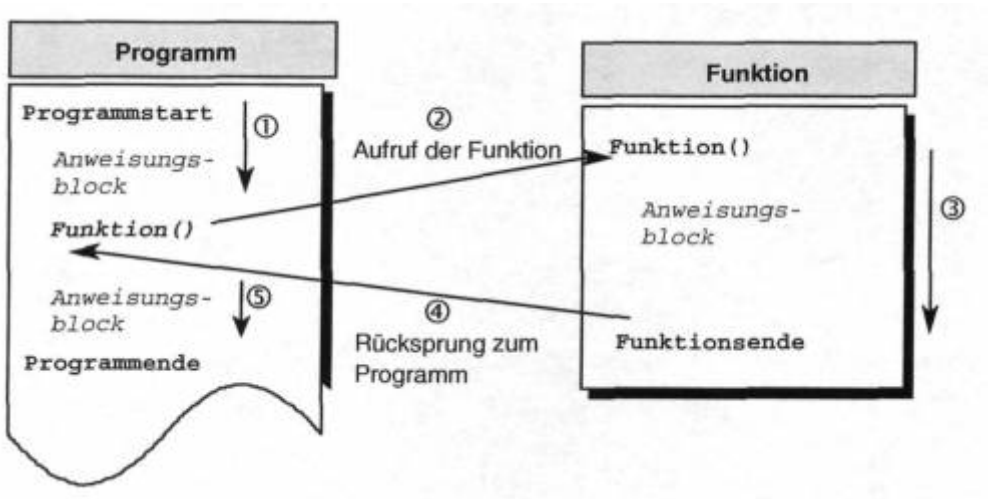
Was sind Funktionen?

Funktionen sind eigenständige Programmteile, die vom Skript beliebig oft aufgerufen und abgearbeitet werden können. Funktionen beinhalten Anweisungen, die innerhalb des Programms mehrmals benötigt werden. Anstatt die Anweisungen mehrfach im Programm zu codieren, wird die entsprechende Funktion aufgerufen, die die gewünschten Anweisungen durchführt.

PHP bietet eine Reihe vordefinierter Funktionen, um bestimmte Standardaufgaben zu lösen. Funktionen, die Sie selbst zur Lösung Ihrer Aufgaben erstellen, werden benutzerdefinierte Funktionen genannt.

Vorteile von Funktionen

- Immer wiederkehrende Abläufe werden nur einmal beschrieben und können danach beliebig oft ausgeführt werden.
- Der Programm-Code wird mithilfe von Funktionen strukturiert und ist dadurch einfacher zu pflegen.
- Änderungen am Programm, die zu einem späteren Zeitpunkt auftreten, lassen sich schneller durchführen, da sich eine Änderung in der Funktion auch auf den Quellcode auswirkt, der diese Funktion verwendet.
- Die Struktur des Programms lässt sich leichter nachvollziehen, da der Quellcode übersichtlicher ist.



<tab>

Eine Funktion wird erst ausgeführt, wenn sie im Programm aufgerufen wird. Dies geschieht über den Namen der Funktion.
Das PHP-Programm wird bis zum Aufruf einer Funktion abgearbeitet (1).
Der Funktionsaufruf (2) erzwingt einen Sprung in die angegebene Funktion.
Jetzt werden die Anweisungen der Funktion abgearbeitet (3).
Mit dem Verlassen der Funktion wird zurück zum Programm gesprungen (4).
Das Programm wird weiter ausgeführt (5).

Eine Funktion erstellen

Syntax und Beschreibung der `function`-Anweisung

```
function Name([Parameter])
{Anweisungsblock;
}
```

- Die function-Anweisung leitet eine Funktion ein.
- Name ist die Bezeichnung der Funktion und sollte einen Bezug zum Inhalt haben, z. B. können Sie eine Funktion, die das Quadrat einer Zahl berechnet, *BerechneQuadratzahl* oder *Quadratzahl* nennen.
- Für den Namen einer Funktion gelten folgende Regeln:
  - Das erste Zeichen muss entweder ein Buchstabe oder ein Unterstrich sein.
  - Der Name darf nur aus Buchstaben und Ziffern bestehen, aber keine Umlaute oder „ß“ und keine Sonderzeichen außer dem Unterstrich „_“ enthalten.
  - Der Name kann Groß- oder Kleinbuchstaben enthalten, wobei zwischen Groß- und Kleinschreibung unterschieden wird: `function Quadratzahl()` ist nicht gleich `function quadratzahl()`.
  - Der Name darf nicht identisch sein mit einem so genannten reservierten Wort (z. B. Befehl aus PHP).
- In den runden Klammern werden die Bezeichnungen der einzelnen Parameter angegeben, für die beim Aufruf der Funktion Werte übergeben werden können.
- Eine Funktion kann keinen, einen oder mehrere Parameter besitzen.

Eine Funktion mit retum-Anweisung erstellen

Syntax und Beschreibung der `return`-Anweisung

- Mit der `return`-Anweisung kann ein Wert zurückgeliefert werden. Sobald die `return`-Anweisung ausgeführt wird, wird eine Funktion verlassen und an die aufrufende Stelle zurückgekehrt.
- Die `return`-Anweisung kann ohne Rückgabewert auch ausschließlich zum Verlassen einer Funktion verwendet werden.

```
function Name([Parameter])
{Anweisungsblock;
 return [$Wert];
}
```

Eine Funktion aufrufen

Syntax und Beschreibung eines Funktionsaufrufs

- Eine Funktion können Sie aus einer beliebigen Stelle des PHP-Codes heraus aufrufen.
- Sie können eine Funktion auch aus einer anderen Funktion heraus aufrufen.
- Eine Funktion wird mit ihrem Namen aufgerufen.

- Die runden Klammern nach dem Funktionsnamen können, falls benötigt, Parameter zur Übergabe von Werten enthalten. Falls keine Werte übergeben werden, bleiben die Klammern leer.

```
<?php
  Anweisungsblock

  Funktionsname();

  Anweisungsblock
?>
```

1.12.2) Mit Funktionen arbeiten

Je nach Verwendungszweck einer Funktion kann eine Funktion mit oder ohne Parameter bzw. mit oder ohne Rückgabewert erforderlich sein.

Funktionen ohne Parameter

Bei einer Funktion ohne Parameter werden bei jedem Aufruf dieselben Anweisungen ausgeführt.

Beispiel: funktionen1.php

```php
<?php

 function Text()
 {
    echo "<p>Hallo Welt!</p>";
 }

 function Linie()
 {
    echo "<p><hr></p>";
 }

 Text();
 Linie();
 Text();
 Linie();

?>
```

Funktionen mit einem oder mehreren Parametern

Bei einer Funktion mit einem oder mehreren Parametern werden bei jedem Aufruf der Funktion in Abhängigkeit von den Parametern ähnliche Aufgaben ausgeführt. Hierbei kann ein Parameter eine Konstante oder eine Variable sein.
Wenn Sie eine Funktion mit mehreren Parametern aufrufen, müssen die Anzahl und die Reihenfolge der Parameter übereinstimmen.

Beispiel: funktionen2.php

Im folgenden Skript werden die Preise pro kg mithilfe einer Funktion mit einem Parameter berechnet.

```
<?php

 function Preis($menge)
 {
    $betrag = 1.2*$menge;
    echo "Der Preis von $menge kg ist: $betrag<br>";
 }

 Preis(2);
 Preis(5);
 Preis(10);
 $einkaufsmenge=20;
 Preis($einkaufsmenge);
?>
```

Die Funktion `Preis()` wird dreimal aufgerufen. Der übergebene Parameter (der Wert in der Klammer) ist bei jedem Aufruf ein anderer. Der Wert in der Klammer wird der Variablen `$menge` übergeben. Mithilfe der Variablen `$menge` wird beim Aufruf der Funktion die Variable `$betrag` berechnet und am Bildschirm ausgegeben.

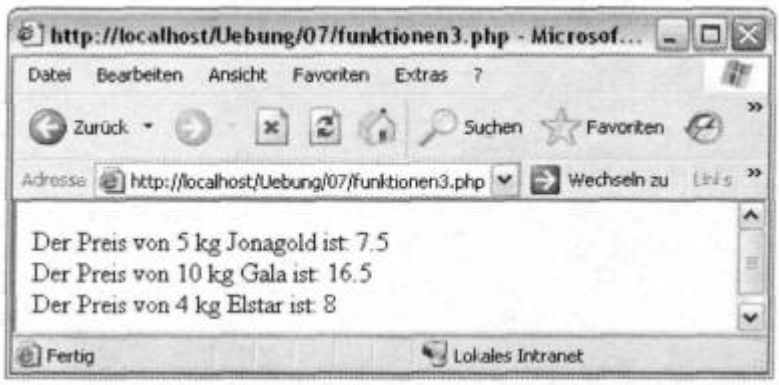
Beispiel: funktionen3.php

Wird eine Funktion mit mehreren Parametern aufgerufen, wird der erste Wert an den ersten Parameter, der zweite Wert an den zweiten Parameter usw. übergeben.

```
<?php
```

```
function Preis($sorte,$preis,$menge)
{
   $betrag = $preis * $menge;
   echo "Der Preis von $menge kg $sorte ist: $betrag<br>";
}

Preis(Jonagold,1.5,5);
Preis(Gala,1.65,10);
Preis(Elstar,2.00,4);

?>
```

Die Funktion `Preis()` wurde mit 3 Parametern versehen: `$sorte, $preis, $menge`.
Beim Aufruf der Funktion werden durch Kommata getrennt drei Parameter übergeben.



Veränderung von Parametern innerhalb einer Funktion

Wenn Sie eine Variable als Parameter an eine Funktion übergeben, dann erhält die Funktion eine Kopie der übergebenen Variablen. Die Übergabe der Parameter als Kopie wird auch als **call-by-value** bezeichnet. Um den Wert einer übergebenen Variablen zu ändern, verändern Sie nur ihre lokale Kopie. Eine Veränderung der Kopie hat keinerlei Rückwirkung auf das Original.

Manchmal ist aber eine Veränderung der übergebenen Variablen erwünscht, beispielsweise um den Preis einer Bestellung abhängig zu machen von der vorgegebenen Menge. Um dies zu erreichen, übergeben Sie den Parameter als Referenz auf das Original. Sie übergeben also nicht, wie bei dem call-by-value-Verfahren, den Inhalt bzw. den Wert einer Variablen, sondern einen Verweis auf die Variable. Diese Vorgehensweise wird **call-by-reference** genannt. Eine Veränderung hat, anders als bei der Übergabe als Kopie, Rückwirkung auf das Original. Eine Referenz auf das Original erhalten Sie, indem Sie vor den Variablennamen ein & setzen, z. B. `&$preis`.

Beispiel: byreference.php

```
<?php

 function Preis($sorte, &$preis, $menge)
 {if ($menge > 10)
  {$preis = 1;
  }
  $betrag = $preis * $menge;
  echo "Der Preis von $menge kg $sorte ist: $betrag EUR <br>";
```

```
 }

 $sorte = Elstar;
 $preis = 2;
 $menge = 8;
 Preis($sorte, &$preis, $menge);

 $sorte = Elstar;
 $preis = 2;
 $menge = 15;
 Preis($sorte, &$preis, $menge);
?>
```

- Die Funktion Preis() wurde mit 3 Parametern versehen: `$sorte,&$preis,$menge`. Die Variable `$preis` wird mithilfe des vorangestellten & als Referenz gekennzeichnet.
- Mithilfe einer `if`-Schleife wird festgelegt, dass der Wert der Variablen `$preis = 1`, wenn `$menge > 10` ist. Beim Aufruf der Funktion werden drei Parameter durch Kommata getrennt übergeben. Da der Wert der Variablen `$menge < = 10` ist, wird zur Preisberechnung der ursprüngliche Wert der Variablen `$preis = 2` verwendet. Wenn aber der Wert der Variablen `$menge > 10` ist, wird der Wert der Variablen `$preis` aufgrund der Bedingung in der `if`-Schleife auf 1 verändert.

Funktionen mit Rückgabewerten

Wenn Sie innerhalb einer Funktion ein Ergebnis ermitteln und dies an den Aufrufer zurückliefern möchten, verwenden Sie die `return`-Anweisung.
Allerdings ist hier zu beachten, dass eine return-Anweisung nicht nur den Rückgabewert festlegt, sondern auch die Ausführung der Funktion sofort beendet. Alle Anweisungen nach einer `return`-Anweisung werden nicht mehr ausgeführt, und der vorhandene Rückgabewert wird an das Programm ausgegeben.

Beispiel: funktionen4.php

```
<?php
 function Summe ($zahll, $zahl2)
 {
     $ergebnis = $zahll + $zahl2;
     return $ergebnis;
 }

 $summel = Summe(10,20);
 echo "Summe1: $summel<br>";
 $zahl3 = 0.11;
 $summe2 = Summe (19,$zahl3);
 echo "Summe2: $summe2<br>";
?>
```

- Die Funktion `Summe()` hat die beiden Parameter `$zahl1` und `$zahl2`.
- Die beiden Parameter werden innerhalb der Funktion addiert und in der Variablen `$ergebnis` gespeichert.

- Der Wert der Variablen `$ergebnis` wird mithilfe der Anweisung return an die aufrufende Stelle zurückgeliefert.
- Der Funktionsaufruf `Summe(l0,20)` liefert in `$ergebnis` den Rückgabewert zurück. Dieser Wert wird der Variablen `$summe` zugewiesen.
- Parameter für Funktionen können sowohl als konstante Werte, als auch über Variablen übergeben werden.

PHP kann nur einen Wert mit `return()` zurückgeben. Wenn Sie mehrer Werte aus einer Funktion zurückgeben möchten, können Sie diese Werte in einem Array zusammenfassen.

Um Variablen die Elemente des zurückgelieferten Arrays zuzuweisen, verwenden Sie die `list()`-Anweisung.

```php
<?php

function zahlen()
{
   $ret = array(2,4,6);
   return $ret;
}

list($var1,$var2,$var3) = zahlen();
echo "Die zurückgelieferten Variablen sind: $var1, $var2, $var3";
?>
```

1.12.3) Der Gültigkeitsbereich von Variablen

Auf Variablen, die Sie außerhalb einer Funktion im PHP-Programm einsetzen, können Sie in Funktionen nicht ohne weiteres zugreifen und mit ihnen arbeiten. Ebenso gibt es innerhalb einer Funktion Variablen, auf die Sie außerhalb der Funktion nicht zugreifen können.

Variablen haben, je nachdem, wo sie definiert sind, unterschiedliche Gültigkeitsbereiche.

- **Lokale Variablen** sind nur innerhalb der Funktion, in der sie definiert wurden, gültig.
- **Globale Variablen**, die außerhalb von Funktionen definiert wurden, haben standardmäßig nur außerhalb der Funktionen ihren Gültigkeitsbereich. Die globalen Variablen stehen innerhalb einer Funktion nicht zur Verfügung. Sollen die globalen Variablen innerhalb einer Funktion gültig sein, müssen sie dort mit dem Schlüsselwort `global` bekannt gemacht werden, z. B. `global $zahl`.
- **Superglobale Variablen** stehen sowohl innerhalb als auch außerhalb von Funktionen zur Verfügung, beispielsweise das assoziative Feld `$_POST`.

Im folgenden Beispiel werden globale und superglobale Variablen gegenübergestellt. Hierzu wird ein Bestell-Formular für Äpfel erstellt. Anhand des PHP-Codes im Auswertungsprogramm wird gezeigt, wann welche Variablen gültig sind.

Beispiel: formular_funktion.html, funktion_var.php

```
<html>
 <body>
  <h2>Apfelkauf</h2>
  Bitte geben Sie die gewünschte Menge ein und wählen Sie eine Apfelsorte:
  <form action="funktion_var.php" method="post">
   Menge: <input name="menge"><br><br>
   Apfelsorte:<br>
   <input type="radio"  name="sorte" value="Jonagold">Jonagold
   <input type="radio"  name="sorte" value="Gala">Gala
   <input type="radio"  name="sorte" value="Elstar">Elstar<br><br>
   <input type="Submit" value="Abschicken">
   <input type="Reset"  value="Zurücksetzen">
  </form>
 </body>
</html>
```

- Mithilfe der Methode POST werden nach Absendung des Formulars folgende Variablen an das Auswertungsprogramm `funktion_var.php` weitergegeben:
- die Variable `$menge`, die vom Benutzer eingetragen wird, und
- die Variable `$sorte`, die der Benutzer mithilfe eines Optionsfeldes auswählt


```php
<?php
 $bestellnummer = "TT1911";
 $bearbeiter = "Mario Caspari";

 function bestellung()
 {
  global $bestellnummer;      // globale Variable
  echo "Bestellnummer: " .$bestellnummer ."<br>";
  echo "Bearbeiter: " .$bearbeiter ."<br>";
  echo "Die von Ihnen eingegebene Menge: ".$_POST["menge"]." kg<br>";
  switch($_POST["sorte"])
  {
     case "Jonagold": $preis=$_POST["menge"]*1.50;  // lokale Variable
                      break;
     case "Gala":     $preis=$_POST["menge"]*1.65;  // lokale Variable
                      break;
     case "Elstar":   $preis=$_POST["menge"]*2.00;  // lokale Variable
                      break;
  }
  echo "Ausgabe des Preises innerhalb der Funktion: $preis<br>";
 }

 echo "Die von Ihnen gewählte Sorte: ".$_POST["sorte"]."<br>";
 bestellung() ;
 echo "Ausgabe des Preises außerhalb der Funktion: $preis<br>";
?>
```

- Die Variablen `$bestellnummer` und `$bearbeiter` werden als **globale** Variablen im Programm definiert.
- Die **globale** Variable `$bestellnummer` wird innerhalb der Funktion `bestellung()` mithilfe des Schlüsselwortes `global` bekannt gemacht.
- Die Variablen `$bestellnummer` und `$bearbeiter` sollen innerhalb der Funktion ausgegeben werden. Da die Variable `$bearbeiter` innerhalb der Funktion nicht mithilfe des Schlüsselwortes `global` bekannt gemacht wurde, wird sie bei der Ausgabe nicht angezeigt.

- Der Wert des **superglobalen** Feldes `$_POST["menge"]` wird innerhalb der Funktion `bestellung()` ausgegeben.
- Innerhalb der Funktion `bestellung()` wird der Wert der **lokalen** Variablen `$preis` berechnet.
- Die **lokale** Variable `''$preis` wird innerhalb der Funktion ausgegeben.
- Der Wert des **superglobalen** Feldes `$_POST ["sorte"]` wird außerhalb der Funktion `bestellung()` ausgegeben.
- Die Funktion `bestellung()` wird aufgerufen.
- Die **lokale** Variable `$preis` soll auch außerhalb der Funktion ausgegeben werden.

1.12.4) PHP-Dateien einbinden mit include und require

Manchmal ist es sinnvoll, wenn Sie selbst definierte Funktionen in eine separate PHP-Datei auslagern, um sie in mehreren Skripten nutzen zu können.

Mithilfe der `include`- bzw. `require`-Anweisung können Sie Dateien, z.B. Funktionen, die Sie benötigen, in Ihr PHP-Programm einbinden.

Mit include und require arbeiten

Die `include()`- bzw. die `require()`-Anweisung binden eine bestimmte Datei, deren Pfad als Argument übergeben wird, in den aktuellen Programmcode ein.

Unterschiede zwischen include und require

Der Unterschied zwischen den beiden Anweisungen zeigt sich, wenn eine Datei fehlerhaft eingebunden wurde:

- Wenn beispielsweise die angegebene Datei nicht vorhanden ist, erzeugt `include()` eine Warnung. Das Skript wird weiter ausgeführt.
- Der Befehl `require()` beendet bei einer fehlenden Datei das Skript sofort mit einer Fehlermeldung.

Syntax und Bedeutung der include()- und require()-Anweisungen

- Die include- und require-Befehle erwarten als Parameter die einzubindende Datei. Befindet sich die Datei nicht im selben Ordner, muss der Pfad angegeben werden.
- Beinhaltet die eingebundene Datei keinen PHP-Code, wird der Inhalt unverändert an den Browser weitergegeben und somit am Bildschirm dargestellt.
- Ist in der Konfigurationsdatei `php.ini` im Bereich `fopen wrappers` die Option `allow_url_fopen = on` gesetzt, so können Sie auch einen URL als Parameter angeben.
- Eingebundene Dateien, die eine Funktion mit Rückgabewert enthalten, werden ebenso behandelt wie Funktionen mit Rückgabewert.

So kann z. B. der Rückgabewert in einer Variablen gespeichert werden mit: `$variable = include(Datei)` .

Innerhalb der eingebundenen Dateien muss der PHP-Code von gültigen PHP-Tags `<?php … ?>` umschlossen sein. Befindet sich kein PHP-Code in den Dateien, können die Tags weggelassen werden.

Um Skripte, die später in einen PHP-Code eingebunden werden sollen, zu kennzeichnen, ist es empfehlenswert, die Dateierweiterung `.inc.php` zu vergeben. Die Bezeichnung `.inc.php` steht hierbei für die Abkürzung von include.

Datei mit include() einbinden

Beispiel: include.php

Zur Veranschaulichung der Arbeitsweise der include()-Anweisung erstellen Sie ein Loginverfahren. Die Prüfung, ob der eingegebene Nickname und das Kennwort korrekt sind, wird in die Datei `prueflogin.inc` codiert. Vom Loginformular `formular_login.html` wird die Datei `include.php` aufgerufen.

Datei: include.php

```
<html>
<body>
 <h4> Loginergebnis über include </h4>
 <?php
  echo "Start<br>Nun wird ein PHP-Skript eingebunden:<hr>";
  echo "Die Rückgabe der Datei ergibt:<br>";
  include ("prueflogin.inc");
  echo "<hr>Hier können weitere Anweisungen der Datei include.php ausgeführt
  werden.";
 ?>
</body>
</html>
```

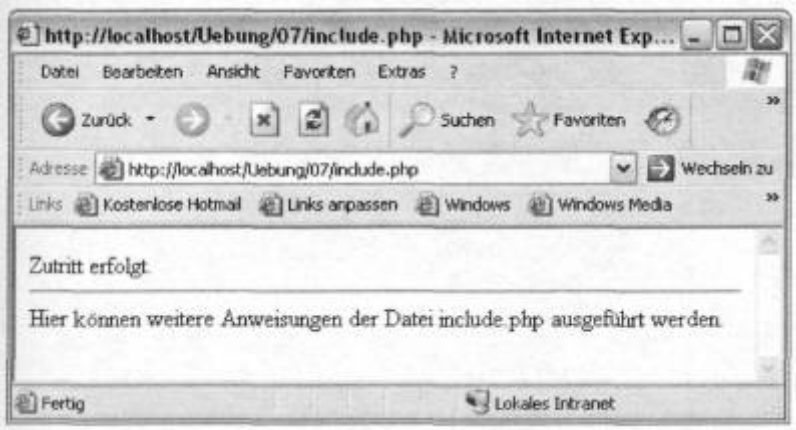
Die `include()`-Anweisung bindet die Datei `prueflogin.inc` ein.

Datei: prueflogin.inc

Innerhalb der Datei prueflogin.inc wird die Kennwortprüfung vorgenommen.

```
<?php
if ($_POST["nickname"] == "Mario" && $_POST["kennwort"] == "geheim")
  echo "Zutritt erfolgt";
else
  echo "Zutritt erfolgt nicht";
?>
```

<tab>



1.12.5) Andere Dateitypen einbinden

Sie können auch andere Dateien, z. B. reine Text- oder HTML-Dateien, in Ihr Programm einbinden.

Innerhalb eines PHP-Skripts soll beispielsweise eine einfache Textdatei eingebunden werden. Der Inhalt der Textdatei wird sofort am Bildschirm ausgegeben. Zuvor wird eine Textdatei mit folgendem Inhalt gespeichert.

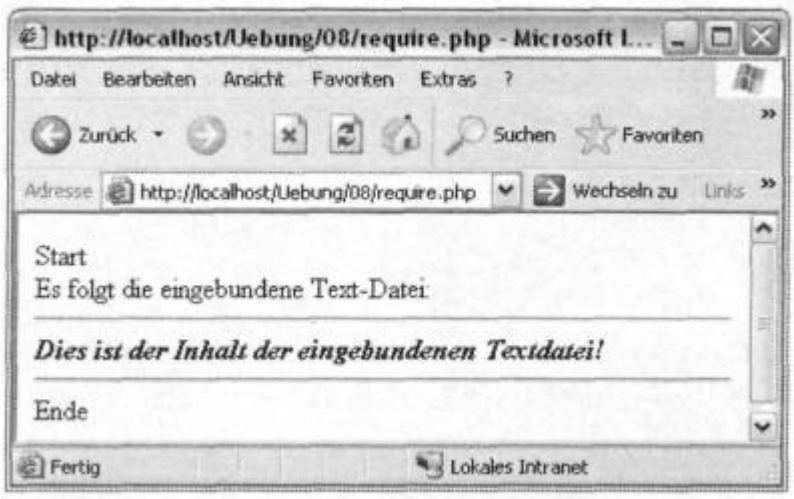
Datei: einfueg.txt

```
<b><i>Dies ist der Inhalt der eingebundenen Textdatei!</i></b>
```

Beispiel: require.php

```
<html>
<body>
 <h4>Eine Textdatei über den Befehl require einbinden</h4>
 <?php
  echo "Start<br>Es folgt die eingebundene Text-Datei:<hr>";
  require("einfueg.txt") ;
  echo "<hr>Ende";
 ?>
</body>
</html>
```

Inmitten des PHP-Codes wird über den Befehl require () eine Textdatei eingebunden. Da diese keinen weiteren PHP-Code, sondern nur Text mit HTML-Befehlen beinhaltet, wird diese von PHP direkt an den Browser weitergeleitet.



1.12.6) Übungen zu Funktionen

Aufgabe 27 - Maximum

Erstelle eine PHP-Seite (mit einer Funktion) welches folgende Aufgabe realisiert:

```
1. Zahl: -Benutzereingabe-
2. Zahl: -Benutzereingabe-
größte Zahl: -gibt die größere der beiden Zahlen aus
```

Aufgabe 28 - Bruch

Schreibe eine PHP-Seite, in der ein Bruch soweit als möglich gekürzt wird. Dazu gibt der Benutzer einen Zähler und Nenner ein. Mit Hilfe einer Funktion, die den ggT der beiden Zahlen mit dem euklidischen Algorithmus berechnet, soll der Bruch gekürzt und wieder ausgegeben werden.

Aufgabe 29 - Quader

Schreibe eine PHP-Seite, in der alle möglichen Berechnungen für einen Quader durchgeführt werden.

- Lese dazu vom Benutzer die Länge, Breite und Höhe ein.
- Überprüfe ob alle Werte eingegeben wurden
- Überprüfe mit is_numeric() ob die eingegebenen Werte Zahlen sind
- Berechne in einer Funktion die Grundfläche
- Berechne in einer Funktion das Volumen
- Berechne in einer Funktion die Oberfläche
- Berechne in einer Funktion die Raumdiagonale
- Speichere bei jedem Programmaufruf, die Ergebnisse wie folgt in eine CSV-Datei.

Bspielhafte CSV-Datei:

```
Laenge;Breite;Hoehe;Grundflaeche;Volumen;Oberflaeche;Raumdiagonale
3;5;3;15;45;78;6.557;
3;2;2;6;12;32;4.123;
```

- Gib am Ende jeweils die ganze CSV-Datei als Tabelle aus!

Laenge	Breite	Hoehe	Grundflaeche	Volumen	Oberflaeche	Raumdiagonale
3	5	3	15	45	78	6.577
3	2	2	6	12	32	4.123
...						

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

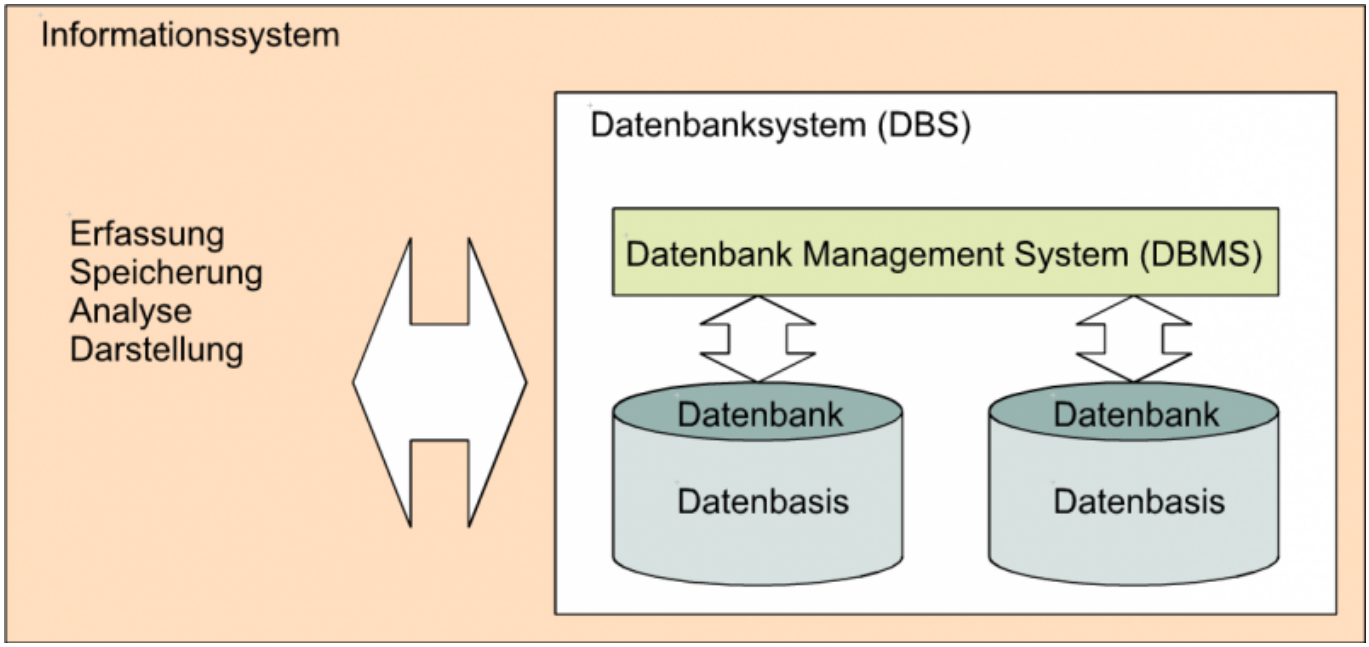
Fakturierung
Kunden
Teile

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.

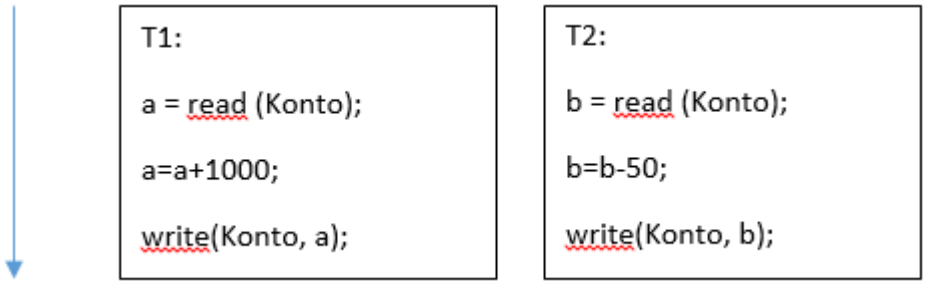
Bsp. zu Dauerhaftigkeit:
Angenommen wir zahlen einen Millionengewinn im Lotto auf unser Konto ein, dann stellt die Dauerhaftigkeit von Transaktionen sicher, dass der Gewinn auch nach einem Systemneustart immer noch auf dem Konto liegt. Für den Wiederanlauf verwenden die meisten DBS ein Log-Protokoll.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

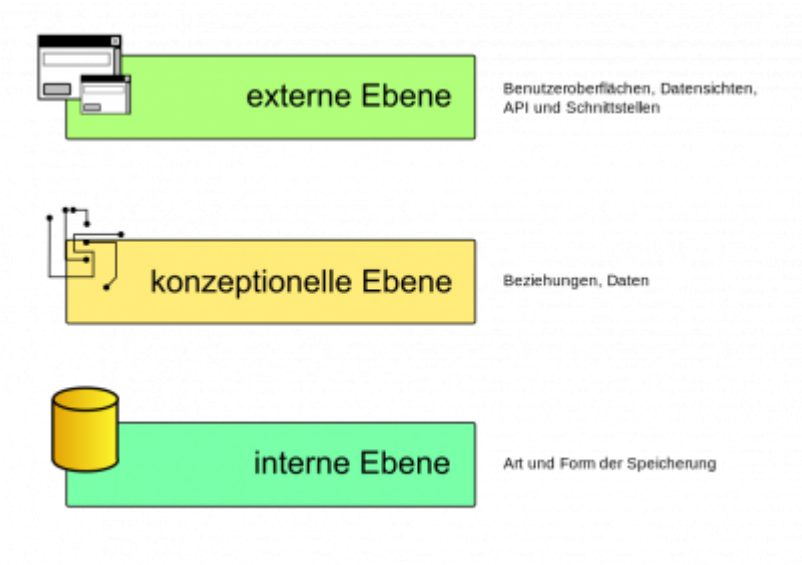
- **Consistency:** Nach Abschluss der Transaktion liegt wieder ein konsistenter Zustand vor, während der Transaktion sind inkonsistente Zustände erlaubt
- **Isolation:** Nebenläufig ausgeführte Transaktionen dürfen sich nicht beeinflussen, d.h. jede Transaktion hat den Effekt, den sie verursacht hätte, 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).

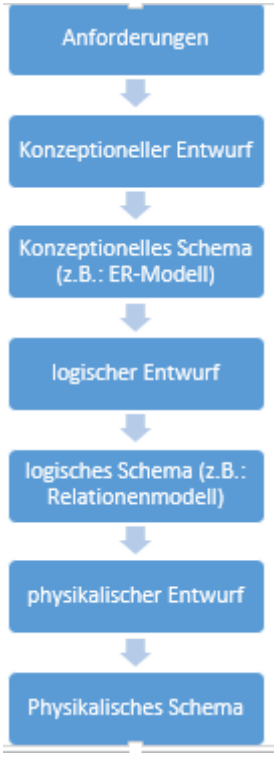
Die Vorteile des Drei-Ebenen-Modells liegen in der

- **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:



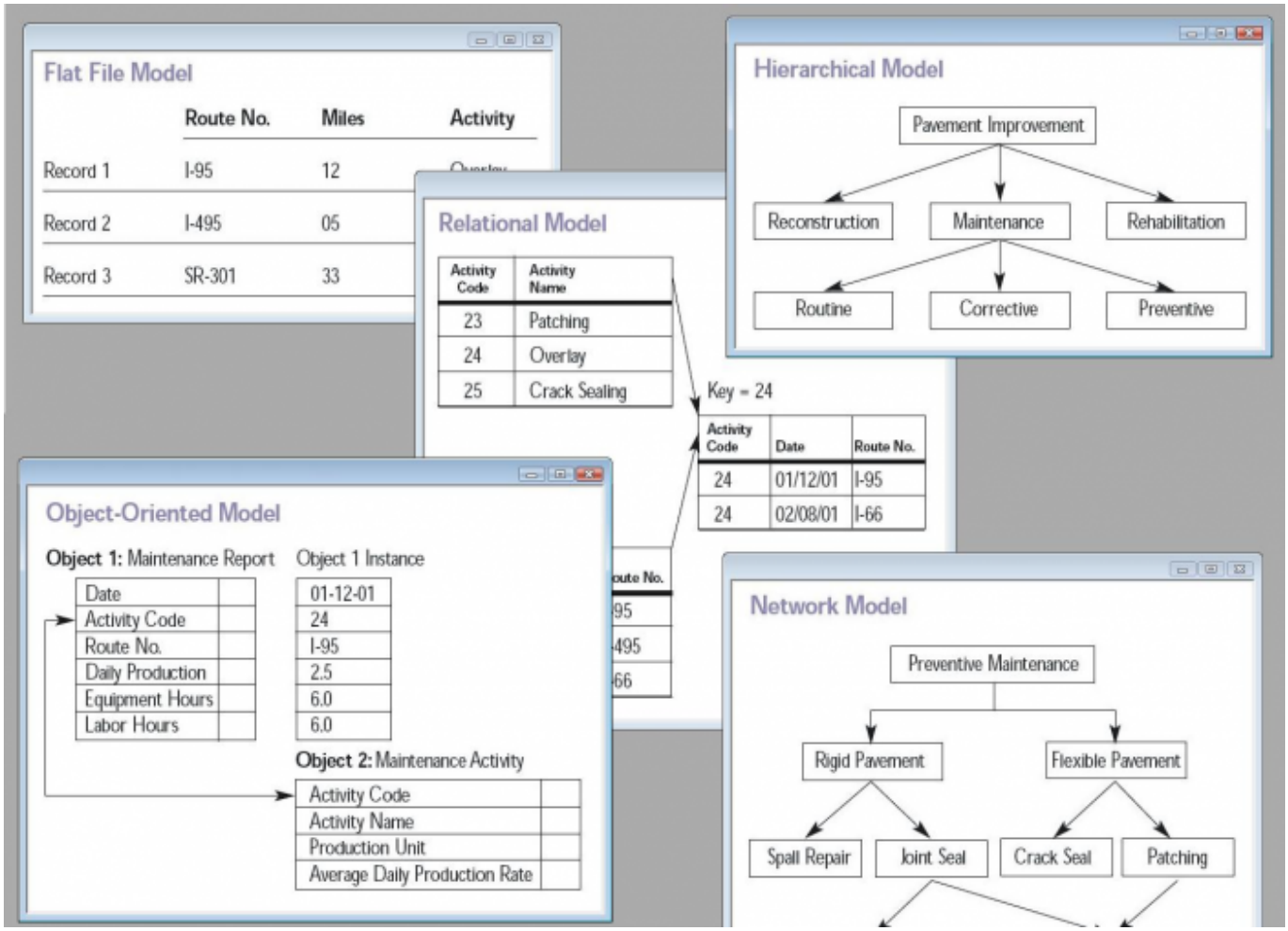
Konzeptionelle Datenmodelle (z.B.: ER-Modell) stehen **problemnahe Modellierungskonzepte** für die ersten Schritte des Datenbankentwurfs zur Verfügung und dienen zur **Kommunikation zwischen Endbenutzern und Datenbankentwicklern**.

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



2.3) ENTITY-RELATIONSHIP-MODELL

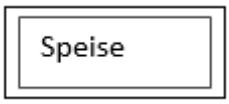
Das ER-Modell wurde im Jahr 1976 von Chen eingeführt. In der Sicht des ER-Modells besteht die Welt aus Entities (Objekttypen) und Relationships (Beziehungen), zwischen den Objekten. Das ER-Modell wurde von Teorey im Jahr 1986 erweitert zum Extended ER-Modell (EER).

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**

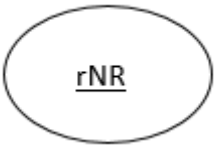
- **Weak Entity**



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.

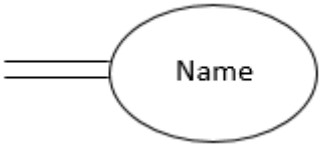


- **beschreibende Attribute** ⇒ sind jene Attribute, die ein Objekt nicht eindeutig kennzeichnen.

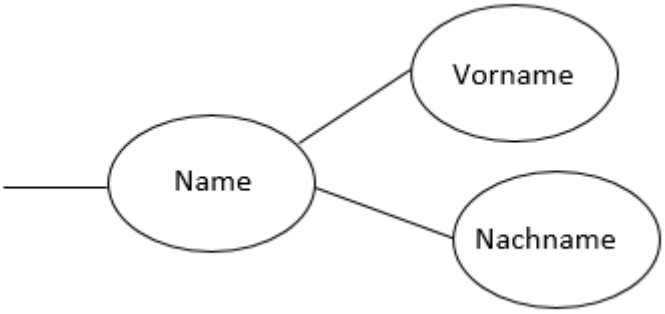
Schlüssel (Keys) können aus **einem oder mehreren Attributen** bestehen. Jede Entität kann mehrere verschiedene Schlüssel haben, wir stellen im ER-Modell aber immer **einen ausgezeichneten Schlüssel als Primärschlüssel (=Primary Key)** dar. Entitäten deren Objekte, nur mit Hilfe von anderen Entitäten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

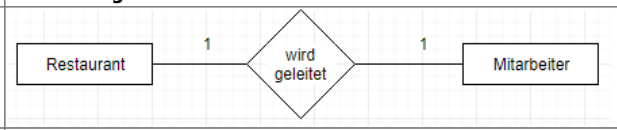
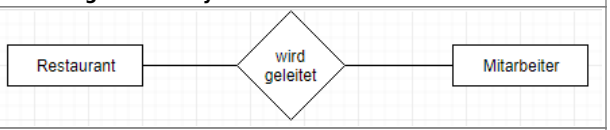
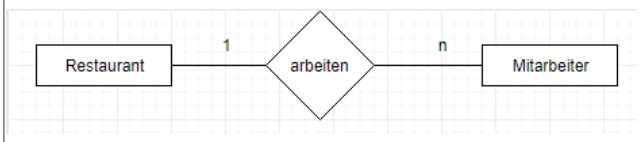
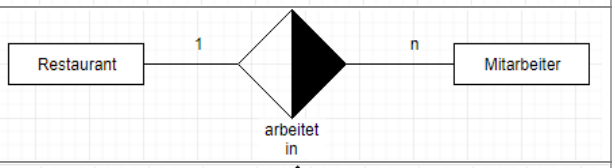
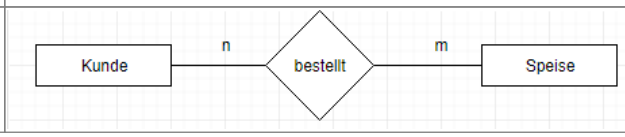
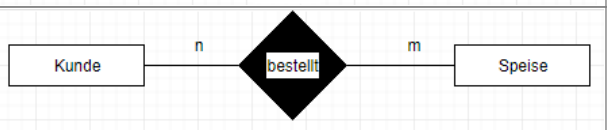


2.3.1.1) Komplexität von Beziehungen

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:

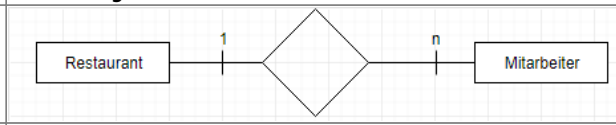
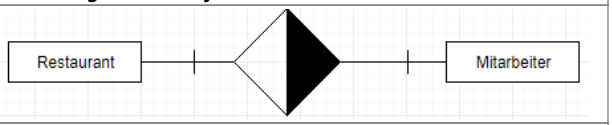
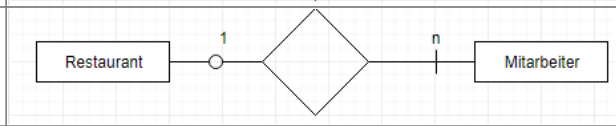
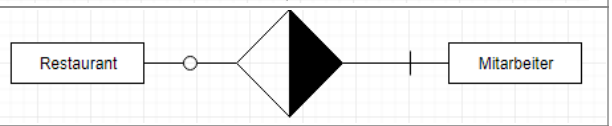
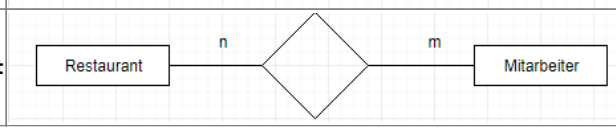
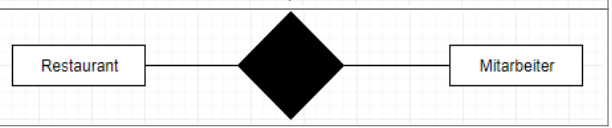
- **1:1** ⇒ Jedes Objekt von A gehört genau zu einem Objekt von B und umgekehrt
- **1:n** ⇒ Jedes Objekt von A gehört zu einem oder mehreren Objekten von B, aber jedes Objekt von B gehört genau zu einem Objekt von A.
- **n:m** ⇒ Jedes Objekt von A gehört zu einem oder mehreren Objekten von B und umgekehrt.

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. Teory 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 Teory
1:1		
1:n		
n:m		

2.3.1.2) Existenz einer Entität in einer Beziehung

Die Kopmplexität 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 Teory
zwingend		
optional		
unbekannt		

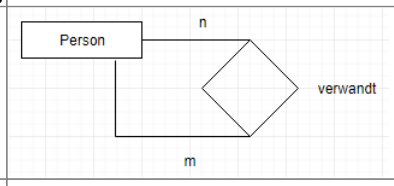
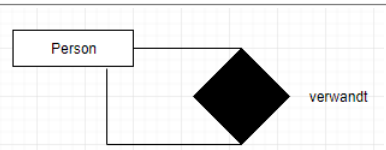
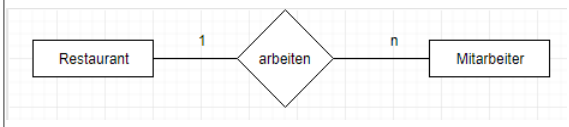
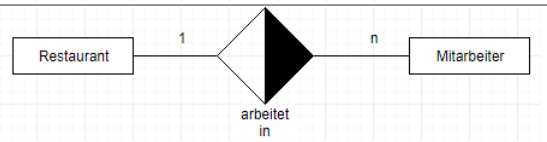
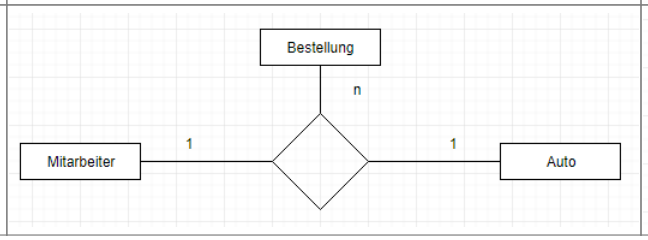
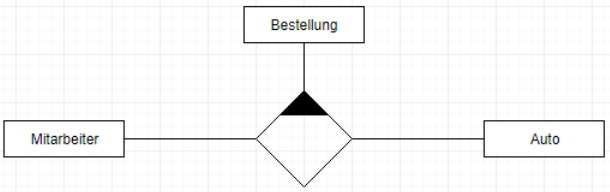
Bsp.:
zwingend ⇒ Ein Restaurant hat hat zumindest einen oder mehrere Mitarbeiter und jeder Mitarbeiter ist genau einem Restaurant zugeordnet.
optional ⇒ Ein Restaurant hat zwingend einen Geschäftsführer. Manche Mitarbeiter sind Geschäftsführer.

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.

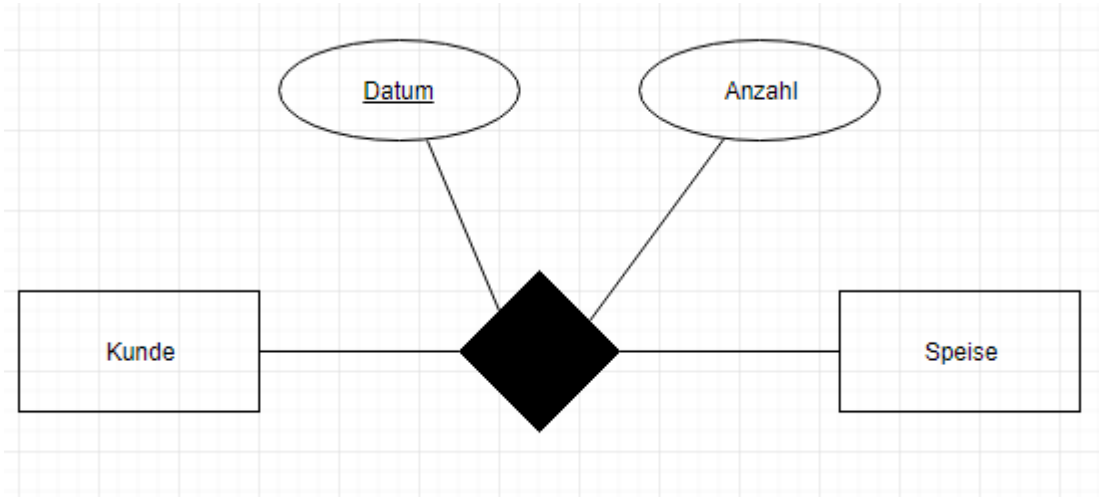
Grad der Beziehung	Darstellung nach Chen	Darstellung nach Teory
unär		
binär		
ternär		

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.

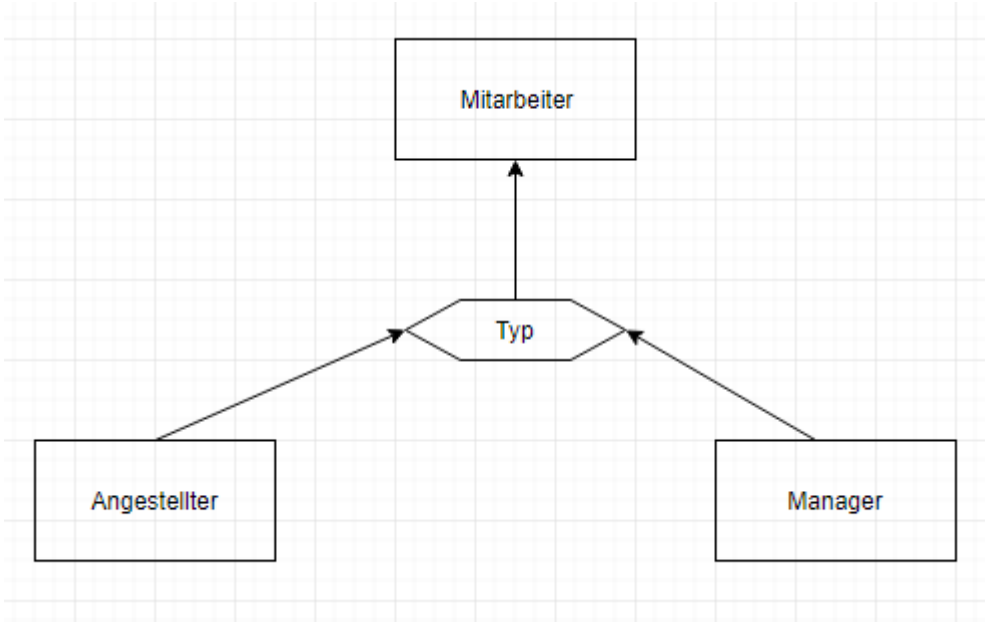
2.3.2) Erweiterte ER-Konstrukte: Die Generalisierung

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.

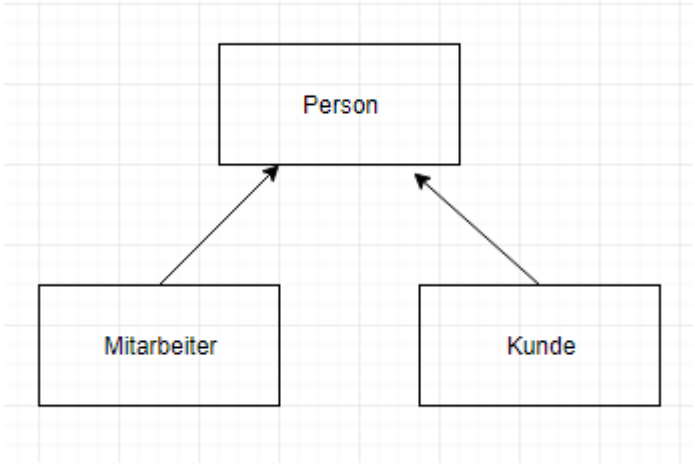
Die Generalisierung gibt an, dass mehrere Entities (Subtyp-Entity) mit bestehenden gemeinsame Attributen zu einer Entity auf einer höheren Ebene (Supertyp-Entity) generalisiert werden können.

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**.

2.4.1 Formalisierung

Eine **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}
```

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 t mit:

```
t(rnr)       = 3
t(name)      = "Green Cottage"
t(haube,typ) = 2, "chinesisch"
t            = 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

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

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

Relation Assistent			
ANr	**AName**	**StudAss**	**Gehalt**
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?

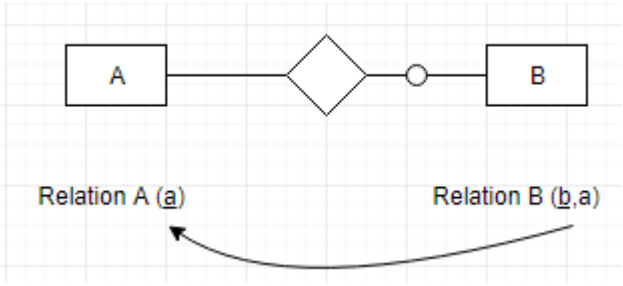
i) Welche Programme (gesucht sind die Namen) sind auf allen Rechnern installiert?

j) Welche Programme laufen auf einem Rechner, der genaue die minimale Leistungsfähigkeit für das Programm besitzt? (Gesucht sind die Paare aus Rechnernummer und Programmnummer)

2.5 Umsetzung des ER-Modells in ein Relationenmodell

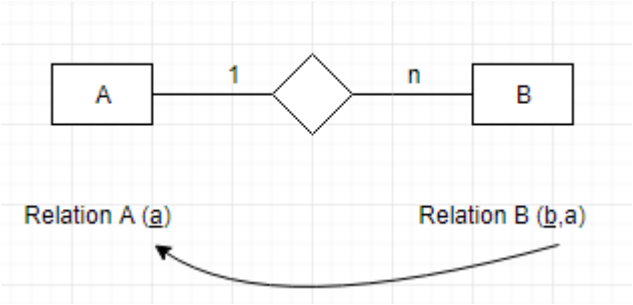
2.5.1 1-zu-1 Beziehung

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.



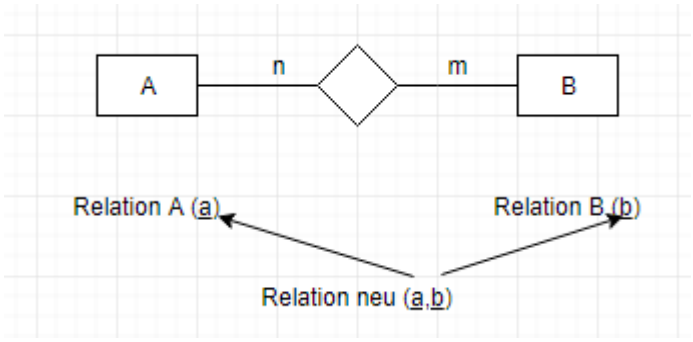
2.5.2 1-zu-n Beziehung

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



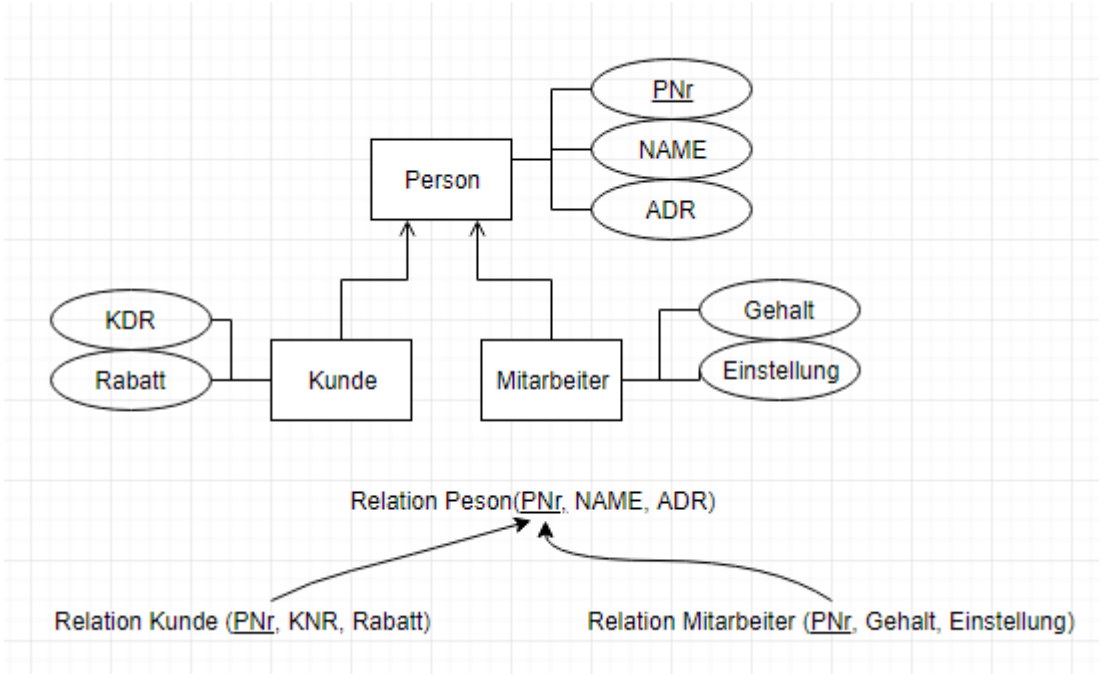
2.5.3 n-zu-m Beziehung

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

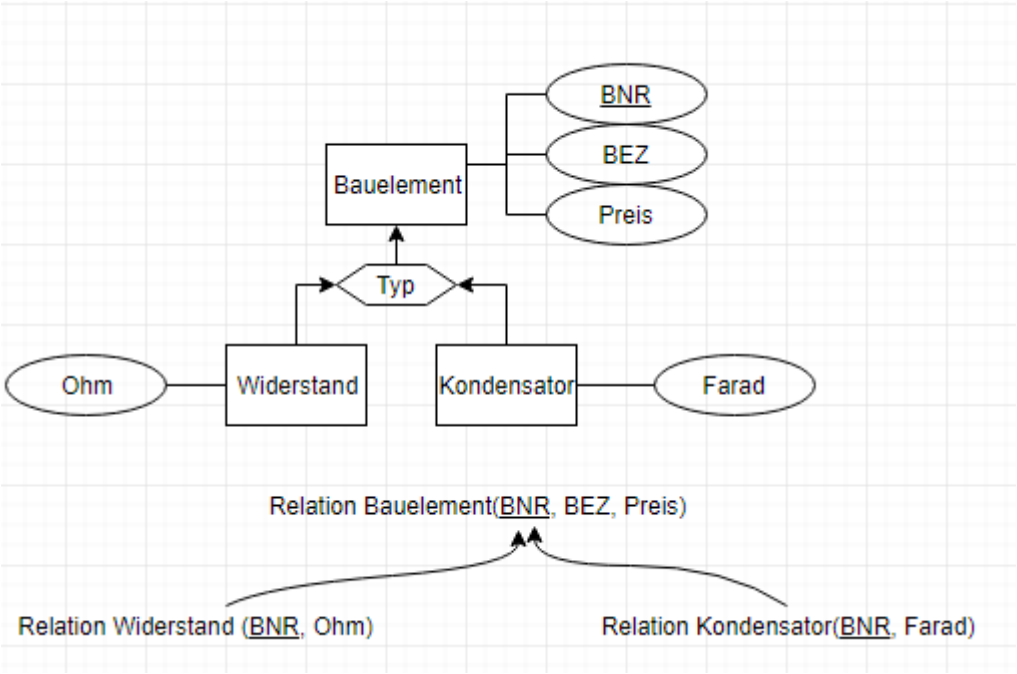


2.5.4 Generalisierung

2.5.4.1 nicht disjunkte Entitäten

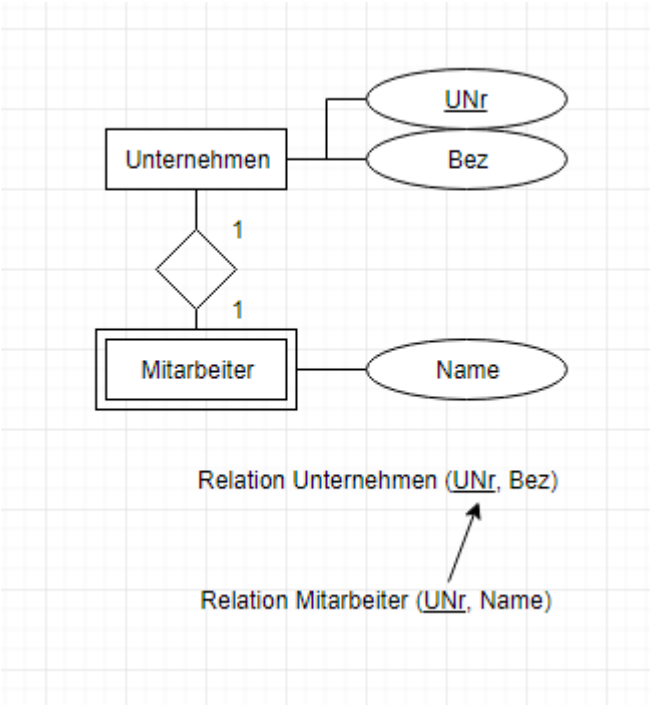


2.5.4.1 disjunkte Entitäten



2.5.5 Weak-Entities

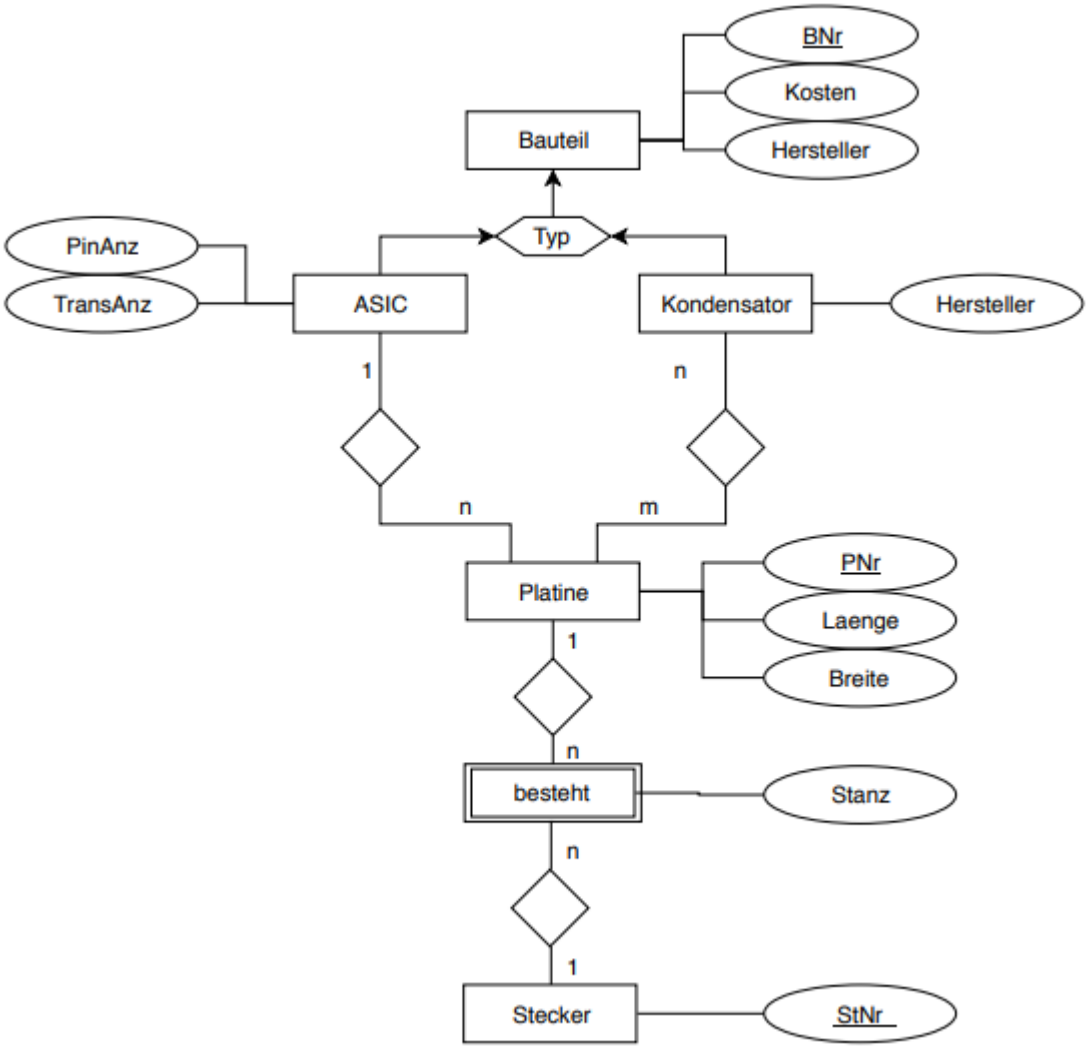
Bei schwachen Entitäten, bei denen die eigenen Attribute nicht ausreichen um ein Tupel eindeutig zu identifizieren, müssen die Schlüsselattribute der damit verbundenen Entitäten zum Schlüssel hinzugenommen werden.



Übungen zur Umsetzung von ER-Modell => Relationenmodell

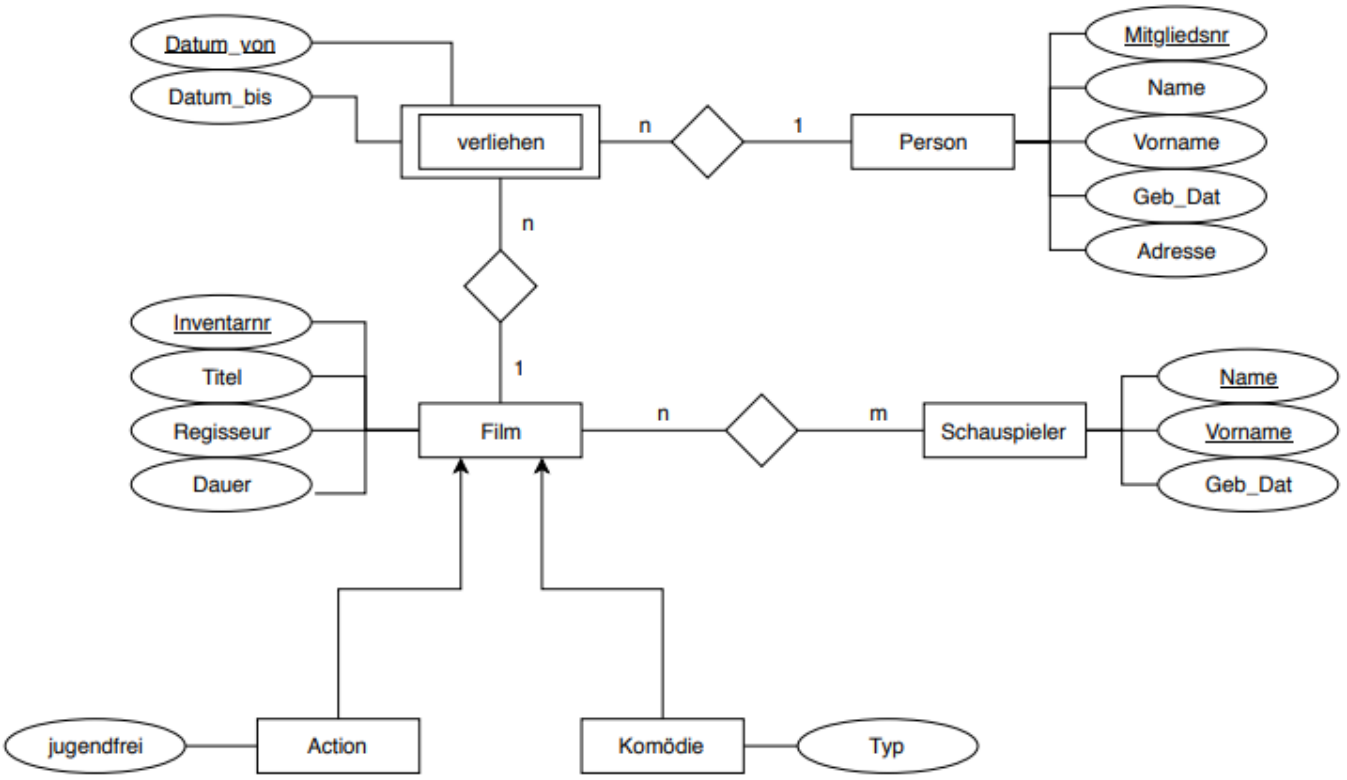
Übung 1

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



Übung 2

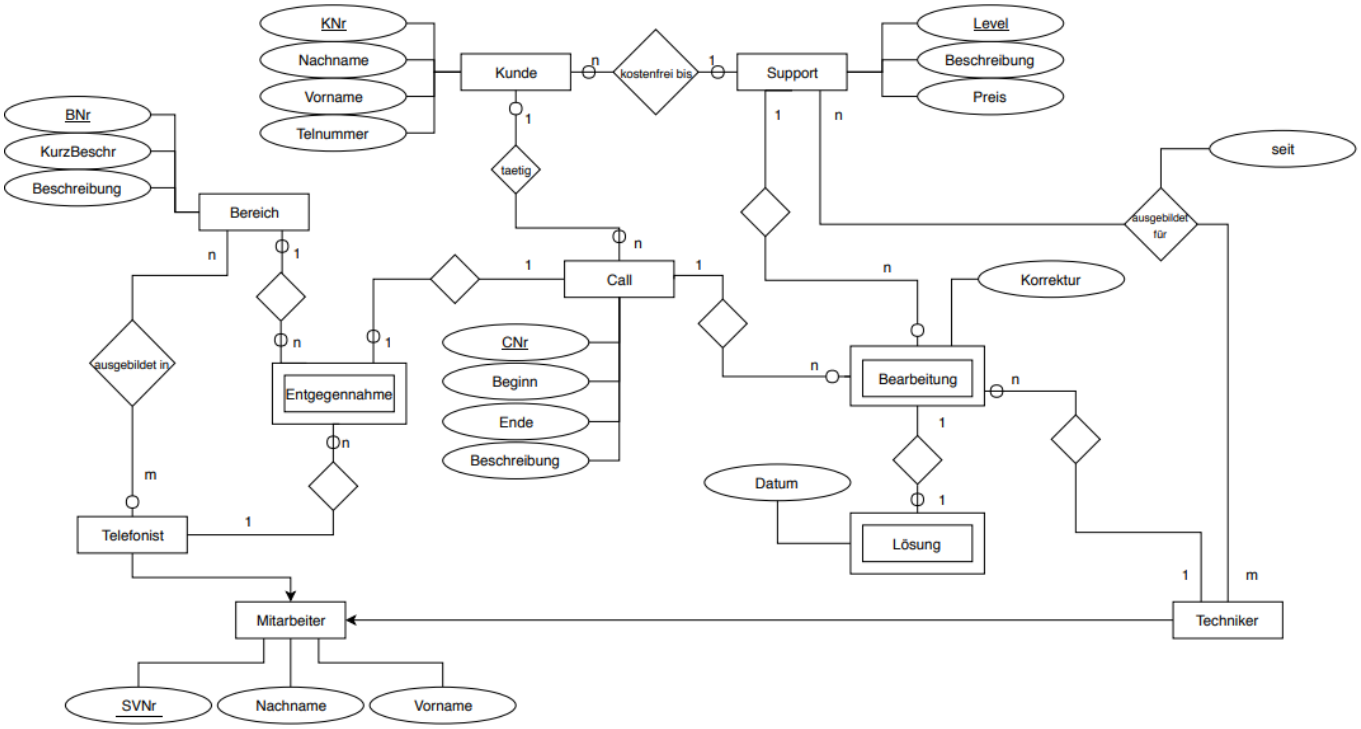
Gegeben ist folgendes ER-Diagramm zur Verwaltung eines Videoverleihs. Entwickeln Sie daraus die Relationen der Datenbank.



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.



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

Permanent link:
http://elearn.bgamstetten.ac.at/wiki/doku.php?id=inf:inf8bi_201819:2:2_05_01

Last update: **2018/11/24 17:15**

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

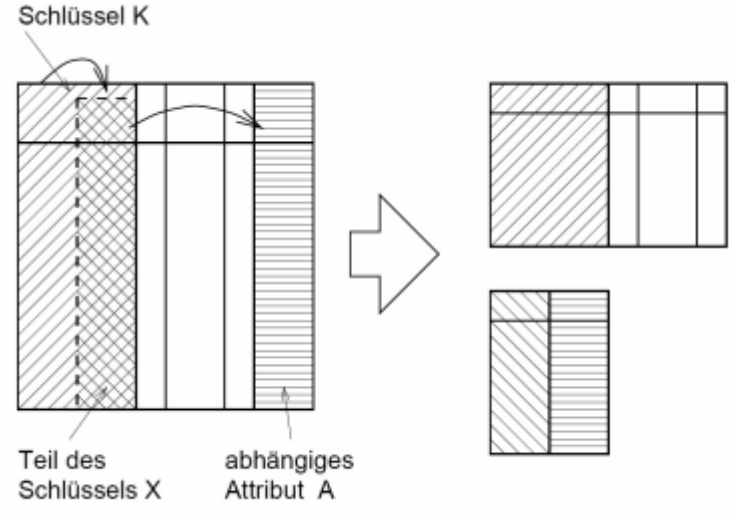
Lösung 1NF

_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	Tattou You	Rolling Stones	Start Me Up
4712	Tattou You	Rolling Stones	Hang Fire
4712	Tattou You	Rolling Stones	Slave

2. Normalform (2NF)

Ein Attribut heißt Prim in einer Relation, wenn es mindestens in einem Schlüssel von R enthalten ist bzw. nicht prim, wenn es in keinem Schlüssel enthalten ist.
Ein Relationenschema R ist in zweiter Normalform, wenn es in 1NF und jedes nicht prime Attribut voll funktional von jedem Schlüssel von R abhängig ist, d.h. nur vom ganzen

Schlüssel und nicht nur von einem Teil abhängig ist.



Beispiel

Gegeben ist folgende Tabelle:

_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	Tattou You	Rolling Stones	1	Start Me Up
4712	Tattou You	Rolling Stones	2	Hang Fire

Problem

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

_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	Tattou You	Rolling Stones	2	Hang Fire

Lösung 2NF

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

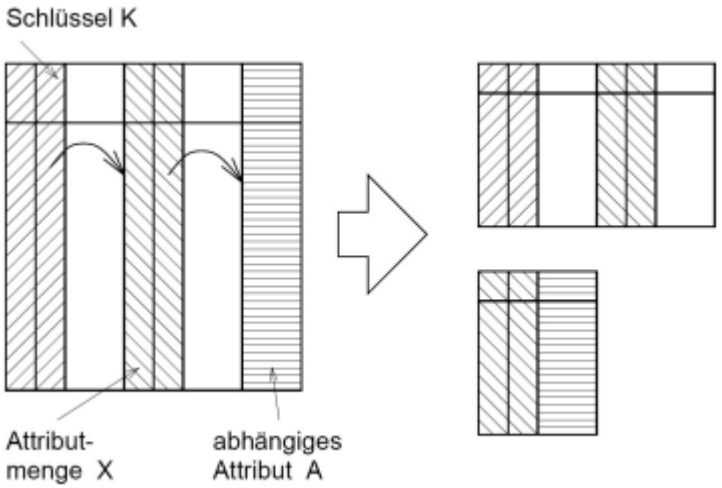
_CD_ID_	_Track_	Titel
4711	1	All That You Can Leave Behind
4711	2	Walk On
4711	3	Kite

_CD_ID_	_Track_	Titel
4712	1	Start Me Up
4712	2	Hang Fire

3. Normalform (3NF)

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

Transitivitätsregel K→X, X→A : K→A



Beispiel

Gegeben sei eine Tabelle mit folgenden Feldern:

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

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

Problem

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

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

Lösung

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

Interpret	Gründungsjahr
U2	1976
Rolling Stones	1962

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

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
  - SMALLINT
  - NUMERIC (p,q)
  - DECIMAL (p,q)
- angenähert numerisch
  - REAL
  - DOUBLE
  - FLOAT
- Zeichenketten
  - CHAR
  - VARCHAR
- Bitketten
  - BLOB (Binary Large Objects)
- Datum und Uhrzeit
  - DATE
  - TIME
  - TIMESTAMP
- 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;
```

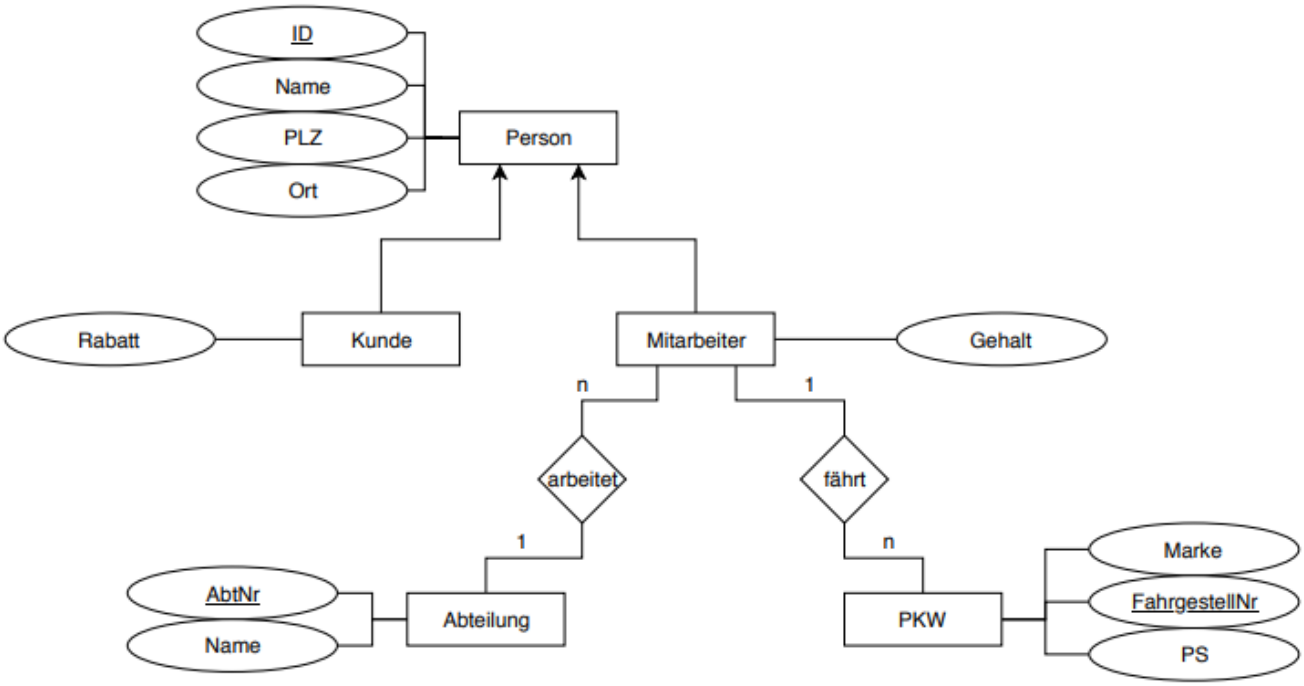
2.7.1.4) ENTFERNEN von Relationenschemata

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_Orte 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_ID 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_AbtNr PRIMARY KEY(ID),
    CONSTRAINT FK_AbtNr FOREIGN KEY(AbtNr) REFERENCES Abteilung(ID),
    CONSTRAINT FK_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_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

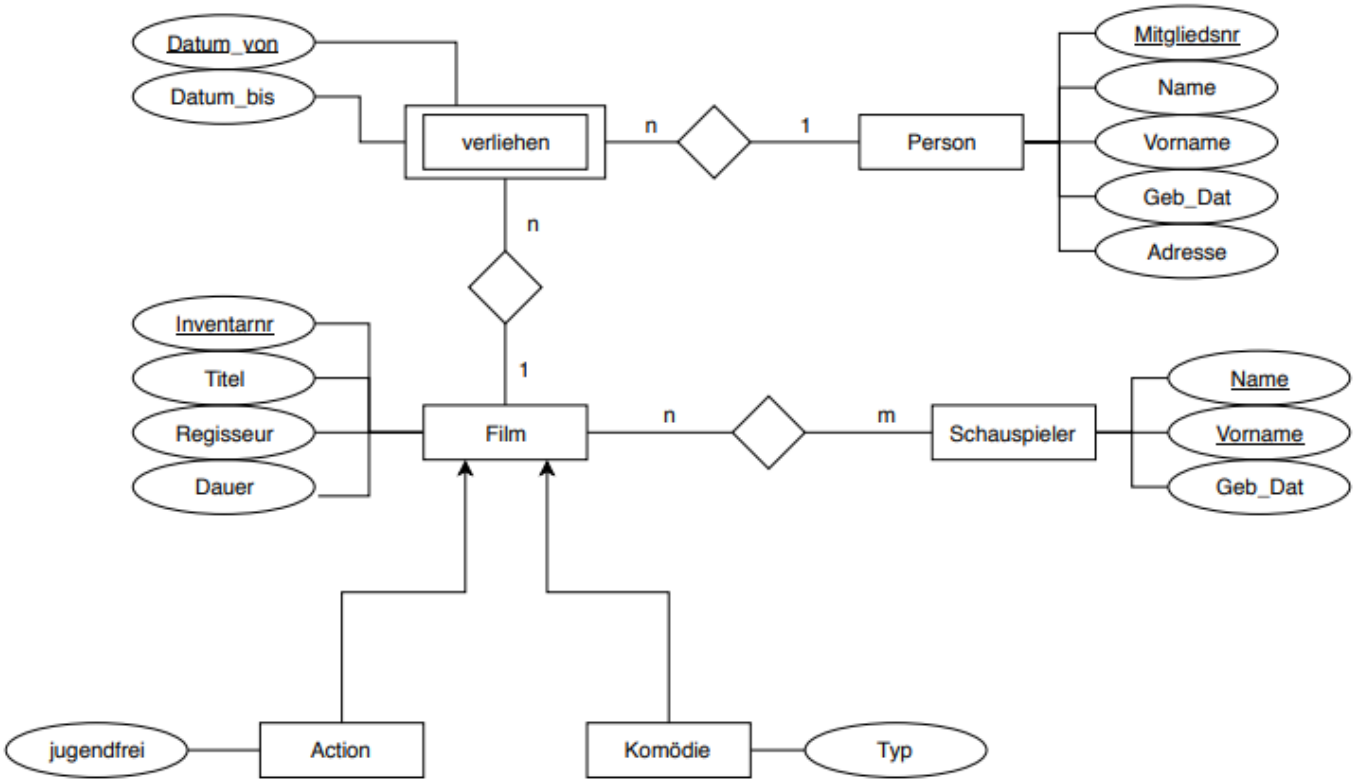
Relation Assistent			
ANr	**AName**	**StudAss**	**Gehalt**
A1	Novak	Brunner	3
A2	Dvorak	Vogt	1
A3	Husak	Vogt	1
A4	Pfeiffer	Brunner	2

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

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);

        while($row=mysqli_fetch_array($erg))
        {
            echo "<table bgcolor=\"#dddddd\" border=\"0\" width=\"600\">
```

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

Datum:	2018-11-15 20:19:20
Name:	Prof. Andreas Lahmer
Eintrag:	Hier sehen sie die SQL-Anbindung mittels PHP in Form eines Gästebuchs!

---

neuen Eintrag hinzufügen

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

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

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

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

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

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

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

            //Formular
        ?>



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

Name:
Text:

Absenden

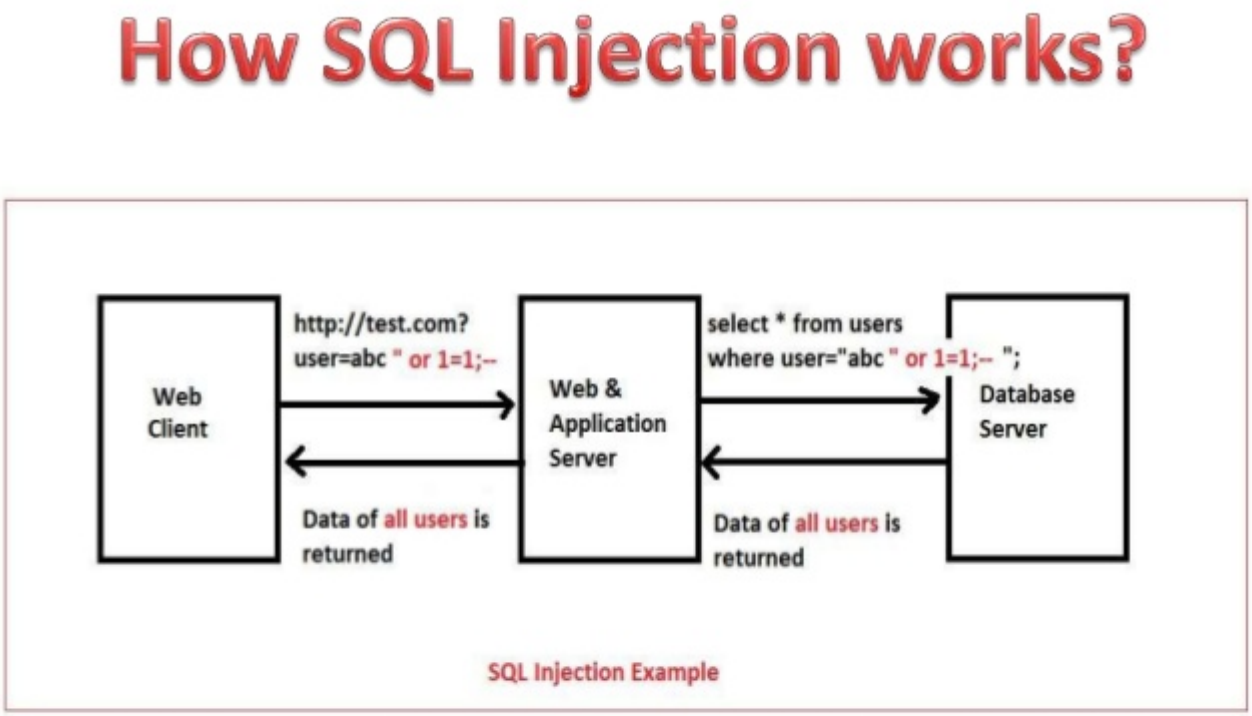
Alle Einträge anzeigen

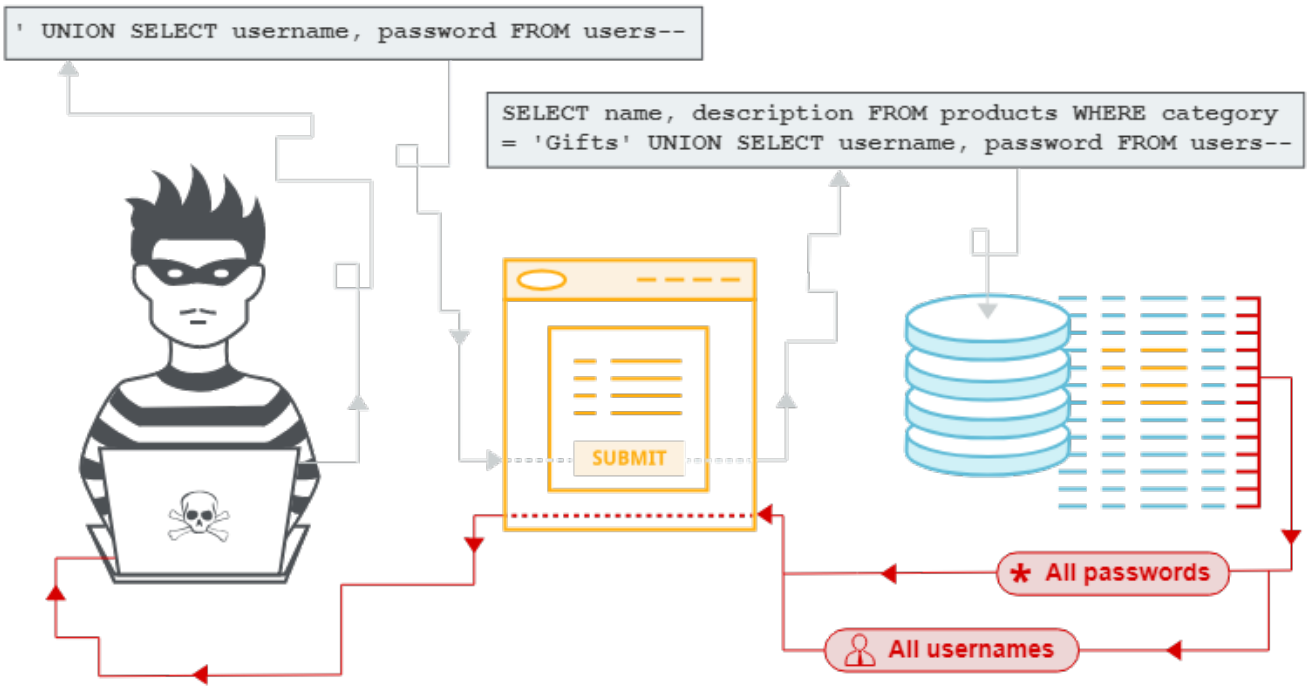
SQL Injection

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

2.8) SQL-Datenbankzugriff mit Visual C++

C++

Zeiger (Pointer)

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

Jede Variable wird in C++ an einer bestimmten Position im Hauptspeicher abgelegt. Diese Position nennt man Speicheradresse (engl. memory address). C++ 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:

Variable	Adresse	Wert
Wert	0x0001	10
*pWert	0x0005	0x0001
*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:

Variable	Adresse	Wert
Wert	0x0001	10 –> 15 –> 23
*pWert	0x0005	0x0001
*pZahl	0x0009	0x0001
...	.	.
...	.	.

Ausgabe:

```
Wert = 23
```

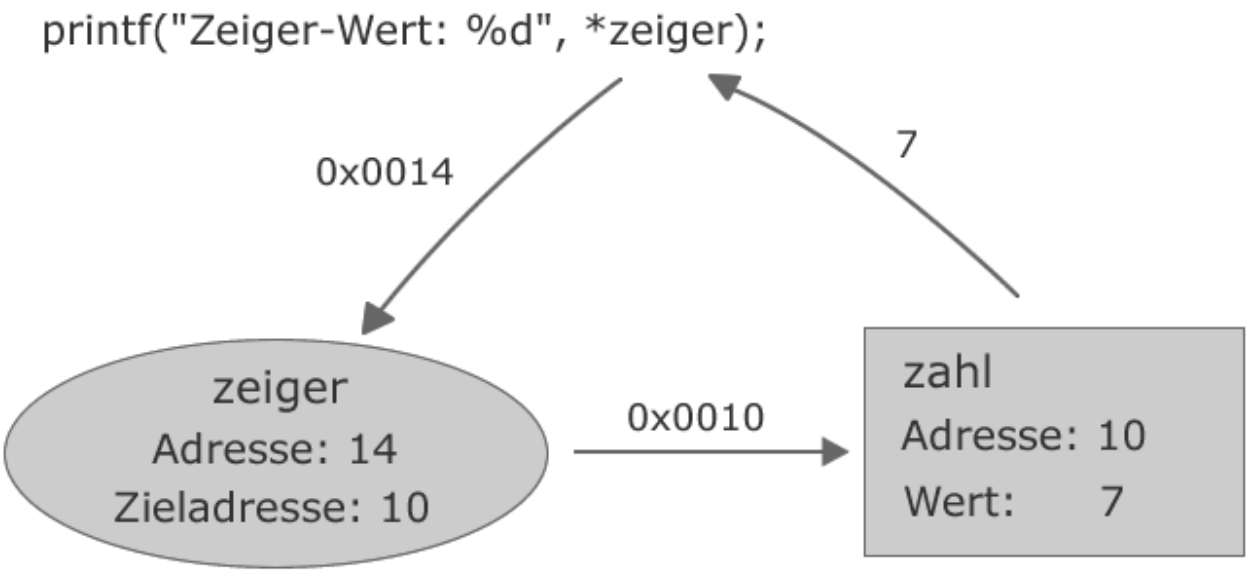
Man nennt das den Zeiger dereferenzieren. Im Beispiel erhalten Sie die Ausgabe Wert = 23, denn pWert und pZahl verweisen ja beide auf die Variable Wert.

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.

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 3.1.1

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 3.1.2

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;
```

Rekursionen

Die **Rekursion** ist ein **spezieller Aufruf von Funktionen**, nämlich wenn Funktionen sich **selbst aufrufen**. Da bei einem Aufruf sich die Funktion wieder selbst aufruft, benötigt die Funktion wie bei den Schleifen eine **Abbruchbedingung**, damit die Selbstaufrufe nicht endlos sind.

Beispiel

In dem folgenden Beispiel, welches lediglich eine Bildschirmausgabe via Rekursion zeigt, wird der **Abbruch anhand einer Zählvariable** entschieden – wie bei den Schleifen. Ist x größer 0 erfolgt eine Ausgabe und ein rekursiver Aufruf mit einem dekrementierten (=verringerten) Zählwert. Ist der Zählwert bei 0 angelangt, erfolgt kein rekursiver Aufruf mehr.

```
#include<stdio.h>

using namespace std;

printLines(int x);

int main() {
    printLines(5);
    return 0;
}

printLines(int x){
    if(x > 0) {
        cout << "\nZeile Nr. " << x);
        printLines(x-1);
    }
}
```

Ausgabe des Programms

```
Zeile Nr. 5
Zeile Nr. 4
Zeile Nr. 3
Zeile Nr. 2
Zeile Nr. 1
```

Beispiel Fakultät

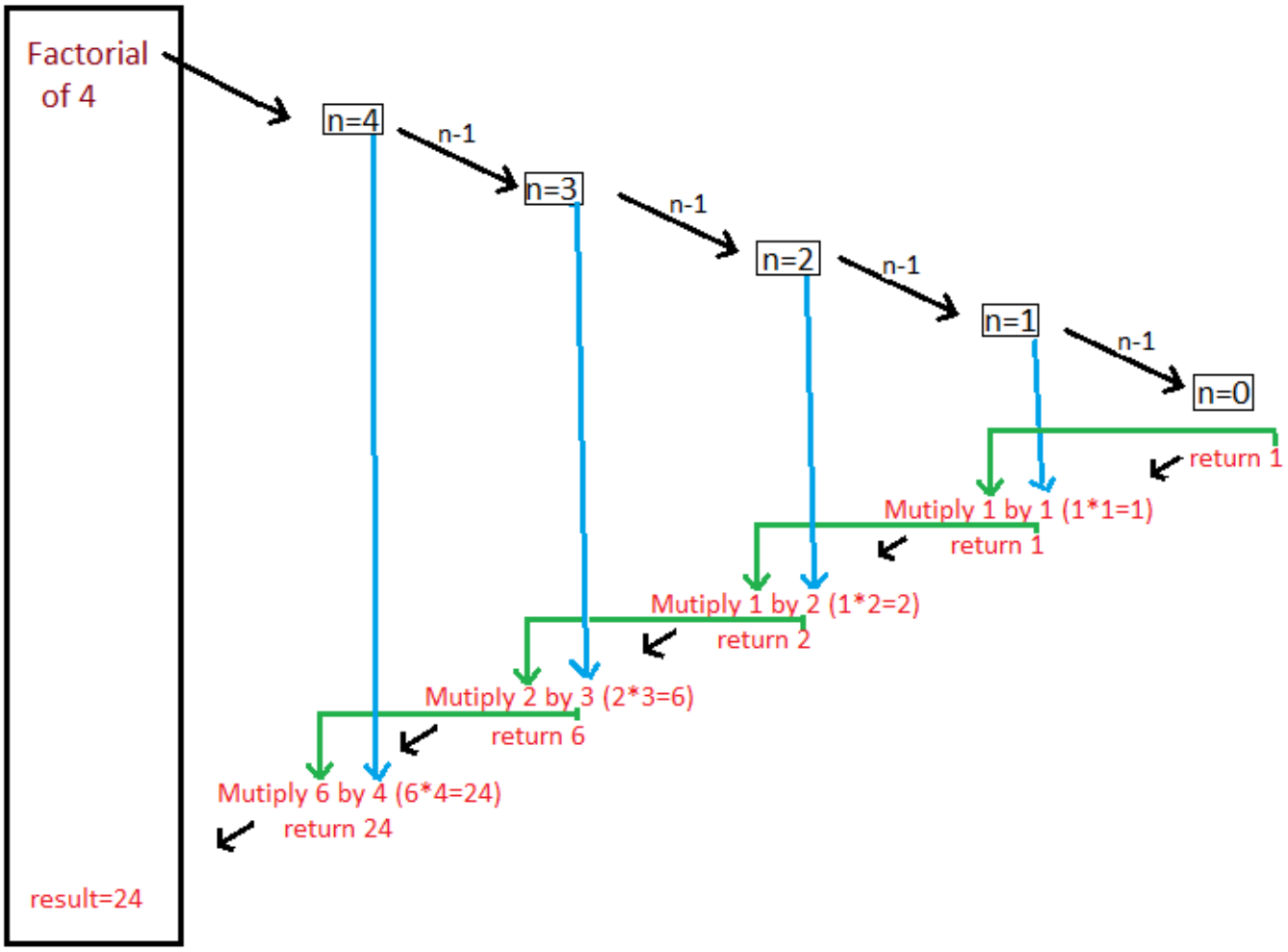
Nun ein sinnvolleres und gern verwendetes Beispiel, die **Berechnung der Fakultät mittels Rekursion**. Bei der Berechnung der Fakultät wird solange ein Produkt mit der dekrementierten Zahl gebildet, bis die Zahl bei der 1 angelangt ist. Zum Beispiel ist die Fakultät Vier: 4! = 4 * 3 * 2 * 1 =

24.

```
#include<stdio.h>

using namespace std;

int fakultaet(int x);

int main() {
    int a = 6;
    cout << "Fakultaet von " << a << " ist " << fakultaet(a);
    return 0;
}

int fakultaet(int x) {
    if(x > 1) {
        return x * fakultaet(x-1);
    }else {
        return 1;
    }
}
```

Ausgabe

```
Fakultaet von 6 ist 720
```



Aufgabe 3.2.1

Schreibe ein C + + - Programm, das die Summe aller Zahlen von 1 bis n (einer vom User eingegebenen Zahl) mithilfe einer Rekursion berechnet. Das Ergebnis der Summe soll im Hauptprogramm ausgegeben werden.

Beispiel

Eingabe:

```
6
```

Summe:

```
1+2+3+4+5+6 = 21
```

Aufgabe 3.2.2

Schreibe ein C + + - Programm, das die Potenzfunktion y = x^n mithilfe einer Rekursion berechnet. Der Funktionswert y, soll im Hauptprogramm ausgegeben werden.

Beispiel

Eingabe:

```
x=2
n=4
```

Ausgabe:

```
y=16 (2*2*2*2)
```

Aufgabe 3.2.3

Schreibe ein C + + - Programm, das die Fibonacci-Zahlen rekursiv berechnet. Die Fibonacci-Zahlen berechnen sich außer bei der Zahl 1 und 2 immer aus den beiden Vorgängern der Zahl.

```
z. B. 1, 2, 3, 5, 8, 13, 21, ...
```

Errechnet werden können sie mittels

```
1+2=3, 2+3=5, 3+5=8, 5+8=13.
```

Die Formel lautet also:

```
F(n+2)=F(n+1) +F(n)
```

Beispiel

Eingabe der Anzahl von Fibonacci-Zahlen:

```
6
```

Ausgabe:

```
1
2
3
5
8
13
```

Struktur - eine zusammengesetzter Datentyp

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

Modell

Vielleicht werden Sie einwerfen, dass ein Auto noch mehr Bestandteile hat. Da gibt es 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.

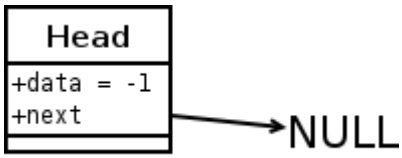
Einfach verkettete Listen zeichnen sich dadurch aus, dass man besonders einfach Elemente einfügen kann, wodurch sie sich besonders gut für Insertion Sort eignen.

Knoten

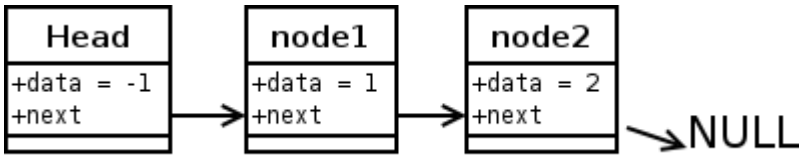
Eine einfach verkettete Liste besteht aus **Knoten, Englisch nodes**, die einen **Zeiger auf das nächste Element** und Daten beinhalten.

```
typedef struct list{

    int data;
    list *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.

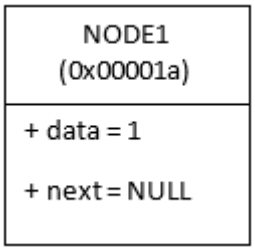
Knoten befüllen & einfügen

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

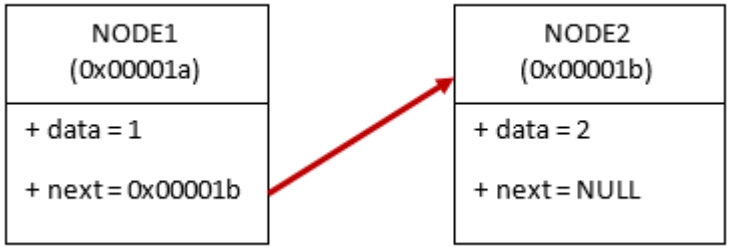
Dazu muss man den next-Zeiger der Liste auf das neue Element setzen, und den next-Zeiger des neuen Element auf den alten Wert des next-Zeigers der Liste:

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

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

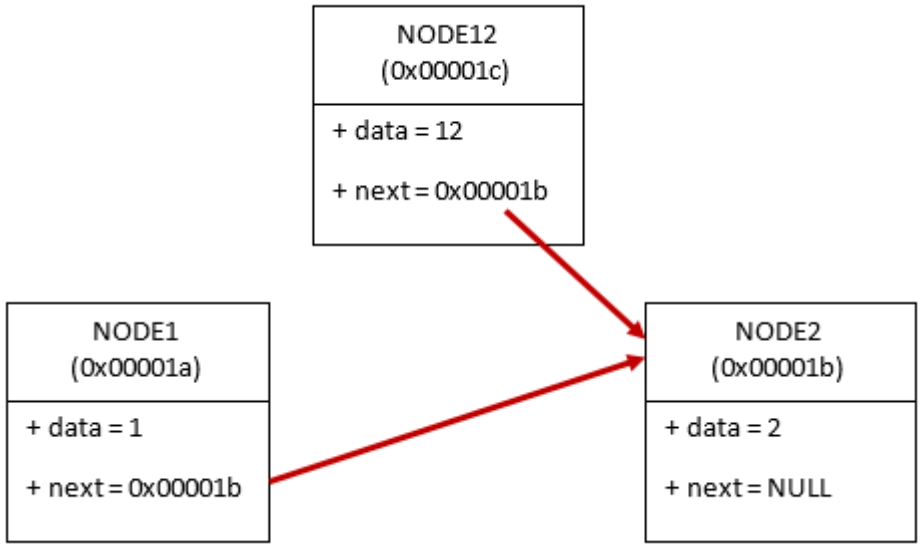



```
    //2. Knoten erstellen
    list *node2;
    node2->data=2;
    node2->next=NULL;
    //1. Knoten zeigt auf 2. Knoten
    node1->next=node2;
```

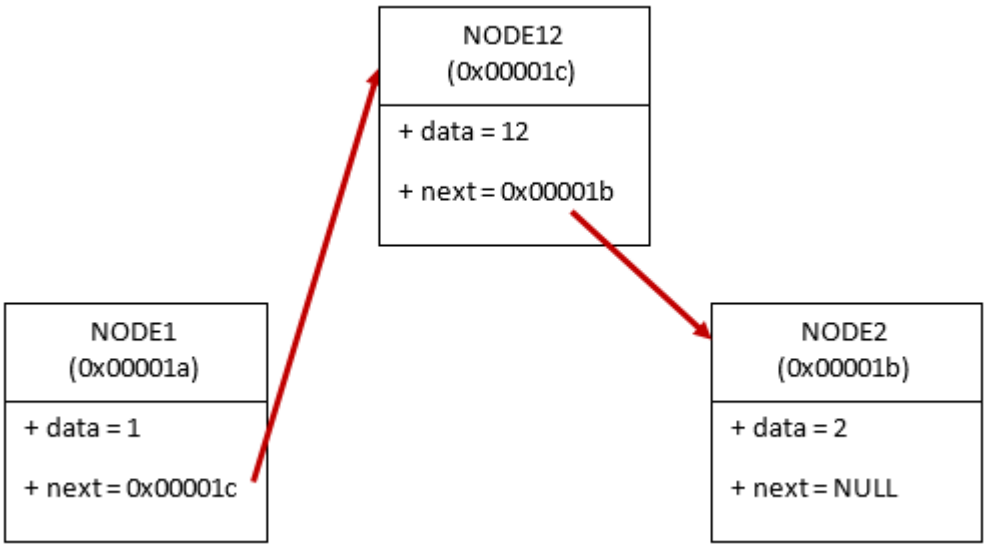



```
    //Knoten 12 zwischen Knoten 1 und 2 einfügen
    list *node12;
    node12->data=12;
    node12->next=node2;
    node1->next=node12;
```

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 list{
```

```
    int data;
    list *next;

};

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


    //Kopf der Liste
    list *head=NULL;


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


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

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

        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.

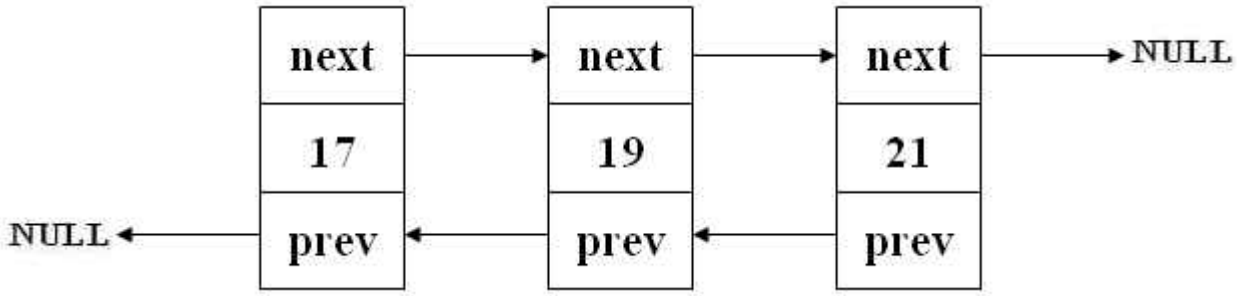
Die Grundidee

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');
```

(Binäre) Bäume

Was haben Bäume mit der Informatik zu tun?? – werden Sie Sich wahrscheinlich gefragt haben, als Sie zum ersten Mal erfahren haben, dass Bäume auch in der digitalen Welt existieren.

Sie werden sehen, dass Bäume in der Informatik einige Gemeinsamkeiten mit den natürlichen Bäumen haben. Zum Beispiel besitzen beide Blätter und Wurzel. Und genau so, wie es in der Natur etliche Arten von Bäumen gibt, existieren auch in der Informatik verschiedene Arten. Jede Art ist spezialisiert und für eine bestimmte Anwendung optimiert. Es gibt jedoch auch Unterschiede zwischen Bäumen der Informatik und natürlichen Bäumen: Sie „wachsen“ nicht in die gleiche Richtung. Bäume in der Informatik wachsen nach unten und haben dementsprechend die Wurzel oben wie ein natürlicher Baum, den Sie um 180 Grad drehen.

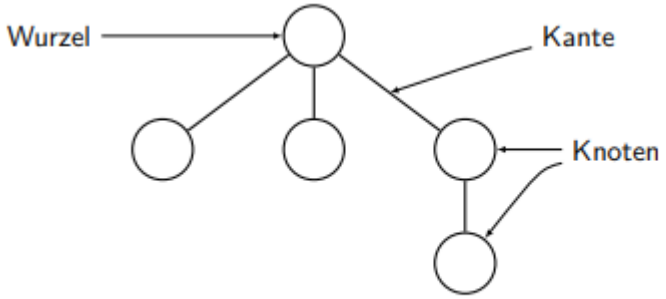


Abbildung Ein Baum mit der Wurzel, mehreren Knoten und Kanten.

Komponenten eines Baums

Ein **Baum (tree)** besteht aus verschiedenen Komponenten. Die wohl wichtigste ist der **Knoten**

Knoten (node) & Kanten (edges)

Ein **Knoten (node)** dient zur Speicherung von Daten. Knoten sind untereinander mit **Kanten (edges)** verbunden.

Wurzel (root) & Blatt (leaf)

Besitzt ein Baum keine Knoten, so ist dieser Baum **leer**. Besitzt er hingegen Knoten, so können diese vom Typ **Wurzel (root)**, **innerer Knoten** oder **Blatt (leaf)** sein.
Jeder nicht leere Baum besitzt genau eine **Wurzel (root)**, welche normalerweise oben gezeichnet wird. Da die Wurzel ganz oben ist, wächst der Baum nach **unten**.

Kind (child) & Vater (father)

Ist ein Knoten nach unten mit **anderen Knoten durch Kanten verbunden**, so sind diese unteren

Knoten seine **Kinder (child)**. Der obere Knoten wird als **Vater (father)** dieser Kinder bezeichnet.

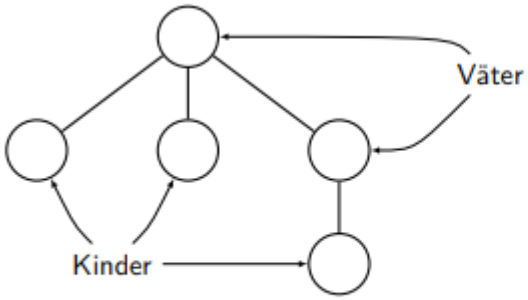


Abbildung Die Wurzel ist Vater von drei Kindern und eines ihrer Kinder (ganz rechts) ist selber Vater eines Kindes.

Ein **innerer Knoten (inner node)** ist ein Knoten mit mindestens einem Kind. Ist ein Knoten kinderlos, so spricht man von einem **Blatt (leaf)**.

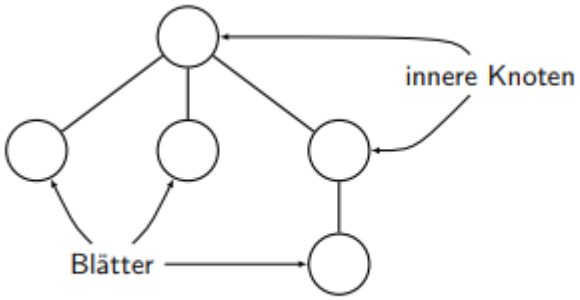


Abbildung Ein Baum mit inneren Knoten und Blättern.

Masse eines Baums

Es gibt verschiedene Messgrößen, welche benutzt werden, um Eigenschaften eines Baumes zu beschreiben.

Ordnung (order)

Die **Ordnung (order)** gibt an, wie viele Kinder ein innerer Knoten *höchstens* haben darf. Hat ein Baum zum Beispiel die Ordnung 2, so dürfen seine Knoten maximal 2 Kinder haben. Ist jedoch nur die grafische Darstellung eines Baumes vorgegeben, so lässt sich seine Ordnung nicht ablesen. An Hand einer Grafik kann nur die Ordnung bestimmt werden, welche der Baum mindestens hat.

Grad (degree)

Der **Grad (degree)** eines Knotens sagt aus, wie viele Kinder ein Knoten effektiv hat. Dabei gilt, dass der Grad jedes Knotens kleiner oder gleich der Ordnung des Baumes sein muss. In der folgenden Abbildung haben zum Beispiel die Knoten a und b den Grad 2 und der Knoten e hat den Grad 5.

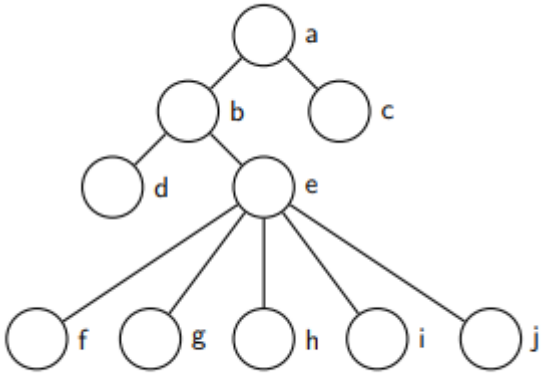


Abbildung Ein Baum mit Ordnung ≥ 5.

Pfad (path)

Der **Pfad (path)** von Knoten *v* nach Knoten *w* gib an, welche Kanten und Knoten besucht werden, wenn man von *v* nach *w* wandert.

Tiefe (depth)

Die **Tiefe (depth)** eines Knotens *v* gibt an, wie lange der Pfad zur Wurzel ist.

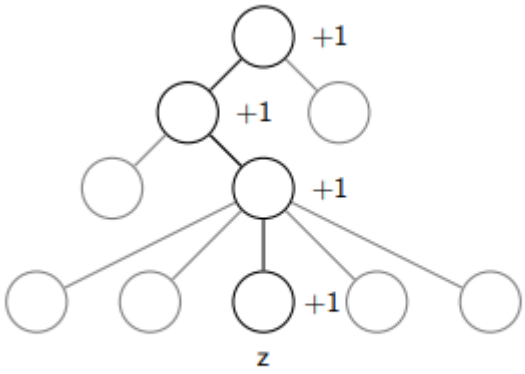


Abbildung Die Tiefe vom Knoten z ist $1 + 1 + 1 + 1 = 4$.

Dabei werden alle Knoten gezählt, welche auf dem direkten Pfad von der Wurzel zum Knoten *v* liegen - inklusive der Wurzel und dem Knoten *v* selbst.

Niveaus (level)

Knoten, die die gleiche Tiefe haben, werden zu **Niveaus (level)** zusammengefasst.

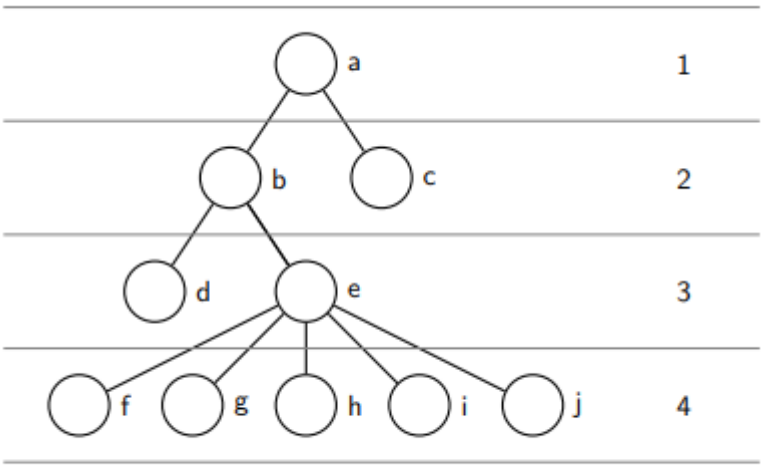


Abbildung Die Knoten d und e haben das gleiche Niveau von 3.

Höhe (height)

Die **Höhe (height)** eines Baumes entspricht der Tiefe des Knotens, welcher am weitesten von der Wurzel entfernt ist.
Somit entspricht die Höhe des Baumes dem größten Niveau aller Knoten des Baumes.

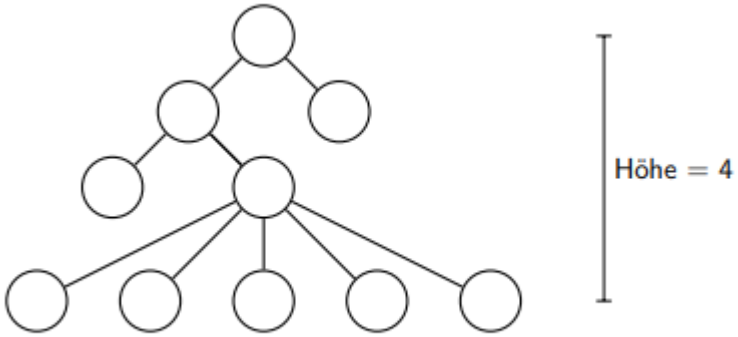


Abbildung Die Höhe dieses Baumes beträgt 4.

Ein Baum heißt ausgefüllt, wenn alle inneren Knoten die maximale Anzahl Kinder haben. Somit ist der Grad jedes Knotens entweder gleich der Ordnung des Baumes (innerer Knoten) oder 0 (Blatt).

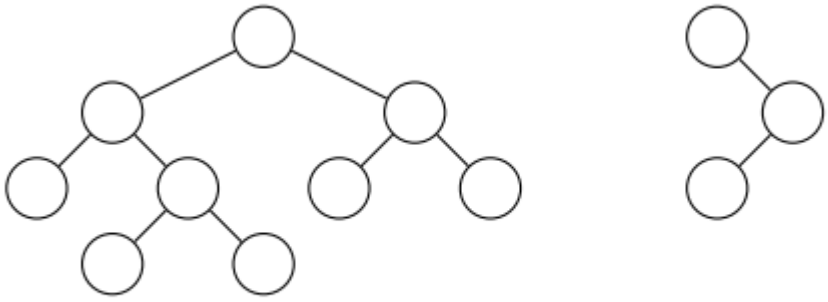


Abbildung Wenn beide Bäume Ordnung 2 haben, dann ist nur der Linke ein ausgefüllter Baum.

Vollständig

Ein Baum heißt **vollständig**, wenn jedes Niveau die maximale Anzahl an Knoten hat. Jeder vollständige Baum ist auch ausgefüllt.

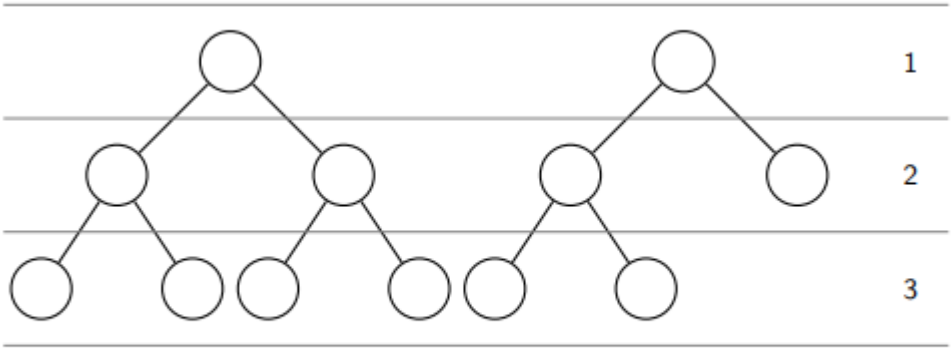


Abbildung Wenn beide Bäume Ordnung 2 haben, dann ist der linke Baum vollständig. Der rechte Baum ist zwar ausgefüllt, aber nicht vollständig.

Brüder/Geschwister

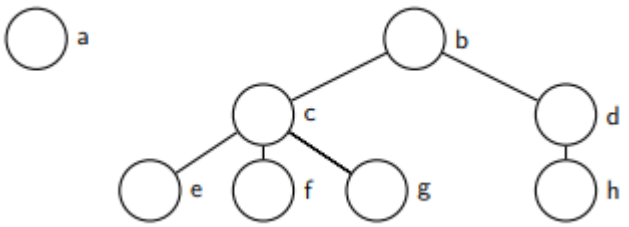
Knoten, welche den gleichen Vater haben, werden auch **Brüder** oder **Geschwister (sibling)** genannt.

⇒ Nun kennst du alle relevanten Begriffe der Bäume, welche für die anstehende Programmierung ein notwendige Grundlage sind.

Aufgaben

1)

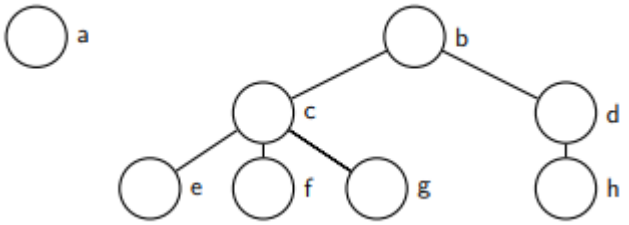
Bestimmen den Typ (Wurzel, innerer Knoten, Blatt, Vater, Kind) aller Knoten in folgender Abbildung. Beachte, dass einzelne Knoten auch von mehreren Typen sein können.



2)

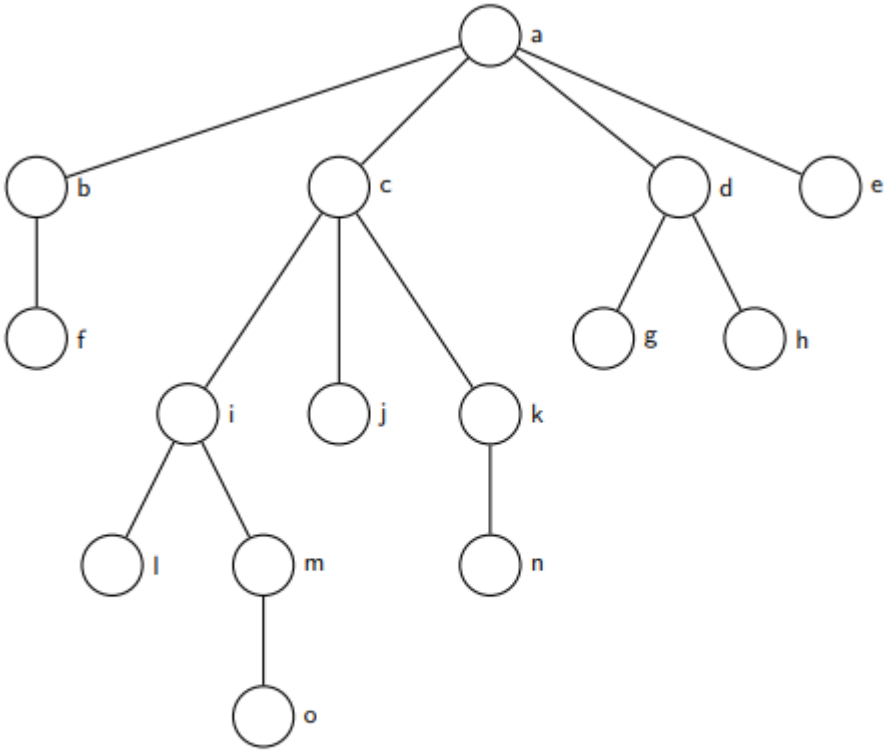
Bestimmen Sie für die folgenden zwei Bäume :

- a) die Höhe,
- b) die Ordnung,
- c) die Niveaus,
- d) ob der Baum ausgefüllt ist,
- e) den Grad aller Knoten
- f) und die Tiefe jedes Knotens



3)

Bestimmen Sie im folgenden Baum die Typen der Knoten. Füllen Sie dazu die vorgegebene Tabelle aus.



	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
Wurzel															
Blatt															
Innerer Knoten															

4)

Bestimmen Sie für jeden Knoten des Baumes der Aufgabe 3 die Tiefe und den Grad.

5)

Zeichnen Sie im Baum der Aufgabe 3 die Niveaus und die Höhe ein. Überlegen Sie sich, welche Ordnung dieser Baum haben könnte, und begründen Sie ihre Entscheidung.

6)

Gegeben sind die drei folgenden Bäume der Ordnung 2. Entscheiden Sie, welche Bäume vollständig und welche ausgefüllt sind, und begründen Sie Ihre Antwort