1 - Objektidentität

Würde die `*`-Methode für `i=1` keine Kopie erzeugen, sondern einfach dasselbe Objekt zurückliefern, so würde beim mutierenden Aufruf `y[0,1] = "b"` dasjenige Objekt, auf das auch `x` zeigt, verändert.
Das Programm würde sich also völlig anders verhalten, als für `i=0` oder `i>1`, wo ja auf jeden Fall neue Objekte angelegt werden müssen, da diese nicht identisch aussehen.

Beispiel:
Für die Variablenbelegung `i = 1` erhielte man, wenn `*` keine Kopie erzeugen würde:

```
Zeile |  x    y
------+----------
 1    | "a"
 2    |      "a"
 3    | "b"  "b"
```

Für die Variablenbelegung `i = 2` erhält man dagegen (`*` erzeugt Kopie):

```
Zeile |  x    y
------+----------
 1    | "a"
 2    |      "aa"
 3    |      "ba"
```

```
def without(s, n)
  if n < 1 then
    # Da wir s nicht verändern, Kopie zurückliefern
    return s.clone();
  else
    # Neuen String bauen
    result = "";

    for i in 0 .. s.length() - 1 do
      # Wenn das aktuelle nicht wieder ein n-tes Zeichen ist...
      if (i + 1) % n != 0 then
        result = result + s[i, 1];
      end;
    end;

    return result;
  end;
end;

str = "Haarfestiger";
puts(without(str, 0));
puts(without(str, 1));
puts(without(str, 2));
puts(without(str, 3));
puts(without(str, str.length()));

def without!(s, n)
  # Falls n < 1 ist können wir direkt zurückkehren
  if n > 0 && n < s.length() then
    # Da wir s verändern, müssen wir uns anders merken, welche
    # Buchstaben wir entfernen und welche nicht
    buchstabe = 1;
```

```
    i = 0;

    while i < s.length() do
      if buchstabe % n == 0 then
        s[i, 1] = "";
        # Hier dürfen wir i nicht hochzählen, weil an der Stelle
        # i nun ohnehin der nächste Buchstabe steht
      else
        i = i + 1;
      end;

      buchstabe = buchstabe + 1;
    end;
  end;
end;

str = "Haarfestiger";
without!(str, 0);
puts(str);

str = "Haarfestiger";
without!(str, 1);
puts(str);

str = "Haarfestiger";
without!(str, 2);
puts(str);

str = "Haarfestiger";
without!(str, 3);
puts(str);

str = "Haarfestiger";
without!(str, str.length());
puts(str);
```

2 - Comicdatenbank

Comicserie

id	titel	webseite
1	Invader Zim	http://www.operationimpendingdoom2.com

Comicschaffende

id	nachname	vorname
1	Vasquez	Jhonen
2	Graham	Ian

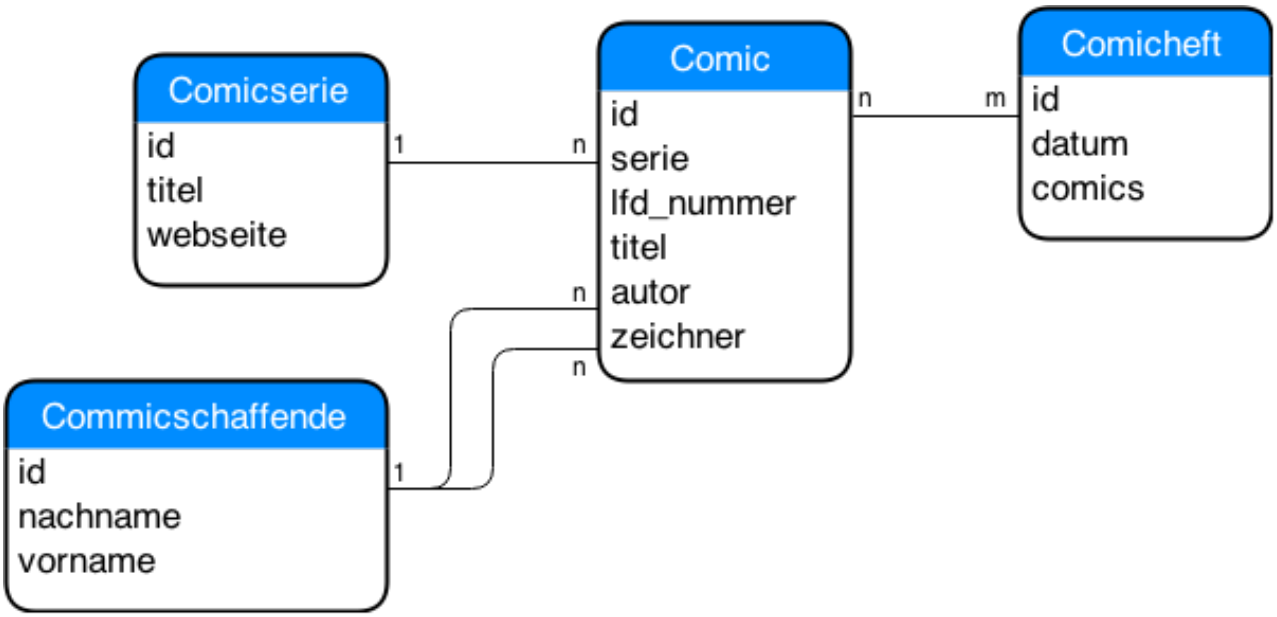


Abbildung 1: dbschema.png

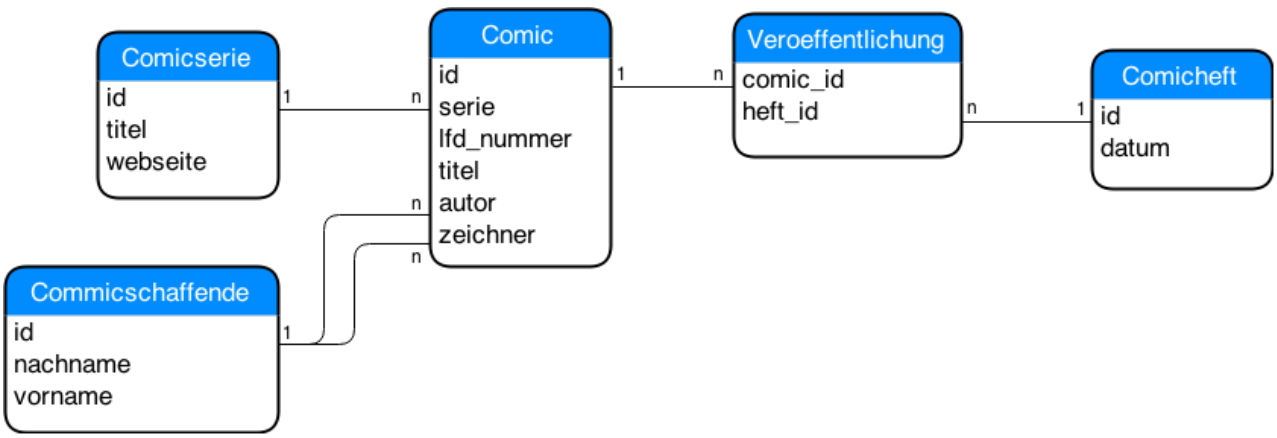


Abbildung 2: dbschema_normalized_(1).png

Comic

id	serie	lfd_nummer	titel	autor	zeichner
1	1	5	Attack of the Saucer Morons	1	2
2	1	6	Battle-Dib	1	2
3	1	7	Hamstergeddon	1	2

Comicheft

id	datum
1	20.05.2014
2	12.10.2013
3	32.11.1991

Veroeffentlichungen

comic_id	heft_id
1	1
1	2
2	2
3	2
3	3

```
SELECT Comic.lfd_nummer, Comic.titel FROM Comicserie INNER JOIN Comic ON Comicserie.id = Comic.serie WH
```

3 - Programmpunkttabelle

PP	n	r	z	s	Ausgabe
1	1337				
2		0			
3			1000		
4				1	
5		1			
6	337				
7			100		
4				3	
5		4			
6	37				
7			10		
4				3	
5		7			
6	7				
7			1		
4				7	
5		14			
6	0				

```
 7  |      |      |    0 |      |
 8  |      |      |      |      |
 9  |      |      |      |      |  14
```

Das Programm berechnet die Quersumme von n und gibt diese aus. Die Quersumme einer Zahl ist die Summe aller vorkommenden Ziffern der Zahl.

4 - Aufzählen und Überprüfen

```
def is_pythagorean_triplet(a, b, c)
  return a**2 + b**2 == c**2
end

for a in 1..100 do
  for b in 1..100 do
    for c in 1..100 do
      if is_pythagorean_triplet(a, b, c) then
        puts("a=" + a.to_s + " b=" + b.to_s + " c=" + c.to_s)
      end
    end
  end
end


# Wir können das Aufzählen von c sparen, wir c auch recht einfach
# mit Hilfe der Wurzelfunktion berechnen können. Für den Test, ob
# es sich dann um ein pythagoreisches Tripel handelt, müssen wir
# die Eigenschaft dann nur noch für das in Frage kommende c überprüfen.
#
# Bsp.: a = 3, b = 5
# Dann ist a * a + b * b = 9 + 25 = 34
# Somit ergibt sich Math.sqrt(34) = 5.830951894845301
# und das mögliche c = Math.sqrt(34).to_i = 5
# Allerdings ist 5 * 5 = 25 != 34.

for a in 1..100 do
  for b in 1..100 do
    c = Math.sqrt(c_hoch_2).to_i
    if c<=100 && c*c == c_hoch_2 then
        puts("a=" + a.to_s + " b=" + b.to_s + " c=" + c.to_s)
    end
  end
end
```

5 - Programmieren mit Arrays

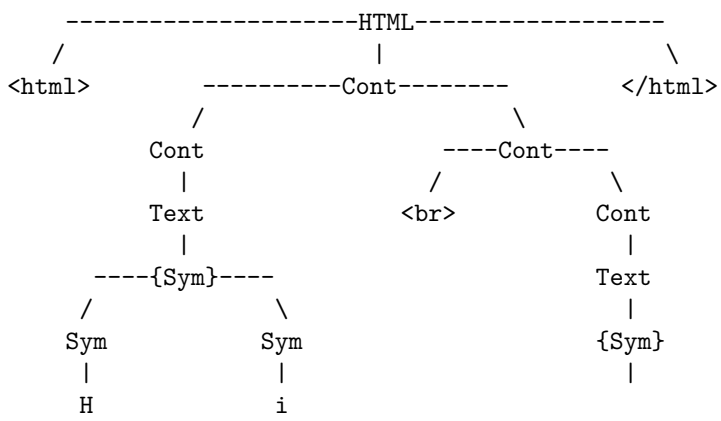
```
def double(a)
  b = []
  for i in 0..a.size-1 do
    b = b + a[i,1] * 2
  end
  return b
end
```

```
def double!(a)
  for i in 0..a.size-1 do
    a[i*2,0] = a[i*2,1]
  end
end

def to_table(a, n)
  b = Array.new(a.size)
  for i in 0..a.size-1
    b[i] = Array.new(n,a[i])
  end
  return b
end
```

6 - Einfaches HTML

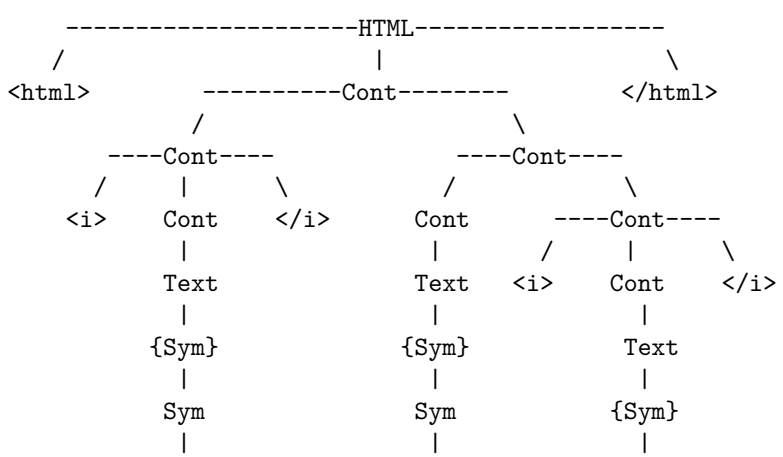
Das Wort `<html>Hi<br></html>` kann abgeleitete werden:



Man beachte, dass die optionale Wiederholung `{Sym}` hinter dem `<br>`-Tag auch zum leeren Wort abgeleitet werden kann.

Das Wort `<html><i><p>Hi</i></p></html>` kann nicht abgeleitet werden, da die Tags `<i>` und `<p>` Klammern sind, welche immer mit dem passenden Tag `</i>` bzw. `</p>` geschlossen werden müssen (2. und 3. Regel für `Cont`).

Das Wort `<html><i>H</i>e<i>y</i></html>` kann abgeleitete werden:



```
H                e              Sym
                                 |
                                 y
```

HTML2 beschreibt einen andere Sprache (insbesondere eine Obermenge der Sprache von HTML). Es wird nicht garantiert, dass die Tags durch das passende Tag geschlossen werden (also durch bzw. durch). Außerdem kann bei HTML2 auch das
-Tag mittels </br> geschlossen werden. Mit HTML2 wäre das zweite Beispiel aus (a) ableitbar.

Ändere die Regel für Text wie folgt ab:

```
Text ::= Sym Text |
```

Da aber auch schon Cond optionale Wiederholungen über die Regel Cont ::= Cont Cont zulässt, wäre auch folgende Änderunge ausreichend:

```
Text ::= Sym |
```

Das leere Wort ist in diesem Fall abenfalls notwendig, da Cont sonst nicht auf das leere Wort abgeleitet werden kann. Alternativ könnte die Regel für das leere Wort auch zu Cont hinzu gefügt werden.

7 - Breitengrade

```
# Da das Gradsymbol nicht so einfach gematcht werden kann, verwenden wir ersatzweise ein kleines o

breitengrad_regexp = /([0-9]|[1-8][0-9]|90)[o]([0-9]|[1-5][0-9])'([0-9]|[1-5][0-9])''(N|S)/


breitengrad_regexp = /([0-9]|[1-8][0-9]|90)[o]([0-9]|[1-5][0-9])'([0-9]|[1-5][0-9])''(N|S)/

print("Breitengrad: ")
str = gets.chop
m = breitengrad_regexp.match(str)
while m==nil || m[0]!=str do  # Wurde wirklich der gesamte String gematcht?
  puts("Kein gueltiger Breitengrad, bitte versuche es erneut.")
  print("Breitengrad: ")
  str = gets.chop
  m = breitengrad_regexp.match(str)
end

dez_grad = m[1].to_f+m[2].to_f/60+m[3].to_f/3600
if m[4]=="S" then
  dez_grad = -dez_grad
end
puts("Als Dezimalgrad: "+dez_grad.to_s)
```