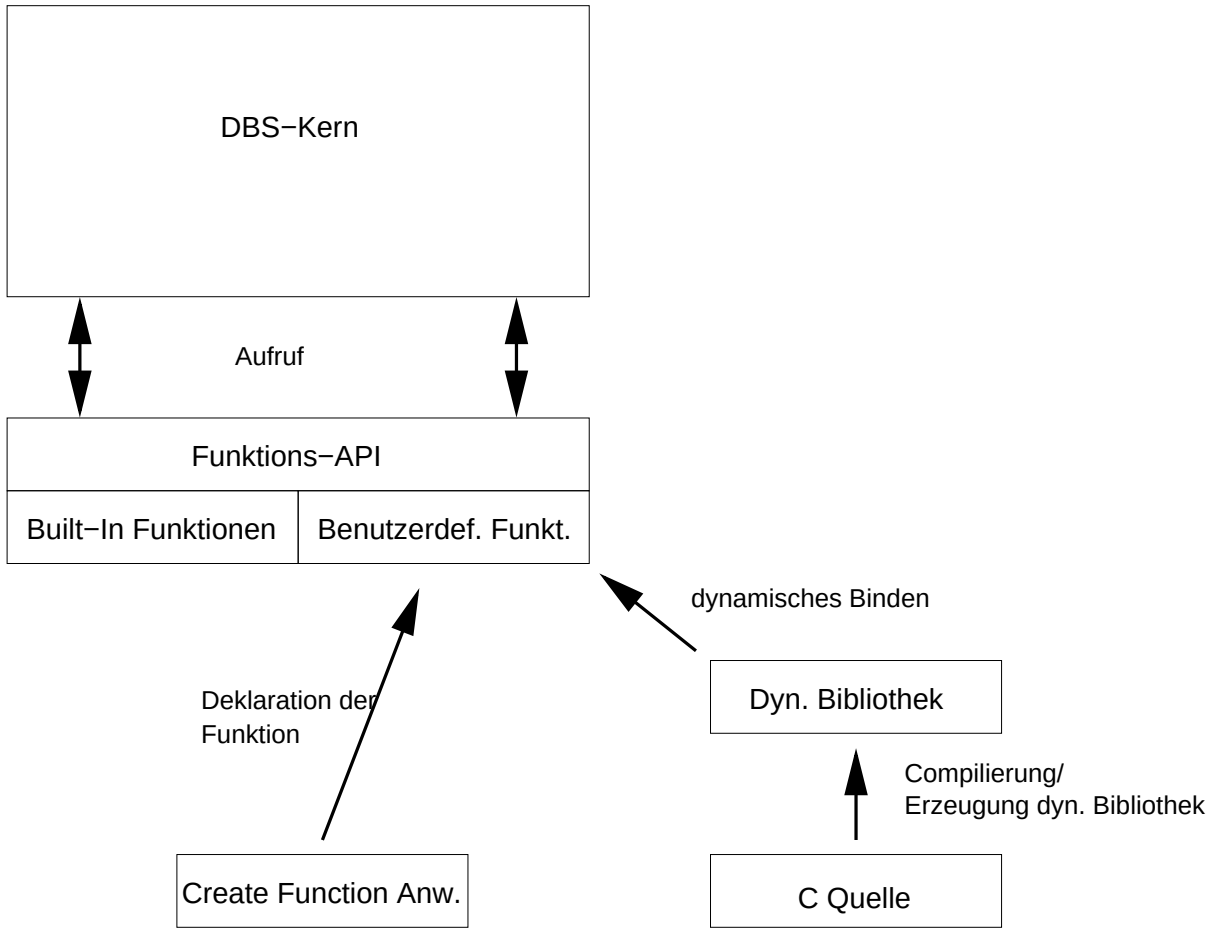
User Defined Functions

Hierbei handelt es sich um Funktionen,

- die vom Benutzer erzeugt und
- unter Verwendung einer speziellen Anweisung (`CREATE FUNCTION`) in eine Datenbank integriert werden.
- Für die Programmierung von UDFs werden PL/SQL, Java oder C eingesetzt.
- Die UDFs erweitern den Vorrat an SQL-Funktionen.
- Prinzipiell kann es sich um eine skalare Funktion, eine Aggregatsfunktion oder eine Tabellenfunktion handeln.

Integration vo externen UDFs in eine DB (1)



Integration von externen UDFs in eine DB (2)

Die Integration besteht aus zwei Schritten:

1. Implementierung

   Auf Basis eines standardisierten APIs wird eine C-Funktion erstellt, übersetzt und in einer dynamischen Bibliothek (shared object) abgelegt.

2. Deklaration

   Mit Hilfe einer `create function` Anweisung wird die UDF dem Datenbanksystem bekannt gemacht. Hierdurch steht eine entsprechende SQL-Funktion zur Verfügung.

   Bei erstmaliger Benutzung der UDF/SQL-Funktion wird die dynamische Bibliothek an den DB-Prozess gebunden und die betreffende C-Funktion aufgerufen.

Implementierung in C

Ein ganz simples Beispiel für Oracle: UDF zum Quadrieren von Integers!

```
int csquare(int n)
{
   return n*n;
}
```

Erzeugung der dynamischen Bibliothek:

```
gcc -c -fpic -Wall square.c
gcc -shared -o square.so square.o
```

Deklaration der externen UDF

In Oracle sind zwei Schritte notwendig:

1. Definition einer Library, die die Implementierung der UDF enthält (für verschiedene UDFs in gleicher Library nur einmal erforderlich):

```
create or replace library square as
'/usr/lib/oracle/xe/app/oracle/product/10.2.0/server/lib/square.so';
```

2. Deklaration der UDF:

```
create or replace function csquare(n in pls_integer) return pls_integer is external
library square
name "csquare"
parameters (n sb4, return sb4);
```

Nutzung der externen UDF

Jetzt in jeder SQL-Anfrage nutzbar, z.B. in der `Select`- oder `Where`-Klausel.

```
select csquare(4) from dual;
```

UDF in PL/SQL

Syntax:

```
CREATE [OR REPLACE] FUNCTION funktionsname(parameter) RETURN datentyp IS
    Variablendeklarationen
BEGIN
    Anweisungen
    ....
    RETURN Ausdruck;
EXCEPTION
    Exception-Handler
    ....
END;
```

☞ Keine SQL-Anweisungen erlaubt, damit die Funktion innerhalb einer Select-Anweisung verwendet werden kann.

Großes Beispiel für UDFs: Approximatives Pattern Matching

Zur Erinnerung: Sei M eine Menge. Eine Funktion $d : M \times M \longrightarrow \mathbb{R}$ heißt *Metrik*, wenn die folgenden Bedingungen erfüllt sind:

- $d(x, y) \geq 0$ für alle $x, y \in M$
- $d(x, y) = 0 \Leftrightarrow x = y$ für alle $x, y \in M$
- $d(x, y) = d(y, x)$ für alle $x, y \in M$
- $d(x, z) \leq d(x, y) + d(y, z)$ für alle $x, y, z \in M$.

(M, d) ist dann ein *metrischer Raum*.

Allgemeines Problem beim Approximativen Pattern Matching

Gegeben seien ein String pat, ein String $text$, eine Metrik d für Strings und ein ganze Zahl $k \geq 0$.

Man finde alle Substrings y von $text$ mit $d(pat, y) \leq k$.

Bemerkungen:

- Für $k = 0$ erhält man das exakte String-Matching Problem
- Das Problem ist zunächst ein “abstraktes” Problem, da nichts über die Metrik d ausgesagt wird.
- Zur Konkretisierung und zur Entwicklung von entsprechenden Algorithmen müssen zunächst sinnvolle Metriken betrachtet werden.

Hamming-Distanz

Für zwei Strings x und y mit $|x| = |y| = m$ ergibt sich die *Hamming-Distanz (Hamming Distance)* durch:

$$d(x, y) = |\{1 \leq i \leq m | x[i] \neq y[i]\}|$$

Bemerkungen:

- Die Hamming-Distanz ist die Anzahl der Positionen, an denen sich x und y unterscheiden. Sie ist nur für Strings gleicher Länge definiert.
- Wird die Hamming-Distanz als Stringmetrik verwendet, so spricht man auch von "string matching with k mismatches".

Beispiel 3.1. *Die Hamming-Distanz der Strings $abcabb$ und $cbacba$ beträgt 4.*

Editierdistanz, Levenstein-Metrik (1)

- Für zwei Strings x und y ist die *Editierdistanz (Edit Distance)* $edit(x, y)$ definiert als die kleinste Anzahl an Einfüge- und Löschoperationen, die notwendig sind, um x in y zu überführen.
- Läßt man zusätzlich auch die Ersetzung eines Symbols zu, so spricht man von einer *Levenstein-Metrik (Levenshtein Distance)* $lev(x, y)$.
- Nimmt man als weitere Operation die Transposition (Vertauschung zweier benachbarter Symbole) hinzu, so erhält man die *Damerau-Levenstein-Metrik* $dlev(x, y)$.

Editierdistanz, Levenstein-Metrik (2)

- Offensichtlich gilt stets $dlev(x, y) \leq lev(x, y) \leq edit(x, y)$.
- Die Damerau-Levenstein-Metrik wurde speziell zur Tippfehlerkorrektur entworfen.
- Wird die Levenshtein-, Damerau-Levenshtein-Metrik oder die Editierdistanz verwendet, dann spricht man auch von “string matching with k differences” bzw. von “string matching with k errors”.

Beispiel: Levenshtein-Metrik

Für $x = abcabba$ und $y = cbabac$ gilt:

$edit(x, y) = 5$

$abcabba \longrightarrow bcabba \longrightarrow cabba \longrightarrow cbba \longrightarrow cbaba \longrightarrow cbabac$

$dlev(x, y) = lev(x, y) = 4$

$abcabba \longrightarrow cbcabba \longrightarrow cbabba \longrightarrow cbaba \longrightarrow cbabac$

$abcabba \longrightarrow bcabba \longrightarrow cbabba \longrightarrow cbabab \longrightarrow cbabac$

Berechnung der Stringdistanz

Gegeben seien zwei Strings x und y. Man ermittle $edit(x, y)$ bzw. $lev(x, y)$ bzw. $dlev(x, y)$ sowie die zugehörigen Operationen zur Überführung der Strings.

Bemerkungen:

- Wenn x und y Dateien repräsentieren, wobei $x[i]$ bzw. $y[j]$ die i-te Zeile bzw. j-te Zeile darstellt, dann spricht man auch vom File Difference Problem.
- Unter UNIX steht das Kommando `diff` zur Lösung des File Difference Problems zur Verfügung.
- Da die Metriken $edit$, lev und $dlev$ sehr ähnlich sind, wird im folgenden nur die Levenstein-Metrik betrachtet.
- Algorithmen für die anderen Metriken erhält man durch einfache Modifikationen der folgenden Verfahren.
- Im folgenden sei $m = |x|$ und $n = |y|$ und es gelte $m \leq n$.

Der algorithmische Ansatz zur Berechnung (1)

- ☞ Lösungsansatz: dynamische Programmierung
- ☞ genauer: berechne die Distanz der Teilstrings $x[1 \ldots i]$ und $y[1 \ldots j]$ auf der Basis bereits berechneter Distanzen.

Der algorithmische Ansatz zur Berechnung (2)

Die Tabelle LEV sei definiert durch:

$$LEV[i,j] := lev(x[1 \ldots i], y[1 \ldots j]) \text{ mit } 0 \leq i \leq m, 0 \leq j \leq n$$

Die Werte für $LEV[i,j]$ können mit Hilfe der folgenden Rekursionsformeln berechnet werden:

- $LEV[0,j] = j$ für $0 \leq j \leq n$, $LEV[i,0] = i$ für $0 \leq i \leq m$
- $LEV[i,j] =$
$$\min\{\, LEV[i-1,j] + 1,$$
$$LEV[i,j-1] + 1,$$
$$LEV[i-1,j-1] + \delta(x[i], y[j])\}$$
- $\delta(a,b) = \begin{cases} 0 & \text{falls } a = b \\ 1 & \text{sonst} \end{cases}$

Bemerkungen zum Lösungsansatz

- Die Rekursionsformel spiegelt die drei Operation Löschen, Einfügen und Substitution wider.
- Die Stringdistanz ergibt sich als $LEV[m, n]$.
- Möchte man nur die Stringdistanz berechnen, so genügt es, sich auf Stufe i der Rekursion die Werte von LEV der Stufe $i - 1$ zu merken.
- Benötigt man die zugehörigen Operationen, speichert man LEV als Matrix und ermittelt die zugehörigen Operationen in einer “Rückwärtsrechnung”.

Der Algorithmus

for $i := 0$ **to** m **do** $LEV[i, 0] := i$ **end**
for $j := 1$ **to** n **do** $LEV[0, j] := j$ **end**
for $i := 1$ **to** m **do**
  for $j := 1$ **to** n **do**

$$LEV[i, j] := \min\{\ LEV[i-1, j] + 1, LEV[i, j-1] + 1, \\ LEV[i-1, j-1] + \delta(x[i], y[j])\}$$

  end
end
return $LEV[m, n]$

Beispiel

Darstellung von LEV als Matrix für $x = cbabac$ und $y = abcabbbaa$:

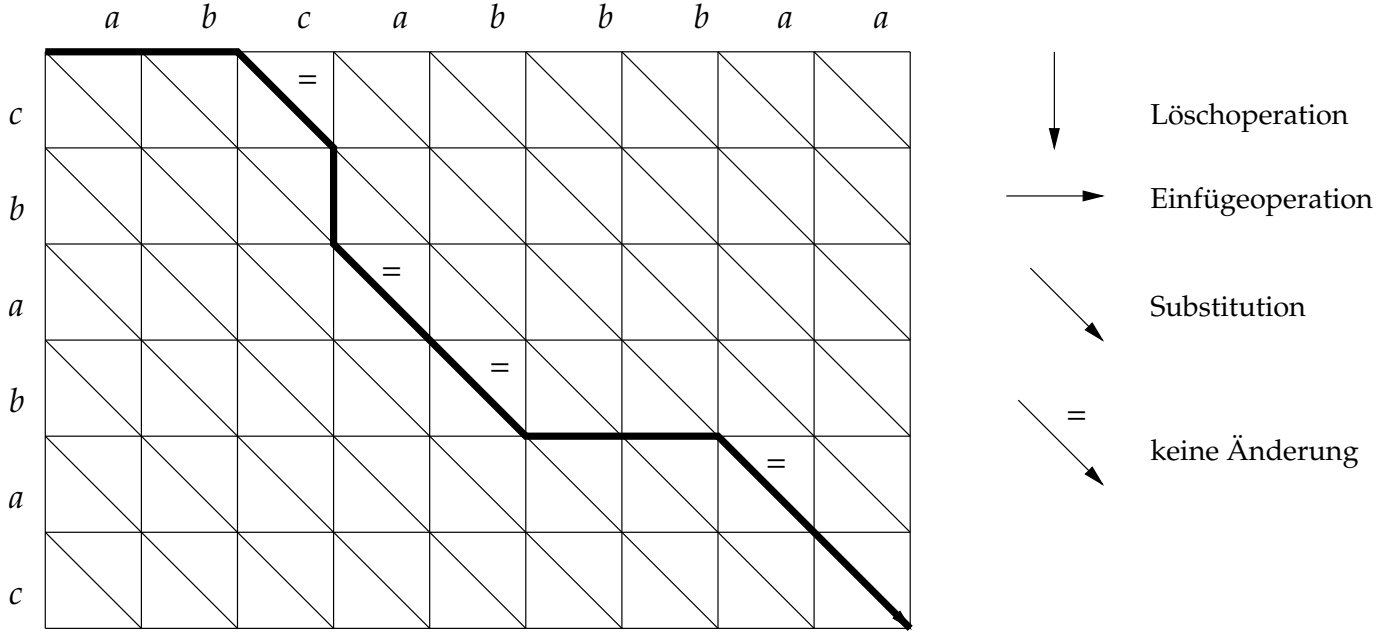
		a	b	c	a	b	b	b	a	a
	0	1	2	3	4	5	6	7	8	9
c	1	1	2	2	3	4	5	6	7	8
b	2	2	1	2	3	3	4	5	6	7
a	3	2	2	2	2	3	4	5	5	6
b	4	3	2	3	3	2	3	4	5	6
a	5	4	3	3	3	3	3	4	4	5
c	6	5	4	3	4	4	4	4	5	5

Die zugehörigen Umwandlungen lauten:

$$cbabac \longrightarrow ababac \longrightarrow abcabac \longrightarrow abcabbac \longrightarrow abcabbbac \longrightarrow abcabbbaa$$

Veranschaulichung

Die Berechnung der Stringdistanz kann als Pfad in einem Graphen veranschaulicht werden.



Jeder Weg entspricht einer möglichen Umwandlung. Der dargestellte Pfad entspricht der folgenden (nicht optimalen) Umwandlung:

$cbabac \longrightarrow acbabac \longrightarrow abcbabac \longrightarrow abcabac \longrightarrow abcabbac \longrightarrow abcabbbac \longrightarrow abcabbbaa$

Ein kürzester Weg würde einer optimalen Umwandlung entsprechen!

Aus der Rekursionsformel und den Bemerkungen folgt:

Die Stringdistanz (für $edit$, lev und $dlev$) kann in Zeit $O(mn)$ und Platz $O(m)$ berechnet werden.