o++o

Die einfachste Programmiersprache

Klaus Benecke

Vorwort

Wir denken, dass o++oPS (ottoPS) die einfachste Programmier- bzw. Anfragesprache ist. Das bedeutet nicht, dass o++oPS aus sehr einfachen Konzepten besteht. Es bedeutet, dass es relativ leicht ist, es zu benutzen. o++oPS besteht aus komplexen Operationen, aber das Lösen von Problemen mit ihr scheint einfacher zu sein als mit anderen Programmiersprachen der gleichen Ausdruckskraft. Mit der natürlichen Sprache verhält es sich ähnlich. Die deutsche Sprache ist nicht einfach, aber bis zu einem bestimmten Grad können bereits Kindergartenkinder sie beherrschen.
Die Grundidee hinter unserer besten Operation – der Strichlistenoperation stroke- ist wahrscheinlich leichter zu verstehen als der Multiplikationsalgorithmus für Dezimalzahlen. Wir denken, dass die Konzepte von o++oPS schwer zu formalisieren sind, sie können aber durch relativ einfache Algorithmen beschrieben werden, so dass in Zukunft fast jeder Nutzer (=Otto Normalverbraucher) sie benutzen kann.
Ist o++oPS eine Programmiersprache?
o++oPS wurde als Endnutzersprache entworfen; nicht um komplexe Datenbankbetriebssysteme oder Compiler zu erstellen. Sie ist für die vielfältigen Alltagsprobleme vom Normalverbraucher gedacht. Alltagsprobleme sind zunächst (ad hoc) Anfragen an Tabellen oder Dokumente oder Kollektionen von Tabellen (Datenbanken) oder von Dokumenten; zweitens Berechnungen wie Bestimmung von Funktionswerten, Nullstellen, Extrema, Rechnen mit Matrizen, Stücklisten-probleme, … . Die wesentlichen neuen Ideen von o++oPS verglichen mit anderen Ansätzen sind mit neuen Operationen für Wiederholgruppen verbunden.

Table of Contents

Einführung

Bis jetzt gibt es keine allgemein akzeptierte Computersprache, die gut genug ist für Otto Normalverbraucher oder wenigstens die nutzerfreundlich und ausdrucksstark genug ist, um an der Schule gelehrt zu werden. Die einzigen "Sprachen", die breit genutzt werden, stehen mit der Nutzung von Suchmaschinen im Zusammenhang. Hier hat der Nutzer lediglich ein, zwei oder drei Worte zu schreiben (ein sehr einfaches Programm) und das System findet Tausend, Millionen oder mehr Ergebnisdokumente. Da der Nutzer in diesem Kontext nicht in der Lage ist, Programme mit einer höheren Selektivität zu schreiben, hofft er dass Baidu, Google und Co. genau die Seiten finden, an denen er interessiert ist. Das stimmt häufig, da Google nach den "wichtigsten" Seiten sucht. Schnippsel der Seiten stehen am Anfang der fast unendlichen Resultatmenge. Aber was kann der Nutzer tun, wenn er eher an einem Dokument mit der Ranknummer 100 1000 oder 100000 interessiert ist? Ferner was kann der Nutzer tun, wenn er eine genaue Vorstellung von tausend gesuchten Dokumenten hat, aber er möchte lediglich kleine Teildokumente dieser zu einem neuen Dokument vereinigt haben?
Diese Fragen sind nicht leicht zu beantworten und zu realisieren, die Sprache o++oPS (ottoPS) zielt darauf, Teile dieser Probleme zu lösen. Wir fassen die wesentlichen Entwurfskriterien und Anforderungen an eine Endnutzersprache oder ein Datenmodell mit entsprechenden Operationen zusammen:

1. Es sollte auf leicht nutzbaren Konzepten mit einer einfachen Syntax basieren.
2. Es sollte ausdrucksstark sein.
3. Es sollte um neue Operationen und Funktionen erweiterbar sein.
4. Es sollte eine präzise Semantik, die auf Algorithmen basiert, haben.
5. Es sollte Anfragen an Tabellen und Dokumente zulassen.
6. Es sollte Anfragen an (ganze) Datenbanken und Kollektionen von Dokumenten (IR-Systeme) zulassen.
7. Es sollte Berechnungen durch naive (brute force) Algorithmen zulassen.
8. Es sollte nutzbar sein durch Menschen mit wenig Interesse für Mathematik und Informatik (Programmierung durch Bauchentscheidungen).
9. Es sollte ausgefeilte Konzepte für breite Anwenderklassen zur Verfügung stellen, so dass es auch für Nutzer mit großem Interesse für Mathematik und Informatik geeignet ist (Programmierung durch Denken).
10. Es sollte Einzeldaten- und Massendatenoperationen adäquat integrieren.
11. Das Resultat von Massendatenoperationen sollte so klein wie möglich sein (Z.B. kein kartesisches Produkt, falls möglich).
12. Es wäre schön, wenn graphische Möglichkeiten auf der Basis von Tabellen bereitgestellt würden.
13. Die Operationen bzw. Operationenfolgen sollten effizient implementierbar sein.
14. Es sollte möglich sein, wenigstens den Kern der Sprache zu optimieren.

o++oPS wurde mit diesen Prinzipien im Kopf Schritt für Schritt entworfen und implementiert. Sie startete als Datenbankanfragesprache für Tabellen mit Wiederholgruppen. Ein Satz mit Wiederholgruppen kann zu einem Merkmal nicht nur einen Wert, sondern auch mehrere Werte bzw. Subsätze enthalten. Ein Studentensatz kann beispielsweise einen Namen, einen Vornamen, ein Stipendium aber auch mehrere Hobbys oder mehrere (KURS, NOTE)-Paare enthalten. In gleicher Weise kann ein Teil eine Nummer oder einen Namen, eine Farbe aber mehrere direkte Unterteile oder Ebenen oder Kanten enthalten. Solche Subtupel (Teilsätze) können wieder Subtupel enthalten. Diese Wiederholgruppen existieren in der EDV seit mehr als 50 Jahren. Sie waren typisch für Magnetbanddateien, hierarchische Systeme wie IMS,… . Später wurden sie durch relationale Datenbanken, die nur flache Tabellen kennen, in Misskredit gebracht. Jetzt werden sie wieder breit

genutzt in XML, JSON, und NoSQL-Systemen. Dennoch gibt es unserer Meinung nach keine allgemein akzeptierte Computersprache, die diese allgemeineren Strukturen vernünftig manipulieren kann.
Mit dem Aufkommen von XML konnten wir unsere Operationen auf die neuen Möglichkeiten beliebiger Tags und dem Auswahloperator (|) verallgemeinern. Daher können wir nicht nur TABellen sondern auch DokuMENTe handhaben. Wir führten den Begriff **Tabment** ein. Ein Tabment kann als eine abstrakte (syntaxunabhängige) Präzisierung des XML-Dokuments verstanden werden. Schrittweise haben wir unsere Sprache **o++oPS** (otto ProgrammierSprache) verbessert. Wir haben binäre Suchbäume in Tabmente eingeführt. Daher konnten wir für mehrere Operationen die Effizienz wesentlich verbessern.
Indexe können ebenfalls als Tabmente angesehen werden. Nun ist es beispielsweise möglich, Anfragen mit geringer Ausgabemenge an eine Wiki-Datei mit 10000 Rezepten in 0.4 Millisekunden zu beantworten (auf einem Laptop mit i7-Intel Prozessor). Unsere Sprache o++oPS ist in OCaml implementiert. Einige Schlüsselworte von o++oPS sind deutsch ('gib' anstelle von SELECT, 'mit' anstelle von WHERE,...), da sie kürzer sind als die entsprechenden englischen Worte, aber die meisten Schlüsselworte sind englisch. Das scheint wichtig zu sein, da Smartphones ein kleines Display haben.
Wenn wir uns bestimmte Anfragen ansehen, hat man den Eindruck, dass sich das o++o Datenmodell zum relationalen Datenmodell verhält wie die Dezimalzahlen zum Römischen System. Römische Zahlen sind leichter zu verstehen bei kleinen Zahlen, aber Berechnungen bei etwas größeren und großen Zahlen sind sehr umständlich.
Als häufigstes Argument gegen eine Endnutzersprache begegnete uns die Frage: Wird der Kunde für ein Produkt bezahlen, für das er auch noch etwas lernen muss?
Wir denken 100 Jahre zurück.
Das Auto war erfunden, aber die meisten Menschen fuhren Eisenbahn oder Pferdekutsche, wenn sie entsprechend vermögend waren. Man glaubte, dass es nie mehr als 1 Million Autos auf der Erde geben würde, da es nicht mehr als 1 Millionen Chauffeure geben kann. Keiner glaubte, dass der durchschnittliche Mensch eine Fahrerlaubnis machen würde.
Heute glauben auch wenige Menschen, dass jemand ein System kauft, für das er Stunden oder Tage lernen muss.
Wir halten folgende Argumente entgegen:

1. Fast alle Menschen der Erde mussten mehrere Jahre lernen, um die Einzeldatenoperationen Addition, Multiplikation, Division und Differenz für die einzelnen Zahlenbereiche zu verstehen. Sind Massendatenoperationen wie Selektion, Berechnungen, Umstrukturierung, Mischen von Tabellen,... nicht genauso wichtig?
2. Selbst wenn man ein "einfaches" Textverarbeitungsprogramm wie MS Word nutzt, muss man bezahlen und man benötigt Wochen und Monate, bevor man alle Möglichkeiten beherrscht.
3. Eine gute Programmiersprache ist viel leichter zu erlernen als deutsch oder eine Fremdsprache. Das Beherrschen einer Computersprache kann in Zukunft Teil der Allgemeinbildung werden.
4. Eine gute Programmiersprache erlaubt es, viele Probleme präziser mit weniger Missverständnissen als bei der natürlichen Sprache zu formulieren.
5. Es ist nicht nötig zu erklären, welche Vorteile jemand mit einer eigenen Fahrerlaubnis oder eigenem Auto hat. Wenn er selbst Probleme mit dem Rechner lösen kann, dann erhöht das die Qualität der Computernutzung, da er die Ergebnisse besser interpretieren kann. Er benötigt keinen Informatiker (Chauffeur). Daher gibt es weniger Kommunikationsprobleme, und er spart die Kommunikationszeit und die Kosten für den Informatiker.
6. Wenn er präzise Anfragen stellen kann, hat er wesentlich geringere Anfrageergebnisse, und

er spart manuellen Suchaufwand. Damit erhöht sich auch das Niveau der Arbeit.

Das Erste was der Nutzer von o++oPS verstehen muss, ist der Begriff des Schemas und der Zusammenhang zu den zugehörigen Tabellen oder Schemen. Danach ist es nicht schwierig, die Anfragemöglichkeiten der Kapitel 4, 5 und 6 zu erfassen. Alle diese Operationen erlauben eine kompakte und leicht lesbare Formulierung von (komplexen) Anfragen. Sie sind auf geschachtelte Mengen (m), Multimengen (bags) (b) und Listen (l) anwendbar und sie sind immer noch neu in der Datenbankwelt.
Berechnungen können als eine "hierarchische Mapfunktion" verstanden werden. Die Umstrukturierungsoperation stroke ist sehr ausdrucksstark, da sie mit Sortierung (m, b) dem Entfernen von Duplikaten (m), Aggregationen (++, min, max, ++1, ++:, ||, &&, **, streuung, variance,...) und der Vereinigung verbunden ist.
Wir kennen keine andere Umstrukturierungsoperation in einem kommerziellen Produkt, die es erlaubt, eine gegebene Hierarchie nur durch Angabe eines Schemas oder der TTDs (Tabment Typ Definition) der gewünschten Tabelle umzustrukturieren. Obwohl unsere Operationen nur sequentiell angewendet werden, decken sie ein breites Anwendungsfeld ab.
In Kapitel 7 wird ein "natürlicher Verbund" (natural Join) und seine ungeschachtelte und geschachtelte Nutzung vorgestellt. Es wird klar, dass wir kein kartesisches Produkt benötigen und auch nicht den flachen relationalen Join. Eine einfache Form von Rekursion wird in Kapitel 8 beschrieben. Mit dieser Endnutzerrekursion können entsprechende Anfragen mit einem minimalen Lernaufwand realisiert werden.
Nachdem in Kapitel 9 gezeigt wurde, dass das Ausgeben von "Hello World" nicht nur eine syntaktische Frage ist, wird in Kapitel 10 versucht zu zeigen, dass o++oPS für alle Fächer in der Schule nützlich ist, aber besonders für Mathematik und Informatik. Es wird verdeutlicht, dass auch Schüler der 9. und 10. Klasse die Anwendungen der Differential- und Integralrechnungen beherrschen können. Ferner wird argumentiert, warum der bekannte Algorithmus der Division aus dem Schullehrplan genommen werden sollte.
Das nächste Kapitel verallgemeinert den „natural join". Im Wesentlichen wird hier die Restrukturierungsoperation auf mehrere Inputtabellen erweitert. Dabei werden weder das kartesisches Produkt noch (versteckte) Joinbedingungen benötigt.
In Kapitel 12 werden neue Lösungen für Stücklisten- und ähnliche Probleme vorgestellt. Kapitel 13 beschreibt einige Vorteile von o++oPS im Kontext eines wohl strukturierten e-Buchs. In Kapitel 14 werden einige Nachteile von heutigen Suchmaschinen präsentiert. Es wird argumentiert, dass jedes Informationssystem auf einer ausdrucksstarken Anfragesprache basieren sollte. Hinter jeder Klickfolge sollte ein lesbares Programm stehen. Kapitel 16 enthält einige Anfragen, deren Ergebnis als Graphik interpretiert werden kann. Grob gesagt, enthält jede Ergebnistabelle (bzw. jedes Tupel von Tabellen) die Koordinaten der Punkte eventuell mit einem Farbwert angereichert. Wie Programme aussehen könnten, wenn sie einfache Videos repräsentieren, wird am Ende des Kapitels gezeigt. Ein Vergleich mit SQL erleichtert den SQL-Kennern den Zugang zur Sprache o++oPS im letzten Kapitel. Wer Fragen zur Optimierung von Anfragen oder zur Spezifikation der Operationen hat oder bestimmte Dateikonzepte näher kennenlernen will, sollte das Buch „o++oPS The simplest Programming Language“ nutzen.
Danksagungen: Ich danke den folgenden Mitstreitern für ihre wertvollen Beiträge zu unserem System bzw. zu Vorgängersystemen:

- Wolfgang Reichstein für die erste Implementation der Umstrukturierungsoperation in einem Schritt in C für HSQ-Dateien
- Dmitri Schamschurko für die erste Implementation des Rumpfes des “gib-aus-mit”-Konstrukts in CAML-Light
- Martin Schnabel für das Konzept und die Implementation von Unterprogrammen und weiterer Konzepte

- Rolf Aretz für die Implementation von H2O-Dateien, einem Vorgänger des DIA-Konzepts
- Andreas Hauptmann für die Verbesserung von vielen Konzepten, die den Entwurf und die Effizienz betreffen, insbesondere für Anfrageoptimierungskonzepte
- Weiter gilt mein Dank Mirko Otto, Stephan Schenkl und Susanne Pape für ihre Unterstützung des o++o Projekts und Thomas Kudraß für die Verbesserungsvorschläge im SQL-Kapitel

1 Strukturierte Tabellen

Das Wichtigste, was der Leser zuerst verstehen muss, ist der Begriff des Schemas mit den Tabellen bzw. Dokumenten, die hierdurch beschrieben werden. Alle Spaltennamen einer Tabelle werden oft als Schema oder Kopf betrachtet. Spaltennamen sind notwendig, um die entsprechenden Werte richtig zu verstehen. Wenn wir strukturierte Tabellen betrachten, ist es vorteilhaft, die Spaltennamen um entsprechende Kollektionssymbole zu erweitern; Z.B. L für Liste.

NAME,	GEBORENIN,	(TAT,	JAHR l)l
Nicolaus Otto	Taunus (De)	Miterfinder des Ottomotors	1876
Otto Normalverbraucher	De	erlernt Autofahren	1960
		erlernt eine Programmiersprache	2020
Otto der Grosse	Altsachsen(De)	Zum Koenig von Deutschland gewaehlt	936
		Die Ungarn auf dem Lechfeld geschlagen	955
		Erster Kaiser des Heil. Roem. Reichs	962
Otto von Bismarck	Preussen(De)	mit Zuckerbrot und Peitsche-Politik	1871
		Emser Depesche	1870
		Erster Reichskanzler von Deutschland	1871
Otto von Guericke	MD (De)	Erfinder der Luftpumpe	1649
		Halbkugelversuch vor dem Kaiser	1654
Otto von Maehren	Maehren	heiratete Euphemia von Ungarn	1086

Tabment 1.1: ottos.tab

Die Tabelle (**TABMENT**=TABelle+dokuMENT) ottos.tab enthält eine Liste von 6 “Personen” und für jede Person eine Wiederholgruppe (TAT,JAHR l) – eine Liste von (TAT,JAHR)-Paaren. Eine Person hat 4 Spalten, ist aber ein Tripel. Die ersten beiden Komponenten sind vom Typ TEXT und die dritte Komponente ist eine Liste von Subtupeln – genauer Paaren. Wir nennen die Attributwerte einer Ebene **Segment**. Das NAME-Segment ist dasselbe wie das GEBORENIN-Segment.
Das dritte NAME-Segment ist:

NAME,	GEBORENIN
Otto der Grosse	Altsachsen(De)

Das erste TAT-Segment von Otto dem Großen ist:

TAT,	JAHR)
Zum Koenig von Deutschland gewaehlt	936

In hsq-Strukturen entspricht jede Zeile einem Segment. Die dritte Person bildet das dritte **Tupel** (den dritten Satz); es ist ein NAME-Tupel. (=GEBORENIN-Tupel):

NAME,	GEBORENIN,	(TAT,	JAHR l)
Otto der Grosse	Altsachsen(De)	Zum Koenig von Deutschland gewaehlt	936
		Die Ungarn auf dem Lechfeld geschlagen	955
		Erster Kaiser des Heil. Roem. Reichs	962

Da die TAT-Tupel keine Wiederholgruppen enthalten, ist ein TAT-(sub-)Tupel dasselbe wie ein TAT-Segment. Wenn wir die obige Tabelle durch eine normale unstrukturierte Tabelle ersetzen würden, würde jedes (NAME, GEBORENIN)-Paar in jeder Zeile erscheinen. Das Schema lautet dann NAME, GEBORENIN, TAT, JAHR l. Das heißt beispielsweise, dass (Otto der Große, Altsachsen(De)) dreimal erscheinen würde. Dann ist es nicht so einfach, die Personen der Tabelle zu zählen. Mit der obigen Tabelle sieht das entsprechende Programm folgendermaßen aus:

Anfrage 1.1: Wie viele Ottos sind in der Tabelle? (Wie viele Elemente (Einträge) enthält die äußere Liste.)

```
aus ottos.tab
++1
```

Hier und im Folgenden benutzen die folgenden Abkürzungen und Schlüsselworte:
aus: Tabelle(n), die angefragt werden.
++1: =Anzahl
ZAHL: ganze Zahl
gib: hat etwas Ähnlichkeit zum “SELECT” von SQL
avec: für die Selektion
sans: ebenfalls für die Selektion
:= Extension (erweitere die gegebene Tabelle um eine neue (komplexe) Spalte)
m: Menge: enthält nur verschiedene Elemente
b: bag: Ein Element kann mehrfach vorkommen.
l: Liste: Die Reihenfolge der Elemente ist von Bedeutung.

Das Ergebnis von Anfrage 1.1 ist eine einfache Tabelle:

```
6
```

oder

```
ZAHL
6
```

Das Schema der Tabelle muss kein Kollektionssymbol enthalten, da die Tabelle genau ein Element enthält. Aus dem gleichen Grund benötigen wir keine Kollektionssymbole in den nächsten beiden Anfragen. Wir erklären die Anfragen in den ersten beiden Kapiteln nicht im Detail. Wir wollen lediglich klären welche verschiedenen Schemata für Tabellen und Dokumente existieren.

Anfrage 1.2: Wie viele Personen und wie viele Taten sind in der Tabelle ottos.tab enthalten?

```
aus ottos.tab
gib ANZPER,ANZTAT
    ANZPER! ++1 NAME
    ANZTAT! ++1 TAT
```

Resultat:

```
ANZPER, ANZTAT
6       12
```

Anfrage 1.3: Gib mir die Person, die in Altsachsen geboren ist.

```
aus  ottos.tab
avec Altsachsen in GEBORENIN
gib  NAME
```

Resultat:

```
NAME
Otto der Grosse
```

Anfrage 1.4: Gesucht ist eine adlige Person.

```
aus  ottos.tab
avec Von in NAME
gib  NAME
```

Resultat:

```
Otto von Bismarck
```

Allerdings wäre es hier besser, nach allen adligen Personen zu fragen:

Anfrage 1.5: Sortiere alle Adligen.

```
aus  ottos.tab
avec Von in NAME
gib  NAMEm
```

Resultat (txt):

```
Otto von Bismarck
Otto von Guericke
Otto von Maehren
```

Um Platz auf dem Schirm oder dem Papier zu sparen, werden in geeigneten Situationen Kollektionen horizontal bzw. horizontal und vertikal arrangiert: (hsq)

```
Otto von Bismarck  Otto von Guericke  Otto von Maehren
```

Anfrage 1.6 a: Zähle alle Taten und bestimme die Anzahl für jedes Jahrhundert. Füge die Personen zu jedem Jahrhundert hinzu.

```
aus ottos.tab
JAHRHUNDERT:=JAHR : 100 zahl +1
gib ANZ_TAT,(JAHRHUNDERT,ANZ_TAT,NAMEm m)
    ANZ_TAT! ++1 TAT
```

Resultat: Tabelle mit drei Segmenttypen: ANZ_TAT, (JAHRHUNDERT, ANZ_TAT) und NAME als tab-Ausgabe mit WIDTH 65.

```
ANZ_TAT, (JAHRHUNDERT, ANZ_TAT,    NAMEm m)
12         10           3          Otto der Grosse
           11           1          Otto von Maehren
           17           2          Otto von Guericke
           19           4          Nicolaus Otto
                                   Otto von Bismarck
           20           1          Otto Normalverbraucher
           21           1          Otto Normalverbraucher
```

Anfrage 1.6 b: Wie Anfrage 1.6 a, nur dass die Personen mit Duplikaten erscheinen sollen.

```
aus ottos.tab
JAHRHUN:=JAHR : 100 zahl +1
gib ANZ_TAT,(JAHRHUN,ANZ_TAT,NAMEb m)
    ANZ_TAT!  ++1 TAT
```

Resultat: Tabelle mit Bag (Multimenge) (WIDTH 60)

```
ANZ_TAT,  (JAHRHUN, ANZ_TAT,   NAMEb m)
12         10       3          Otto der Grosse
                                Otto der Grosse
                                Otto der Grosse
           11       1          Otto von Maehren
           17       2          Otto von Guericke
                               Otto von Guericke
           19       4          Nicolaus Otto
                               Otto von Bismarck
                               Otto von Bismarck
                               Otto von Bismarck
           20       1          Otto Normalverbraucher
           21       1          Otto Normalverbraucher
```

Hier steht b für Bag (Multimenge). Bis jetzt haben wir nur Tabellen mit ineinander verschachtelten Kollektionen kennengelernt. In der folgenden Tabelle enthält jedes Tupel drei unabhängige Kollektionen.

```
NAME,           RESIDENZm,     FRAUl,     LANDl m
Otto der Grosse Magdeburg      Editha     Sachsen
                Memleben       Adelheid   Thüringen
                                          Bayern
                                          Franken
                                          Schwaben
                                          Italien
                                          Böhmen
                                          Holland
                                          Lothringen
                                          Friesland
Karl IV.        Prague         Margareth  Böhmen
                Tangermünde    Anna       Schlesien
                               Anna       Brandenburg
                               Elisabeth  Italien
                                          Ungarn
```

Tabment 1.2: kaiser.tab

In dieser Tabelle stehen 'Memleben' und 'Otto der Große' in Beziehung zueinander genau wie 'Adelheid' und 'Otto der Große'. Das heißt aber nicht, dass 'Adelheid' und 'Memleben' zueinander in Beziehung stehen. Daher ist die folgende Umstrukturierung bedeutungslos.

Anfrage 1.7: Anfrage mit leerem Resultat:

```
aus kaiser.tab
gib NAME,RESIDENZ,FRAU m
```

`gib NAME,RESIDENZm,FRAUm m` ist jedoch sinnvoll.

2 Strukturierte Dokumente

Wir benutzen die Bezeichnung Tabment für strukturierte Tabellen und strukturierte Dokumente. Daher kürzen wir den Typ eines Tabments durch TTD (Tabment Typ Definition) ab. Eine TTD vervollständigt die durch das Schema gegebene Information. Die TTD für die obige Tabelle ottos.tab lautet beispielsweise:

```
TABMENT! OTTOS
OTTOS! NAME,GEBORENIN,(TAT,JAHR l)m
NAME GEBORNENIN TAT! TEXT
JAHR! ZAHL
```

TEXT und ZAHL sind elementare Typen, die nicht weiter präzisiert werden müssen. Jedes Tabment mit Namen wird von einem Tag eingeschlossen, der aus dem Dateinamen verkürzt um den Suffix besteht. Daher kann unsere Tabelle auch im Dokumentstil oder auch als hierarchisch sequentielle Datei (HSQ) repräsentiert werden.

```
<OTTOS>
  <NAME>Nicolaus Otto</NAME>
  <GEBORENIN>Taunus (De)</GEBORENIN>
  <PAAR>
    <TAT>Miterfinder des Ottomotors</TAT>
    <JAHR>1876</JAHR>
  </PAAR>
  <NAME>Otto Normalverbraucher</NAME>
  <GEBORENIN>De</GEBORENIN>
  <PAAR>
    <TAT>erlernt Autofahren</TAT>
    <JAHR>1960</JAHR>
  </PAAR>
  <PAAR>
    <TAT>erlernt eine Programmiersprache</TAT>
    <JAHR>2020</JAHR>
  </PAAR>
  <NAME>Otto der Grosse</NAME>
  <GEBORENIN>Altsachsen(De)</GEBORENIN>
  <PAAR>
    <TAT>Zum Koenig von Deutschland gewaehlt</TAT>
    <JAHR>936</JAHR>
  </PAAR>
  <PAAR>
    <TAT>Die Ungarn auf dem Lechfeld geschlagen</TAT>
    <JAHR>955</JAHR>
  </PAAR>
  <PAAR>
    <TAT>Erster Kaiser des Heil. Roem. Reichs</TAT>
    <JAHR>962</JAHR>
  </PAAR>
  ...
</OTTOS>
```

Tabment 2.1: Tabelle ottos.tab in XML (Auszug)

```
Nicolaus Otto  Taunus (De)
 Miterfinder des Ottomotors  1876
Otto Normalverbraucher  De
 erlernt Autofahren  1960
 erlernt eine Programmiersprache  2020
Otto der Grosse  Altsachsen(De)
 Zum Koenig von Deutschland gewaehlt  936
 Die Ungarn auf dem Lechfeld geschlagen  955
 Erster Kaiser des Heil. Roem. Reichs  962
Otto von Bismarck  Preussen(De)
 mit Zuckerbrot und Peitsche-Politik  1871
 Emser Depesche  1870
 Erster Reichskanzler von Deutschland  1871
Otto von Guericke  MD (De)
 Erfinder der Luftpumpe  1649
 Halbkugelversuch vor dem Kaiser  1654
Otto von Maehren  Maehren
 heiratete Euphemia von Ungarn  1086
```

Tabment 2.2: ottos.tab im hsq-Format

Jetzt betrachten wir Teile eines größeren realen Dokuments:
“Grundgesetz der Bundesrepublik Deutschland”.
Eine mögliche TTD folgt:

```
GRUNDGESETZ! KAPITELl
KAPITEL! KNO,KTITEL,ARTIKELl l
ARTIKEL! ANO,ATITEL,(PNO,PARAGRAPH l)
PARAGRAPH ATITEL CTITEL! TEXT
KNO ANO PNO! ZAHL
```

Diese TTD ist in den Primärdaten sichtbar:

```
<GRUNDGESETZ>
<KAPITEL>
  <KNO>1</KNO>
  <KTITEL>Die Grundrechte</KTITEL>
   <ARTIKEL>
    <ANO>1</ANO>
    <ATITEL>Menschenwürde – Menschenrechte – Bindung der Grundrechte</ATITEL>
      <PNO>1</PNO>
      <PARAGRAPH>Die Würde des Menschen ist unantastbar. Sie zu achten und zu schützen ist
                 Verpflichtung aller staatlichen Gewalt. </PARAGRAPH>
      <PNO>2</PNO>
      <PARAGRAPH>Das Deutsche Volk bekennt sich darum zu unverletzlichen und unveräußerlichen
        Menschenrechten als Grundlage jeder menschlichen Gemeinschaft, des Friedens und der
        Gerechtigkeit in der Welt.</PARAGRAPH>
      <PNO>3</PNO>
      <PARAGRAPH>Die nachfolgenden Grundrechte binden Gesetzgebung, vollziehende Gewalt und
        Rechtsprechung als unmittelbar geltendes Recht.</PARAGRAPH>
  </ARTIKEL>
  <ARTIKEL>
    <ANO>2</ANO>
    <ATITEL>Persoenliche Freiheit</ANO>
      <PNO>1</ANO>
      <PARAGRAPH>Jeder hat das Recht auf die freie Entfaltung seiner Persönlichkeit, soweit er
        nicht die Rechte anderer verletzt und nicht gegen die verfassungsmäßige Ordnung oder das
        Sittengesetz verstößt.</PARAGRAPH>
      <PNO>2</PNO>
      <PARAGRAPH>Jeder hat das Recht auf Leben und ...</PARAGRAPH>
  </ARTIKEL>    ....
  <ARTIKEL>
  <ANO>14</ANO>
  <ATITEL>Eigentum Erbrecht Enteignung</ATITEL>
      <PNO>1</PNO>
      <PARAGRAPH>Das Eigentum und das Erbrecht werden gewährleistet. Inhalt und Schranken werden
         durch die Gesetze bestimmt.</PARAGRAPH>
      <PNO>2</PNO>
      <PARAGRAPH>Eigentum verpflichtet. Sein Gebrauch soll zugleich dem Wohle der Allgemeinheit
          dienen.</PARAGRAPH>
      <PNO>3</PNO>
      <PARAGRAPH>Eine Enteignung ist nur zum Wohle der Allgemeinheit zulässig. Sie darf nur durch
         Gesetz oder auf Grund eines Gesetzes erfolgen, das Art und Ausmaß der Entschädigung
regelt.
         Die Entschädigung ist unter gerechter Abwägung der Interessen der Allgemeinheit und der
         Beteiligten zu bestimmen. Wegen der Höhe der Entschädigung steht im Streitfalle der
         Rechtsweg vor den ordentlichen Gerichten offen.</PARAGRAPH>
  </ARTIKEL>
  <ANO>15</ANO>
  <ATITEL>...
<KAPITEL>
  <KNO>2</KNO>
  <KTITEL>Der Bund und die Länder</KTITEL>
  <ARTIKEL>
    <ANO>20</ANO>
    ...
      </ARTIKEL>
</KAPITEL>...
</GRUNDGESETZ>
```

Tabment 2.3: Grundgesetz.xml

Aufgrund der gegebenen Struktur (Tags, TTD) können folgende Anfragen formuliert werden:

Anfrage 2.1: Gib alle Artikel, die zwei gegebene Worte enthalten.

```
aus grundgesetz.xml
avec ARTIKEL! Erbrecht Eigentum # oder avec Erbrecht & Eigentum
```

Anfrage 2.2: Gib mir Artikel 22.

```
aus  grundgesetz.xml
avec ANO=22
gib  ARTIKEL
```

Anfrage 2.3: Gesucht sind alle Artikel mit einem bestimmten Wort im Titel.

```
aus  grundgesetz.xml
avec ARTIKEL! Enteignung in ATITEL
```

Anfrage 2.4: Gib alle Artikel mit Titel und Nummer der Kapitel die das Wort “Bund” im Titel enthalten.

```
aus  grundgesetz.xml
avec KAPITEL! Bund in KTITEL
gib  ANO,ATITEL l
```

Anfrage 2.5: Berechne für jedes Kapitel die Anzahl der Artikel und Paragraphen.

```
aus grundgesetz.xml
gib KNO,AANZ,PANZ m
    AANZ!  ++1 ANO
    PANZ!  ++1 PNO
```

Jetzt betrachten wir ein weiteres komplexeres Dokument mit TTD. Es benutzt Alternativen (|). Es stammt von den XQuery use-cases (C+07):

```
REPORT1! ABSCHNITTl
SECTION! TITEL,INHALT
INHALT! TEXT|NARKOSE|VORBEREITUNG|SCHNITT|AKTION|BEOBACHTUNG l
VORBEREITUNG! TEXT|AKTION l
SCHNITT! TEXT|GEOGRAPHIE|INSTRUMENT l
AKTION! TEXT|INSTRUMENT l
TITEL NARKOSE BEOBACHTUNG GEOGRAPHIE INSTRUMENT! TEXT
```

```
<REPORT1>
  <ABSCHNITT>
    <TITLE>Verfahren</TITLE>
    <INHALT>
  Die Patientin wurde in den Operationssaal gebracht, wo sie in Rückenlage
platziert wurde und
<NARKKOSE> induziert unter Vollnarkose. </NARKOSE>
<VORBEREITUNG>
<AKTION>Es wurde ein Foley-Katheter gelegt, um die Blase </AKTION> zu
dekomprimieren und der Bauch wurde dann steril vorbereitet und drapiert.
</VORBEREITUNG>
<SCHNITT>
Ein gekrümmter Einschnitt wurde gemacht
<GEOGRAPHIE> in der Mittellinie sofort infraumbilical </GEOGRAPHIE>
 und das subkutane Gewebe wurde geteilt
 <INSTRUMENT> Elektrokauterisation verwenden. </ INSTRUMENT>
 </SCHNITT>
 Die Faszie wurde identifiziert und
 <AKTION> # 2 0 Maxon Nähte wurden auf jeder Seite der Mittellinie
angeordnet.
 </AKTION>
 <SCHNITT>
 Die Faszie wurde geteilt mit
<INSTRUMENT> electrocautery </ INSTRUMENT>
 und das Peritoneum eingegeben.
</SCHNITT>
 <BEOBACHTUNG>Der Dünndarm wurde identifiziert.</BEOBACHTUNG>
 und
 <AKTION> das <INSTRUMENT>Hasson-Trokar</INSTRUMENT>
 wurde unter direkter Visualisierung gelegt.
 </AKTION>
 <AKTION>Das <INSTRUMENT>Trokar</INSTRUMENT>unter Verwendung der
      Nähte wurde an der Faszie befestigt.
</AKTION>
</INHALT>
</ABSCHNITT>
</REPORT1>
```

Tabment 2.4: report1.xml

In report1.tbt ist INHALT eine Liste von Elementen, wobei jedes Element entweder vom Typ TEXT, NARKOSE, VORBEREITUNG, SCHNITT, AKTION, oder BEOBACHTUNG ist. Im obigen Dokument ist das erste Element einfacher TEXT, das zweite ist vom Typ NARKOSE, das dritte vom Typ VORBEREITUNG, ... Durch obige Tags werden beispielsweise folgende Anfragen ermöglicht:

Anfrage 2.6: Welche Instrumente wurden beim zweiten Schnitt benutzt?

```
aus  report1.xml
gib  SCHNITTl
avec SCHNITT pos = 2
gib  INSTRUMENTl
```

Anfrage 2.7: Welches sind die beiden zuerst benutzten Instrumente?

```
aus  report1.xml
gib  INSTRUMENTl
avec INSTRUMENT pos<3
```

3 Eine Universitätsdatenbank

Wir betrachten eine nicht-relationale Datenbank, die aus einer flachen und zwei strukturierten Tabellen besteht:

```
uni.db! FAKS,STUDENTEN,KURSE
faks! FAK,DEKAN,BUDGET,STUDKAPAZI m
studenten! STID,NAME,ORT?,STIP,FAK,(KURS,NOTE m),
                                   (PROJ,STUNDEN m),CV m
CV! TEXT|GEBOREN|SCHULE|VATER|MUTTER l
kurse! KURS,LEHRENDER,(ISBN,TITEL m)m
```

Die unterstrichenen Spaltennamen sind Schlüssel. In der entsprechenden Kollektion kann ein Schlüsselwert stets nur einmal vorkommen. CV wird in den Beispielen nicht betrachtet. Die letzten beiden Tabellen ohne CV können durch die folgenden 5 flachen Relationen repräsentiert werden.

studenten1: STID,NAME,ORT?,STIP,FAK m
examen1: STID,KURS,NOTE m
projekte1: STID,PROJ,STUNDEN m
kurse1: KURS,LEHRENDER m
kurs_buecher1: KURS,ISBN,TITEL m

```
FAK,  DEKAN,  BUDGET, STUDKAPAZI m
Kunst Sitte    2000   600
Inf   Reichel 10000   500
Mathe Dassow   1000   200
Philo Hegel    1000    10
```

Tabment 3.1: faks.tab

```
STID, NAME,  ORT?,    STIP,FAK,  (KURS,       NOTE m), (PROJ, STUNDEN m)m
1234  Ernst  Oehna    500  Mathe  Algebra     1        Fritz  4
                                  Logik       2        Otto   2
                                  Geschichte  1
1245  Sophia Berlin   400  Inf    Algebra     3        Mao    5
                                  Datenbanken 1        Ming   4
                                  Otto        1        Otto   6
3456  Clara  Oehna    450  Inf    Datenbanken 1
                                  OCaml       2
4567  Ulrike          400  Kunst                        Monet 10
5678  Kaethe Gerwisch   0  Kunst  Repin       1        Monet 20
                                  Apel        1
```

Tabment 3.2: studenten.tab

```
KURS,       LEHRENDER, (ISBN,         TITEL m)m
Algebra     Reichel     0138-3019     Structural Induction on Partial Alg
                        3-8244-2099-6 Strukturierte Tabellen
Otto        Benecke     3-8244-2099-6 Strukturierte Tabellen
                        0-7167-8069-0 Principles of Database Systems
Datenbanken Saake       0-7167-8069-0 Principles of Database Systems
                        0-321-31256-2 Database Systems an Application-Or
```

Tabment 3.3: kurse.tab

```
STID, NAME,  ORT?,    STIP,FAK m
1234  Ernst  Oehna    500  Mathe
1245  Sophia Berlin   400  Inf
3456  Clara  Oehna    450  Inf
4567  Ulrike          400  Kunst
5678  Kaethe Gerwisch   0  Kunst
```

Tabment 3.4: studenten1.tab

```
STID, KURS,       NOTE m
1234  Algebra     1
1234  Geschichte  1
1234  Logik       2
1245  Algebra     3
1245  Datenbanken 1
1245  Otto        1
3456  Datenbanken 1
3456  OCaml       2
5678  Apel        1
5678  Repin       1
```

Tabment 3.5: examen1.tab

```
STID, PROJ, STUNDEN m
1234  Fritz  4
1234  Otto   2
1245  Mao    5
1245  Ming   4
1245  Otto   6
4567  Monet 10
5678  Monet 20
```

Tabment 3.6: projekte1.tab

Die obigen Tabellen und die folgenden Programme beziehen sich auf tab-Dateien. Es könnten aber auch Datenbankdateien sein. Die Programme brauchten dafür nicht geändert werden. So können sie unmittelbar in unserer Onlineversion (http://ottoPS.eu) genutzt werden.

4 Selektion (avec-, sans-Klausel)

Durch eine Bedingung werden Tupel oder Subtupel spezifiziert. Durch einen mit-Teil (mit-Klausel) überleben die spezifizierten (Sub) Tupel und bei einer ohn-Klausel werden sie verworfen. Daher verändert sich das Schema und die TTD des betrachteten Tabments bei einer Selektion nicht. Spaltennamen oder Tags werden in o++oPS in Großbuchstaben geschrieben. Das trifft auch dann zu, wenn Spaltennamen im gegebenen Tabment Kleinbuchstaben enthalten. Sie müssen mit einem Buchstaben oder dem "_" Zeichen beginnen. Ein Wort oder ein Text-Wert, der nicht in """ eingeschlossen ist, muss einen Kleinbuchstaben enthalten. Bei der Gleichheit von Wörtern oder Texten wird dann die Groß- bzw. Kleinschreibung nicht berücksichtigt. D.h., z.B, dass in der folgenden Anfrage Berlin auch klein geschrieben werden könnte, ohne dass sich das Anfrageergebnis ändert.

Anfrage 4.1: Gesucht sind alle Studenten von Berlin und Oehna, die schlechte Ergebnisse haben mit ihren schlechten Ergebnissen.

```
aus  studenten.tab
avec ORT in “Berlin Oehna“ # selektiert Studenten (STID-Tupel)
avec NOTE>2                # selektiert Studenten und Examen
```

Resultat:

```
STID, NAME,  ORT?,  STIP, FAK,  (KURS, NOTE m),(PROJ, STUNDEN m)m
1245  Sophia Berlin 400  Inf     Algebra 3      Mao   5
                                                Ming  4
                                                Otto  6
```

Die erste Bedingung drückt aus, dass der ORT-Wert in der zweielementigen Liste [Berlin Oehna] enthalten sein soll. Dies ist in diesem Fall eine Abkürzung für die Bedingung (Disjunktion)

```
avec ORT=Berlin | ORT=Oehna
```

Zwischenresultat nach der ersten Bedingung:

```
STID, NAME,  ORT?,  STIP, FAK,  (KURS,       NOTE m),(PROJ, STUNDEN m)m
1234  Ernst  Oehna  500  Mathe   Algebra     1        Fritz 4
                                 Logik       2        Otto  2
                                 Geschichte  1
1245  Sophia Berlin 400  Inf     Algebra     3        Mao   5
                                 Datenbanken 1        Ming  4
                                 Otto        1        Otto  6
3456  Clara  Oehna  450  Inf     Datenbanken 1
                                 OCaml       2
```

Auf das Ergebnis der ersten Bedingung wird die zweite "Bedingung" angewandt. Die zweite Bedingung ist eine Abkürzung der folgeden 2 Bedingungen:

```
avec STID! NOTE>2                  # selektiert STID-Tupel
avec KURS! NOTE>2                  # selektiert KURS-Tupel
```

Die erste dieser Bedingungen drückt aus, dass wir komplette Studenten selektieren, für die (KURS, NOTE)-Subtupel mit einer NOTE 3 oder größer existiert. Wir schreiben den Existenzquantor nicht, da er hinter jeder Bedingung versteckt ist. “#” ist das **Kommentarsymbol**. Es kann benutzt werden, um die Bedeutung eines Programmschrittes zu schreiben. Programmcode kann ebenfalls auskommentiert werden, um die Ergebnisse anderer Operationen zu isolieren. Die obige

Zwischentabelle ist Resultat des folgenden auskommentierten Programms:

```
aus  studenten.tab
avec ORT in “Berlin Oehna“ # selektiert Studenten
#avec NOTE>2               # selektiert Studenten und Examen
```

Anfrage 4.2: Gib für **alle** Studenten von Oehna und Berlin alle Resultate mit einer 3 oder schlechter.

```
aus  studenten.tab
avec ORT in “Berlin Oehna“
KURS avec NOTE>2   # selektiert Examen aber keine Studenten
gib  NAME,ORT,(KURS,NOTE m)b
```

Resultat:

```
NAME,  ORT,  (KURS,   NOTE m)
Clara  Oehna
Ernst  Oehna
Sophia Berlin Algebra 3
```

Nach Anwendung der beiden Bedingungen wurde eine Umstrukturierung (siehe Kapitel 6) angewandt. Daher hat sich das Schema der Tabelle geändert und das Ergebnis ist sortiert.

Anfrage 4.3: Gesucht sind alle Studenten von Oehna und Berlin mit allen Prüfungen, die eine 3 oder eine schlechtere Note haben.

```
aus  studenten.tab
avec ORT in “Berlin Oehna“
avec STID! NOTE>2 # selektiert nur Studenten aber keine Examen
gib  NAME,ORT,(KURS,NOTE m)b
```

Resultat:

```
NAME,  ORT,  (KURS,       NOTE m)b
Sophia Berlin Algebra     3
              Datenbanken 1
              Otto        1
```

Anfrage 4.4: Gesucht sind alle Studenten, die nur Einsen und wenigstens eine Eins haben.

```
aus  studenten.tab
avec NOTEm = {1}
```

Resultat:

```
STID, NAME,  ORT?,    STIP, FAK,  (KURS, NOTE m),(PROJ, STUNDEN m)m
5678  Kaethe Gerwisch 0     Kunst  Apel  1        Monet 20
                                    Repin 1
```

Für die Evaluierung der Bedingung wird für jeden Studenten seine Liste von Noten in eine Menge überführt, d.h. es werden Duplikate eliminiert. Daher ist die Menge von Ernst {1 2 1} = {1 2} und

die Menge {1 1} von Sabine ist gleich {1}. **Zwei Mengen M1 und M2 sind gleich**, wenn jedes Element von M1 in M2 vorkommt und umgekehrt jedes Element von M2 in M1 vorkommt. In anderen Worten: zwei Mengen M1 und M2 sind gleich, wenn 'M1 inc M2 & M2 inc M1' gilt. in2 ist die mengentheoretische Inklusionsrelation. Wenn wir andererseits Studenten mit genau zwei Einsen wünschen, können wir Multimengen nutzen: NOTEb = {{1 1}} (b kürzt bag ab). Wenn auch die Reihenfolge der Noten wichtig ist, können wir Listen benutzen, z.B.: NOTEl = [1 2 1],... .

Anfrage 4.5: Gesucht sind alle Studenten, die im Algebrakurs eine 1 haben.

```
aus  studenten.tab
avec STID! KURS=Algebra & NOTE=1
gib  STID,NAME,(KURS,NOTE m)m
```

Resultat:

```
STID, NAME,  (KURS,      NOTE m)m
1234  Ernst   Algebra    1
              Geschichte 1
              Logik      2
```

Anfrage 4.6: Gesucht sind alle Studenten, die eine Prüfung in Algebra abgelegt haben und eine 1 haben (nicht unbedingt in Algebra).

```
aus  studenten.tab
avec STID! KURS=Algebra
avec STID! NOTE=1
gib  STID,NAME,(KURS,NOTE m)m
```

Resultat:

```
STID, NAME,  (KURS,       NOTE m)m
1234  Ernst   Algebra     1
              Geschichte  1
              Logik       2
1245  Sophia  Algebra     3
              Datenbanken 1
              Otto        1
```

Anfrage 4.7: Gesucht sind alle Studenten, die Prüfungen in Algebra und Datenbanken abgelegt haben.

```
aus  studenten.tab
avec STID! KURS=Algebra
avec STID! KURS=Datenbanken
```

Resultat:

```
STID, NAME,  ORT?,  STIP, FAK, (KURS,       NOTE m),(PROJ, STUNDEN m)m
1245  Sophia Berlin 400  Inf    Algebra     3        Mao   5
                                Datenbanken 1        Ming  4
                                Otto        1        Otto  6
```

Zwischenergebnis nach der ersten Bedingung:

```
STID, NAME,  ORT?,  STIP,FAK,   (KURS,       NOTE m),(PROJ, STUNDEN m)m
1234  Ernst  Oehna  500  Mathe   Algebra     1        Fritz 4
                                 Logik       2        Otto  2
                                 Geschichte  1
1245  Sophia Berlin 400  Inf     Algebra     3        Mao   5
                                 Datenbanken 1        Ming  4
                                 Otto        1        Otto  6
```

Wenn wir beide Bedingungen durch & (und) verbinden würden (siehe unten), enthielte diese Bedingung nur einen EXIST-Operator. Dann gibt es kein Subtupel, welches beide Bedingungen gleichzeitig erfüllt. Das Ergebnis wäre in jedem Fall leer.

```
avec STID! KURS=Algebra & KURS=Datenbanken
```

Es sei angemerkt, dass beide Bedingungen von Anfrage 4.7 auch durch eine Bedingung ersetzt werden können.

Anfrage 4.7b: Gesucht sind alle Studenten, die Prüfungen in Algebra und Datenbanken abgelegt haben.

```
aus  studenten.tab
avec “Algebra Datenbanken“ in KURSm
```

Anfrage 4.8: Gesucht sind für jeden Studenten, der Algebra abgelegt hat, alle weiteren Kurse, die er abgelegt hat.

```
aus  studenten.tab
avec STID! KURS=Algebra      # selektiert Studenten
sans KURS! KURS=Algebra      # selektiert Examen
gib  NAME,KURSm b
```

Resultat:

```
NAME,  KURSm b
Ernst  Geschichte Logik
Sophia Datenbanken Otto
```

Anfrage 4.9: Gesucht sind alle Studenten, die das Wort Otto irgendwo enthalten.

```
aus  studenten.tab
avec Otto
```

Resultat:

```
STID, NAME,  ORT?,  STIP, FAK,  (KURS,       NOTE m),(PROJ, STUNDEN m)m
1234  Ernst  Oehna  500  Mathe   Algebra     1        Fritz 4
                                 Logik       2        Otto  2
                                 Geschichte  1
1245  Sophia Berlin 400  Inf     Algebra     3        Mao   5
                                 Datenbanken 1        Ming  4
                                 Otto        1        Otto  6
```

Anfrage 4.10: Drucke von allen Tupeln der Universitätsdatenbank (, zu denen ich Zugriff habe), die Tupel, die das Wort Apel enthalten.

```
aus faks.tab,studenten.tab,kurse.tab # später: aus uni.db
avec Apel
```

Resultat in ment-Form:

```
<TABM>
  <FAKS/>
  <STUDENTEN>
    <STID>5678</STID>
    <NAME>Kaethe</NAME>
    <ORT>Gerwisch</ORT>
    <STIP>0</STIP>
    <FAK>Kunst</FAK>
    <KURS>Apel</KURS>
    <NOTE>1</NOTE>
    <KURS>Repin</KURS>
    <NOTE>1</NOTE>
    <PROJ>Monet</PROJ>
    <STUNDEN>20</STUNDEN>
  </STUDENTEN>
  <KURSE/>
</TABM>
```

Bis jetzt haben wir nur Bedingungen behandelt, die nach dem Inhalt selektieren. Aber fast die gleiche Bedeutung haben Selektionen, die die Reihenfolge der Elemente ausnutzen. Das ist nicht nur für Listen von Bedeutung sondern auch für "relationale Anwendungen". Wir betrachten hier nur zwei Beispiele.

Anfrage 4.11: Gib für jeden Studenten aus Oehna mit Examen das letzte Examen.

```
aus  studenten.tab
avec ORT=Oehna
avec NOTE pos- = 1
gib  STID,NAME,(KURS,NOTE m)m
```

Resultat:

```
STID, NAME,  (KURS, NOTE m)m
1234  Ernst   Logik 2
3456  Clara   OCaml 2
```

Die Funktion pos (pos-) gibt die Positionsnummer(Positionsnummer rückwärts) eines (Sub-) elements der entsprechenden Kollektion zurück. Daher ist pos(NOTE) das Gleiche wie pos(KURS) und pos-(NOTE) das Gleiche wie pos-(KURS).

Anfrage 4.12: Gib für die besten 3 Studenten die 2 besten Examen. Wir löschen vorher Ulrike, da sie keine Noten hat und somit der Durchschnitt nicht existiert.

```
aus  studenten.tab
avec NOTE=NOTE
gib  DUR,NAME,FAK,(NOTE,KURS m)m
     DUR!  ++: NOTE
avec NAME pos < 4
avec NOTE pos < 3
```

Resultat:

```
DUR,          NAME,  FAK,  (NOTE, KURS m)m
1.            Kaethe Kunst  1     Apel
                            1     Repin
1.33333333333 Ernst  Mathe  1     Algebra
                            1     Geschichte
1.5           Clara  Inf    1     Datenbanken
                            2     OCaml
```

Für dieses Beispiel ist es ausreichend zu wissen, dass die Studenten durch die gib-Anweisung nach DUR und die Kurse nach NOTE sortiert werden. Die gib-Klausel wird in Kapitel 6 detaillierter erklärt. Die folgende Anfrage ist keine Selektion, da sich das Schema der Tabelle ändert. Sie hat aber einen ähnlichen Effekt.

Anfrage 4.13: Gib mir das letzte Element der Studentendatei.

```
studenten.tab nth -1
```

Resultat:

```
STID, NAME,  ORT?,    STIP, FAK,  (KURS, NOTE m),(PROJ, STUNDEN m)
5678  Kaethe Gerwisch 0     Kunst  Apel  1        Monet 20
                                    Repin 1
```

Anfrage 4.14: Gib mir die Studenten mit dem höchsten Stipendium.

```
aus  studenten.tab
avec STIP= STIPm- nth 1 # m- sortiert alle STIPs abwärts
```

Resultat:

```
STID, NAME, ORT?, STIP,FAK, (KURS,      NOTE m),(PROJ, STUNDEN m)m
1234  Ernst Oehna 500  Mathe Algebra    1        Fritz 4
                             Logik      2        Otto  2
                             Geschichte 1
```

5 Berechnungen

Anfrage 5.1a: Berechne die Bruttowerte mehrerer Preise.

```
  3.18 55.88 17.90 * 1.19
```

Resultat (hsq und tab):

```
3.7842  66.4972  21.301
```

oder

```
PZAHLl
3.7842 66.4972 21.301
```

PZAHL ist eine Zahl mit Punkt (früher Kommazahl). Anstelle der zweistelligen Operation * kann man auch einstellige Operationen, wie ++, ++:, … benutzen. Dann besteht das Ergebnis lediglich aus einem Wert. Diese Operationen sind auch anwendbar, wenn das gegebene Tabment Texte enthält. Diese verbleiben dann unverändert. Wendet man dagegen einstellige Funktionen wie die Sinus-, Cosinus- oder Quadratwurzelfunktion (sqrt) an, so wird jede Zahl in den entsprechenden Funktionswert überführt, und die anderen elementaren Daten bleiben wieder unverändert. Die dritte Wurzel kann man beispielsweise durch die zusätzliche Zeile

```
hoch 1:3
```

ziehen.

Anfrage 5.1b: Berechne die Brutto Werte mehrerer Preise und gib auch die gegebenen Werte mit aus.

```
  3.18 55.88 17.90
  BRUTTO:=PZAHL*1.19
```

Resultat:

```
PZAHL, BRUTTO l
 3.18   3.7842
55.88  66.4972
17.9   21.301
```

Anfrage 5.1c: Berechne die Bruttowerte von NETTO-Werten.

```
  [NETTO! 3.18 55.88 17.90]
  BRUTTO:=NETTO*1.19
```

Resultat:

```
NETTO, BRUTTO l
 3.18   3.7842
55.88  66.4972
17.9   21.301
```

Anfrage 5.2: Wandle alle Nettopreise einer Tabelle in Bruttopreise um.

```
<TAB!
ARTIKEL,   PREIS l
OttoRAMDB  500
OttoWiki    10
OttoCalc    20
!TAB>
  * 1.19
```

Resultat:

```
ARTIKEL,  PREIS l
OttoRAMDB 595.
OttoWiki   11.9
OttoCalc   23.8
```

Jeder Wert der Tabelle wurde mit 1.19 multipliziert. TEXT-Werte werden nicht verändert bei arithmetische Operationen.

Anfrage 5.3: Berechne die Bruttowerte jeder Position und summiere alle Bruttowerte.

```
<TAB!
ARTIKEL,   PREIS, ANZ m
OttoRAMDB  500      20
OttoWiki    10     200
OttoCalc    20    4000
!TAB>
  TOTAL:=PREIS*ANZ*1.19
  TOTALSUM:=TOTALl ++
```

Resultat:

```
TOTALSUM,  (ARTIKEL,  PREIS, ANZ, TOTAL m)
109480.     OttoCalc   20    4000 95200.
            OttoRAMDB 500      20 11900.
            OttoWiki   10     200  2380.
```

Anfrage 5.4: Drücke das Stipendium aller Informatikstudenten in Dollar aus.

```
  aus  studenten.tab
  avec FAK=Inf
  DOL:=STIP*1.22
```

Resultat:

```
STID, NAME,  ORT?,  STIP,FAK,  DOL, (KURS,       NOTE m),(PROJ, STUNDEN m) m
1245  Sophia Berlin 400  Inf   488.  Algebra     3        Mao   5
                                    Datenbanken 1        Ming  4
                                    Otto        1        Otto  6
3456  Clara  Oehna  450  Inf   549.  Datenbanken 1
                                    OCaml       2
```

Anfrage 5.5: Zahle jedem Studenten für jedes seiner Projekte 100 Euro.

```
aus studenten.tab
BONUS:= PROJl ++1 *100
gib STID,NAME,BONUS m
```

Resultat:

```
STID, NAME,  BONUS m
1234  Ernst  200
1245  Sophia 300
3456  Clara    0
4567  Ulrike 100
5678  Kaethe 100
```

Anfrage 5.6: Gib jedem Studenten von Oehna einen Bonus, der von seinen Noten abhängt.

```
aus  studenten.tab
avec ORT=Oehna
DUR  := NOTEl ++:
BONUS:=1000 : DUR
gib  STID,NAME,DUR,BONUS m
rnd  2
```

Resultat:

```
STID, NAME, DUR,  BONUS m
1234  Ernst 1.33  750.
3456  Clara 1.5   666.67
```

Mit der Hilfe von rnd (runde) wird jeder Wert der Tabelle auf 2 Stellen nach dem Komma gerundet. Wenn rnd nicht anwendbar ist, bleibt der Wert wieder unverändert.

Anfrage 5.7: Berechne den BMI (body mass index) für jedes Gewicht jeder Person.

```
<TAB!
NaME,       LAeNGE, (ALTER, GeWICHT l)m
Klaus       1.68       18     61
                       30     65
                       56     80
                       61     75
Kathi       1.70       18     55
                       40     70
!TAB>
  BMI:= GEWICHT : LAENGE : LAENGE
  rnd 2
```

Resultat:

```
NAME, LAENGE,  (ALTER,GEWICHT,BMI l)m
Klaus 1.68      18    61      21.61
                30    65      23.03
                56    80      28.34
                61    75      26.57
Kathi 1.7       18    55      19.03
                40    70      24.22
```

Beachten sie die zwei verschiedenen Schreibweisen des Gewichts und der Länge. Beachten sie weiterhin, dass die Formel nicht nur auf Zeilen angewandt wird, die einen LAENGE-Wert enthalten. Das ist möglich, da unser System das gegebene Schema versteht.

Anfrage 5.8: Die Studenten der Mathe Fakultät bekommen einen Bonus von 900 Euro, der Informatik 800 Euro und der Rest bekommt 700 Euro.

```
aus studenten.tab
BONUS:= if FAK=Mathe then 900 else
        if FAK=Inf   then 800 else 700
gib STID,NAME,FAK,BONUS m
```

Resultat:

```
STID, NAME,  FAK,  BONUS m
1234  Ernst  Mathe 900
1245  Sophia Inf   800
3456  Clara  Inf   800
4567  Ulrike Kunst 700
5678  Kaethe Kunst 700
```

6 Restrukturierung (gib-Klausel)

Die Restrukturierungs Operation (stroke) erlaubt die Restrukturierung eines beliebigen Tabments in ein beliebiges anderes Tabment nur durch Vorgabe des Schemas oder der TTD des gewünschten Tabments. Zusätzlich können Aggregationen durchgeführt werden, Duplikate eliminiert werden, Vereinigungen und Sortierungen vorgenommen werden. Es können sogar joins realisiert werden (allerdings in mehreren Schritten).

Anfrage 6.1: Illustriere die Kollektionssymbole.

```
aus studenten.tab
gib FAKm,FAKb,FAKl,FAKm-,FAKb-,FAKl-,FAK?
```

Resultat:

FAKm,	FAKb,	FAKl,	FAKm-,	FAKb-,	FAKl-,	FAK?
Kunst	Kunst	Mathe	Mathe	Mathe	Kunst	Mathe
Inf	Kunst	Inf	Inf	Inf	Kunst	
Mathe	Inf	Inf	Kunst	Inf	Inf	
	Inf	Kunst		Kunst	Inf	
	Mathe	Kunst		Kunst	Mathe	

Die STID-Segmente (Typ: (STID, NAME, ORT?, STIP, FAK)) werden nacheinander in jede der gegebenen FAK-Kollektionen eingefügt. KURS- und PROJ-Segmente werden nicht beachtet.

Anfrage 6.2: Sortiere Studenten nach FAK und NAME.

```
aus studenten.tab
gib FAK,NAMEb m
```

Resultat:

FAK,	NAMEb m
Inf	Clara Sophia
Kunst	Kaethe Ulrike
Mathe	Ernst

STID-Segmente werden zunächst in die FAK-Ebene und dann tiefer in die NAME-Ebene - Segment nach Segment - eingefügt. KURS- und PROJ-Segments werden wiederum nicht berührt.

Anfrage 6.3: Sortiere die Studenten nach FAK und NAME, wobei das Resultat eine flache Tabelle ist.

```
aus studenten.tab
gib FAK,NAME m
```

Resultat:

FAK,	NAME m
Inf	Clara
Inf	Sophia
Kunst	Kaethe
Kunst	Ulrike
Mathe	Ernst

Wenn wir m durch b ersetzen, ändern sich die Ergebniswerte nicht.

Anfrage 6.4 a: Sortiere die Fakultäten abwärts nach BUDGET und zweitens nach Studentenkapazität.

```
aus faks.tab
gib BUDGET,STUDKAPAZI,FAK m-
```

Resultat:

```
BUDGET,  STUDKAPAZI, FAK m-
 10000   500         Inf
  2000   600         Kunst
  1000   200         Mathe
  1000    10         Philo
```

Anfrage 6.4 b: Sortiere die Fakultäten nach Budget und zusätzlich nach Studentenkapazität (Duplizierung der Tabelle).

```
aus faks.tab
gib BUDGET,FAK m-,(STUDKAPAZI,FAK m-)
```

Resultat:

```
BUDGET, FAK m-,  (STUDKAPAZI, FAK m-)
 10000   Inf      600         Kunst
  2000   Kunst    500         Inf
  1000   Philo    200         Mathe
  1000   Mathe     10         Philo
```

Anfrage 6.5: Packe die Prüfungsergebnisse jedes Studenten nach Fakultät (Gruppiere bereits gruppierte Daten auf neue Art).

```
aus studenten.tab
gib FAK,(KURS,NOTE b)m
```

Resultat:

```
FAK,  (KURS,       NOTE b)m
Inf    Algebra     3
       Datenbanken 1
       Datenbanken 1
       OCaml       2
       Otto        1
Kunst  Apel        1
       Repin       1
Mathe  Algebra     1
       Geschichte  1
       Logik       2
```

Hier werden die STID-Segmente in die FAK-Ebene eingefügt. Sie können nicht tiefer eingefügt werden, da sie weder KURS- noch NOTE-Werte enthalten. Die entsprechenden Prüfungsmultimengen sind dann stets zunächst leer. Dann wird jedes KURS-Segment ((KURS, NOTE)-Paar) um ihr übergeordnetes STID-Segment verlängert. Diese verlängerten Segmente können Schritt für Schritt in die entsprechende Multimenge eingefügt werden. Das verlängerte Segment hat den Typ: (STID, NAME, ORT?, STIP, FAK, KURS, NOTE) PROJ-Segmente werden nicht benötigt.

Anfrage 6.6 a: (Spezielle Selektion mit gib-Klausel) Gib alle Studenten, für die ein ORT-Eintrag existiert, mit diesem Eintrag. Gib zusätzlich die gegebene Kollektion für Vergleichszwecke.

```
aus studenten.tab
gib NAME,ORT m,(NAME,ORT? m)
```

Resultat:

```
NAME,  ORT m,  (NAME,  ORT? m)
Clara  Oehna    Clara  Oehna
Ernst  Oehna    Ernst  Oehna
Kaethe Gerwisch Kaethe Gerwisch
Sophia Berlin   Sophia Berlin
                Ulrike
```

In der ersten Menge fordert der Nutzer vollständige Paare. Da für Ulrike kein Paar existiert, kann sie nicht im ersten Ergebnis erscheinen.

Anfrage 6.6 b: (Selektion nur mit gib-Klausel) Gib alle Studenten mit nichtleeren Kollektionen.

```
aus studenten.tab
gib STID,NAME,FAK,KURS,NOTE m
gib STID,NAME,FAK,(KURS,NOTE m)m
```

Resultat:

```
STID, NAME,  FAK,  (KURS,       NOTE m)m
1234  Ernst  Mathe  Algebra     1
                    Geschichte  1
                    Logik       2
1245  Sophia Inf    Algebra     3
                    Datenbanken 1
                    Otto        1
3456  Clara  Inf    Datenbanken 1
                    OCaml       2
5678  Kaethe Kunst  Apel        1
                    Repin       1
```

Zwischenresultat nach der ersten gib-Klausel:

```
STID, NAME,  FAK,  KURS,       NOTE m
1234  Ernst  Mathe Algebra     1
1234  Ernst  Mathe Geschichte  1
1234  Ernst  Mathe Logik       2
1245  Sophia Inf   Algebra     3
1245  Sophia Inf   Datenbanken 1
1245  Sophia Inf   Otto        1
3456  Clara  Inf   Datenbanken 1
3456  Clara  Inf   OCaml       2
5678  Kaethe Kunst Apel        1
5678  Kaethe Kunst Repin       1
```

Um das Zwischenresultat zu erhalten, wird zuerst versucht, STID-Segmente einzufügen. Das ist nicht möglich da KURS-Werte nicht existieren. Dann wird wieder jedes KURS-Segment um sein übergeordnetes Segment verlängert. Diese verlängerten Segmente werden prüfungsweise und studentenweise eingefügt.

Anfrage 6.6 c: (Unkontrollierte Selektion) Gib für jeden Namen die erste Note, falls eine existiert. Drucke weitere Kollektionen für Vergleichszwecke.

```
aus studenten.tab
gib (NAME,NOTE? m),(NAME,NOTE m),(NAME,NOTE b),(NAME,NOTEb m)
```

Resultat:

```
NAME,  NOTE? m, (NAME,  NOTE m),(NAME,  NOTE b),(NAME, (NOTE b)m)
Clara  1         Clara  1        Clara  1        Clara   1 2
Ernst  1         Clara  2        Clara  2        Ernst   1 1 2
Kaethe 1         Ernst  1        Ernst  1        Kaethe  1 1
Sophia 3         Ernst  2        Ernst  1        Sophia  1 1 3
Ulrike           Kaethe 1        Ernst  2        Ulrike
                 Sophia 1        Kaethe 1
                 Sophia 3        Kaethe 1
                                 Sophia 1
                                 Sophia 1
                                 Sophia 3
```

In die erste und letzte Kollektion können STID-Segmente eingefügt werden, da nur Namen gefordert sind.

Anfrage 6.7: (Restrukturierung) Kehre die gegebene Strukturierung um. D.h., vertausche KURS und NAME.

```
aus studenten.tab
gib KURS,(NAME,NOTE b)m
```

Resultat:

```
KURS,       (NAME,  NOTE b)m
Algebra      Ernst  1
             Sophia 3
Apel         Kaethe 1
Datenbanken  Clara  1
             Sophia 1
Geschichte   Ernst  1
Logik        Ernst  2
OCaml        Clara  2
Otto         Sophia 1
Repin        Kaethe 1
```

Hier wird zuerst versucht, STID-Segmente einzufügen. Da KURS-Werte nicht existieren, kann nicht eingefügt werden. Daher werden die verlängerten KURS-Segmente zuerst in die Kursebene und dann in die NAME-Ebene eingefügt.

Anfrage 6.8: (Restrukturierung mit zusätzlichen Tags) Kehre die gegebene Strukturierung um. D.h. vertausche KURS und NAME. Zusätzlich sind Wurzel- Tupel und Subtupeltags zu erzeugen.

```
aus studenten.tab
gib KURSE
    KURSE!    KURSTUPm
    KURSTUP!  KURS,EXAMSTUPb
    EXAMSTUP! NAME,NOTE
```

Resultat (XML-Auszug): Die gleichen Werte wie im vorherigen Resultat.

```
<KURSE>
  <KURSTUP>
    <KURS>Algebra</KURS>
    <EXAMSTUP>
      <NAME>Ernst</NAME>
      <NOTE>1</NOTE>
    </EXAMSTUP>
    <EXAMSTUP>
      <NAME>Sophia</NAME>
      <NOTE>3</NOTE>
    </EXAMSTUP>
  </KURSTUP>
  <KURSTUP>
    <KURS>Apel</KURS>
    <EXAMSTUP>
      <NAME>Kaethe</NAME>
      <NOTE>1</NOTE>
    </EXAMSTUP>
  </KURSTUP>
  <KURSTUP>
    <KURS>Datenbanken</KURS>
    <EXAMSTUP>
      <NAME>Clara</NAME>
      <NOTE>1</NOTE>
    </EXAMSTUP>
    <EXAMSTUP>
      <NAME>Sophia</NAME>
      <NOTE>1</NOTE>
    </EXAMSTUP>
  </KURSTUP>
  <KURSTUP>
    <KURS>Geschichte</KURS>
    <EXAMSTUP>
      <NAME>Ernst</NAME>
      <NOTE>1</NOTE>
    </EXAMSTUP>
  </KURSTUP>

  ...
</KURSE>
```

Jetzt wollen wir die bekannten Operationen Vereinigung, Durchschnitt und Mengendifferenz illustrieren, ohne sie direkt zu benutzen.

Anfrage 6.9: Gib die Vereinigung der STID-Werte von 2 Tabellen.

```
aus examen1.tab,projekte1.tab # ein Paar von Tabellen
gib STIDb
```

Resultat (width 50):

```
STIDb
1234 1234 1234 1234 1234 1245 1245 1245 1245
1245 1245 3456 3456 4567 5678 5678 5678
```

Wenn wir B durch M ersetzen, erhalten wir jeden STID nur einmal:

Resultat:

```
STIDm
1234 1245 3456 4567 5678
```

Falls wir wissen wollen, von welcher Datei jeder STID ist, können wir diese Informationen in folgender Weise anfordern.

```
gib STID,KURS?,PROJ? b
```

Resultat:

```
STID, KURS?,      PROJ? b
1234              Fritz
1234              Otto
1234  Algebra
1234  Geschichte
1234  Logik
1245              Mao
1245              Ming
1245              Otto
1245  Algebra
1245  Datenbanken
1245  Otto
3456  Datenbanken
3456  OCaml
4567              Monet
5678              Monet
5678  Apel
5678  Repin
```

Anfrage 6.10: Bilde den Durchschnitt von zwei Dateien mit verschiedenen Schemata.

```
aus examen1.tab,projekte1.tab
gib STID,KURS?,PROJ? m
gib STID,KURS,PROJ m
# Diese Aufgabe kann auch durch Selektionen realisiert werden.
gib STIDm
```

Resultat:

```
STIDm
1234 1245 5678
```

Zwischenresult nach der ersten gib-Klausel:

```
STID, KURS?,      PROJ? m
1234  Algebra     Fritz
1245  Algebra     Mao
3456  Datenbanken
4567              Monet
5678  Apel        Monet
```

Anfrage 6.11 a: Mengendifferenz: Gib alle STIDs von examen1, die nicht in projekte1 enthalten sind.

```
aus examen1.tab
rename STID to STUDID
,projekte1.tab     # es entsteht wieder ein Paar von Tabellen
sans STUDID inm STIDm
gib  STUDIDm
```

Resultat:

```
STUDIDm
3456
```

Anfrage 6.11 b: Mengendifferenz, aber mit verschachtelter Anfrage (siehe Kapitel 7).

```
aus  examen1.tab
sans STID inm  begin aus projekte1.tab
                     gib STIDm end
gib  STIDm
```

Resultat:

```
STIDm
3456
```

Anfrage 6.12: (Gruppierung mit Aggregation) Berechne die Anzahl aller Studenten und die Anzahl für jede Fakultät. Sortiere die Studenten nach FAK und NAME.

```
aus studenten.tab
gib ANZ,(FAK,ANZ,(NAME,STID b)m)
    ANZ:=STID ! ++1
```

Resultat:

```
ANZ,  (FAK,  ANZ,  (NAME,  STID b)m)
5      Inf   2      Clara  3456
                    Sophia 1245
       Kunst 2      Kaethe 5678
                    Ulrike 4567
       Mathe 1      Ernst  1234
```

Anfrage 6.13: (Restrukturierung mit Aggregation) Gib die totale Summe aller Stipendien und die Summe für jeden Kurs. Sortiere die Sätze nach KURS.

```
aus studenten.tab
gib SU,(KURS,SU m)
    SU:=STIP ! ++
```

Resultat:

```
SU,  (KURS,        SU m)
1750  Algebra      900
      Apel           0
      Datenbanken 850
      Geschichte   500
      Logik        500
      OCaml        450
      Otto         400
      Repin          0
```

Hierbei ist es interessant, dass die Summe der inneren SU-Werte größer ist als der äußere SU-Wert. Das liegt daran, dass bestimmte Kurse in mehr als einem Studentensatz vorkommen.

Anfrage 6.14: Gesucht ist der Name des Studenten mit dem STID 1245.

```
aus  studenten.tab
avec STID = 1245
gib  NAME
```

Resultat:

```
NAME
Sophia
```

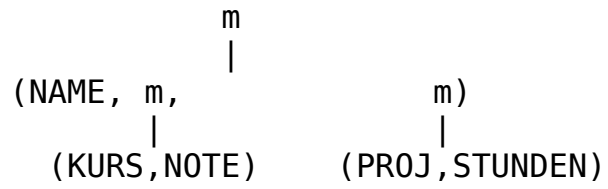
Anfrage 6.15: Verteile die Studenten der Fakultäten Informatik und Kunst in 2 unabhängige Tabellen.

```
aus studenten.tab
STIDCS := if FAK=Inf then STID
STIDART:= if FAK=Kunst then STID
gib STIDCS,NAME,ORT? m,(STIDART,NAME,ORT? m)
```

Resultat:

```
STIDCS, NAME,  ORT? m,   (STIDART, NAME,  ORT? m)
1245    Sophia Berlin     4567     Ulrike
3456    Clara  Oehna      5678     Kaethe Gerwisch
```

Der Begriff des hierarchischen Weges ist wichtig für alle Operationen. Seine Definition basiert auf "schmalen" Schemen. Alle Kollektionssymbole außer '?' sind echte Kollektionssymbole.
Ein **Schema s ist schmal**, wenn für 2 beliebige echte Kollektionssymbole c und c’ gilt, entweder c ist in c' enthalten oder c' in c. Felder f1 und f2 eines Schemeas s befinden sich auf einem **hierarchischen Pfad** (kurz **HP**) **on s**, wenn das Schema, das durch Vergessen aller Felder außer f1 und f2 entsteht, schmal ist.
X,Ym,Zm m ist nicht schmall, aber X,Y?,Z? m. PROJ und KURS befinden sich in
NAME, (KURS, NOTE m) , (PROJ, STUNDEN m)m, nicht auf einem hierarchischen Pfad; im Gegensatz zu PROJm and KURS. Das wird in der graphischen Representation des Schemas sichtbar.



Anfrage 6.16: Bringe zwei Felder die nicht auf einem HP auf einen HP.

```
aus studenten.tab
gib KURS,PROJ m # in jedem Fall leer
```

Resultat:

```
KURS, PROJ m
```

Anfrage 6.17: Sortiere und gruppiere die Studenten, die einen Algebra KURS abgelegt haben, nach den entsprechenden Noten und sortiere sie nach Name und drucke alle ihre Projekte.

```
aus  studenten.tab
avec KURS=Algebra
strip
gib NOTE,(NAME,(PROJ,STUNDEN m)b) m
```

Resultat:

```
NOTE, (NAME, (PROJ, STUNDEN m)b)m
1       Ernst   Fritz 4
                Otto  2
3       Sophia  Mao   5
                Ming  4
                Otto  6
```

Obwohl PROJ und NOTE nicht auf einem HP liegen, ist die Projekt-Kollektion nicht leer. Das liegt an der vorhandenen Bedingung und der danach angewandten strip-Operation. Jede (KURS,NOTE)-Kollektion enthält höchstens ein Element. Das ensprechende l wird jetzt durch strip durch eine „optionale Kollektion“ ersetzt. Danach befinden sich NOTE und PROJ auf einem HP. Diese sogenannte „strip“ Operation kehrt in Kapitel 11 wieder. Ohne die strip Funktion hätten wir das folgende Endresultat:

```
NOTE,  (NAME, (PROJ, STUNDEN m)b)m
1       Ernst
3       Sophia
```

7 Der einfache „Join“ (:=-Klausel) und geschachtelte Anfragen

Das Mischen oder Verbinden von Informationen zweier Tabellen heißt “Join”. In unserem Ansatz ist der Join zweier flacher Tabellen nicht wieder flach. Wir benötigen keine weitere Join Operation. Mit := können vernünftige Strukturen generiert werden.

Anfrage 7.1: Erweitere die Studenten um Prüfungsdaten.

```
aus studenten1.tab
:=  examen1.tab at FAK
```

First part of the Resultat:

```
STID, NAME,  ORT?,    STIP, FAK,  (STID, KURS,       NOTE m)m
1234  Ernst  Oehna    500  Mathe   1234  Algebra     1
                                   1234  Geschichte  1
                                   1234  Logik       2
                                   1245  Algebra     3
                                   1245  Datenbanken 1
                                   1245  Otto        1
                                   3456  Datenbanken 1
                                   3456  OCaml       2
                                   5678  Apel        1
                                   5678  Repin       1
1245  Sophia Berlin   400  Inf     1234  Algebra     1
                                   1234  Geschichte  1
                                   1234  Logik       2
                                   1245  Algebra     3
                                   1245  Datenbanken 1
                                   1245  Otto        1
                                   3456  Datenbanken 1
                                   3456  OCaml       2
                                   5678  Apel        1
                                   5678  Repin       1
3456  Clara  Oehna    450  Inf     1234  Algebra     1
                                   1234  Geschichte  1
                                   1234  Logik       2
  ...
```

Das Resultat enthält 5*10=50 Subtupel. Um die 10 gewünschten Subtupel zu erhalten benötigen wir eine zusätzliche Bedingung.

Anfrage 7.2 a: Füge zu jedem Studentensatz alle zugehörigen Prüfungssätze hinzu (“Strukturierter left outer join”).

```
aus  studenten1.tab
:=   examen1.tab at FAK
avec KURS! STUDENTEN1/STID=EXAMEN1/STID
```

Resultat:

```
STID, NAME,  ORT?,    STIP, FAK, (STID, KURS,       NOTE m)m
1234  Ernst  Oehna    500   Mathe  1234  Algebra     1
                                  1234  Geschichte  1
                                  1234  Logik       2
1245  Sophia Berlin   400   CS     1245  Algebra     3
                                  1245  Datenbanken 1
                                  1245  Otto        1
3456  Clara  Oehna    450   CS     3456  Datenbanken 1
                                  3456  OCaml       2
4567  Ulrike          400   Kunst
5678  Kaethe Gerwisch   0   Kunst  5678  Apel        1
                                  5678  Repin       1
```

Wenn wir Ulrike streichen wollen, müssen wir lediglich den Ebenenspezifizierer 'KURS:' entfernen. Jedes Tabment xyz.tab hat den Wurzeltag xyz. Daher ergibt sich die folgende TTD durch die obige Erweiterung:

```
TABMENT! STUDENTEN1
STUDENTEN1! STID,NAME,ORT?,STIP,FAK,EXAMEN1 m
EXAMEN1! STID,KURS,NOTE m
NOTE,STIP,STID! ZAHL
KURS,FAK,ORT,NAME! TEXT
```

Diese TTD erlaubt eine präzise Spezifikation von Spaltennamen auch wenn ein Name zweifach vorkommt. EXAMEN1/KURS ist hier das Gleiche wie KURS, da KURS nur einmal auf der rechten Seite der TTD auftritt. Genauso entspricht der Pfad STUDENTEN1/EXAMEN1/KURS dem KURS. Nur der Pfad STID ist nicht „präzise“, da er zweimal vorkommt. STUDENTEN1/STID ist der Schlüssel der Tabelle studenten1 und EXAMEN1/STID=STUDENTEN1/EXAMEN1/STID der Schlüssel der examen1 Tabelle.
In einem Pfad X/Y/Z wird gefordert, dass Z auf der rechten Seite von Y und Y auf der rechten Seite von X vorkommt. X ist der Vatertag von Y und Y der Vatertag von Z. Es ist kein Tag zwischen X und Y und zwischen Y und Z (in der XML Repräsentation). Wenn wir die Zwischentags nicht kennen, können wir auch die Schreibweise X//Z benutzen. Hier sind beliebig viele Tags zwischen X und Z zugelassen. Das heißt, dass dieser Tagpfad äquivalent zu einem vollständigem Pfad X/X1/X2/…/Xn/Z für geeignete Tags X1,…,Xn ist. Daher ist hier STUDENTEN1//KURS gleichwertig zu KURS und dem vollständigen Pfad STUDENTEN1/EXAMEN1/KURS.

Anfrage 7.2 b: Anfrage 7.2a als geschachtelte Anfrage.

```
aus studenten1.tab
:=  begin aus  examen1.tab
          avec STID=STID' end   at FAK
```

Geschachtelte Anfragen werden in “{}” eingeschlossen. Wenn wir uns in der inneren Anfrage auf einen Spaltennamen außerhalb der inneren Anfrage beziehen wollen, müssen wir lediglich ein “'” anhängen. Daher ist STID' der Schlüssel von studenten1.

Anfrage 7.3: Versuche die gegebene Tabelle studenten aus den 3 gegebenen Tabellen zu erzeugen.

```
aus studenten1.tab
:=  begin aus  projekte1.tab
          avec STID=STID'
          gib  PROJ,STUNDEN m end   at FAK
:=  begin aus  examen1.tab
          avec STID=STID'
          gib  KURS,NOTE m end  at FAK
```

Resultat: studenten.tab aber ohne das Innere l

Anfrage 7.4: Erzeuge eine Tabelle der Verschachtelungstiefe 3.

```
aus faks.tab
weg STUDKAPAZI # weg: entferne die Spalte
:=  begin aus  studenten1.tab
          avec FAK=FAK'
          weg  FAK      end     at BUDGET
:=  begin aus  examen1.tab
          avec STID=STID'
          weg  STID      end     at STIP
```

Resultat:

```
FAK,  DEKAN,  BUDGET,  (STID, NAME,  ORT?,    STIP,  (KURS,       NOTE m)m)m
Inf   Reichel 10000     1245  Sophia Berlin   400     Algebra     3
                                                      Datenbanken 1
                                                      Otto        1
                        3456  Clara  Oehna    450     Datenbanken 1
                                                      OCaml       2
Kunst Sitte    2000     4567  Ulrike          400
                        5678  Kaethe Gerwisch   0     Apel        1
                                                      Repin       1
Mathe Dassow   1000     1234  Ernst  Oehna    500     Algebra     1
                                                      Geschichte  1
                                                      Logik       2
Philo Hegel    1000
```

Wenn wir nur wenige Spalten löschen wollen, sollten wir die weg-Klausel anstelle der gib-Klausel benutzten. Es sei angemerkt, dass oben eine Tabelle der Tiefe 3 entsteht die tiefste Verschachtelungstiefe im Programm aber lediglich 2 ist.

8 Rekursive Programme

Rekursivität ist ein mächtiges Werkzeug, um Funktionen und Datenstrukturen prägnant zu beschreiben. Es erfodert jedoch Zeit, um verstanden zu werden. Daher führen wir hier nur eine einfach zu handhabende Art der rekursive Erweiterung ein. Vollrekursive Funktionen können auch in o++oPS definiert werden, sie werden aber in diesem Buch nicht beschrieben.

Anfrage 8.1: Wie wächst ein Betrag von 100 Euro bei 1 und wie bei 9 % Zinsen in einer Periode von 20 Jahren.

```
aus 2016 .. 2037 tags JAHR
BETRAG1:= first 100. next  BETRAG1 recpred*1.01 at JAHR
BETRAG9:= first 100. next  BETRAG9 recpred*1.09 at BETRAG1
rnd 2
```

Resultat:

```
JAHR, BETRAG1, BETRAG9 l
2016  100.     100.
2017  101.     109.
2018  102.01   118.81
2019  103.03   129.5
2020  104.06   141.16
2021  105.1    153.86
2022  106.15   167.71
2023  107.21   182.8
2024  108.28   199.26
2025  109.37   217.19
2026  110.46   236.74
2027  111.57   258.04
2028  112.68   281.27
2029  113.81   306.58
2030  114.95   334.17
2031  116.1    364.25
2032  117.26   397.03
2033  118.43   432.76
2034  119.61   471.71
2035  120.81   514.17
2036  122.02   560.44
```

Die neue BETRAG1 Spalte wird durch 2 Formeln definiert. Dem ersten Jahr der Liste von Jahren wird der Wert der ersten Formel zugewiesen. Der zweite Wert und alle weiteren Werte werden durch die zweite Formel berechnet. Hier ist BETRAG1 pred der Wert des Vorgängers. Daher erhalten wir 100*1.01=101 für den zweiten Wert. Für den dritten Wert ergibt sich dann 101 *1.01=102.01. In der selben Weise werden alle folgenden Werte nacheinander mit der zweiten Formel berechnet.

Anfrage 8.2: Berechne das BIP-Wachstum in Westdeutschland, Ostdeutschland und China in den unten stehenden Jahren bei den gegebenen Wachstumszahlen.

```
<TAB!
JAHR,  WGINC,   EGINC ,    CINC l
1988  100.      100.      100.
1989    3.9       1.85      4.2
1991   11.09    -47.8      13.56
1992    1.7       6.2      14.3
1993   -2.6       8.7      13.9
1994    1.4       8.1      13.1
1995    1.4       3.5      11.
1996    0.6       1.6       9.9
1997    1.5       0.5       9.2
1998    2.3       0.2       7.8
1999    2.1       1.8       7.6
2000    3.1       1.2       8.4
2001    1.1      -0.6       8.3
2002    0.1       0.2       9.1
2003   -0.1      -0.3      10.
2004    1.6       1.3      10.1
2005    0.8      -0.2      11.3
2006    3.8       3.4      12.7
2007    3.3       2.9      14.2
2008    1.        0.6       9.6
2009   -6.1      -3.9       9.2
2010    4.3       3.2      10.6
2011    3.8       1.9       9.5
2012    0.4       0.6       7.7
2013    0.1      -0.1       7.7
2014    1.6       1.4       7.4
!TAB>
  WGER:= first WGINC
         next  WGER recpred*(WGINC:100+1)     at CINC
  EGER:= first EGINC
         next  EGER recpred*(EGINC:100+1)     at WGER
  CHINA:=first CINC
             next  CHINA recpred*(CINC:100+1)     at EGER
  rnd 1
```

Resultat:

```
YEAR, WGINC, EGINC, CINC, WGER, EGER, CHINA l
1988 100.    100.  100.   100.  100.   100.
1989   3.9     1.8   4.2  103.9 101.8  104.2
1991  11.1   -47.8  13.6  115.4  53.2  118.3
1992   1.7     6.2  14.3  117.4  56.5  135.2
1993  -2.6     8.7  13.9  114.3  61.4  154.
1994   1.4     8.1  13.1  115.9  66.3  174.2
1995   1.4     3.5  11.   117.5  68.7  193.4
1996   0.6     1.6   9.9  118.3  69.8  212.5
1997   1.5     0.5   9.2  120.   70.1  232.1
1998   2.3     0.2   7.8  122.8  70.2  250.2
1999   2.1     1.8   7.6  125.4  71.5  269.2
2000   3.1     1.2   8.4  129.3  72.4  291.8
2001   1.1    -0.6   8.3  130.7  71.9  316.
2002   0.1     0.2   9.1  130.8  72.1  344.8
2003  -0.1    -0.3  10.   130.7  71.9  379.3
2004   1.6     1.3  10.1  132.8  72.8  417.6
2005   0.8    -0.2  11.3  133.8  72.7  464.8
2006   3.8     3.4  12.7  138.9  75.1  523.8
2007   3.3     2.9  14.2  143.5  77.3  598.2
2008   1.      0.6   9.6  144.9  77.8  655.6
2009  -6.1    -3.9   9.2  136.1  74.7  715.9
2010   4.3     3.2  10.6  141.9  77.1  791.8
2011   3.8     1.9   9.5  147.3  78.6  867.1
2012   0.4     0.6   7.7  147.9  79.1  933.8
2013   0.1    -0.1   7.7  148.1  79.  1005.7
2014   1.6     1.4   7.4  150.4  80.1 1080.1
```

Da im Statistischen Jahrbuch von Deutschland keine Zahlen für Ostdeutschland für die erste Hälfte von 1990 angegeben waren, haben wir 1990 übersprungen. Das Wachstum von Westdeutschland war 1990 und 1991 5.7 und 5.1 und das Wachstum von China 3.9 und 9.3 Prozent. Das BIP der DDR von 1989 haben wir 2:1 in DM umgerechnet. Als Quelle für das Wachstum von China diente die englische Wikipedia ("China Economy"); als Quelle für Deutschland diente:
http://www.pdwb.de/w_biprei.htm und “Statista”.
Insbesondere wenn dieses Wachstum für viele Länder berechnet werden soll, sollten Tupeloperationen benutzt werden. Damit ergibt sich folgendes kurzes Programm:

```
...
gib JAHR,INC l
    INC! WGINC,EGINC,CINC
ALLINC:=first ALLINC next ALLINC recpred*(INC tup:100+1) at INC
sans INC
rnd 1
```

Anfrage 8.3: Berechne die Nullstelle der Sinusfunktion im Intervall [3, 4] durch Intervallhalbierungen.

```
LI,RE l := first 3.,4.
           next  if   LI recpred +(RE recpred)*0.5 sin <0
                 then LI recpred, (LI recpred + RE recpred *0.5)
                 else LI recpred+RE recpred*0.5,(RE recpred)
              while RE-LI>0.00001
avec LI pos- = 1
```

Resultat:

LI,	RE l
3.14158630371	3.1416015625

Zwischenresultat:

LI,	RE l
3.	4.
3.	3.5
3.	3.25
3.125	3.25
3.125	3.1875
3.125	3.15625
3.140625	3.15625
3.140625	3.1484375
3.140625	3.14453125
3.140625	3.142578125
3.140625	3.1416015625
3.14111328125	3.1416015625
3.14135742188	3.1416015625
3.14147949219	3.1416015625
3.14154052734	3.1416015625
3.14157104492	3.1416015625
3.14158630371	3.1416015625

Wir wissen dass das exakte Resultat pi=3.141592653... ist. Offensichtlich ist pi immer zwischen der linken (LI) und der rechten Intervallgrenze (RE).

Anfrage 8.4: Berechne die ersten 7 Fibonacci Zahlen.

```
[X! 1 .. 8]
FIBS:= first 0,1
       next  FIBS recpred nth 2,(FIBS recpred ++)   at X
```

Resultat:

X,	ZAHL,	ZAHL l
1	0	1
2	1	1
3	1	2
4	2	3
5	3	5
6	5	8
7	8	13

Anfrage 8.5: Berechne das Pascaldreieck.

```
aus 0 .. 11 tags X
Y:=first [Z! 1]
   next  begin aus Y pred
               W:= first 1 next Z recpred+Z    at Z
               weg Z
               rename W to Z
               add1 1 tag Z    end                  at X
```

Resultat:

```
X,   Zl l
 0   1
 1   1 1
 2   1 2 1
 3   1 3 3 1
 4   1 4 6 4 1
 5   1 5 10 10 5 1
 6   1 6 15 20 15 6 1
 7   1 7 21 35 35 21 7 1
 8   1 8 28 56 70 56 28 8 1
 9   1 9 36 84 126 126 84 36 9 1
10   1 10 45 120 210 252 210 120 45 10 1
```

9 „Hello World“-Spielerei

Anfrage 9.1: Drucke zwei Worte.
```
Hallo otto
```

Resultat mit Metadaten:
```
TABMENT! WORTl
WORTl
Hallo otto
```

WORT ist ein spezieller Texttyp. Ein Wort enthält kein Leerzeichen (Trennzeichen).

Anfrage 9.2: Gib mir ein Paar von zwei Worten.
```
Hallo, otto
```

Resultat mit Metadaten:
```
TABMENT! WORT,WORT
WORT, WORT
Hallo otto
```

Anfrage 9.3: Drucke einen Text mit Leerzeichen.
```
"Hallo otto"
```

Resultat mit Metadaten:
```
TABMENT! TEXT
TEXT
Hallo otto
```

Anfrage 9.4: Verbinde zwei Worte.
```
Hallo +text " " +text otto # +text: Konkatenation von Texten
```

Resultat:
```
TEXT
Hallo otto
```

Anfrage 9.5: Drucke einen Gruß mit einer Liste von Worten.
```
GRUSS:=[Hallo otto]
```

Resultat mit Metadaten:
```
TABMENT! GRUSS
GRUSS! WORTl
GRUSS
Hallo otto
```

Anfrage 9.6: Drucke zwei Worte in zwei Spalten.
```
LIEBER:=Hallo
GRUSS :=otto
```

Resultat:

```
LIEBER,GRUSS
Hallo  otto
```

Anfrage 9.7: Drucke einen Text mit Leerzeichen und Spaltennamen.

```
GRUSS:=“Hallo otto“
```

Resultat:

```
GRUSS
Hallo otto
```

Anfrage 9.8: Sortiere eine Menge von Wörtern.

```
GRUSS:={otto Hallo}
```

Resultat (ment) mit Metadaten:

```
<META>
TABMENT! GRUSS
GRUSS! WORTm
</META>
<GRUSS>
Hallo
otto
</GRUSS>
```

```
WORTm
Hallo otto
```

Anfrage 9.9: Nimm eines der Worte als Metadatum.

```
HALLO:=otto
```

Resultat:

```
HALLO
otto
```

10 o++oPS in der Schule

Es gibt viele mögliche Anwendungen von o++oPS in der Schule, insbesondere in Mathe und Informatik. Aber auch in anderen Fächern kann o++oPS genutzt werden, um bestimmte Daten von gegebenen Tabellen oder Dokumenten oder von der Wikipedia zu extrahieren. Wir wollen nicht alle möglichen Anfragen präsentieren. Wir beschränken uns hier auf "brute force Algorithmen" für den Mathematikunterricht. Das sind die einfachsten, d.h. die besten vom Standpunkt der Methodik. Da diese Algorithmen im Hauptspeicher ablaufen, behandeln wir keine Effizienzfragen. Wir haben bereits einen effizienten Intervallhalbierungsalgorithmus für eine Nullstelle betrachtet. Jetzt wollen wir uns einen einfacheren Algorithmus ansehen. Wir hoffen, dass es der einfachste Algorithmus für eine Nullstelle ist. Das Kapitel endet mit Programmen für Noten und Betrachtungen, die auch für den Kindergarten interessant sein könnten.

Anfrage 10.1: Berechne die Nullstelle der Sinusfunktion im Intervall [3, 4] in einfacher Weise.

```
[X! 3 ... 4! 0.00001]
avec X sin < 0
avec X pos = 1
```

Resultat:

```
Xl
3.1416
```

Anfrage 10.2: Berechne die ganzzahligen Nullstellen des Polynoms X^2–15X+56.

```
[X! -100 .. 100]
avec X polynom [1 -15 56] = 0 # oder X hoch 2 – (15*X) + 56=0
```

Resultat:

```
Xl
7 8
```

Mit den folgenden Programmen soll gezeigt werden, dass Schüler, denen die Differential- und Integralrechnung nicht vermittelt wurde, sehr wohl ihre Anwendungen meistern können, welche da wären:

1. Wie groß ist die Fläche unter einer Kurve?
2. Was ist das lokale Maximum einer Funktion?

Anfrage 10.3 a: Berechne die Fläche unter dem Kreis mit dem Radius 2 im Intervall [0, 2].

```
[X! 0.0001 ... 2!0.0001]
HOEHE := 4-(X*X) sqrt
RECHTECK:=HOEHE*0.0001
++ RECHTECK
```

Resultat (hsq):

```
3.14149223791
```

Anfrage 10.3 b: Anfrage 10.3a, aber kürzer und präziser

```
aus [X! 0.000001 ... 2!0.000001]
++ 4 - X*X sqrt *0.000001
```

Resultat (hsq):

```
3.14159165328
```

Jetzt zum brute force Algorithmus für das lokale Maximum.

Anfrage 10.4: Gesucht ist das Maximum der Sinusfunktion im Intervall [0, 3].

```
[X! 0. ... 3.! 0.00001]
avec X+0.000001 sin - (X sin) >= 0.
avec X pos- = 1
Y:=X sin
```

Resultat:

```
X,      Y
1.57079 0.99999999998
```

Anfrage 10.5: Berechne die Sinusfunktion und eine Näherung der ersten Ableitung im Intervall [0, 4].

```
[X! 0 ... 4,0.01]
SINUS := X sin
ABLEITUNG:=X+0.0001 sin  - (X sin*10000)
```

Resultat: Es besteht aus 401 Zeilen. o++oPS kann die Punkte in einem Bild ausgeben.

```
X,   SINUS,             ABLEITUNG l
0.    0.                 0.999999998333
0.01  0.00999983333417   0.999949498758
0.02  0.0199986666933    0.999799005067
0.03  0.0299955002025    0.999548532308
0.04  0.0399893341866    0.999198105529
0.05  0.0499791692707    0.998747759772
0.06  0.0599640064794    0.998197540071
...
3.99 -0.75022832823     -0.661141325258
4.   -0.756802495308    -0.653605779651
```

Die folgenden Beispiele basieren auf einer gedachten Tabelle mit Noten:

```
NAME,    (FACH,      KLAUSURl,  NOTEl l)l
Einstein  German     1 3        1 2 1 3 1
          Physics    1 1        1 2 1 1 1 1 1
          Algebra    1 2        1 1 1 1 2
          Art        3          3 3 2 1
Gauss     German     2 3        1 2 1 2
          Algebra    1 1        1 1 1 1
Guericke  Physics    1 1        1 1 2 1
          German     2 1        1 2 1 1 1
          Algebra    1 1        2 1 1 1 2
Newton    Physics    1 1        2 1 1 2 1
Konfuzius Philosophy 1 1        1 1 1 1 1
          Chinese    1 1        2 2 1 1 1
Marx      Economy    1 1        1 2 1 2 1
          Philosophy 1 2        1 3 1
Brecht    German     1 1        1 1 1 1 1 1 1 1
          Philosophy 1 2        2 1 1 1 2
Cantor    Set theory 1 1        1 1 1
```

Tabment 10.1: genies.tab

Anfrage 10.6: Berechne die gewichteten Durchschnitte für jede Person und jedes Fach und die Gesamtdurchschnitte. Sortiere die Daten.

```
aus genies.tab
gib DUR,(NAME,DUR,(FACH,DUR m)m)
    DUR:=KLAUSURl ++: *0.6 + (NOTEl ++: *0.4)! ++:
rnd 1
```

Resultat:

```
DUR,  (NAME,     DUR,  (FACH,      DUR m)m)
1.4    Brecht    1.2    German     1.
                        Philosophy 1.5
       Cantor    1.     Set theory 1.
       Einstein  1.7    Algebra    1.4
                        Art        2.7
                        German     1.8
                        Physics    1.
       Gauss     1.5    Algebra    1.
                        German     2.1
       Guericke  1.2    Algebra    1.2
                        German     1.4
                        Physics    1.1
       Konfuzius 1.1    Chinese    1.2
                        Philosophy 1.
       Marx      1.4    Economy    1.2
                        Philosophy 1.6
       Newton    1.2    Physics    1.2
```

Hier bemerken wir, dass die Anfrage nicht vollständig korrekt ist, da Einstein kein zweites Examen in Kunst hat. Da Einstein nur eine Klausur hat, kann man seinen gewichteten Durchschnitt anzweifeln. Solche Situationen sind in der deutschen Schule nicht zulässig. Wenn ein Schüler eine bestimmte Klausur nicht geschrieben hat, muss er sie wiederholen oder er bekommt eine 6. Wenn wir den Durchschnitt berechnen wollen, nachdem das erste Examen abgeschlossen wurde, können wir die folgende Formel benutzen:

```
DUR:=KLAUSURl nth 1 *0.3 + (NOTEl ++: *0.7) ++:
```

Anfrage 10.7: Gesucht sind alle Fächer und Personen, die eine 1 gefolgt von einer 3 bekommen haben.

```
aus  genies.tab
avec NAME FACH! NOTE=1 & NOTE succ=3 #succ:successor=Nachfolger
```

Resultat:

```
NAME,   (FACH,      KLAUSURl, NOTEl l)m
Einstein German     1 3       1 2 1 3 1
Marx     Philosophy 1 2       1 3 1
```

Jetzt wenden wir uns noch "einfacheren" Problemen zu. Diese Probleme könnten nicht nur für die Schule sondern auch für den Kindergarten interessant sein. Zurzeit besteht Konsens, dass die Addition die einfachste und die Division die komplizierteste der Grundrechenoperationen ist. Das

scheint falsch zu sein. Ich machte ein kleines Experiment mit einem 3 und einem 6 jährigem Kind aus dem Kindergarten. Die Aufgabe bestand darin, 10 Äpfel auf 4 Kinder zu verteilen. Die 4 Kinder wurden durch Photographien repräsentiert. Weder das 6 Jahre alte Kind noch das 3 jährige hatten ein Problem. Sie erreichten dasselbe Ergebnis für die **Division**, welches durch die folgende Tabelle repräsentiert wird:

```
KIND,    APFELl l
Ernst    | | |
Clara    | | |
Ulrike   | |
Sophia   | |
```

Tabment 10.2: 10 dividiert durch 4.tab

Was können wir von dem Experiment lernen?

1. Kinder können keinen Apfel teilen. Sie haben keine klare Vorstellung von 1/2 oder 2/3, …, so dass die gewöhnliche Division nicht gelehrt werden kann.
2. Es gibt trotzdem keinen Rest bei der Division, so dass nichts weggeworfen werden muss.
3. Es ist leichter mit Strichen zu rechnen als mit Dezimalzahlen.

Lasst uns die **Addition** betrachten - die nächst einfache Operation. Die einfachste Repräsentation der Zahl 3 sind drei Striche. Das Gleiche trifft auf die anderen natürlichen Zahlen zu. Jetzt betrachten wir lediglich die 3 und die 4. Dies seien die Äpfel von Ernst und Clara.
ernst.tab= [APFEL! 3 mal |]
clara.tab= [APFEL! 4 mal |]
Wir müssen diese durch Listen oder Multimengen repräsentieren, da die Menge {APFEL! | |} das Gleiche ist wie {APFEL! |}. Das Ergebnis jeder Operation würde dann 1 sein.

Anfrage 10.8: Addition 3+4:

```
aus ernst.tab,clara.tab
gib APFELl
```

Resultat:

```
APFELl
|||||||
```

Ist das Ergebnis als Dezimalzahl gewünscht, so nützt das folgende Programm:

Anfrage 10.8 b: Addition 3+4:

```
[APFEL! 3 mal | ], [APFEL! 4 mal |]
++1 APFEL
```

Es ist klar, dass solch eine Operation nur geringen Aufwand in der Lehre erfordert. Aber was ist das Ergebnis von 3 Äpfeln und 4 Birnen? Da ein Paar von Tabmenten wieder Tabment ist, ist folgendes Tabment Resultat dieser “Addition”:

```
[APFEL! | | | |],[BIRNE! | | |]
```

Die **Multiplikation** kann ebenfalls in einer einfachen Weise gehandhabt werden. Wir betrachten die einfache Frage, wie viele Äpfel werden für 4 Kinder benötigt, wenn jedes Kind 3 Äpfel

bekommen soll:

Anfrage 10.9: Multiplikation 4*3:
```
aus [KIND! 4 mal |]
:=  [APFEL! 3 mal |] at KIND
++1 APFEL
```

Zwischenergebnis nach der :=-Klausel:
```
KIND, APFELl l
|     | | |
|     | | |
|     | | |
|     | | |
```

Dieser Multiplikationsalgorithmus ist nicht nur leichter verständlich, er verdeutlicht auch, dass die Multiplikation die Berechnung der Fläche eines Rechtecks ist. Wie wir bereits wissen, ergibt sich das obige Zwischenergebnis auch durch das folgende Programm.

```
aus [KIND! 4 mal |], [APFEL! 3 mal | ]
gib KIND,APFELl l  atom! APFELl
```

Die **Subtraktion** für Kollektionen mit verschiedenen Elementen kann durch die Selektion ausgedrückt werden:

Anfrage 10.10 a: Subtraktion mit Mengen.
```
aus  {Ernst Clara Ulrike}
sans WORT inm {Ulrike Sophia}
```

Resultat:
```
WORT m
Clara Ernst
```

Wenn wir von Personen durch Striche abstrahieren, können wir zwei "Personen" durch ihre Position unterscheiden und somit den gleichen Algorithmus anwenden.

Anfrage 10.10 b: Subtraktion im unären System (ohne spezielle Operation).
```
aus  [PERSON! 5 mal |]
POS_PER:=PERSON pos
sans POS_PER in  begin [KIND! 2 mal |]
                       POS_KIND:=KIND pos
                       gib POS_KINDl end
gib PERSONl
```

Resultat:
```
PERSONl
| | |
```

Wir schließen dieses Kapitel mit den folgenden Gedanken:

1. Das Resultat unserer “arithmetischen Operationen” sind keine Zahlen sondern Tabellen.
2. Mit Tabellen kann man leichter rechnen als mit Zahlen, da ihr Abstraktionsniveau niedriger

ist.

3. Wir sollten den Dezimalzahlalgorithmus für die Division aus dem Lehrplan entfernen, genau wie der Algorithmus des Wurzelziehens entfernt wurde. Die Übungen für die anderen drei arithmetischen Operationen könnten reduziert werden, um beispielsweise Platz für eine Computersprache zu schaffen.
4. Rechnen mit Strichen (BAR) ist wichtiger als Rechnen mit Dezimalzahlen. Es ist wesentlich leichter die bekannten Gesetzte der Kommutativität, Assoziativität,... im unären System als im Dezimalsystem zu beweisen.
5. Dezimalzahlen bleiben wichtig für eine kompakte Darstellung großer Zahlen. Daher müssen Transformationen zwischen beiden Zahlsystemen gelehrt werden.
6. In Deutschland werden nichtganze Zahlen immer noch mit Komma geschrieben werden. Hier sollte die Schule mit der Ersetzung des Kommas durch den Dezimalpunkt vorangehen.

11 Ein Join für Strukturierte Tabellen (igib)

Das letzte Beispiel in Kapitel 6 kann man auch durch zwei aufeinanderfolgende gib-Anweisungen lösen. Damit lassen sich auch zwei Felder, die sich nicht auf einem HP befinden, in bestimmten Situationen auf einen hierarchischen Weg bringen. Das Problem wird noch wichtiger, wenn relationale Datenbanken gegeben sind. In einem Tupel solcher flacher Tabellen befinden sich lediglich die Felder einer Tabelle auf einem hierarchischen Weg. Daher ist eine gewöhnliche gib-Klausel nicht besonders ausdrucksstark. Relationale Systeme lösen das Problem durch Joins, die bekanntermaßen auf dem kartesischen Produkt basieren. Hierfür müssen Join-Bedingungen benutzt werden. In [Gol08] werden Experimente mit Studenten beschrieben. Sie zeigten, dass das Vergessen von Join-Bedingungen der häufigste semantische SQL-Fehler ist.
Für das Konstrukt igib, das in diesem Kapitel beschrieben wird, müssen Join-Bedingungen nicht geschrieben werden. Ferner basiert das igib Konstrukt überhaupt nicht auf dem kartesischen Produkt. In diesem Kapitels präsentieren wir einige typische Bespiele für das igib Konstrukt. Es scheint leicht zu sein, igib zu benutzen, seine Definition ist sicher etwas komplizierter.

Anfrage 11.1: Gib für alle Studenten, die nicht in Gerwisch wohnen, die einen KURS mit 1 abgeschlossen haben und die zeitintensive Projekte haben, die sehr guten Prüfungen und die zeitintensiven Projekte. Gruppiere die Studenten nach Wohnort.

```
aus  studenten1.tab,examen1.tab,projekte1.tab
sans ORT=Gerwisch
avec NOTE=1
avec STUNDEN>2
igib ORT,(NAME,(KURS,NOTE m),(PROJ,STUNDEN m)b)m
```

Resultat:

```
ORT,  (NAME,  (KURS,       NOTE m),(PROJ, STUNDEN m)b)m
Berlin Sophia  Datenbanken 1        Mao   5
               Otto        1        Ming  4
                                    Otto  6
Oehna  Ernst   Algebra     1        Fritz 4
               Geschichte  1
```

Dieses Resultat könnte beispielsweise durch das Einschieben einer gib-Klausel ermittelt werden:

```
gib STID,(NAME, ORT? ?),(KURS,NOTE m),(PROJ,STUNDEN m)m
```

Das zugehörige Zwischenresultat:

```
STID, (NAME,  ORT?)?,  (KURS,       NOTE m),(PROJ, STUNDEN m)m
1234   Ernst  Oehna     Algebra     1        Fritz  4
                       Geschichte  1
1245   Sophia Berlin    Datenbanken 1        Mao    5
                       Otto        1        Ming   4
                                            Otto   6
3456   Clara  Oehna     Datenbanken 1
4567   Ulrike                                Monet 10
5678                    Apel        1        Monet 20
                       Repin       1
```

Das Zwischenresultat erhält man durch Einfügen aller selektierten studenten1-Sätze (NAME-Segmente), anschließend der entsprechenden examen1-Sätze (KURS-Segmente) und schließlich

aller selektierten project1-Sätze (PROJ-Segmente). Jede dieser Einfügungen startet mit dem STID-Level. Die weiteren Einfügungen erfolgen in das (NAME, ORT?)-Level, (KURS, NOTE)-Level bzw. (PROJ, STUNDEN)-Level. Sabine wurde nicht eingefügt, ihr STID resultierte von examen1- und projekte1-Einfügungen. Entsprechende Daten und die Daten von Ulrike verschwinden auf Grund von fehlenden ORT-Werten. Clara verschwindet aufgrund von fehlenden PROJ-Werten.
Lass uns den allgemeinen Fall für das igib Konstrukt betrachten:

Algorithmus für igib:
Das igib Konstrukt hat die folgende Form:

```
aus  t1,t2,...,tn
weg  ueberfluessige Felder
igib s
```

Hierbei sind t1,...,tn Kollektionen, wobei es erlaubt ist, dass Felder in mehr als einer Kollektion vorkommen. Diese Felder repräsentieren unsere "join Bedingungen". Nichtstandard Join Bedingungen können durch eine vorangehende rename-Anweisung ausgedrückt werden. Ein Feld darf auf einem HP nicht mehrfach vorkommen. Alle Felder, die nicht in s vorkommen und nicht für den Join gebraucht werden, werden gleich zu Beginn gestrichen. Das igib Konstrukt ist durch eine Folge von einfachen Join-Schritten bzw. Doppelschritten (siehe Anfrage 11.4) gefolgt von

```
gib s
```

definiert. Die Folge der Joinschritte endet, wenn kein Joinschritt mehr anwendbar ist. Wenn ein Doppelschritt möglich ist, wird dieser realisiert. Dazu muss es eine nichtschmale Tabelle geben, die dann um zusätzliche Attribute erweitert wird und die zweite Tabelle muss so erweitert werden, dass ihre Daten in die erweiterte nichtschmale Tabelle eingefügt werden kann. Die nichtschmale Tabelle muss ein cf-Feld (siehe unten) enthalten, das sich nicht in der Tiefe 1 befindet.

Algorithmus für einen einfachen Joinschritt:
Wir beginnen mit m Kollektionen (zu Anfang ist das eine Teilmenge von t1,t2,…,tn), die paarweise die gemeinsamen Felder cf haben. Ein Joinschritt ist eine Folge von drei konzeptuellen Schritten, wobei in bestimmten Situationen einzelne Schritte übersprungen werden können:

```
aus c1,c2,...,cm
cf-intersection                          # Ergebnistyp: cfm
gib  cf,s1',s2',...,sm' m
,c1,c2,…,cm
gib2 cf,s1',s2',...,sm' m                # Joinsubschritt
strip                                    # Stripschritt
```

s1',...,sm' sind die um cf reduzierten Schemen von c1,...,cm. Im gib-Schritt bleiben die s1',…,sm' Kollektionen leer. gib2 stimmt mit gib überein, nur dass keine neuen cf-Werte eingefügt werden.
Nach jedem Joinschritt werden alle Kollektionen c1,c2,...,cm durch das Resultat des Joinschritts ersetzt.
Für das erste Beispiel gilt m=3 und cf={STID}. Jetzt betrachten wir ein Beispiel mit 2 Joinschritten.

Anfrage 11.2: Gruppiere und sortiere Studentennamen mit schlechten Noten nach Ort und Fakultät und gib die schlechten Kurse der Studenten aus.

```
aus  faks.tab,studenten1.tab,examen1.tab
avec NOTE>2
igib ORT,(FAK,DEKAN,(NAME,(KURS,NOTE m)b)m)m
```

Resultat:

```
ORT,  (FAK, DEKAN,  (NAME,  (KURS,   NOTE m)b)m)m
Berlin Inf  Reichel  Sophia  Algebra 3
```

Im folgenden werden die Joinschritte einfach durch Umstrukturierungen realisiert:

```
aus  faks.tab,studenten1.tab,examen1.tab
avec NOTE>2
gib  STID,(NAME,FAK,ORT? ?),(KURS,NOTE m)m ,FAKS
gib  FAK,DEKAN?,(NAME,FAK,ORT? ?),(KURS,NOTE m)m
gib  ORT,(FAK,DEKAN,(NAME,(KURS,NOTE m)b)m)m
```

In Zeile 3 wird FAKS als atomar betrachtet. In dieser Lösung realisierten wir zunächst einen Joinschritt zwischen examen1 und studenten1 und dann einen weiteren zwischen dem Resultat und faks.
Die zweite Lösung zuerst einen Joinschritt zwischen studenten1 und faks und dann das Resultat mit examen1 zu verschmelzen ist wahrscheinlich nicht ratsam, da weder eine Bedingung für faks noch eine Bedingung für studenten1 vorhanden ist.

Anfrage 11.3: Gib alle Studenten aus Oehna mit Dekan, Kursen und Projekten.

```
aus  studenten.tab,faks.tab
avec ORT=Oehna
igib STID,NAME,FAK,DEKAN,KURSm,PROJm m
```

Resultat:

```
STID, NAME, FAK,  DEKAN,   KURSm,     PROJm m
1234  Ernst Mathe Dassow   Algebra    Fritz
                           Geschichte Otto
                           Logik
3456  Clara Inf   Reichel  Datenbanken
                           OCaml
```

Das Programm wird durch den folgenden Joinschritt realisiert:

```
aus  studenten.tab,faks.tab
avec ORT=Oehna
gib  FAK,DEKAN?,(STID,NAME,PROJm,KURSm m)m
gib  STID,NAME,FAK,DEKAN,KURSm,PROJm m
```

Anfrage 11.4: Füge die Spalte Lehrender zu den Kursen der Studenten der Informatikfakultät.

```
aus  studenten.tab,kurse.tab
avec FAK=Inf
igib NAME,ORT?,(KURS,LEHRENDER,NOTE m),PROJm m
```

Resultat:

```
NAME,  ORT?, (KURS,         LEHRENDER ,NOTE m),PROJm m
Clara  Oehna   Datenbanken Saake      1
Sophia Berlin  Algebra     Reichel    3       Mao
               Datenbanken Saake      1       Ming
               Otto        Benecke    1       Otto
```

Würden wir hier einen einfach Joinschritt realisieren, wären die PROJm-Kollektionen in jedem Fall leer, da zu Projekten keine Kurse existieren:

```
aus  studenten.tab,kurse.tab
avec FAK=Inf
gib  KURS,LEHRENDER?,(NAME,ORT?,NOTEm,PROJm m)m
```

Das Problem wird durch einen Doppelschritt gelöst, dabei wird zunächst die Kurse-Tabelle um alle Attribute erweitert, damit sie anschließend in die erweiterte studenten-Tabelle eingefügt werden kann. Die erweiterte Studententabelle hat das Schema:
`STID, NAME, ORT?, (KURS, NOTE, LEHRENDER? m), PROJm m`
Damit der Lehrende in diese Tabelle eingefügt werden kann, muss die Kurse-Tabelle um STID, NAME, KURS und NOTE erweitert werden:

```
aus  studenten.tab,kurse.tab
avec FAK=Inf
gib  KURS,LEHRENDER?,(STID,NAME,NOTE m)m
```

Durch die nächste Anfrage wird klar, dass manchmal Joinschritte völlig vermieden werden könnten.

Anfrage 11.5: Gib alle Studenten von großen Fakultäten, die eine gute Note in Algebra haben. Strukturiere Studenten nach FAK und ORT und sortiere sie nach NAME.

```
aus  faks.tab, studenten1.tab, examen1.tab
avec STUDKAPAZI>300
avec NOTE<4 & KURS=Algebra
igib FAK,(ORT,NAMEb m)m
```

Resultat:

```
FAK,  (ORT,   NAMEb m)m
Inf    Berlin Sophia
```

Das interne optimierte Programm könnte wie folgt aussehen:

```
aus  studenten1.tab
avec FAK in begin aus  faks.tab
                  avec STUDKAPAZI>300
                  gib  FAKm end
avec STID in begin aus  examen1.tab
                   avec NOTE<4 & KURS=Algebra
                   gib  STIDm end
gib  FAK,(ORT,NAMEb m)m
```

Wir beschließen das Kapitel mit der Bemerkung, dass das igib Konstrukt auch sinnvoll eingesetzt werden kann, wenn nur eine Tabelle gegeben ist. Wir könnten in Anfrage 6.17a gib durch igib ersetzen. In diesem Fall wenden wir den Joinstep auf eine Tabelle an. Dabei genügt es, den „Stripschritt“ anzuwenden. Wenn eine Kollektion in einer Tabelle nur leere Kollektionen oder Singletons enthält, ersetzt strip alle diese Kollektionen durch optionale Schemen. Dadurch kann die Anzahl der hierarschichen Wege verringert werden.

```
KURS,   NOTE m          KURS,   NOTE ?
Algebra x               Algebra x
```

(Singleton) (optionale „Kollektion“)

Jetzt kann das Programm einfach durch Ersetzen von strip und gib durch igib modifiziert werden:

```
aus  studenten.tab
avec KURS=Algebra
igib NOTE,(NAME,(PROJ,STUNDEN m)b) m
```

Die selektive Kraft von **igib** kann mit dem **Join** verglichen werden, aber ohne kartesisches Produkt und mit strukturiertem Resultat. Ferner enthält das igib Ergebnis in der Regel weniger Redundanz als das join Ergebnis.

12 Stücklisten und ähnliche Probleme

Wenn eine Kollektion M1 von n Elementen gegeben ist, dann wird durch “lists k” die Kollektion aller Listen der Länge k von Elementen aus M1 gebildet. Leider ist diese Kollektion bereits für relativ kleine k (n^k) relativ groß.

Anfrage 12.1: Drucke alle Listen der Länge zwei von drei Elementen.

```
1 .. 4 tags X
lists 2
```

Resultat:

```
Xl l
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

Wenn wir nur die Listen wollen, die alle Elemente enthalten, könnten wir das in folgender Weise tun.

Anfrage 12.2 a: Berechne alle Permutationen von 4 Elementen.

```
1 .. 5 tags X
lists 4
avec  Xm = {1 2 3 4}
```

Resultat:

```
Xl l
1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 2 3
1 4 3 2
2 1 3 4
2 1 4 3
2 3 1 4
2 3 4 1
2 4 1 3
2 4 3 1
3 1 2 4
3 1 4 2
3 2 1 4
3 2 4 1
3 4 1 2
3 4 2 1
4 1 2 3
4 1 3 2
4 2 1 3
4 2 3 1
4 3 1 2
4 3 2 1
```

Da obiges Programm ineffizient ist, haben wir eine spezielle Operation implementiert. Das folgende Programm liefert dasselbe Resultat:

Anfrage 12.2 b: Berechne alle Permutationen von 4 Elementen.

```
[X! 1 .. 5]
permutations
```

Wir sehen keine wichtigen Anwendungen für die Potenzmengenoperation. Daher haben wir keine spezielle Operation implementiert. Die Potenzmenge kann mit der vlists Operation generiert werden. vlists k erzeugt alle Listen maximaler Länge k.

Anfrage 12.3: Berechne die Potenzmenge einer 4 elementigen Menge.

```
[X! 1 .. 5]
vlists 4
P := if Xl ++1=1 then 1,2
P2:= P pos
gib P2,Xm m
gib Xm m atom! Xm
```

Resultat:

```
X m m

1
2
3
4
1 2
1 2 3
1 2 3 4
1 2 4
1 3
1 3 4
1 4
2 3
2 3 4
2 4
3 4
```

Wir nehmen an, dass x der direkte Vorgesetzte von y ist. Dann gibt der transitive Abschluss alle Paare (x, z), so dass x Vorgesetzter von z ist.

Anfrage 12.4: transitiver Abschluss mit strukturierter Ausgabe

```
<TAB!
X,Y l
1 2
2 3
2 5
3 6
4 7
!TAB>
vlists 4
avec X succ nth 1 = Y | X pred nth 2 = X | Xl ++1=1
START:= Xl nth 1
END:= Ylv nth 1 at START
gib START,ENDm m
```

Resultat:

```
START, ENDm m
1      2 3 5 6
2      3 5 6
3      6
4      7
```

Typische Stücklistenprobleme können ebenfalls mit der vlists-Operation gelöst werden. Dann werden aber zu viele Listen generiert, und die gegebene Tabelle muss flach sein. Daher implementierten wir eine weitere Funktion paths, die auf eine Tabelle des Typs
KEY1,…,(KEY2,…l)m angewandt wird. Hier werden die Schlüssel als Knoten eines Graphen interpretiert, und jedes KEY2-Subtupel kann als Kante vom übergeordneten KEY1-Wert zum KEY2-Wert interpretiert werden. wege generiert aus einer solchen Tabelle eine Tabelle vom Typ (KEY2,…l)l, wobei der Nachfolger jedes KEY2-Subtupels das direkte Unterteil des Vorgängers ist. Der erste KEY2-Wert ist direktes Unterteil des gegebenen zweiten Arguments von paths.

Anfrage 12.5: Berechne die totale Anzahl der Vorkommen aller Unterteile und Baugruppen des Hundai i40 mit Aluminium Felge und ohne Klimaanlage.

```
<TAB!
OT,       EIGENSCHAFT, (UT,         ANZ)l m
Buchse    zylindrisch
Felge                  Alufelge   1
                       Stahlfelge 1
i30       modern       Rad        4
                       Motor      1
                       Karosse    1
i40       schnell      Rad        4
                       Reserverad 1
                       Karosse    1
                       Klimaanl   1
                       Motor      1
Karosse   blau
Klimaanl  robust
KolRing   rund
Kolben    leicht       KolRing    2
                       Buchse     1
Motor     schwer       Kolben     6
                       Schraube   8
Rad       rund         Schraube   5
                       Reifen     1
                       Felge      1
Reifen    schwarz
Schraube  stabil
!TAB>
avec UT! OT=Felge -> UT=Alufelge # MUSS Bedingung
sans UT! OT=i40 & UT = Klimaanl  # KANN Bedingung
wege i40
HILFSANZ:=first ANZ next HILFSANZ pred*ANZ at ANZ
avec UT! UT pos-=1
gib  UT,TOTAL m TOTAL:= HILFSANZ ! ++
```

Resultat:

```
UT,         TOTAL m
Alufelge     4
Buchse       6
Felge        4
Karosse      1
Kolben       6
KolRing     12
Motor        1
Rad          4
Reifen       4
Reserverad   1
Schraube    28
```

13 Erste Ideen für ein e-Buch für Mathematik

Schulbücher sind auch sehr gut strukturiert. Daher ist eine Anfragesprache für sie wichtiger als für Romane und vergleichbare Bücher. Obwohl viele gut strukturierte wissenschaftliche Bücher existieren, wollen wir uns hier auf Bücher für den Mathematikunterricht beschränken.
Die folgenden Vorteile werden mit einem elektronischen Buch mit XML-ähnlichem Kern anvisiert.

1. Ein Schüler kann sein Buch selbst erweitern. Das können Erklärungen des Lehrers, selbst berechnete Lösungen von Übungen oder zusätzliche Erklärungen zu Definitionen oder Theoremen sein.
2. Der Lehrer kann Teile von Büchern aller Schüler extrahieren und sie automatisch korrigieren lassen oder er kann Bemerkungen zu Definitionen oder Sätzen kontrollieren.
3. Viele Konzepte können durch Bilder, Videos und Animationen illustriert werden.

Die TTD eines Mathematikbuchs:

ClaraMathe8! KAPITELl
KAPITEL! KTITEL, ABSCHNITTl
ABSCHNITT! ATITEL, (TEXT | UEBUNG | DEF | THEO | BEWEIS | BILD | VIDEO | BEMERKUNG l)
UEBUNG! UENO, INHALT, LOESUNG
DEF, THEO, BEWEIS, BEMERKUNG! (TEXT | FETT l)
IMAGE, VIDEO! TITEL, JPG
TITEL, LOESUNG, FETT, BEMERKUNG, INHALT, CTITEL, STITEL, UENO! TEXT

Anfrage 13.1: Gib alle Definitionen und Theoreme der Geometriekapitel der e-Bücher der Klassen 6 und 7.

```
aus  ClaraMathe6,ClaraMathe7
avec Geometrie inm KTITLE
gib  DEF|THEO l  # oder gib DEFl,THEOl
```

Anfrage 13.2: Gesucht sind alle Übungen mit Lösungen der 8ten Klasse, die das Wort Pythagoras enthalten.

```
aus  ClaraMathe8
avec UEBUNG! Pythagoras
gib  UEBUNGl
```

Neben Anfragen an strukturierte Dokumente muss auch das Navigieren in Dokumenten möglich sein. Es ist wichtig, ein bestimmtes Wort, eine bestimmte Definition oder einen bestimmten Abschnitt in einem Dokument oder einer Menge von Dokumenten zu finden. Wir wollen dies wieder durch kurze Beispiele illustrieren. Wir nehmen an, dass dieser Editor, wie jeder andere Editor, einen "Pfeil" bietet, der es erlaubt, zum nächsten Objekt mit der spezifizierten Eigenschaft zu springen.

Find 13.1: Springe zur Definition der Primzahlen.

```
Primzahl inm FETT # oder präziser:  DEF! Primzahl inm FETT
```

Find 13.2: Springe von Theorem zu Theorem, die das Wort Primzahl enthalten.
```
THEO! Primzahl
```

Find 13.3: Springe zum Beginn des Abschnitts 3 des Kapitels 2.
```
ABSCHNITT! 2.3 in ATITEL # oder einfach 2.3 in ATITEL
```

Find 13.4: Springe zu einer Definition, die das Wort prim und den Stamm Zahl enthält.
```
DEF! Prim & Zahl*
```

Find 13.5: Springe zu Übung 3.4.12.
```
“3.4.12“ in UENO
```

14 Bemerkungen zu Suchmaschinen

Suchmaschinen sind die Softwareprodukte, die am weitesten verbreitet sind, da sie leicht zu nutzen sind und schnelle Antwortzeiten besitzen. Ein „Suchmaschinenprogramm“ besteht häufig nur aus einem, zwei oder drei Worten. Da der Suchraum aus weit mehr als einer Milliarde Seiten besteht, wundert sich niemand, dass die Antwortmenge aus 1000, Millionen oder mehr Dokumenten besteht.

Welches Kriterium nutzt die Suchmaschine, um die 3 oder 10 ersten Dokumente zu präsentieren. Google wählt die Seiten mit den größten Rangwerten. Welche Rangfunktion das System nimmt ist Firmengeheimnis. Die Situation ist auch tatsächlich noch schlechter. Die Firmen ändern die Rangfunktionen häufig, um zu garantieren, dass niemand in der Lage ist, die Seiten so zu preparieren, dass sie an die Spitze kommen. Dies ist auch ein Machtinstrument der Suchmaschinenfirmen.

Wie kann man dem langfristig begegnen?

Wir müssen eine neue Generation von Suchmaschinen bauen, mit denen der Nutzer besser beschreiben kann, an welchen Informationen er interessiert ist:

1. Es müssen höher selektive Bedingungen für die Seitenspezifikation zugelassen werden.

Das setzt zum Beispiel voraus, dass Dokumente des Internets oder von Teilen des Internets inhaltliche Tags besitzen. Wir sollten mit einer einfachen Art von Tags beginnen. Lediglich Worte oder kurze Texte werden getaggt. Dann sind Bedingungen "Magdeburg in GEBOREN" möglich. Es ist klar, dass solch eine Bedingung eine höhere Selektionskraft hat als die zwei Worte Magdeburg und geboren zusammen. Wenn wir höhere selektive Bedingungen haben, haben wir nicht Millionen sondern vielleicht nur Tausend oder weniger Ergebnisdokumente. Die Bedeutung der Rangfunktion sinkt dadurch. Genauso sollten Bedingungen der Art „LAENGE>2000“ möglich sein.

2. Der Nutzer kann seine Rangfunktion selber wählen.

Unser Nutzer beherrscht eine Anfragesprache und kennt mehrere Rangfunktionen, so dass er die geeignete Rangfunktion für seine Anfrage selbst wählen kann. Das gewöhnliche Sortieren von Seiten oder Teilen von Seiten betrachten wir hierbei ebenfalls als Rangfunktion.

3. Gib den Seiten eine Struktur, die die Suchmaschine versteht.

Die Seiten des Internets können nach Kapiteln mit Titeln,... strukturiert werden. Wenn die Suchmaschine das versteht, können Teile von Dokumenten selektiert werden und zu neuen Dokumenten verschmolzen werden. Die Suchmaschine benötigt dann entsprechende spezielle Indexe. Das bedeutet insbesondere, dass wir uns mehr für den Inhalt der Internetseiten interessieren sollten als für ihr Aussehen.

4. Restrukturierung einer oder mehrerer Seiten.

In diesem Buch haben wir viele Anfragen präsentiert, durch welche klar wurde, was wir hiermit meinen.

15 Klassische Nutzerschnittstellen versus o++oPS

Wenn ein wirklich neues Problem auftritt, muss der Nutzer oder sein Informatiker einen neue ad hoc Anfrage oder ein Programm formalisieren. Aber häufig muss der Nutzer eine Aufgabe mehrfach abarbeiten oder ein Problem lösen, welches einem bereits gelöstem sehr ähnlich ist. Daher muss es möglich sein, entsprechende Anfragen oder Programme zu speichern. Dann kann der Nutzer das selektierte Programm wiederholt laufen lassen oder er kann vorher bestimmte Werte, Spaltennamen oder Tabellennamen modifizieren.
Wenn man sehr viele gespeicherte Programme hat, kann das sehr unübersichtlich werden. Hier könnte ein Nutzerinterface Abhilfe verschaffen. Heutige Nutzerinterfaces haben den Nachteil, dass **unklar ist, was das Resultat eines Klicks bzw. einer Klickfolge ist.** Lasst uns das Problem an zwei Beispielen des Systems der Arbeitsagentur Deutschlands beschreiben:

- Obwohl ich selbst 3 Jobs eingegeben habe, die das Wort OCaml (eine französische Programmiersprache) enthalten, ergab die Suche eines Arbeitssuchenden nach OCAML keinen Job. Da OCaml relativ selten in den Daten des Arbeitsamtes vorkommt, d.h., das „Programm“ “OCaml“ eine hohe Selektivität hat, wäre dies ein schönes einfaches und vor allem nützliches Schlagwort. Das betrachtete System lässt dies dennoch nicht zu. Das System zwingt den Nutzer noch weitere Eingaben zu machen. Stellt man „Selbstständigkeit“ bzw. „Praktikum/Trainee“ ein, so ergibt sich jeweils ein Job, obwohl mindestens 2 Praktikantenstellen vorhanden sind. Da das System nicht auf einer Anfragesprache basiert, ist völlig unklar, ob das ein Fehler ist oder beabsichtigt war. Ich würde es als Fehler interpretieren.
- Sucht man auf der anderen Seite nach Arbeitskräften, die über „Experten“- Kenntnisse oder „Erweiterte Kenntnisse“ zu HASKELL (einer schottischen Programmiersprache) verfügen, so besteht keine Möglichkeit, dies auszudrücken. Gibt man „HASKELL“ mit einigen Zusatzinformationen ein, so liefert die Volltextsuche mehr als 200 Personen (Wie viele genau?). Diese verfügen aber fast alle nur über Grundkenntnisse. Hier ist sehr viel manueller Suchaufwand erforderlich. Dieser Mangel ist auch deshalb nicht einzusehen, da jeder Jobsuchende beim Erstellen seiner Anzeige den Grad der Beherrschung mit eingegeben hat. Präzisiert man jetzt noch den Beruf „Informatiker/in(Hochschule)“, so muss man den Abschluss, z.B. „Diplom(FH)“, zusätzlich angeben. Dann hat man zwar nur 88 Arbeitssuchende, es ist aber unklar, ob hier „Diplom(Uni)“ oder Master mit eingeschlossen sind,... .

Eine ähnliche Situation trat bei der Suche nach einem Einbaukühlschrank auf. Nachdem ich Einbaumöbel angeklickt hatte und mich weiter durch das System klickte, um den richtigen Kühlschrank zu finden, war es völlig unklar, ob meine "Anfrage im Hintergrund" noch auf Einbaumöbel eingeschränkt war oder ob ich mich wieder in "normalen" Kühlschränken bewegte.
Wie können diese Probleme gelöst werden?
Wir müssen die Systeme auf der Basis einer ausdrucksstarken Anfragesprache entwickeln, und wir müssen jede Anforderung in dieser Sprache ausdrücken. Daher fordern wir: **hinter jedem Klick oder jeder Klickfolge muss ein endnutzerlesbares Programm stehen.**
Was sind die Vorteile eines solchen Systems?
Wir gehen davon aus, dass die wesentlichen Operationen der Endnutzer Anfragen sind und dass der Nutzer die Sprache, die diese ausdrückt versteht.
Wenn eine Nutzeranfrage durch ein Interface generiert wurde, kann der Nutzer erkennen, ob seine Intention bzw. sein Interesse realisiert wurde. Wenn nötig modifiziert er einfach die Anfrage. Daher ist eine Endnutzersprache auch für normale Endutzerschnittstellen der obersten Ebene von großer Bedeutung.

Dadurch, dass der Kenner einer Anfragesprache weiß, welche Anfragen möglich sind, kann er seine Wünsche erst richtig artikulieren. Er kann die Möglichkeiten, die ein vernünftiges System bietet, weit besser ausschöpfen. Dadurch kann er viel manuellen Aufwand einsparen. Der Manager benötigt keinen EDV-Experten mehr. Das angefragte System ist keine totale Blackbox mehr, sondern nur die Implementation des Systems. Wie das System genau arbeitet, welche Indexe, welche Prozessoren, ... es wie benutzt, muss der Nutzer nicht wissen.
Den Vorteil von *Massendatenoperationen* kann man vielleicht mit folgendem *Einzeldaten-operationsproblem* vergleichen. Nehmen wir an, ein Nutzer hat ein System, das die Flächeninhalte, Volumina, ... von einzelnen Rechtecken, bzw. Quadern und Dreiecken, u.s.w. bestimmen kann. Wenn der Nutzer aufgefordert wird, die Länge und Breite eines Rechtecks einzugeben und er kennt sich mit der Multiplikation nicht aus, könnte er leicht die Länge in Metern und die Breite in Dezimetern eingeben. Dieses Problem existiert für alle, die in der Schule die Multiplikation verstanden haben, nicht. Analog sollte der Nutzer verschiedene Bedingungen, Umstrukturierungen, Aggregationsanweisungen, mengenweise Berechnungen, Operationen zum Verschmelzen von Tabellen und Dokumenten verstanden haben.

Jetzt betrachten wir das Problem am Beispiel Serienbrief. Viele Textverarbeitungssysteme verfügen über diese Komponente. Durch eine Klickfolge kann der Nutzer aus einer Personendatei und einer Briefdatei entsprechende Briefe generieren. Was sich hinter der Klickfolge vom Standpunkt von o++oPS verbirgt, wird unten im Programm verdeutlicht. Wir betrachten wieder nur inhaltliche Fragen. Styleinformationen müssten gegebenenfalls noch hinzugefügt werden. Es wird deutlich, dass dieses Programm leicht an neue Situationen (neue Personendatei mit anderen Spaltennamen, neue Briefstruktur,...) angepasst werden kann.
Die TTD der gegebenen Dokumente:

TABMENT! PERSONEN
PERSONEN! PERSONl
PERSON! NAME, VORNAME, SEX, ADDRESSE
ADDRESSE! STRASSE, NO, PLZ, ORT
NAME, VORNAME, SEX, STRASSE, NO, ORT! TEXT
PLZ! ZAHL

TABMENT! BRIEF
LETTER! ABSENDER, INHALT
ABSENDER, INHALT! TEXT

Anfrage 15.1: Schreibe einen Serienbrief an Personen einer Datei, die einer bestimmten Bedingung genügen mit unterschiedlichen Anreden für männliche und weibliche Personen.

```
aus personen.tab
avec ORT in [Magdeburg Hadmersleben]
:=   brief.xml at PERSON
STRASSENO := STRASSE +text " " +text NO at ABSENDER
PLZORT := PLZ text +text " " +text ORT  at STRASSENO
ANREDE:= if SEX=w then "Liebe Freundin “
         else “Lieber Freund “ +text VORNAME +text ", "
         at  PLZORT
weg PERSON
```

Abschließend bringen wir noch zwei Vorteile von Endnutzerschnittstellen, die auf einer Anfragesprache basieren:

1. Da der Kern einer Anfrage der meisten Endnutzerprobleme mit einer einzigen Anfrage gelöst werden kann, ist weit weniger Programmieraufwand erforderlich als in konventionellen Ansätzen. Das hängt auch mit der Erweiterbarkeit einer Endnutzersprache zusammen. Das heißt z.B., dass komplexe anwenderabhängige Funktionen in :=-Klauseln und Bedingungen benutzt werden können.
2. Da die Programmierung in höheren Sprachen schneller geht als in prozeduralen Sprachen, wird die Entwicklungszeit verkürzt und das System kann mit niedrigeren Kosten entwickelt werden.

16 Graphik

Wir wollen die einfachen geometrischen Vorstellungen, die in jeder Schule gelehrt werden, unterstützen. Ein Bild ist eine Menge von Punkten, wobei ein Punkt aus Koordinaten und gegebenenfalls einem Farbwert besteht. In der Geometrie besteht ein Segment oder Kreis aus unendlich vielen Punkten. Wir können jedoch davon ausgehen, dass jeder Endnutzer versteht, dass solch ein Modell nicht unmittelbar realisiert werden kann. Er weiß, dass das menschliche Auge oder ein Computer oder TV-Bildschirm nur mit endlich vielen Punkten arbeitet. Wir denken, dass gegenwärtige und zukünftige Computer leistungsstark genug sind, um Bilder mit Millionen Punkten mit mengenweisen Operationen zu verarbeiten. Wir hoffen, dass die ersten Nutzer unseres Systems Schüler mit Interesse für Geometrie und Mathematik sein werden.
Durch die folgende Tabelle werden lediglich drei Punkte beschrieben, die visualisiert werden können, wenn man auf den Bild-Button klickt. Diese Punkte sind aufgrund ihrer geringen Größe aber schwer zu erkennen.

Graphik 16.1: Drei Punkte
```
<TAB!
X,Y l
1 2
1 3
4 5
!TAB>
```

Graphik 16.2: 70 Punkte
```
  [X! 0 ... 7!0.1]
  Y:= X sin
```

Wenn wir 0.1 durch 0.01 ersetzen, dann ist die Sinusfunktion als komplette Kurve sichtbar.
Die folgenden Programme unseres Prototyps generieren drei Rechtecke der Tricolore bzw. die drei Bestandteile der deutschen Fahnen.

Graphik 16.3: Trikolore
```
  R1 := 1 ... 3!0.005 tags X
  :=  0 ... 3!0.005 tags Y at X
  R2 := R1 +tup (2,0)
  R3 := R1 +tup (4,0)
  RGB := (0.,0.,1.) leftat R1 # blue
  RGB := (1.,1.,1.) leftat R2 # white
  RGB := (1.,0.,0.) leftat R3 # red
```

Graphik 16.4: Deutsche Fahne.

```
R1:= 1.6 ... 5,0.005 tags X
:= X sin *0.3 + 2 ... X sin 3 *0.3 3,0.005 tags Y
R2:=R1 -tup (0,1)
R3:=R1 -tup (0,2)
RGB:=Black  leftat R1
RGB:=Red    leftat R2
RGB:=Yellow leftat R3
```

Resultat: Detail der deutschen Fahne

Im letzten Programm ist R1 vom Typ X,Yl l. R1 wurde durch die ersten zwei Zeilen generiert. R1 wurde mittels := in Zeile drei um R2 erweitert. R2 unterscheidet sich von R1 nur durch die Y-Koordinaten, die um 1 vermindert wurden. Insgesamt wurde ein Tabment von Typ (RGB, R1, RGB, R2, RGB, R3) erzeugt. Aufgrund der Verwendung der Sinusfunktion weht die Fahne. Jeder Punkt nimmt die Farbe des vorangehenden Farbwertes an.
Jetzt wenden wir eine "Datenbankselektion" auf obiges Bild an:

Graphik 16.5: Die deutsche Fahne mit einer Selektion.

```
R1:=1.6 ... 5,0.003 tags X
:= X sin + 2 ... X sin *0.3 + 3,0.003> tags Y at X
avec X – 2 hoch 2 + (Y – 2.7 hoch 2) > 0.01
R2:=R1 -tup (0,1)
R3:=R1 -tup (0,2)
RGB:=black leftat R1
RGB:=red leftat R2
RGB:=yellow leftat R3
```

17 Ein Kurzvergleich mit SQL

SQL (Structured Query Language) war eine übersichtliche Datenbanksprache, die zuerst in den Arbeiten [C+76] und [AC75] vorgeschlagen wurde. SQL basiert wesentlich auf der Relationenalgebra, welche aus den folgenden Operationen besteht:

1. Selektion
2. Projektion
3. Join oder kartesisches Produkt
4. Mengentheoretische Operationen Vereinigung, Differenz und Durchschnitt
5. Umbenennung von Spaltennamen

Durch obige Arbeiten wurden die folgenden Dinge zur Relationenalgebra hinzugefügt.

1. arithmetische Operationen
2. Aggregationen
3. Group By
4. Select (ohne Distinct)
5. Order By
6. Nested Queries
7. Nulls

Später wurde SQL um viele weitere Konzepte (outer joins, limit, cube, …) erweitert, so dass es nun etwas unhandlicher geworden ist. Daher ist die Frage der Ausdruckskraft nicht das Wichtigste sondern die Frage der leichten Nutzbarkeit.

Wir beginnen mit Bemerkungen zu 3 der vorangehenden Anfragen. Um Anfrage 4.7 in SQL auszudrücken, müssen wir die gegebene Studententabelle in 3 erste Normalformtabellen zerlegen, wie wir gesehen haben. Da das Ergebnis der entsprechenden SQL Anfrage auch in erster Normalform sein muss, enthält es beispielsweise 100 Tupel für jeden Ergebnisstudent, der 10 Prüfungen und 10 Projekte hat (kartesisches Produkt). Weiterhin benötigt die SQL Anfrage 4 Joinbedingungen. Die Zwischenanfrage

```
SELECT *
FROM students1
WHERE “Algebra” in (SELECT COURSE
                    FROM exams1
                    WHERE students1.STID=exams1.STID)
```

enthält lediglich students1 Daten und die Anfrage

```
SELECT *
FROM students1, exams1
WHERE students1.STID=exams1.STID and KURS=”Algebra”
```

enthält lediglich examen Daten. Eine Bedingung KURS=”Datenbanken” hinzuzufügen ist sinnlos.

Anfrage 4.8 erfordert in SQL ebenfalls einen Join. Ferner ist es nicht möglich, eine 1.NF-Relation zu erzeugen, die gleichzeitig Menge und Multimenge ist.

Anfrage 6.12 beispielweise erfordert Joins und ein GROUP BY. Wegen des GROUP BY DEPT, können die Daten der einzelnen Studenten nicht im SELECT-Teil erscheinen. Ferner muss die totale Aggregation CNT1 ohne Group By in einer separaten Unteranfrage berechnet werden.

Die relationale Vollständigkeit von o++oPS kann bewiesen werden, indem die Operationen der Relationenalgebra ausgedrückt werden. Dafür seien

r1,r2: A1,A2,...,An m und

s1: B1,B2,...,Bk m

drei Relationen, in denen r1 und s1 gemeinsame Spaltennamen haben können.

Anfrage 17.1: Vereinigung von r1 und r2:

```
  aus r1,r2
  gib A1,A2,...,A m
```

oder

```
  aus r1 +m r2
```

Anfrage 17.2: mengentheoretischer Durchschnitt von r1 und r2:

```
  aus  r1
  avec A1,A2,...,An m inm r2
```

oder

```
  aus  r1,r2
  igib A1,A2,…,An m
```

oder:

```
  aus  r1,r2
  R11:=R1/A1
  R21:=R2/A1
  gib  A1,A2,...,An,R11m,R21m m
  avec R11=R11
  avec R21=R21
  weg  R11, R12
```

Anfrage 17.3: Mengendifferenz: r1 \ r2:

```
  aus  r1
  sans A1,...,An m inm r2
```

Es sei angemerkt, dass die mengentheoretischen Operationen Vereinigung, Durchschnitt und Differenz im Kontext von strukturierten Tabellen nicht mehr die große praktische Bedeutung besitzen. Der Durchschnitt der beiden folgenden Tabellen ist beispielsweise leer:

```
NAME,  FACHm m
Ernst  Mathe
       Deutsch

NAME,  FACHm m
Ernst  Mathe
       Physik
```

Anfrage 17.4: (natürlicher) Verbund r1 |×| s1, falls A1=B1 und A2=B2:

```
aus  r1
:=   s1 at An
avec A1=B1 & A2=B2
gib  A1,A2,...,An,B3,B4,...,Bk m
```

Anfrage 17.5: Selektion:

```
aus r1
avec Ai = ai # oder Ai = Aj
```

Anfrage 17.6: Projektion von r1 auf eine Teilmenge {C1,C2,...,Ck} ⊆ {A1,A2,...,An}.

```
aus r1
gib C1,C2,..., Ck m
```

Anfrage 17.7: Renaming: ersetze A1 durch C1:

```
aus r1
rename A1 to C1
```

oder

```
aus r1
C1  :=A1
gib C1,A2,...,An m
```

Jedes Ergebnis der obigen 7 Anfragen kann abgespeichert werden und wieder als Input weiterer Operationen dienen. Daher ist **o++oPS is relational vollständig**.

Um weitere Möglichkeiten von SQL zu beschreiben, beziehen wir uns im Folgenden auf Tabellen, die Multimengen repräsentieren:

```
r1,r2: A1,A2,....,An b
s1: B1,B2,...,Bk b
```

Anfrage 17.8: "SELECT" (Projektion ohne Duplikatelimination)

```
SELECT A3,A5,A6,A9
FROM r1
```

```
aus r1
gib A3,A5,A6,A9 l
```

oder

```
aus r1
weg A1,A2,A4,A7,A8,A10,...,An
```

Anfrage 17.9: Duplikate elimination:

```
SELECT DISTINCT A3,A5,A6,A9
FROM r1

  aus r1
  gib A3,A5,A6,A9 m
```

Anfrage 17.10: order by:

```
SELECT A1,A2,A3,A4,A5,A6
FROM r1
ORDER BY A2,A4

  aus r1
  gib A2,A4,A1,A3,A5,A6 b
```

```
SELECT A1,A2,A3,A4,A5,A6
FROM r1
ORDER BY A2 DESC A4 ASC

  aus r1
  gib A2,(A4,A1,A3,A5,A6 b) m
  gib A1,A2,A3,A4,A5,A6 l
```

Es sei angemerkt, dass das Zwischenergebnis der letzten Anfrage bereits besser ist als das Endergebnis. Es ist aber nicht so dicht am SQL Ergebnis.

Anfrage 17.11: group-by:

```
FROM r1
GROUP BY A3,A5

  aus r1
  gib A3,A5,(A1,A2,A4,A6,...,An b)m
  gib A1,A2,…,An l
```

Das obige SQL Statement ist nicht vollständig. Wenn wir es durch ein beginnendes SELECT * vervollständigen, erscheint eine Fehlernachricht, da das Ergebnis des GROUP BY Statements ein Tupel für jede Gruppe zurückgeben muss und die Tupel in erster Normalform sein müssen. Wenn wir das GROUP BY unmittelbar in SQL definieren würden, könnten wir die obige o++oPS Anfrage nehmen. Hierbei ist wieder das Zwischenresultat besser als das Endergebnis.

Anfrage 17.12: group-by in Aktion:

```
SELECT A3,A5,sum(A8)
FROM r1
GROUP BY A3,A5

  aus r1
  gib A3,A5,CNT1 m
      CNT1:= A8 ! ++
```

Die obige o++oPS Anfrage erlaubt ebenfalls Attribute, die funktional abhängig von A3, A5 sind. Weiterhin könnten beliebige Attribute wie A7 mit einem Fragezeichen hinzugefügt werden.

Anfrage 17.13: having:

```
SELECT A3,A5,sum(A4)
FROM r1
GROUP BY A3,A5
HAVING sum(A4) > 600

  aus  r1
  gib  A3,A5,SUMA4 m
       SUMA4:=  A4 ! ++
  avec SUMA4 > 600
```

Anfrage 17.14: kartesisches Produkt:

```
SELECT *
FROM r1,s1

  aus r1
  :=  s1 at An
  gib A1,A2,..An,B1,B2,...,Bk b
```

Hier ist das Zwischenresultat wieder dem Endergebnis vorzuziehen.

Anfrage 17.15: left outer join:

```
SELECT *
FROM r1 left join s1 on A1=B1

  aus  r1
  :=   s1 at An
  avec B1! A1 = B1
```

Da das Ergebnis der letzten Anfrage vom Typ A1,A2,...,An,(B1,...,Bk m)m ist, werden die Null-Werte durch innere Kollektionen repräsentiert.
Eine zweite Lösung mit flachem Output:

```
  aus  r1
  :=   s1 at An
  avec A1=B1
  gib  A1,A2,...,An,B1,B2,...,Bk b
  ,    begin aus  r1
             sans A1 in  begin aus s1
                                 gib B1m end end
  gib A1,A2,...,An,B1?,B2?,...,Bk? b
```

Das Ergebnis der vorletzten Zeile könnte man wieder als Endergebnis betrachten. Es ist allerdings vom Typ A1,...,An,B1,..Bk b, (A1,...,An b).

Anfrage 17.16: outer join:

```
SELECT *
FROM r1 full outer join s1 on A1=B1
```

```
  aus  r1
  :=   s1 at An
  avec A1 = B1
  gib  A1,A2,...,An,B1,B2,...,Bk m
  ,begin aus  r1
         sans A1 inm begin aus s1
                            gib B1k m end end
  , begin aus  s1
          sans B1 inm  begin aus r1
                               gib A1m
  gib A1?,A2?,...,An?,B1?,B2?,...,Bk? b
```

Anfrage 17.17: limit:

```
SELECT *
FROM r1
LIMIT 10
```

```
  aus  r1
  avec A1 pos <= 10
```

```
SELECT *
FROM r1
LIMIT 5 OFFSET 10
```

```
  aus  r1
  avec A1 pos > 10
  avec A1 pos <= 5
```

oder

```
  aus r1
  avec A1 pos>10 & A1 pos<=15
```

Abschließend wollen wir den cube-Operator betrachten. Dabei betrachten wir nur ein Beispiel aus [KBL06] Seite 723. Dieses Beispiel bezieht sich auf eine Faktentabelle
`SALES: Market_Id,Product_Id,Time_Id,Sales_Amt m`
der Seite 714. Eine Zeile (x,y,z,w) repräsentiert den Betrag w der Verkäufe in Dollar des Marktes mit Id x und Produkt des Id y und dem Zeitintervall mit der Id z.

Anfrage 17.18: cube Operator:
```
SELECT S.Market_id,S.Product_Id,SUM(S.Sales_Amt)
FROM Sales S
GROUP BY CUBE(S.Market_Id,S.Product_Id)

  aus Sales.rel
  gib SU,(MARKET_ID,SU,(PRODUCT_ID,SU m)m),(PRODUCT_ID,SU m)
      SU:= SALES_AMT ! ++
```

Wenn wir drei Dimensionen in der Zielstruktur hätten, erschiene die SU-Spalte 8 mal in der Zielstruktur:
`gib SU,(D1,SU,(D2,SU,(D3,SU m)m),(D3,SU m)m),(D2,SU,(D3,SU m)m),(D3,SU m)`
Wenn die Anzahl der Dimensionen weiter steigt, sollte eine Abkürzung implementiert werden:
`gib cube(DIM1,DIM2,...,DIMn) AGG! …`
Wir wollen das Kapitel mit einen kleinen Vergleich zwischen SQL und o++o beenden.

Merkmal	**SQL**	**o++o**
Objekte	flache Tabellen	strukturierte Tabellen und Dokumente
materialisierte Joins	in wenigen Situationen	ja, da Wiederholgruppen unmittelbar gespeichert werden
Null-Werte	ja (nur für eine Spalte und eine Zeile)	nein
optionale Werte	nein	ja (für ein beliebiges Schema)
Selektion nach der Position	nur limit	pos, pos-, succ, pred, ...
group by, having	ja	nein (dafür gib, mit)
Restrukturierung	nein	ja
Aggregationen	spaltenweise	gleichzeitig vertikal und horizontal
aggregation types	++:, ++1, max, min, ++:	++:, ++1, max, min, ++:, prd (Produkt), all, exs, median, variance, linreg (lineare Regression)
cube	ja	möglich ohne Null-Werte und ohne optionale Werte
SFW-construct	ja	nein, aber igib
debugging	nur Gesamtergebnis eines SFW-Konstrukts präsentierbar	Resultat nach jeder Operation repräsentierbar
Programmierung durch Denken	ja	ja
Programmierung durch Bauentscheidungen	?	ja (z.B.: igib,)

Table 17.1: Vergleich von SQL und o++oPS

Wir überlassen es dem Leser, die Anfragen der vorangehenden Kapitel in SQL zu übersetzen. Wir bemerken lediglich, dass es problematisch ist, die einfache Anweisung

```
:=+ students1,exams1,projects1
```

, bei der sich die Studententabelle ergibt, mit SQL auszudrücken.

```
SELECT *
FROM students1, exams1, projects1
WHERE STUDENTS1.STID=EXAMS1.STID AND STUDENTS1.STID=PROJECTS1.STID
```

zum Beispiel enthält für einen Studenten mit 20 Prüfungen und 10 Projekten 200 Zeilen, die den Studenten beschreiben.

18 Das abstrakte Tabment

Der Begriff Tabment ist abstrakt. Das heißt, dass es viele Repräsentationen für ein Tabmentobjekt gibt. Die sehr allgemeine, abstrakte, rekursive Definition besteht aus 6 kurzen generierenden Regeln:

1. Jeder (elementare) Wert v ist ein Tabment (El_tab(v)).
1. Jedes Tabment t kann durch einen einfachen Namen n getaggt werden (Tag0(n,t)).
2. Wenn wir n Tabmente t1, t2, ..., tn haben, dann existiert stets ein Tupel dieser Tabmente (Tuple_t(t1,t2,…,t2)).
3. Für jedes Schema s, jedes Kollektionssymbol C und n Tabmente t1, t2,…, tn des Typs s existiert eine Kollektion dieser Tabmente (Coll_t(C,s,t1,t2,…,tn)).
4. Für jede Liste von Schemen s1,s2,…,sn und jedem Tabment t der Typen s1, s2,... oder sn existiert eine Alternative (Alternate_t(s1,s2,…,sn,t)).
5. Es existiert ein leeres Tabment vom leeren Typ (Empty_t).

Wir illustrieren die Definition zunächst durch einfache Beispiele:
8 und Ernst sind Tabmente aufgrund von Regel 1.
Wenn wir beide Werte durch ALTER bzw. NAME taggen, erhalten wir nach Regel 2 die Tabmente <ALTER: 8>=<ALTER>8</ALTER> und <NAME: Ernst>=<NAME>Ernst</NAME>.
Jetzt können wir ein Tupel (Paar) dieser zwei Tabmente durch Regel 3 bilden:

```
<ALTER: 8>,<NAME: Ernst>=   ALTER,NAME
                            8     Ernst
```

Wir können 3 drei weitere ähnliche Paare desselben Typs bilden. Dann ergibt sich mit Regel 4, dem Kollektionssymbol L, dem Schema (ALTER,NAME) und den vier Paaren ein Tabment

```
ALTER,NAME l
8     Ernst
5     Clara
3     Ulrike
3     Sophia
=
<ALTER>8</ALTER>
<NAME>Ernst</NAME>
<ALTER>5</ALTER>
<NAME>Clara</NAME>
<ALTER>3</ALTER>
<NAME>Ulrike</NAME>
<ALTER>3</ALTER>
<NAME>Sophia</NAME>
```

.

Obwohl in der ersten Repräsentation die Tags lediglich einmal erscheinen, ist klar welche Werte zu NAME und welche zu ALTER gehören. Die zweite Darstellung ist nur eindeutig, wenn der Typ `ALTER,NAME l` des Objekts bekannt ist. Theoretisch könnte es auch ein Tupel dieser 4 Paare sein.
Jeder strukturierte Satz (Tupel) der Datei ottos kann in gleicher Weise repräsentiert werden. Das dritte Subtupel kann beispielsweise in folgender Weise erzeugt werden:

```
  NAME,             GEBORENIN,    (TAT,                                  JAHR l)
  Otto der Grosse   Altsachsen(De) Zum Koenig von Deutschland gewaehlt     936
                                   Die Ungarn auf dem Lechfeld geschlagen  955
                                   Erster Kaiser des Heil. Roem. Reichs    962
= Tuple_t(Tag0(NAME,El_tab(Otto der Grosse)),Tag0(GEBORENIN,El_tab(Altsachsen(De))),
  Coll_t(L,(ACT,YEAR),
```

```
Tuple_t(Tag0(TAT,El_tab(Zum Koenig von Deutschland gewaehlt)),
        Tag0(JAHR,El_tab(936))),
Tuple_t(Tag0(TAT,El_tab(Die Ungarn auf dem Lechfeld geschlagen)),
        Tag0(JAHR,El_tab(955))),
Tuple_t(Tag0(TAT,El_tab(Erster Kaiser des Heil. Roem. Reichs)),
        Tag0(JAHR,El_tab(962))))
```

Wenn wir dieses Tupel mit otto3 bezeichnen und entsprechend das i-teTupel durch ottoi, dann kann die gesamte Tabelle ottos in folgender Weise generiert werden:

`ottos=Coll_t(List,(NAME,GEBORENIN,(TAT,JAHR)l),otto1,otto2,…,otto6)`

Die gesamte Datei besitzt noch einen zusätzlichen Tag:

`ottos.tab = Tag0(OTTOS,ottos).`

Jetzt wollen wir die obigen abstrakten Regeln etwas detaillierter beschreiben.

1) El_tab(v): In unserem Model muss für jeden elementaren Wert bekannt sein von welchem Typ er ist. Wir unterscheiden zwischen TEXT, ZAHL (integer), PZAHL (float), BOOL (Bool'scher Wert true oder false), WORT, and BAR (nur ein Wert | ist erlaubt). Manchmal wir der Text durch “” eingeschlossen.

2) Tag0(NAME,t): Wir haben gesehen, dass in der tab-Darstellung die Tags von elementaren Werten nicht geschrieben werden; sie wurden im Tabellenkopf angegeben. Weitere Tags werden unterdrückt, damit sie nicht die tabellarische Ordnung zerstören. Sie können aber im Kopf des Lesers erzeugt werden. In der XML Darstellung wird ein Tag0 durch einen Beginn und Endtag (z.B. <NAME>, </NAME>) gekennzeichnet. Wenn der Endtag in der gleichen Zeile wie der Beginntag ist, sollte er durch > abgekürzt werden. Noch intuitiver scheint die folgende Schreibweise zu sein: `<NAME: value>`

Der Typ von Tag0(NAME,t) ist NAME und NAME besitzt dann den Typ von t.

3) Tuple_t(t1,t2,…,tn): Häufig werden Tabmente durch Komma getrennt. Das Komma wird im Wert-Teil der tab-Darstellung eingespart, da es bereits im Kopf enthalten ist. Der Typ des Tupels setzt sich aus den Typen der Komponenten zusammen. Komponenten werden im allgemeinen horizontal arrangiert.

4) Coll_t(c,s,t1,t2,...tn): Im Gegensatz zu Tupeln wächst der Kopf einer Kollektion nicht mit der Anzahl n der beteiligten Tabmente. „s c“ ist der Kopf (Typ). Im Fall von optionalen Elementen schreiben wir s? oder (s)?. Die Tabmente t1,t2,…,tn werden in der Regel vertikal angeordnet. Um Platz auf dem Papier oder Bildschirm zu sparen, werden Kollektionen des Typs NAMEc auch horizontal angeordnet. In der XML-Repräsentation werden sie immer vertikal arrangiert. Kollektionen bestehen in der Regel aus mehr Tabmenten als Tupel. In Ihnen kann selektiert werden.

5) Alternate_t(s1,s2,…,sn,t) ist vom Typ (s1 | s2 | ...| sn) und t muss vom Type s1 oder s2,… oder sn sein. Ein einzelnes solches Tabment hat wenig praktische Bedeutung. Erst im Rahmen einer entsprechenden Kollektion macht das Sinn:

LEBENSLAUF: TEXT | WICHTIG l ist ein Text, der im Innern WICHTIG-Tags enthalten kann.

<LEBENSLAUF>

Ich wurde <WICHTIG: am 18.03.1953 in Hadmersleben geboren>. Mein <WICHTIG: Vater war Bäcker>. Meine Mutter …

</LEBENSLAUF>

Hier ist „Ich wurde “ vom typ TEXT, das zweite Element vom Typ WICHTIG und das dritte wieder vom Typ TEXT. Genau genommen sind alle drei Elemente vom Typ (TEXT | WICHTIG).

6) Empty_t ist nicht so wichtig. Durch diese Regel wird ein einziges Element erzeugt. Es dient häufig als Fehlerwert. Es hat das leere Schema als Typ.

19 Konkrete Tabmente

Wir haben bereits mehrere verschiedene Repräsentationen von Tabmenten kennengelernt: tab, xml, csv, hsq. Das sind aber noch nicht alle Möglichkeiten, die bei dem allgemeinen Tabmentbegriff möglich sind. Relationen von Datenbanken oder gesamte Datenbanken kann man ebenfalls als Tabmente auffassen. Sie schöpfen allerdings die Möglichkeiten des Tabmentbegriffs nicht voll aus. Das gilt auch für csv-Dateien. Diese Dateien haben dafür in der Regel den Vorteil in der Effizienz oder in der leichteren Verarbeitung in Anwendungen. In diesem Abschnitt wollen wir die für unser System neu entworfenen Dateistrukturen beleuchten. Die folgenden drei Dateitypen benutzen alle eine einheitliche Darstellung der Metadaten. Sie wurden in erster Linie entworfen, um ein bequemes Arbeiten mit relativ allgemeinen Strukturen zu erlauben. Bei den meisten anderen Konzepten stand eine effiziente Verarbeitung der Daten im Vordergrund.

Eine Datei hat einen Namen, diesen Namen, reduziert um den Punkt mit der Endung, nutzt o++o als Tag, der alle Daten der Datei umschließt. Existiert bereits ein solcher Tag um die gesamte Datei, so wird dieser durch den reduzierten Dateinamen ersetzt.

1. Die tab-Struktur

Sehen wir uns noch einmal die Tabelle studenten.tab aus Kapitel 3 an:

```
STID, NAME,  ORT?,    STIP,FAK,  (KURS,       NOTE m),(PROJ, STUNDEN m)m
1234  Ernst  Oehna    500  Mathe  Algebra     1        Fritz  4
                                  Logik       2        Otto   2
                                  Geschichte  1
1245  Sophia Berlin   400  Inf    Algebra     3        Mao    5
                                  Datenbanken 1        Ming   4
                                  Otto        1        Otto   6
3456  Clara  Oehna    450  Inf    Datenbanken 1
                                  OCaml       2
4567  Ulrike          400  Kunst                        Monet 10
5678  Kaethe Gerwisch   0  Kunst  Repin       1        Monet 20
                                  Apel        1
```

Eine gewöhnliche tab-Struktur beginnt mit einer Metadatenzeile, wobei die Metadaten lediglich aus einem Schema bestehen und folgenden Zeilen für die Primärdaten.
Diese übersichtliche Gliederung in Spalten ist vorteilhaft, wenn die Spaltenanzahl nicht zu groß ist. Ein Wert einer Spalte beginnt mit dem ersten Buchstaben des Spaltennamens und endet ein Zeichen vor dem Beginn der nächsten Spalte. Dadurch steht zwischen zwei benachbarten Spalten mindestens ein Leerzeichen. Die Werte Kaethe und Gerwisch haben hier beispielsweise die maximale Länge, die durch die Leerzeichen im Schema vorgegeben sind. Will man jetzt beispielsweise eine Studentin Brunhilde mit einfügen, so müsste man alle Zeilen um drei Leerzeichen ergänzen. Damit müssten zwischen NAME und ORT? jetzt ebenfalls drei Leerzeichen mehr stehen. Dieser Nachteil ist in der hsq-Struktur behoben. Das Problem könnte man auch im Rahmen der tab-Struktur beheben, indem man die Leerzeichen durch Tabulatoren in den Primärdaten ersetzt. Der Wert eines Tabulators müsste sich dann aus der Anzahl der Zeichen des Spaltennamens und der Anzahl der Leerzeichen ergeben. Das hätte aber den Nachteil, dass man diese Tabellen nicht mehr mit einem gewöhnlichen Texteditor erstellen kann.
Um den Platz auf dem Bildschirm oder Papier besser auszunutzen, werden die Spaltenwerte in bestimmten Situationen horizontal angeordnet. Das betrifft alle letzten Spalten, die vom Typ

SPALTEc sind, wobei c M, B oder L sein kann.

```
NAME, SEX,  KMAl,  MATHEl,  KPHYl,  PHYSIKl l
Ernst m     1 2    1 2 1    1 1     2 1 1
                   3 1
Clara w     1      1 2 1    1 2     3 1 2
```

Tabment 19.1: studenten2.tab mit 4 horizontalen Spalten

Hier hat Ernst die Noten 1 2 1 3 1 in Mathe.

2. Die hsq-Struktur

Gibt man die Studententabelle als hsq-Struktur aus, so hat sie bei „normalen“ Metadaten das folgende Aussehen:

```
STID NAME ORT STIP FAK
 KURS NOTE
  PROJ STUNDEN
1234 Ernst Oehna 500 Mathe
 Algebra 1
 Geschichte 1
 Logik 2
  Fritz 4
  Otto 2
1245 Sophia Berlin 400 Inf
 Algebra 3
 Datenbanken 1
 Otto 1
  Mao 5
  Ming 4
  Otto 6
3456 Clara Oehna 450 Inf
 Datenbanken 1
 OCaml 2
4567 Ulrike _ 400 Kunst
  Monet 10
5678 Kaethe Gerwisch 0 Kunst
 Apel 1
 Repin 1
  Monet 20
```

Tabment 19.2 studenten.tab als hsq ausgegeben

Hier wird jedem Segment eine Zeile gewidmet. Das Segment der ersten Kollektion wird nicht eingerückt. Wenn es jedoch eine nullte Ebene gibt, ist das zugehörige Segment das Einzige das nicht eingerückt ist. Die Prüfungssegmente beginnen mit einem Leerzeichen und die Projektsegmente mit zweien. Die einzelnen Felder der Segmente werden in der Regel durch ein

Leerzeichen getrennt. Nach einem TEXT-Feld müssen jedoch zwei Leerzeichen folgen. Dadruch können Textfelder einzelne Leerzeichen enthalten. In der sechst-letzten Zeile erkennen wir, dass leere optionale Werte durch „_“ gekennzeichnet werden müssen, da ansonsten die 400 dem Wohnort zugewiesen werden würde. Wollen wir Daten als hsq-Struktur erfassen, so können wir uns einfach ein Schema vorgeben, wie in obiger tab-Datei oder wir geben die Metadaten vollständig an (siehe unten). Im ersten Fall werden die einzelnen elementaren Datentypen vom System bestimmt. Um die damit verbundenen Effizienzverluste bei größeren Dateien zu vermeiden gibt man die vollständigen Metadaten an, die durch <META! und !META> geklammert sein müssen.

3. Die ment-Struktur

Betrachten wir jetzt studenten.tab im Dokumentstyle:

```
<META!
TABMENT! STUDENTEN
STUDENTEN! STID,NAME,ORT?,STIP,FAK,(KURS,NOTE m),(PROJ,STUNDEN m)m
NOTE STID STIP STUNDEN! ZAHL
FAK KURS NAME ORT PROJ! TEXT
!META>
<STUDENTEN>
  <STID>1234</STID>
  <NAME>Ernst</NAME>
  <ORT>Oehna</ORT>
  <STIP>500</STIP>
  <FAK>Mathe</FAK>
  <KURS>Algebra</KURS>
  <NOTE>1</NOTE>
  <KURS>Logik</KURS>
  <NOTE>2</NOTE>
  <KURS>Geschichte</KURS>
  <NOTE>1</NOTE>
  <PROJ>Fritz</PROJ>
  <STUNDEN>4</STUNDEN>
  <PROJ>Otto</PROJ>
  <STUNDEN>2</STUNDEN>
  <STID>1245</STID>
  <NAME>Sophia</NAME>
  <ORT>Berlin</ORT>
  <STIP>400</STIP>
  <FAK>CS</FAK>
  <KURS>Algebra</KURS>
  <NOTE>3</NOTE>
  <KURS>Datenbanken</KURS>
  <NOTE>1</NOTE>
  <KURS>Otto</KURS>
  <NOTE>1</NOTE>
  <PROJ>Mao</PROJ>
  <STUNDEN>5</STUNDEN>
  <PROJ>Ming</PROJ>
```

```
  <STUNDEN>4</STUNDEN>
  <PROJ>Otto</PROJ>
  <STUNDEN>6</STUNDEN>
  <STID>3456</STID>
  <NAME>Clara</NAME>
  <ORT>Oehna</ORT>
  <STIP>450</STIP>
  <FAK>CS</FAK>
  <KURS>Datenbanken</KURS>
  <NOTE>1</NOTE>
  <KURS>OCaml</KURS>
  <NOTE>2</NOTE>
  <STID>4567</STID>
  <NAME>Ulrike</NAME>
  <STIP>400</STIP>
  <FAK>Kunst</FAK>
  <PROJ>Monet</PROJ>
  <STUNDEN>10</STUNDEN>
  <STID>5678</STID>
  <NAME>Kaethe</NAME>
  <ORT>Gerwisch</ORT>
  <STIP>0</STIP>
  <FAK>Kunst</FAK>
  <KURS>Repin</KURS>
  <NOTE>1</NOTE>
  <KURS>Apel</KURS>
  <NOTE>1</NOTE>
  <PROJ>Monet</PROJ>
  <STUNDEN>20</STUNDEN>
</STUDENTEN>
```

Tabment 19.3 studenten.ment

Das Dokument wäre etwas übersichtlicher, wenn man Tupel- und Subtupeltags einführt:

```
<META!
TABMENT! STUDENTEN
STUDENTEN! STUDENTm
STUDENT! STID,NAME,ORT?,STIP,FAK,PRUEFUNGl,PROJEKTm
PROJEKT! PROJ,STUNDEN
PRUEFUNG! KURS,NOTE
NOTE STID STIP STUNDEN! ZAHL
FAK KURS NAME ORT PROJ! TEXT
!META>
<STUDENTEN>
  <STUDENT>
    <STID>1234</STID>
    <NAME>Ernst</NAME>
    <ORT>Oehna</ORT>
    <STIP>500</STIP>
    <FAK>Mathe</FAK>
```

```
    <PRUEFUNG>
      <KURS>Algebra</KURS>
      <NOTE>1</NOTE>
    </PRUEFUNG>
    <PRUEFUNG>
      <KURS>Logik</KURS>
      <NOTE>2</NOTE>
    </PRUEFUNG>
    <PRUEFUNG>
      <KURS>Geschichte</KURS>
      <NOTE>1</NOTE>
    </PRUEFUNG>
    <PROJEKT>
      <PROJ>Fritz</PROJ>
      <STUNDEN>4</STUNDEN>
    </PROJEKT>
    <PROJEKT>
      <PROJ>Otto</PROJ>
      <STUNDEN>2</STUNDEN>
    </PROJEKT>
  </STUDENT>
  <STUDENT>
    <STID>1245</STID>
    <NAME>Sophia</NAME>
    <ORT>Berlin</ORT>
    <STIP>400</STIP>
    <FAK>CS</FAK>
    <PRUEFUNG>
      <KURS>Algebra</KURS>
      <NOTE>3</NOTE>
    </PRUEFUNG>
    <PRUEFUNG>
      <KURS>Datenbanken</KURS>
      <NOTE>1</NOTE>
    </PRUEFUNG>
    <PRUEFUNG>
      <KURS>Otto</KURS>
      <NOTE>1</NOTE>
    </PRUEFUNG>
    <PROJEKT>
      <PROJ>Mao</PROJ>
      <STUNDEN>5</STUNDEN>
    </PROJEKT>
    <PROJEKT>
      <PROJ>Ming</PROJ>
      <STUNDEN>4</STUNDEN>
    </PROJEKT>
    <PROJEKT>
      <PROJ>Otto</PROJ>
```

```
      <STUNDEN>6</STUNDEN>
    </PROJEKT>
  </STUDENT>
  <STUDENT>
    <STID>3456</STID>
    <NAME>Clara</NAME>
    <ORT>Oehna</ORT>
    <STIP>450</STIP>
    <FAK>CS</FAK>
    <PRUEFUNG>
      <KURS>Datenbanken</KURS>
      <NOTE>1</NOTE>
    </PRUEFUNG>
    <PRUEFUNG>
      <KURS>OCaml</KURS>
      <NOTE>2</NOTE>
    </PRUEFUNG>
  </STUDENT>
  <STUDENT>
    <STID>4567</STID>
    <NAME>Ulrike</NAME>
    <STIP>400</STIP>
    <FAK>Kunst</FAK>
    <PROJEKT>
      <PROJ>Monet</PROJ>
      <STUNDEN>10</STUNDEN>
    </PROJEKT>
  </STUDENT>
  <STUDENT>
    <STID>5678</STID>
    <NAME>Kaethe</NAME>
    <ORT>Gerwisch</ORT>
    <STIP>0</STIP>
    <FAK>Kunst</FAK>
    <PRUEFUNG>
      <KURS>Repin</KURS>
      <NOTE>1</NOTE>
    </PRUEFUNG>
    <PRUEFUNG>
      <KURS>Apel</KURS>
      <NOTE>1</NOTE>
    </PRUEFUNG>
    <PROJEKT>
      <PROJ>Monet</PROJ>
      <STUNDEN>20</STUNDEN>
    </PROJEKT>
  </STUDENT>
</STUDENTEN>
```

Tabment 19.4 Studenten2.ment

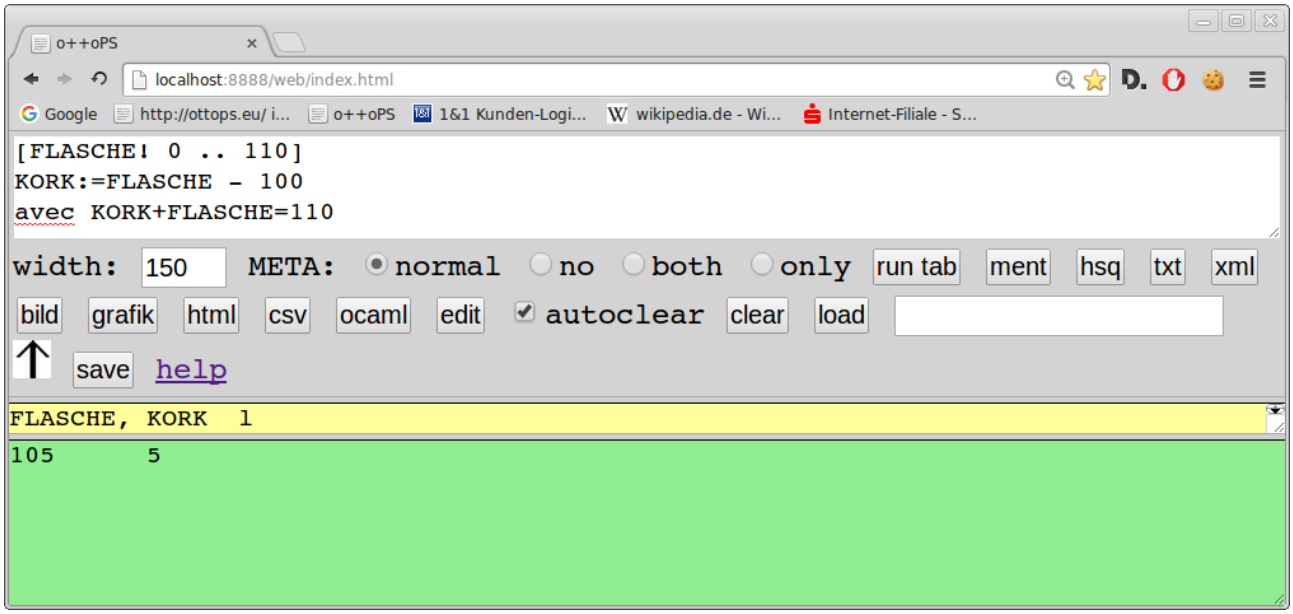
Allerdings benötigt diese Darstellung noch mehr Zeilen. Wenn in einem Tabment zunächst kein Tag für das Gesamtdokument vorkommt, generiert das System einen Tag TABM. Bis auf die Metadatendarstellung stimmen beide Repräsentationen mit entsprechenden XML-Darstellungen überein. Die Primärdaten von ment-Dateien sind also auch Primärdaten von XML-Dateien.

20 Das Interface für Windows und Linux

Vom Standpunkt der Komplexität des Interfaces ist es am einfachsten, ein Programm bzw. eine Anfrage einfach einzutippen. Bei Anfragen an eine Tabelle oder ein Dokument müssen die Tabellen-, Dokument- bzw. Spaltennamen genauso geschrieben werden, wie sie in der TTD vorgegeben sind. Hier können sich leicht Fehler einschleichen. Daher kann man einen Dateinamen einfach anklicken. Anschließend kann man sich die Metadaten ausgeben lassen und kann die entsprechenden Namen dahin kopieren, wo es gewünscht ist. Die Funktionsnamen findet man in der Schlüsselwortedatei.

Die o++o Weboberfläche besteht aus vier Bereichen: oben dem Programmfeld, unten dem Ausgabefeld (grün) und dazwischen dem Kommandofeld und dem Metadatenfeld (gelb).

Screenshot:



Programmfeld

Das Programmfeld ist eine Editorkomponente. Hier werden o++o-Programme eingegeben und modifiziert.

Kommandofeld

Ausführen

Die Buttons "tab", “ment“, "xml", "bild", “grafik“, "html", “hsq“ und "ocaml" führen das Programm aus dem Eingabefeld aus.

- Der Button "tab" formatiert die Ausgabe als Tabelle. Zusätzlich kann hier vorher im Eingabefeld "width" die Tabellenbreite eingestellt werden. Wird die Checkbox "meta" aktiviert, so wird die TTD ausgegeben. Bei den anderen vier Ausgabearten werden dann die Metadaten vor den eigentlichen Daten ausgegeben.
- "bild" setzt voraus, dass das Anfrageergebnis sich als Punktmenge interpretieren lässt. Diese wird dann ausgegeben.
- Die Nutzung “grafik“ kann durch Styleinformationen untersetzt werden. “bild“ ist ein

Spezialfall von “grafik“. Hierbei müssen lediglich die Zeilen

```
tag COLL
FORMAT:= ##
<TAB!
SELECT,     TYPE,  STYLEl m
COLL        svg
!TAB>
```

hinzugefügt werden.

- "html" zeigt den Quellcode des HTML-Dokuments (Tabelle) an.
- "xml" erzeugt ein XML-Dokument. Die Checkbox "meta" gibt hier an, ob die XML-DTD eingefügt werden soll oder nicht.
- “hsq“ gibt die Ergebnismenge als hierarchisch sequentielle Datei aus. Zu welcher Ebene der Tabelle ein Segment gehört, ergibt sich aus der Anzahl der führenden Leerzeichen der Zeile. Die einzelnen Felder werden durch 2 Leerzeichen voneinander getrennt.
- "ocaml" zeigt die Ausgabe als OCaml-Term an. Dies ist in erster Linie für Entwickler interessant.

Die Checkbox "autoclear" bewirkt bei den Ausgabeformen "tab", "xml" und "ocaml", dass die alten Ergebnisse gelöscht werden, bevor die neue Ausgabe angezeigt wird. Bei der Image- und HTML-Ausgabe wird diese Angabe ignoriert, da sowieso immer das alte Dokument durch das neue ersetzt wird.

Laden und Speichern

Die Buttons "load"/ "save", der Pfeil und das Eingabefeld dazwischen sind für das Laden und Speichern verantwortlich. "load" ermöglicht es, o++o-Programme und Dateinamen einzuladen. Mit dem "save" Button kann man entweder o++o-Programme oder die Ausgabe abspeichern. Der Pfeil entscheidet dabei, ob Programm oder Ausgabe abgespeichert wird. Durch Anklicken des Pfeils kann man dessen Richtung ändern. Den Dateinamen muss man vor dem Abspeichern im Eingabefeld daneben eingeben.

Ausgabefeld

Hier werden die Ergebnisse von o++o-Programmen angezeigt. Bewegt man den Mauszeiger in die rechte obere Ecke wird der "show" Button angezeigt. Mit diesem kann man die Ausgabe in einem neuen Fenster öffnen.

Ein Vorschlag für das Vorgehen des o++o-Nutzers

1. Falls möglich, ein o++o-Programm laden, das annähernd das Gewünschte leistet.
2. Das o++o-Programm modifizieren falls nötig (weitere oder andere Programmzeilen, andere Tabellennamen, andere Werte, andere Spaltennamen oder andere Operationen).
3. Falls kein ähnliches o++o-Programm gefunden wurde: Tabellenname laden ("tab", "xml", hsq oder "csv").
4. "meta" und "tab" anklicken und Spaltennamen aus der TTD an gewünschte Stelle kopieren
5. Alle Worte der Datei oder einer Spalte extrahieren und passende Worte oder Elemente an die gewünschte Stelle kopieren. Die Worte einer Spalte XX lassen sich durch folgendes Programm ermitteln:

```
aus datei
gib XXm
```

1. In einem zweiten Window oder einem zweiten Tab des Browsers o++o noch einmal starten und geeignete Schritte 1 -5 realisieren und Programmteile, Tabellennamen, Spaltennamen oder Werte in das erste Arbeitsfeld kopieren bis das Programm gewünschte Form besitzt.

Anhang A: Schlüsselworte von o++o

SCHLÜSSELWORT	KATEGORIE	BESCHREIBUNG
+	binäre algebraische Operation	**Addition**: 3+6=9 (2,3,Paul)+4=(6,7,Paul)
*	binäre algebraische Operation	**Multiplikation**: (2,3,Paul)*4=(8,12,Paul); tab*z1: jede Zahl von tab wird mit z1 multipliziert; der Rest bleibt unverändert tab*si=tab*1;tab*no=tab*0; tab1*tab2 sonst: beide Tabmente müssen den gleichen Typ haben
-	binäre algebraische Operation	**Subtraktion**: 7-3 = 4: Differenz
:	binäre algebraische Operation	**Division**: 6 : 4 = 1.5
++	unäre algebraische Operation	**Summe**: [1 2 3] ++ ergibt 6 1, Wladimir, 8 ++ ergibt 9
**	unäre algebraische Operation	**Produkt**: [2 3 7]** = 42
--	unäre algebraische Operation	**Mehrfachsubtraktion**: [55 3 4] -- = 48
::	unäre algebraische Operation Aggregation	**Mehrfachdivisionen**: [64 2 2 2] :: = 8
++:	unäre algebraische Operation Aggregation	**Durchschnitt**: 1 2 2 1 ++: = 1.5

++1	unäre algebraische Operation Aggregation	**Anzahl** (Statistik): `[5 6] ++1 =2`; Anzahlen der (komplexen) Zeilen jeder Komponente;
		`[2 3 5 7 8 9], {rr tt zz uu},99 ++1`
		`= (6,4,1)`

+m, +b, +l	binäre Kollektionsoperation	mengentheoretische Vereinigung, es können aber auch Elemente zur Kollektion hinzugefügt werden, …., das Kollektionssymbol gibt den Zieltyp an
-m, -b, -l	binäre	**Mengendifferenz**: Differenz von flachen Kollektionen

	Kollektionsoperation	<TAB! X,Y l 1 2 3 4 !TAB> -l ## <TAB! X,Y l 3 4 1 5 !TAB> = X, Y l 1 2
:m, :b, :l	binäre Kollektionsoperation	mengentheoretischer Durchschnitt,….
*m, *b, *l	binäre Kollektionsoperation	kartesisches Produkt
,	binäre Tabmentoperation	**Paarbildung:** 1,6,(otto,9) = 1,6,otto,9 (4-Tupel) 1,otto =(1,otto) ist vom (Typ ZAHL,WORT) 1,2 ungleich 1.2
;	Basissymbol	**Semikolon:** kann vor einem binären Operator stehen; das zweite Argument reicht dann bis zum nächsten Semikolon bzw. bis zum Zeilenende: 1+2 ;* 3+4 ;*2+3 = 105
„ “ bzw. „ “	Basiszeichen	[1 2 4] = Liste von 3 Zahlen trennt auch Werte und Operationen; Eine Einrückung (vier Leerzeichen) verbindet neue Zeile mit der vorangehenden zur logischen Einheit
=	Relationssymbol	ORT = Hadmersleben selektiert beispielsweise alle Personen, die in Hadmersleben wohnen
:=	Basissymbol	**Zuweisung**: BRUTTO:=NETTO + TARA neuname := ausdruck (neuname wird mit dem Resultat der Berechnung von ausdruck belegt); Aggregationszuweisungen kommen auch in der gib-Klausel vor
::=	Basissymbol	Einer vorhandenen Spalte werden neue Werte zugewiesen. n ::= e **ersetze** den Inhalt der „Spalte“ n durch die Werte des Ausdrucks e
=:	Basissymbol	**Zuweisung**: 1,2 =: $XX

&	binäre Boolesche Operation	**Konjunktion (und):** si & no = no
|	binäre Boolesche Operation	**Disjunktion** (oder): si | no = si
->	binäre Boolesche Operation	**logische Implikation**: si `-> no = no`
<->	binäre Boolesche Operation	**logische Äquivalenz: no** `<-> no = si`
>, <, <=, >=	Relationssymbol	`3>4` ist falsch (no)
“	Anführungszeichen	wird benutzt um Texte mit Leerzeichen zu einer Einheit zu verschmelzen Otto =“Otto“, “Es regnet heute“ ist ein Text; (Es regnet heute) ist eine Liste von Wörtern.

+text	binäre Textoperation	**Konkatenation (Verkettung)**:
		`Hallo +text ” “ +text Otto =`
		`“Hallo Otto”`
		Der Typ der ersten Tabelle bleibt unverändert.

{}	Klammern für Mengen von Werten	{1 1}={1}
{{ }}	Klammern für Multimengen von Werten	{{1 1}} ungleich {{1}}
[ ]	Klammern für Listen von Werten	[1 3 4]
#	Basissymbol	**Kommentar**: der folgende Text der Zeile ist Kommentar
$	Basissymbol	bezeichnet eine (Tabment) **variable**: $X:=(8,9)
(#, #)	Begrenzer	**Kommentar: Beginn/Ende** von mehrzeiligem Kommentar
|	Basissymbol	**Alternative**: X|Y l; **Strich** (bar): [| | |]

+tup	Mengenoperation	`<TAB!`
		`X,Y l`
		`1 2`
		`3 4`
		`!TAB>`
		`+tup (4,5)`
		`=`
		`X,Y l`
		`5 7`
		`7 9`
		zu jedem **Tupel** des Tabments wird ein zweites Tupel addiert

*tup	Mengenoperation	**Tupelmultiplikation:** (2,3) *tup (4,5)= (8,15)
-tup	Mengenoperation	**Tupeldifferenz**:

:tup	Mengenoperation	**Tupeldivision**:

*mat	Matrizenoperation	**Matrixmultiplikation**: (1,2) *mat [2 3]=8
		`<TAB!`
		`X1,X2,X3 l`
		`1 0 2`
		`0 2 0`
		`0 0 8`
		`!TAB>`
		`*mat ##`
		`<TAB!`
		`X1, X2, X3 l`
		`1. -0. -0.25`
		`-0. 0.5 -0.`
		`0. -0. 0.125`
		`!TAB>`
		`=`
		`X1, X2, X3 l`
		`1. 0. 0.`
		`0. 1. 0.`
		`0. 0. 1.`

-1mat	Matrizenoperation	`<TAB!`
		`X1,X2,X3 l`
		`1 0 2`
		`0 2 0`
		`0 0 8`
		`!TAB>`
		`-1mat`
		`=`
		`X1, X2, X3 l`
		`1. -0. -0.25`
		`-0. 0.5 -0.`
		`0. -0. 0.125`
		inverse Matrix

<, >	Begrenzer	**Beginn/Ende von Tags:** `<HSQ: ; :TAB>`

/	Basissymbol	`AUTOR/NAME:` in der XML-Repräsentation ist der **Tag** NAME is **direkt in** AUTOR **enthalten**; das wird z.B. benötigt, wenn die folgenden zwei Typen gegeben sind:
		`AUTOR! NAME, VORNAME`
		`EDITOR! NAME, VORNAME, FIRMA`

//	Basissymbol	`AUTOR//VOR`: zwischen AUTOR und VOR sind mehrere Tags erlaubt;
		`AUTOR! NAME, FIRMA`
		`NAME! VOR, MITTEL?, NACH`
		aber `AUTOR/NAME/VOR` ist effizienter

!	Basissymbol	**Begrenzer**: `[X! 77 88]`

		wird bei einzeiligen Tabmenten und dreistelligen Operationen benutzt
&&	unäre Aggregation	Statistik: **für alle** Aggregation: `si,66,si && = si`

||	Aggregation	Statistik: **Existenzaggregation:**
		`WEIBLICHl ||`
		`1=2,7=8`
		`||`
		`= no`

..	binäre Operation mit Listenoutput	`1 .. 5 **`
		`= 120`
		(entspricht hier der Fakultätsfunktion)

...	Operation mit Liste als Ergebnis	`1 ... 4!2` = 1 3 x ... y ! z: alle Zahlen x, x+z, x+2z,..., x+n*z: x+n*z<=y

..x	Operation mit Liste als Ergebnis; Zufallszahlen-generator	`1 ..x 6 !5`
		=
		`1 4 2 6 1`

1in	Relation	`[1 2] 1in “1 3 4“ = si` ein Wort bzw. eine Zahl der linken Seite ist in der rechten enthalten.
abs	unäre Zahlenoperation	der absolute Betrag einer (P)Zahl: -7 abs = 7 6 abs = 6

add	Mengenoperation	t1 add t2: **füge** t2 in t1 bzgl. gleicher Spaltennamen **ein**; der Typ von t1 ist der Ergebnistyp; ähnlich zur gib-Klausel
		`<TAB!`
		`X,Yl m`
		`1 2 3`
		`4 5`
		`!TAB>`
		`add &&`
		`<TAB!`
		`Y,X`
		`6 1`
		`!TAB>`
		`=`
		`X, Yl m`
		`1  2 3 6`
		`4  5`

at	Schlüsselwort	**Erweitere an einer Position** `n:=e at n2`: erweitere rechts neben n2 neben einfachen Namen sind auch NAMEm, NAMEb, NAMEl und NAMEa zugelassen. Dann wird eine Ebene

		höher erweitert.
atom	gib-Klausel	atomares Subtabment `atom! HOBBYl` HOBBYl wird während der Umstrukturierung als Gesamtheit transferiert
aus	Schlüsselwort	`aus studenten.tab`

avec	Schlüsselwort	**Selektion**
		`avec GEHALT > 5000`
		nur die spezifizierten (komplexen) Zeilen verbleiben im Ergebnis
		`avec ABTEILUNG!  GEHALT >5000`
		alle Abteilungen, in denen ein Angestellter mehr als 5000 verdient.
		`avec 5000`
		nur Tupel, die 5000 enthalten, verbleiben

BAR	Datentyp	enthält nur ein Element (!); daher erst BARl sinnvoll
begin, end	Begrenzer	**Beginn/Ende** eines Unterprogramms
BOOL	Datentyp	enthält 2 Wahrheitswerte (si,no); zu Ehren des englischen Mathematikers George Boole

comp	binäre Operation	`NAME,   VORNAME,ORT`
		`Miller  Paul    Magdeburg`
		`comp ORT`
		ergibt `MD;` siehe auch `nth`

cos	trigonometrische Operation	**Kosinus**: 0 `cos=1.`
csv	Suffix	**csv-Datei** kann als Input oder Output von o++oPS Programmen dienen

det	Matrizenoperation	`<TAB!`
		`X1,X2,X3 l`
		`1  0  2`
		`0  2  0`
		`0  0  8`
		`!TAB>`
		`det`
		`= 16.`
		Determinante

div	binäre algebraische Operation	**ganzzahlige Division** `7 div 3 = 2`
divrest	binäre algebraische Operation	**ganzzahlige Division mit Rest**: `7 divrest 3 = 2,1`
gib	Basisoperation	**restrukturiere, sortiere, aggregiere, vereinige, eliminiere Duplikate,...**:

		`aus studenten.tab`
		`gib FAK,(ORT,NAMEm m)m`
		transformiere ein Tabment in ein Tabment mit gegebenen Schema bzw. gegebener TTD

giball	Basisoperation	`giball X | Y l` **Liste aller** X- und Y-**Elemente** (beliebige Tiefe); entspricht dem Doppelslash ...//X|Y von XPath
gibtop	Basisoperation	`gibtop Xl` entspricht dem Slash: t/X: Liste aller X-Subtabmente von t, die in der obersten Ebene von t vorkommen.
hoch	binäre algebraische Operation	2 hoch 3 = 8

horizontal	Umstrukturierungsoperation	`<TAB!`
		`NAME, (FACH,NOTE m)m`
		`Paul  Ma  1`
		`  Deu  2`
		`Sophia Ma  2`
		`  Bio  1`
		`  Deu  1`
		`!TAB>`
		`horizontal FACH`
		=
		`NAME,  NOTEBIO?, NOTEDEU, NOTEMA l`
		`Paul  2  1`
		`Sophia 1  1  2`

hsq	Suffix	In- und Outputdatei; jeder Zeile entspricht ein Segment; die Felder eines Segments werden mit einem oder mehr Leerzeichen voneinander getrennt
it then else	Schlüsselwort	`if 1=1 then 2 else 3 = 2` `if 1<1 then 2 else 3 = 3`
if then	Schlüsselwort	`XX := if ORT=Halle then gut` XX erhält nur einen Eintrag, wenn die Person aus Halle ist, d.h. es wird um XX? erweitert.
inm	Relationssymbol	sind linke und rechte Seite Kollektionen gleichen Typs (bis auf Tags, so stellt inm die „mengentheoretische“ Inklusion dar, [1 3] inm [1 4 3] =si ist die linke Seite vom Elementtyp der rechten, so ist inm die Elementrelation 2 inm { 6 7 2} =si ansonsten ist „in“ nicht definiert (siehe in)
in	Relationssymbol	X in Y: linke und rechte Seiten werden in Mengen von Wörtern und Zahlen transformiert und es wird getestet, ob die linke Menge Teilmenge der rechten ist. “1 2 1“ in “1 2“ = si

		“1 2 3“ in “1 1 2“ = no
leftat	Schlüsselwort	`n:=e leftat n2`: siehe oben **at**

like	Boolesche Operation	`Hadmersleben like "?admers*"`
		`= si`
		`'?'`: repräsentiert ein Zeichen
		`'*'`: null oder mehrere Zeichen

linreg	Aggregation	`<TAB!`
		`FLASCHENPREIS, VERKAUFTEMENGE l`
		`20 0`
		`16 3`
		`15 7`
		`16 4`
		`13 6`
		`10 10`
		`!TAB>`
		`linreg`
		`=`
		`Y0, ANSTIEG`
		`19.7321428571 -0.982142857143`

lists	binäre Basisoperation	**Liste von Listen der Elemente:**
		`[X! 1 2] lists 2`
		`=`
		`Xl l`
		`1 1`
		`1 2`
		`2 1`
		`2 2`

ln	unäre algebraische Operation	**natürlicher Logarithmus**; e ln =1.
m, m-, b, b-, l, l-, a, s, ?	Kollektionssymbole	m: Menge, m-: Menge umgekehrt sortieren, b (bag) Multimenge, l: Liste, a: Kollektion vom ANY-Typ, s: Strom (Stream noch nicht angefangen); ?: optionaler Wert; die Kollektionssymbole werden postfix notiert und können ohne Leerzeichen an ein Tag angehängt werden.
mal	Tabmentoperation	5 mal "Ich liebe Dich!" = Ich liebe Dich! Ich liebe Dich! Ich liebe Dich! Ich liebe Dich! Ich liebe Dich!

max	Aggregation	Statistik: **Maximum**
		`1.1,2,Hallo`
		`max = 2.`

median	Aggregation	`(2 6 3 2),7,8`
		`median`
		ergibt
		`4.5`

ment	Suffix	Dokumentdarstellung eine Tabments; unterscheidet sich von XML durch vereinfachte Angabe der Metadaten

min	Aggregation	Statistik: **Minimum**
		`1.1,2,Hallo`
		`min`
		ergibt `1.1`

no	Booloperation	**Boolesche Konstante:** Wahrheitswert falsch; entspricht Antwort no (spanisch)

not	unäre Boolesche Operation	`si not = no`
		Negation
		`X not not = X`

nth	binäre Operation	n-te Komponente bzw. n-tes Element `emperors.tab nth 2 comp NAME`

nurtext	unäre Textoperation	`1,a1,Butter`
		`nurtext = a1 Butter`
		Verkettung aller TEXT-und WORT-Werte eines Tabments (Operation kann auch durch andere Operationsfolgen ausgedrückt werden)

polygon	unäre algebraische Operation	`[X,Y! 0 0  1 1  0 1]` zeichnet die 2 Strecken (0,0) bis (1,1) und (1,1) bis (0,1)
polynom	binäre algebraische Operation	`[3 1 4] polynom 2` berechnet den Funktionswert von X*X*3+X+4 an der Stelle 2
pos	unäre Positionsoperation	`NOTE pos < 5` die ersten vier Noten
pos-	unäre Positionsoperation	`NOTE pos- < 5` die letzten 4 Noten
pred	unäre Positionsoperation	**Vorgänger** NAME pred: NAME-Wert des Vorgängers
pred_n	binäre Positionsoperation	`NOTE pred_n 3` dritter **Vorgänger** innerhalb einer Kollektion
PZAHL	Datentyp	**Zahl mit Punkt** (früher Kommazahl=Float): 2.34
pzahl	Konvertierungs-operation	**konvertiere** eine Zahl oder einen Text in eine **PZAHL**
pzahll	unäre Tabmentoperationen	alle Pzahlen eines Tabments werden ausgegeben (keine Typkonvertierungen)
rename to	Basisoperation	`rename X to Y`: **ersetze** jeden Spaltennamen X durch den Namen Y
recpred	unäre Positionsoperation	**Vorgänger** innerhalb von :=: AMOUNT recpred AMOUT-Wert des Vorgängers
rest	binäre	Rest der ganzzahligen Division:

	algebraische Operation	`7 rest 3 = 1`

rnd	binäre algebraische Operation	`[2.1436 5.88] rnd 1`
		=
		`2.1 5.9`
		`z rnd n`: **runde** z auf n Ziffern nach dem Punkt

sans	Schlüsselwort	**Selektion**
		`sans ORT=Magdeburg`
		`sans Magdeburg`
		sans: ohne die spezifizierten (komplexen) Tupel

satzl	Textoperation	`LEBENSLAUF satzl` **Liste aller Sätze**; Das Resultat ist vom Typ: SATZl
seg	unäre Operation	`X seg ++`: Durchschnitt aller Zahlen des Segments, das X enthält
si	Boolesche Konstante	Wahrheitswert **wahr (entspricht der Antwort ja)**
sin	unäre trigonometrische Operation	`3.14159 sin` =2.65358979335e-06 **Sinusfunktion**
sqrt	unäre algebraische Operation	**Quadratwurzel** `4 sqrt` =2.
streuung	unäre Aggregation (mad)	[1 2 5 3 5 1] streuung = 1.5

strip	unäre Basisoperation	`<TAB!`
		`X,Y?,Zl,Wm m`
		`1  2  3`
		`  4`
		`!TAB>`
		`strip`
		=
		`(X, Y?, Z?,  Wm)?`
		`1  2  3`
		`  4`
		Alle Kollektionssymbole, zu denen jede Kollektion höchstens 1 Element enthält, werden durch ? ersetzt.

subtext	dreistellige Textoperation	`aBCdE subtext 2 ! 3` = BCd text subtext beg ! len: beg beginnt bei 1 an zu zählen, len ist die Länge des Ergebnistextes; der zweite und dritte Inputwert muss eine ganze Zahl sein
subtext2	dreistellige Textoperation	`aBCdEfgh subtext2 "B" ! fg` =CdE text3 subtext2 text1 ! text2: Text von text3 zwischen text1 und text2

succ	unäre Positionsoperation	`NOTE succ`: **Nachfolger** innerhalb einer Kollektion; das gesamte (Sub)Tupel ist der Nachfolger
succ_n	binäre Positionsoperation	`NOTE succ_n 3`: dritter **Nachfolger** innerhalb einer Kollektion
tab	Suffix	ein **Tabment in tabellarischer Sicht**
TABMENT	Tag	**virtueller Tag** um das gegebene Tabment
tag	unäre Tabmentoperation mit Parameter	`1 tag X` t1 tag ROOT1: um t1 wird ein ROOT1-Tag gesetzt

tags	unäre Tabmentoperation mit Parameter	`0 .. 3 tags X`
		`=`
		`Xl`
		`0 1 2 3`

TEXT	Datentyp	**text Datentyp** (string)

++text	unäre Textoperation	`13.2,[ab cc],Bc`
		`++text`
		`=13.2 ab cc Bc`
		alle Werte werden in **Text** umgewandelt und verbunden

textend	binäre Textoperation	`asdfgh textend 4`
		`=fgh`
		Rest des Textes ab der spezifizierten Position

textend-	binäre Textoperation	`asdfgh textend- 4`
		`=dfgh`
		Rest des Textes ab der spezifizierten Position von hinten gezählt

++text2	binäre Textoperation der zweite Inputwert ist der Separator	`0 .. 10 ++text2 ","`
		`=`
		`0,1,2,3,4,5,6,7,8,9,10`

textindex	binäre Textoperation	`“Heute wird schoenes Wetter“`
		`textindex wir`
		`=7`

textcut	binäre Textoperation	`abcdecefcg textcut c`
		`= ab de ef`
		Der Text wird in Liste von Texten bzgl. des Separators geteilt.

time	algebraische Operation ohne Inputwert	`X:= time`
		`=1449251939.91`
		Systemzeit; muss in der Regel zweimal angewandt werden, um die Differenz zu bilden

tup	unäre Operation	`X tup ++` Summe aller Zahlen aller X-Tupel
untag	unäre Basisoperation	`1 tag XX untag`

		=1 untag(Tag(n,t'))=t'; streiche den äußersten Begin- (und End) Tag
upper	unäre Textoperation	1.2,aW upper = 1.2 AW (hsq Ausgabe) jeder Kleinbuchstabe wird in einen Großbuchstaben umgewandelt; der Rest verbleibt unverändert.
variance	Aggregation	[1 2 4 6] variance = 4.91666666667
vertical	Basisoperation	<TAB! NAME, BIO?,DEUTSCH?, MATHE? l Paul 2 1 Sophia 1 1 2 !TAB> vertical ## FACH,NOTE l:=BIO,DEUTSCH,MATHE = NAME, (FACH, NOTE m)m Paul DEUTSCH 2 MATHE 1 Sophia BIO 1 DEUTSCH 1 MATHE 2 vertical X,Y m :=C,D,E: die Spaltennamen C,D,E erscheinen in der Spalte X und die entsprechenden Werte in Spalte Y
vlists	unäre Basisoperation	variabel lange Listen; die Operation stimmt mit lists überein, nur dass alle kürzeren Listen noch im Ergebnis eingeschlossen sind.
weg	Basisoperation	weg XX Y: vergiss die Spalten (Tags) XX und Y
wege	Basisoperation	eine gegebene Tabelle tab: SUP,XX,…,(SUB,YY,…l)m wird als gerichteter, gewichteter, zyklenfreier Graph mit Kanten von SUP nach SUB interpretiert. tab wege sup0 ist die Liste aller Wege von sup0 bis zum „Endknoten“. Sie ist vom ((SUB,YY,…)l)l. <TAB! SUP, (SUB,ANZ l)m t0 t1 t2 t3 t1 5 t4 6

		t4 t2 3 t0 2 !TAB> wege t3 = (SUB, ANZ l)l t4 6 t0 2 t4 6 t2 3 t4 6 t1 5 (Die Leerzeilen wurden eingefügt.)
wortl, wortb, wortm	unäre Textoperation	"We are 6." wortm ={6 are we} alle **Worte** eines Tabments
wortsep	unäre Textoperation	„We are 6.“ wortesep =[„We“ „ “ „are“ „ “ „6“ „.“]
xml	Suffix	studenten.xml: **XML-file**
ZAHL	Datentyp	beliebig große ganze Zahlen (bigInt)
zahl	unäre Konvertierungs-operation	**konvertiere** TEXT oder PZAHL **in ZAHL** "12" zahl =12
zahl1	unäre Konvertierungsoperation	"24:5:33" zahl1 =24 **erste Zahl** im Text; der Text muss mit einer Ziffer beginnen
zahl2	unäre Konvertierungsoperation	"24.05" zahl2 =5 **zweite Zahl** im Text
zahl3	unäre Konvertierungsoperation	"24:AA:5::087" zahl3 =87 **dritte Zahl** im Text
zahll	unäre Tabmentoperationen	alle Zahlen eines Tabments werden ausgegeben (keine Typkonvertierungen)

Anhang B: Syntax

Wir beschränken uns zunächst auf die Aufzählung einiger Besonderheiten der Syntax.

- Eine Zeile, die mit 4 Leerzeichen beginnt, bildet mit der vorangehenden eine logische Einheit.
- In o++o-Programmen können bisher nur tab- und hsq-Strukturen vorkommen. Die anderen Strukturen (ment, csv, xml) müssen als Datei gespeichert werden und können dann über den Dateinamen angesprochen werden.
- Kopiert man ein o++o-Programm in das Programmfeld, so werden die Anführungszeichen häufig nicht richtig erkannt; es kommt eine Syntaxfehlernachricht; man muss die Anführungszeichen gegebenenfalls erneut eintippen.
- Zur Zeit können Umlaute nicht richtig verarbeitet werden. UTF8 ist in Vorbereitung.

main	CommandList EOF
	EOF
CommandList	CommandListCore
	Trenner CommandList
CommandListCore	Command
	CommandListCore Trenner Command2
	CommandListCore Trenner
Command	Command2
	Expr
Command2	AUS Expr
	WAS DOLLAR NAME
	WAS DOLLAR2 NAME
	DOLLAR NAME IS Expr
	DOLLAR2 NAME IS Expr
	NAME IS Expr Where2
	NAME IS Expr Where2
	Oper1
	NOT
	Oper2_all Expr
	Oper3 Expr AUSRUFE Expr
	VERTICAL NAME COMMA NAME COLL IS NAME DDOT NAME
	VERTICAL NAME COMMA NAME COLL IS NAME DDOT
	Stroke
	Strokeplus
	IS Expr Where2
	NAME IS FIRST Expr NEXT Expr Where2
	Names COLL IS WHILE Expr FIRST Expr NEXT Expr Where2
	Sel4 Values
	Sel4 Values Log_agg
	Sel4 PathNames AUSRUFE Values

	Sel4 PathNames AUSRUFE Values Log_agg
	Sel3 Expr
	Sel3 PathNames AUSRUFE Expr
	MIT_LIKE Expr
	PathName IMPLIR Expr
	RENAME PathName TO NAME
	GIBALL Scheme
	GIBTOP Scheme
	WEG PathNames
	NUR PathNames
	TAG NAME
	TAGS NAME
	COMP NAME
	Agg Expr
Where2	epsilon
	AT FpathNames
	LEFTAT FpathNames
	AT
Sel3	AVEC
	SANS
Log_agg	EX
	ALL
Stroke	GIB Scheme
	Stroke NAME AUSRUFE Scheme
	Stroke ATOM AUSRUFE Scheme
	Stroke NAME IS Expr AUSRUFE Agg
Scheme	Scheme COLL
	ANYCOLL
	LPAREN Scheme RPAREN
	Scheme COMMA Scheme
	Scheme OR Scheme
	Name2
	NAMEC
	NAMEC MINUS
Agg	SUM
	MAX
	MIN
	PROD

	EX
	ALL
	COUNT
	AVG
	VARIANZ
	STREUUNG
	MEDIAN
	LINREG
Names	$\text{NAME}^{+}_{\text{COMMA}}$
PathName	PATHNAME
	NAME
PathNames	PathName
FpathName	PATHNAME
	NAME
	NAMEC
FpathNames	FpathName
Name2	NAME
	ZAHL
	PZAHL
	TEXT
	WORT
	BOOL
	BAR
Trenner	EOL
	DSEMI (;;)
Expr	NAME
	PATHNAME
	NAME Strich
	NAME PREDTUP
	NAME SUCCTUP
	NAME SUCC_N Expr
	NAME PRED_N Expr
	NAME PRED
	NAME SUCC
	Expr TAG NAME
	Expr TAGS NAME
	Expr COMP NAME
	DOLLAR NAME
	DOLLAR2 NAME
	TIME
	LPAREN Expr RPAREN

LT GT
LBRACK RBRACK
LCURL RCURL
LCURL2 RCURL2
Values
Value
LBRACK Values RBRACK
LCURL Values RCURL
LCURL2 Values RCURL2
LCURL NAME AUSRUFE Values RCURL
LCURL2 NAME AUSRUFE Values RCURL2
LBRACK NAME AUSRUFE Expr RBRACK
LBRACK Names AUSRUFE Values RBRACK
LCURL Names AUSRUFE Values RCURL
LCURL2 Names AUSRUFE Values RCURL2
IF Expr THEN Expr ELSE Expr
Expr Oper3 Expr AUSRUFE Expr
Expr MAL OR
Expr Oper2_or Expr
Expr Oper2_lt Expr
Expr SEMI Expr
Expr Oper2 Expr
IF Expr THEN Expr
Expr Oper1
Expr GIBALL Scheme
Expr GIBTOP Scheme
NAME POS
PATHNAME POS
NAME RPOS
PATHNAME RPOS
NAME POS2
PATHNAME POS2
NAME RPOS2
PATHNAME RPOS2
NAME TUP
PATHNAME TUP
NAME Strich TUP
PATHNAME Strich TUP
NAME SEG
PATHNAME SEG
NAME ALLSEGS

	PATHNAME ALLSEGS
	NAMEC
	NAMEC MINUS
	BEGIN CommandList END
	TABLE
	XMLTABLE
	MENTTABLE
	HSQTABLE
	TXTTABLE
	XML
	TAB
	CSV
	HSQ
	TXT
	MENT
Strich	STRICH
Value	MINUS INTEGER
	MINUS FLOAT
	INTEGER
	FLOAT
	PI
	RETURN
	STRING
	STRING2
	STRING3
	SI
	NO
Values	Value
Oper1	SUM (++)
	ALL (&&)
	EX (||)
	MAX
	MIN
	COUNT (++1)
	PROD (**)
	AVG (++:)
	DIVDIV (::)
	MINUSMINUS (--)
	VARIANZ
	STREUUNG
	MEDIAN

ZAHLEN
PZAHLEN
LINREG
TEXT
UPPER
LOWER
STRIP
ZAHL
SATZL
SATZB
SATZM
LETTERS
WORTM
WORTSEP
WORTB
WORTL
PZAHL
INVERSMAT
DET
PERMUTATIONS
CROSSTAB
ZAHL1
PZAHL1
ZAHL2
ZAHL3
ABS
UNTAG
UNTAGCOLL
SQRT
SIN
COS
EHOCH
LN
LI
ME
FILLPOLYGON
POLYGON
NATSEL
NATJOIN

Oper2_all Oper2
Oper2_lt

	Oper2_or
Oper2	Oper2_union
	Oper2_plus
	Oper2_minus
	Oper2_plusmat
	Oper2_multmat
	Oper2_textsep
	Oper2_mal
	NTH
	Oper2_comma
Oper2_mal	DDOT (..)
	MAL
Oper2_or	OR (|)
	AND (&)
	IMPLI (->)
	EQUI (<->)
Oper2_union	UNIONM (+m)
	UNIONB (+b)
	UNIONL (+l)
	EXCEPTM (-m)
	INTERSECTM (:m)
	CARTM (*m)
	EXCEPTB (-b)
	INTERSECTB (:b)
	CARTB (*b)
	EXCEPTL (-l)
	INTERSECTL (:l)
	CARTL (*l)
	MITKEYS
	ADD
	ADD1
	ROUND (rnd)
	LISTS
	VLISTS
	PATHS (wege)
	PATHSCYC (wegecyc)
	ROTATE
	POLYNOM
	TEXTEND
	TEXTEND2
	TEXTCUT

	TEXTINDEX
Oper2_comma	COMMA
Oper3	SUBTEXT
	SUBTEXT2
	Oper3_zufall
	INSIDE
Oper3_zufall	DDDOT (...)
	ZUFALL(..x)
Oper2_lt	LT (<)
	GT (>)
	LE (<=)
	GE (=)
	NE (!=)
	EQ (=)
	IN
	INM
	ONEIN (1in)
	LIKE
Oper2_plus	PLUS (+)
	MULT (*)
	DIV (:)
	DIVINT (div)
	MOD (rest)
	DIVREST
	HOCH
Oper2_minus	MINUS (-)
Oper2_plusmat	PLUSTUP (+tup)
	MINUSTUP (-tup)
Oper2_multmat	DIVTUP (:tup)
	MULTTUP (*tup)
	MULTMAT (*mat)
Oper2_textsep	PLUSTEXT (+text)
	TEXTSEP (text2)

Literatur

[AB84] S. Abiteboul, N. Bidot, „Non First Normal Form Relations: An Algebra Allowing Data Restructuring“ Rapports de Recherche No347, Institute de Recherce en Informatique et en Automatique, Rocquencourt, France, Nov. 1984

[AC75] M. M. Astrahan, D. D. Chamberlain, „Implementation of a Structured English Query Language“, Communications of the ACM 18 10, Oct. 1975 pages 580-587

[Ben80] K. Benecke, „Signaturketten und Operations- und Mengenformen über Signaturketten“, Dissertation A, TH Magdeburg 1980

[Ben82] K. Benecke. “AFUL- Eine Anfragesprache für Datenbanken”. In Weiterbildungszentrum für mathematische Kybernetik und Rechentechnik/Informationsverarbeitung, Studientexte Datenbanken TU Dresden Heft 59, pages 100 – 107, 1982.

[Ben88] K. Benecke, "Hierarchische Datenstrukturen", Habilitation, Technische Universität Magdeburg, 1988

[Ben91] K. Benecke, "A powerful Tool for Object-Oriented Manipulation", in "Object-Oriented Databases: Analysis, Design & Construction (DS-4) R.A. Meersmann, W. Kent, S. Khosla (editors), Elsevier Science Publisher B.V. (North Holland) 1991, S. 95-121

[Ben98] K. Benecke, “Strukturierte Tabellen – Ein neues Paradigma für Datenbank- und Programmiersprachen”, Deutscher Universitätsverlag, ISBN 3-8244-2099-6 Wiesbaden 1998

[Ben16] K. Benecke, „o++oPS The simplest Programming Language“, BoD, 2016, ISBN 978-3-7412-4281-6

[BH11] K. Benecke, A. Hauptmann, “Does the School Need a Tabular Computer Language?” http://www.infonomics-society.org/IJDS/Does%20the%20School%20Need%20a%20Tabular%20Computer%20Language.pdf, pages 520-527, 2011.

[BSH16] K. Benecke, M. Schnabel, A. Hauptmann. Internet server for OttoPS. http: //ottoPS.eu, 2016.

[B+10] S. Boag, D. Chamberlain, et al. XQuery 1.0: An XML query language. http://www.w3.org/TR/xquery/, 2010.

[C+76] D. Chamberlain and et al. SEQUEL 2: A unified approach to data definition, manipulation and control. IBM Journal of Research of Development, 20(6):560 – 575, 1976.

[C+07] D. Chamberlain et al. XML Anfrage Use Cases. http://www.w3.org/TR/xmlquery-use-cases, 2007.

[Cod70] E.F.Codd, „A Relational Model of Data for Large Shared Data Banks“, Communications of the ACM, Vol. 13, No. 6 June 1970, S. 377-387

[Cod72] E. F. Codd, „Relational Completeness of Datenbanken Sublanguages“, in R. Rustin (ed.), Datenbanken Systems, Prentice- Hall, Englewood Cliffs, N.J., 1972

[Cod79] E.F. Codd, „Extending the Datenbanken Relational Model to capture more Meaning“, ACM Transactions on Datenbanken Systems, Vol. 4 , No 4, Dec. 1979, S. 379-434

[DL89] P. Dadam, V. Linnemann, "Advanced Information Management (AIM): Advanced Datenbanken Technology for integrated applications“, IBM System Journal Vol. 28, No 4 1989

[EN94] R. Elmasri, S.B. Navathe, „Fundamentals of Datenbanken Systems“, The Benjamin/ Lummings Publishing Company, Inc. Redwood City, California 1994

[Gol08] C. Goldberg, “Do you know SQL? About semantic errors in database queries”, http://dbs.informatik.uni-halle.de/sqllint/tlad09.pdf

[Hau10] A. Hauptmann, “OttoQL: Probleme der Implementation nichtrelationaler Datenbank-sprachen (mit besonderer Berücksichtigung der logischen Optimierung)”. Studienarbeit, Uni Magdeburg, benecke-systeme, 2010.

[KBL06] M. Kifer, A. Bernstein, P. M. Lewis “Datenbanken Systems An Application Oriented Approach”, Person International Edition, Complete Version, ISBN 0-321-31256-2, 2006, 1235 Seiten

[KR71] H. Kaphengst, H. Reichel, „Algebraische Algorithmentheorie“, VEB Robotron, Wiss. Informationen und Berichte, Nr. 1/71 Reihe A, Sommer 1971

[Kud15] T. Kudraß “Taschenbuch Datenbanken”, 2. Auflage, Hanser Leipzig, ISBN 978-3-446-43508-7, 2015

[MGL+10] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, T. Vassilakis, “Dremel: Interactive Analysis of Web-Scale Datasets”, Proc. VLDB 2010, Singapore, 10 pages.

[MMH13] Y. Minsky, A. Madhavapeddy, J. Hickey, “Real World OCAML- Functional Programming for the masses”, O'Reilly, 2013, ISBN 978-1-449-32391-2

[MRS08] C. D. Manning, P. Raghavan, H. Schütze, “An Introduction to information retrieval” Cambridge University Press, http://nlp.stanford.edu/IR-book/html/htmledition/irbook.html

[OCA15] “Objective Caml Tutorial”, http://mirror.ocamlcore.org/ocaml-tutorial.org/index-2.html

[Pla14] H. Plattner “A Course in In-Memory Data Management”, Springer 2014, ISBN 9783642552700

[Rei90] W. Reichstein. “Implementation der Strichlistenoperation Operation stroke in C”, Praktikumsbeleg, VE CS Magdeburg, 1990.

[Sch96] D. Schamschurko, „Implementation des Rumpfes des GIB-AUS-MIT-Konstrukts in CAML-Light“, Praktikumsbeleg, DeTeCSM Magdeburg, Betreuer: K. Benecke, März 1996, 123 Seiten

[SHL75] N. C. Shu, B. C. Housel, V. Y. Lum, „CONVERT- A High Level Translation Definition Language for Data Conversion“, Communications of the ACM, Vol.18 Nr.10,Oct., 1975, S. 557-567

[SLPW89] G. Saake, V. Linnemann, P. Pistor, L. Wegener, "Sorting Grouping and Duplicate Elimination in the Advanced Information Management Prototyp, Proc. 15. Intern. Conference on Very Large Databases 1989, S. 307-316

[SSU91] A. Silberschatz, M. Stonebraker, J. Ullman, „Database Systems: Achievements and Opportunities, CACM, Oct. 1991 Vol. 34, No 10, S. 110-120

[Ull82] J. D. Ullman, „Principles of Database Systems“, Computer Science Press, Rockville, Maryland 1982

[Zem85] H. Zemanek, „Formal Definition the Hard Way“, Proc. IFIP TC2 Working Conference, Wien 1985; North Holland, S. 411-417

Liste der Anfragen

Liste von Tabmenten

Alphabetical Index