



FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Ein Bash-zu-SQL-Übersetzer für die in-situ Dateianalyse

Maximilian E. Schüle





FAKULTÄT FÜR INFORMATIK

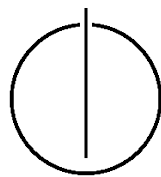
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelorarbeit in Informatik

Ein Bash-zu-SQL-Übersetzer für die in-situ Dateianalyse

A Bash to SQL Compiler for in-place File Analysis

Autor:	Maximilian E. Schüle
Themensteller:	Prof. Alfons Kemper, Ph.D.
Betreuer:	Tobias Mühlbauer, M.Sc.
Datum:	15. Februar 2015



Ich versichere, dass ich diese Abschlussarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 5. Januar 2015

Maximilian E. Schüle

Abstract

This paper is about bash scripts analyzing flat files (commonly known as CSV files) that will be compared to SQL queries doing the same on relational databases.

Methods for analyzing data will be tested on their efficiency and how they could be adapted for use with relational databases. That is why some useful familiar unix command line tools as well as newer innovations using the command line will be presented. But in the end, it shows the disadvantage of common scripts in fields of duration and maintainability.

The first goal is a implementation of the TPC-H benchmark only using unix command line tools, afterwards a bash to SQL converter will be presented, which makes query processing on flat file databases much easier.

As a result of performance tests using TPC-H benchmarks, times about up to 15 minutes for processing a query, command line tools should be replaced by an equivalent SQL query.

The presented Bash2SQL compiler converts each command to SQL statements and connects SQL queries, where their origin commands are in one pipeline. The produced SQL query can stand alone or be embedded in a new bash script.

Zusammenfassung

Diese Arbeit beschäftigt sich mit Bash-Skripten, die tabellenähnliche CSV-Dateien auslesen, im Vergleich zu ihnen äquivalenten SQL-Abfragen, die auf relationalen Datenbanken arbeiten.

Es wird der Frage nachgegangen, welche Methoden in der Wissenschaft zur Datenanalyse eingesetzt werden, wie effizient diese sind und untersucht, **wie diese für relationale Datenbanksysteme angepasst werden können**. Das Ziel ist zu zeigen, dass die bisherigen Methoden langsam und kompliziert sind und eine Konvertierung in SQL möglich ist. Diese Arbeit stellt die für die Implementierung nötigen Kommandos vor und untersucht auch neuere Erfindungen, die Abfragen auf Textdateien erleichtern.

Damit wird zuerst ein Vergleich geschaffen, in dem der TPC-H Benchmark ausschließlich unter Verwendung der Unix-Kommandos mit Shell-Skripten implementiert wird und im Anschluss dazu ein Übersetzer entworfen, mit dem Skripte automatisiert in SQL-Abfragen konvertiert werden. Der Performanzvergleich der TPC-H Benchmarks zeigt deutlich den Optimierungsbedarf, Abfragezeiten von bis 15 Minuten bei bis zu 10 GB großen sind zu viel. Auch der Einsatz verschiedener Hilfsmittel können die Abfragen nicht beschleunigen, sodass eine Konvertierung notwendig wird.

Der entwickelte Bash2SQL-Compiler übersetzt jedes Kommando in eine SQL-Abfrage und verschachtelt über eine Pipeline verbundene Kommandos, sodass am Ende ein gültiger SQL-Ausdruck entsteht, eventuell eingebunden mittels eines SQL-Kommandos in ein Bash-Skript.

Inhaltsverzeichnis

Abstract	vii
1. Einführung	1
1.1. Motivation	1
1.2. Textbasierte Datenbanken	2
1.3. Vorgehen	4
2. Bash statt SQL	5
2.1. Relationale Algebra der Unix-Shell im Vergleich zu SQL	5
2.1.1. Grundlage	5
2.1.2. Selektion	6
2.1.3. Projektion	6
2.1.4. Vereinigung	7
2.1.5. Kreuzprodukt	7
2.1.6. Mengendifferenz	7
2.1.7. Umbenennung	8
2.1.8. Relationaler Verbund	8
2.1.9. Gruppierung und Aggregation	10
2.2. Performanzmessungen	11
2.2.1. TPC-H Benchmarks	11
2.2.2. Implementierung mit Shell-Skripten	12
2.2.3. Optimierung durch Parallelisierung	16
3. Vergleich	19
3.1. SQL-ähnlicher Syntax	19
3.1.1. Datamash	19
3.1.2. csvtool	21
3.1.3. Fsdb	21
3.2. Abfragen mit SQL	22
3.2.1. txt-sushi	22
3.2.2. csvfix	23
3.2.3. csvkit	25
3.2.4. querycsv.py	25
3.2.5. gcsvsql	26
3.2.6. Mynodbcsv	26
3.2.7. shql	26
4. Parser mit Yacc und Lex	27
4.1. Vorwissen zu Yacc und Lex	27

4.2.	Der Bash2SQL-Übersetzer mit Yacc	28
4.2.1.	Der Quellcode	28
4.2.2.	Der Parser	28
4.2.3.	Der Lexer	31
4.2.4.	Bedienung des Bash2SQL-Übersetzers	32
5.	Parser mit ANTLR	33
5.1.	Konfiguration	33
5.1.1.	Installieren der Bibliothek	33
5.1.2.	Starten von ANTLR	33
5.1.3.	Vorwissen zu ANTLR	34
5.2.	Der Bash2SQL-Übersetzer mit ANTLR in C	34
5.2.1.	Neues Konzept	35
5.2.2.	C-Quellcode	36
5.2.3.	Die Grammatik	38
5.2.4.	Lexer-Regeln	40
5.2.5.	Bedienung	41
5.3.	Der Bash2SQL-Übersetzer mit ANTLR in C++	41
5.3.1.	Unterschiede: C vs. C++ mit ANTLR	42
5.3.2.	Die Klasse TheQuery	43
5.3.3.	Der Parser näher betrachtet	46
5.3.4.	Bedienung	50
5.3.5.	Rückübersetzung einer TPC-H Abfrage	50
6.	Ausblick	53
6.1.	Schleifen/Bedingungen	53
6.2.	Sprache awk separat	53
6.3.	Vor Übersetzen zusammenfügen	53
6.4.	Fazit	54
Appendix		57
A.	TPC-H-Abfragen	57
A.1.	Abfragen	57
A.1.1.	Abfrage 1	57
A.1.2.	Abfrage 2	58
A.1.3.	Abfrage 3	60
A.1.4.	Abfrage 4	61
A.1.5.	Abfrage 5	62
A.1.6.	Abfrage 6	64
A.1.7.	Abfrage 7	65
A.1.8.	Abfrage 8	67
A.1.9.	Abfrage 9	69
A.1.10.	Abfrage 10	71
A.1.11.	Abfrage 11	72

A.1.12. Abfrage 12	73
A.1.13. Abfrage 13	74
A.1.14. Abfrage 14	75
A.1.15. Abfrage 15	76
A.1.16. Abfrage 16	77
A.1.17. Abfrage 17	78
A.1.18. Abfrage 18	79
A.1.19. Abfrage 19	80
A.1.20. Abfrage 20	81
A.1.21. Abfrage 21	82
A.1.22. Abfrage 22	84
A.2. Parallelisierte Abfragen	85
A.2.1. Abfrage 2	85
A.2.2. Abfrage 3	87
A.2.3. Abfrage 4a	88
A.2.4. Abfrage 4b	89
A.2.5. Abfrage 5	90
A.2.6. Abfrage 7	92
A.2.7. Abfrage 8	94
A.2.8. Abfrage 9	96
A.2.9. Abfrage 10	98
A.2.10. Abfrage 11	99
A.2.11. Abfrage 12	100
A.2.12. Abfrage 13	101
A.2.13. Abfrage 14	102
A.2.14. Abfrage 15	103
A.2.15. Abfrage 16	104
A.2.16. Abfrage 17	105
A.2.17. Abfrage 18	106
A.2.18. Abfrage 19	107
A.2.19. Abfrage 20	108
A.2.20. Abfrage 21	109
A.2.21. Abfrage 22	111
A.3. Die Ergebnisse	113
A.3.1. SF=1	113
A.3.2. SF=10	114
B. Vergleich	115
B.1. Datamash Abfragen	115
B.1.1. Abfrage 1	115
B.1.2. Abfrage 2	116
B.1.3. Abfrage 3	118
B.1.4. Abfrage 4	119
B.1.5. Abfrage 5	120
B.1.6. Abfrage 6	121
B.1.7. Abfrage 7	122

B.1.8. Abfrage 8	124
B.1.9. Abfrage 9	126
B.1.10. Abfrage 10	127
B.1.11. Abfrage 11	128
B.1.12. Abfrage 12	129
B.1.13. Abfrage 13	130
B.1.14. Abfrage 14	131
B.1.15. Abfrage 15	132
B.1.16. Abfrage 16	133
B.1.17. Abfrage 17	134
B.1.18. Abfrage 18	135
B.1.19. Abfrage 19	136
B.1.20. Abfrage 20	137
B.1.21. Abfrage 21	138
B.1.22. Abfrage 22	139
B.1.23. Ergebnisse	140
C. Der Bash2SQL-Compiler	141
C.1. Yacc und Lex	141
C.1.1. Makefile	141
C.1.2. bash2sql.lex	141
C.1.3. bash2sql.y	142
C.2. ANTLR mit C	145
C.2.1. Makefile	145
C.2.2. SimpleBashSQL.g	145
C.3. ANTLR mit C++	159
C.3.1. Makefile	159
C.3.2. Query.hpp	159
C.3.3. Query.cpp	162
C.3.4. SimpleBashSQL.g	174
Literaturverzeichnis	191

1. Einführung

1.1. Motivation

Die Welt wächst zusammen, die Kommunikation steigt und mit ihr auch das Datenvolumen. Waren es 2005 geschätzte 130 Exabyte (130 Mrd. GB) an erzeugten Daten weltweit, so hat sich die Menge bis 2012 um das 20-fache auf 2837 Exabyte gesteigert und 2015 soll die Marke der Zettabyte gebrochen werden, ganze 40000 Exabyte werden an Daten erwartet, die Menge verdoppelt sich jährlich.[22]

Der Begriff Big Data ist weit gefasst, verstehen die meisten darunter den Rohstoff zur Analyse digitaler Profile von Menschen, so sind ihr Kern doch Daten, denn das Potential der Digitalisierung steigt ins Unermessliche. Daten werden maschinell generiert, allein mit RFID-Chips werden Tiere gebrandmarkt, Menschen geben ihre Arbeitszeiten durch, selbst Mülltonnen werden damit gekennzeichnet. Neue Technologie ergibt bessere Sensoren mit automatisierteren Abläufen, damit nützt sie vor allem der Forschung: noch mehr Ergebnisse, noch genauere Vorhersagen machen noch präziseres Reagieren in Klimaforschung, Geologie und Physik möglich.

Für den Austausch benötigt man die Daten und damit ein geeignetes Datenformat, eines hat sich schlichtweg durchgesetzt: die CSV-Datei - ein portables und menschenlesbares Format zur Darstellung tabellen-ähnlicher Daten. Aber zur Analyse allein reicht ein Datenformat nicht aus, es müssen Abfragen darüber laufen, die einem Zugriff verschaffen. Die Lösung sind ganz klar relationale Datenbanken mit einer eigenen Abfragesprache, dennoch scheint unter Wissenschaftlern und Firmen eine Ablehnung gegenüber proprietären Datenbanksystemen zu herrschen, da sie ihre Daten lieber in Textform und CSV-Dateien halten.

Schaut man in einem Standardwerk zur Datenhaltung für Geowissenschaftler nach, so schlägt es gleichrangig zwei Konzepte vor: textbasierte Datenbanken (wie CSV-Dateien) und relationale Datenbanken. Der anschließende Vergleich erkennt: textbasierte Datenbanken benötigen zum Zugriff "ein Skript, dass die Datei öffnet und in der ganzen Datei sucht"[12, S. 19]. Diese Skripte sind auf unixoiden Systemen meist Bash-Skripte die schwierig zu schreiben und zu warten und in der Ausführung sehr langsam sind.

Doch zurück zu großen Datenmengen, was passiert wenn die Daten zu groß werden? Die Abfragezeit mit Skripten explodiert. Will man ein Datenbanksystem zu Hilfe ziehen, so unterstützen sie CSV-Dateien nur unzureichend, sie müssen zuerst geladen werden oder, wie es Oracle anbietet, als *external table* in die Datenbank eingebunden, beides ist teuer. Programme, die für den CSV-Import in Datenbanken sorgen, gibt es genügend. Darum stellt diese Arbeit ein Verfahren vor, wie Skripte in SQL-Abfragen umgewandelt werden. Jetzt wird das Beste aus beiden Welten kombiniert - Bash-Skripte mit SQL-Abfragen. Soweit wie möglich sollen alle Kommandos eines Skripts in eine äquivalente SQL-Anweisung übersetzt werden. Um dennoch die Funktionen eines Skripts nicht zu verlieren, soll es möglich sein, die Abfrage in das Skript einzubinden, also innerhalb eines Skripts auf eine

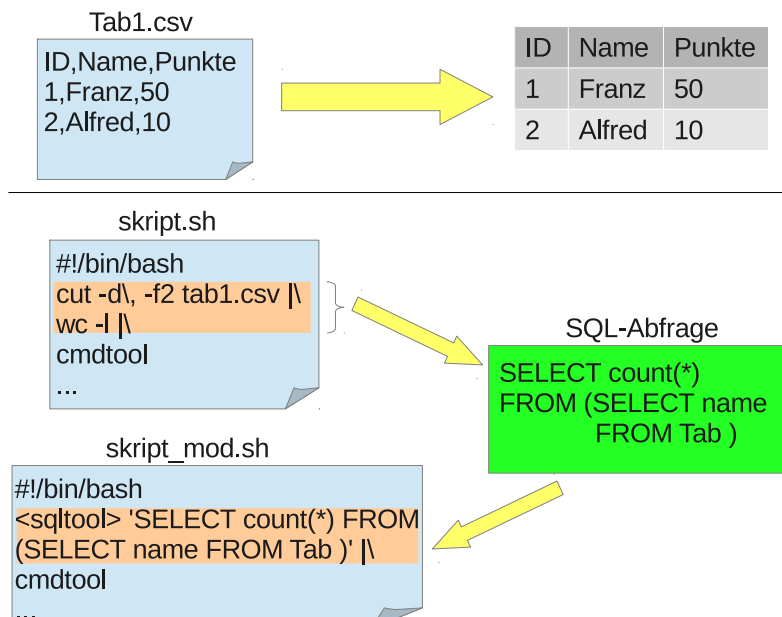


Abbildung 1.1.: Grundidee: Daten werden vorab in eine Datenbank geladen, anschließend werden die Skripte übersetzt

Datenbank zugreifen, die einem das gewünschte Ergebnis zurückliefert (*SQL inline* mittels Kommandos). Dazu wird ein Übersetzer geschrieben, der Shell-Skripte einliest und sie komplett oder teilweise in SQL-Abfragen übersetzt (siehe Abb. 1.1). So können die Vorteile eines Datenbanksystems mit einer einfachen deklarativen Sprache (SQL), parallelisierter Anfrageverarbeitung und deren Geschwindigkeit ausgenutzt werden, die Speicherung der Daten erfolgt aber weiterhin in CSV-Dateien.

1.2. Textbasierte Datenbanken

Das Jahr 1970 bedeutete einen Umbruch im Bereich der Datenbanken, Edgar F. Codd veröffentlichte sein Papier über das relationale Modell, der Grundlage aller relationaler Datenbanken. Es erlaubt, mehrere Relationen zu verbinden, ohne dass der Benutzer sich um die interne Repräsentation kümmern muss [8]. Dennoch dauerte es weitere neun Jahre bis Larry Ellison und Bob Miner den ersten Prototyp einer relationalen Datenbank mit ihrer Firma Software Development Laboratories auf den Markt brachten, bei IBM sogar zwei Jahre länger [11]. Dennoch werden Datensätze gerne noch in textbasierten Datenbanken (engl.: flat file databases) wie CSV-Dateien wegen ihrer Einfachheit gespeichert.

Textbasierte Datenbanken gibt es schon immer, sobald eine Person ein Adressbuch von Kontakten mit Namen und Adressen pflegt, so ist das eine textbasierte Datenbank. Sobald ein Kontakt den Wohnsitz wechselt oder die Anzahl an Freunden wächst, muss die Da-

Franz Winkler	Am Winkl 5, 80000 Musterstadt
Xaver Ziegler	Maurergasse 19, 80000 Musterstadt

Abbildung 1.2.: Beispiel für textbasierte Datenbank

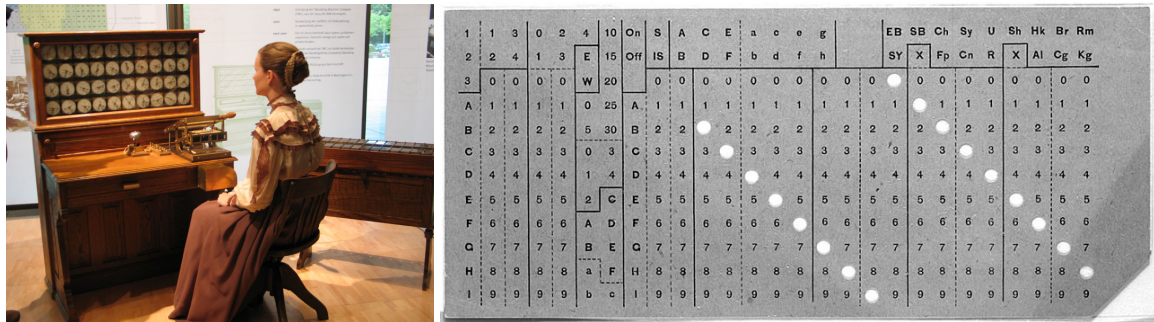


Abbildung 1.3.: Hollerith Maschine [4] mit Lochkarte [wikipedia.de]

tenbank aktualisiert werden, ist sie geordnet, bedeutet das in manchen Fällen, die Daten komplett neuzuschreiben, ein unnötiger Aufwand.

Mit diesem Problem konfrontiert hat der deutschstämmige US-Amerikaner Hermann Hollerith bereits 1884 das erste Datenformat erfunden, [15, S.48], abgeschaut von Schaffnern in Zügen, die die Fahrkarten an bestimmten Stellen gelocht haben um Wiederverwendung auszuschließen, hat er die Lochkarte entwickelt. Das System wurde 1890 erstmals bei einer Volkszählung eingesetzt, wobei die Lochkarten die Informationen über Geschlecht und Alter enthielten und elektrische Maschinen in 3,4 Sekunden eine Karte auslasen. Weitere Aufträge auf der ganzen Erde folgten, Hollerith gründete die Computing Tabulating Recording Company (CTR), aus der 1924 schließlich IBM hervorging [4].

Textbasierte Datenbanken sind mittlerweile weit verbreitet, in Unix-Systemen (entwickelt Mitte der 1960-er Jahre) werden so Daten zum Beispiel zur Passwort- oder Gruppenverwaltung gespeichert. (siehe Abb. 1.4) Jeder Datensatz wird durch einen Zeilenvorschub mit Wagenrücklauf (engl.: carriage return und line feed, CRLF) abgeschlossen, einzelne Felder durch ein Trennzeichen (engl.: delimiter) separiert [9].

Ein Beispiel für textbasierte Datenbanken sind CSV-Dateien (Comma-Separated Values) manchmal auch DSV-Dateien (Delimiter-Separated Values) genannt, also durch Komma

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
```

Abbildung 1.4.: Auszug aus /etc/passwd eines Unix-Systems mit Trennzeichen ':'

oder anderes Trennzeichen separierte Tabellen, bei denen in den meisten Fällen ein Komma oder ein Semikolon die Spalten trennt. Erstmals erwähnt als Teil der Fortran Spezifikation zwischen 1968 und 1972 für listenorientierte Ein- und Ausgabe (Common Format and MIME Type for Comma-Separated Values (CSV) Files) [1, S. 17], haben sich CSV-Dateien mittlerweile zum Standard im Datenaustausch entwickelt. Inzwischen existiert sogar eine Richtlinie für CSV-Dateien, herausgegeben von der Internet Engineering Task Force in der RFC4180. [19] Nach dieser Richtlinie sollen CSV-Dateien folgende Regeln einhalten:

- jeder Datensatz ist in einer Zeile gespeichert, beendet durch **CRLF**
- der letzte Datensatz benötigt keinen **Zeilenvorschub** (CRLF)
- die erste Zeile kann eine Kopfzeile (engl.: header) sein und muss für jede Spalte einen **Bezeichner** enthalten
- die Felder sind durch Kommata getrennt, **nach der letzten Spalte darf kein Komma folgen**; Leerzeichen sind Teil eines Feldes
- jedes Feld kann in Anführungszeichen (doppelte Hochkommata) stehen
- **Kommata und CRLF als Teil eines Feldes müssen in Anführungszeichen eingeschlossen sein**
- **Felder in doppelten Hochkommata (double quoted fields) sind durch einen doppelte Hochkommata eingeklammert, falls dieser im Feld auftaucht wird er mit dem doppelten Hochkommata escaped.**

Die Vorteile solcher Datenbanken liegen in ihrer Einfachheit, sie speichern alle Informationen, es werden keine weiteren Informationen benötigt, es ist leicht, damit Informationen zu importieren und exportieren, da außer einem Trennzeichen und einem finalen Zeilenvorschub keine Konventionen einzuhalten sind.

Die Nutzung textbasierter Datenbanken bei Abfragen birgt aber Nachteile, so lassen sich die Daten speichern und verschicken, ein Ändern einzelner Datensätze erweist sich als schwierig, ohne die komplette Datei zu überschreiben. Da das Format nur wenigen Beschränkungen unterliegt, enthält in manchen Fällen die erste Zeile die Feldbezeichner, in anderen bereits den ersten Inhalt. Aber das gravierendere Problem liegt in der Anzahl der Datensätze, jeder Datensatz muss zeilenweise ausgelesen werden, da auch keine Konventionen befolgt werden, ist auch nicht von einer Sortierung auszugehen. Somit ist die **Performanz solcher Datenbanken auch um einiges schlechter, wie dieses Kapitel anhand von Benchmarks noch zeigt.**

1.3. Vorgehen

Im Nachfolgenden werden zuerst grundlegende Befehle erklärt, wie sie in Skripten zur Analyse von Textdateien vorkommen und auf bereits existierende Ansätze verwiesen, wie Daten in Textdateien bewältigt werden. Abschließend führt die Arbeit in die Welt der Parser und zeigt einen Ansatz auf, wie die Skripte übersetzt werden.

2. Bash statt SQL

Das Ziel ist zwar, am Ende die Bash-Kommandos zu ersetzen, aber um die Kommandos zu ersetzen, wird zuerst die Idee benötigt, wie sähe denn eigentlich so eine SQL-Anfrage in der Bash aus, also nur mithilfe der klassischen Unix-Befehle, die textbasierte Datenbanken auslesen. Daher erklärt dieses Kapitel zuerst, wie solche Datenbanken nur von Kommandos wie `cat`, `cut`, `awk` und `sed` analog zu SQL-Abfragen ausgelesen werden können. Anschließend werden mit diesen Befehlen Datenbank-Benchmarks implementiert und die Zeit der Abfragen bei großen Datenmengen gemessen. Der Vergleich mit neueren Programmen erfolgt dann im nächsten Kapitel.

Punktetabelle:			Zeittabelle:	
ID	Name	Punkte	ID	Zeit
1	Franz	50	1	44
2	Alfred	10	2	88
3	Marie	27	3	67

Abbildung 2.1.: Beispiel Datenbank

2.1. Relationale Algebra der Unix-Shell im Vergleich zu SQL

Welche Strukturen eines Shell-Skripts sind in welche SQL-Anweisungen zu übersetzen? Um dies besser vergleichen zu können, werden im Folgenden Ausdrücke der relationalen Algebra als Befehle der Unix-Shell ausgedrückt und eine äquivalente Abfrage in SQL angegeben. Als Grundlage für die relationale Algebra dienen Operatoren aus dem Buch Datenbanksysteme [3] und in diesem Kapitel werden ausschließlich die grundlegenden Kommandos der Unix-Shell verwendet, heute auch bekannt als GNU core utilities [10].

2.1.1. Grundlage

Zuerst einmal liegt eine solche textbasierte Datenbank vor, bevor relationale Algebra angewandt wird, soll nur der Inhalt ausgegeben werden, da nutzt ein einfacher SQL-Befehl wie:

```
SELECT * FROM Punktetabelle
```

Solche Anfragen können mit Befehlen wie `cat`, `more`, `less`, ... übersetzt werden und finden sich häufig, wenn vorher Daten durchgepipet werden.

```
cat Punktetabelle;
```

2.1.2. Selektion

Wenn jetzt Tupel ausgewählt werden, die ein Prädikat erfüllen sollen, also entspricht dies der Selektion, dann sind zwei Fälle zu unterscheiden: Äquivalenz: $\sigma_{ID=3}(Tabelle)$ und Vergleich $\sigma_{ID<3}(Tabelle)$ oder in SQL:

```
SELECT * FROM Punktetabelle WHERE ID=3;
SELECT * FROM Punktetabelle WHERE ID<3
SELECT * FROM Punktetabelle WHERE Name='Marie'
```

Die allgemeine Lösung nutzt *awk*, mit dem alle Vergleichsfunktionen einer höheren Programmiersprache implementiert sind, hierbei sind die Felder durch $\$1, \dots, \n bezeichnet, $\$0$ bezeichnet alle Felder, die Option *-F*, bezeichnet das Feldtrennzeichen (Delimiter), anschließend folgt ein Muster und der Befehl, der ausgeführt wird ('*pattern {CMD}*'), in diesem Fall zuerst die Bedingung $1==3$ und der Befehl zur Ausgabe (*print \$0*) für alle Spalten (das *SELECT ** der SQL).

```
$ awk -F, '$1==3 { print $0 }' Punktetabelle.csv
$ awk -F, '$1>3 { print $0 }' Punktetabelle.csv
$ awk -F, '$2="Marie" { print $0 }' Punktetabelle.csv
```

Andere grundlegende Kommandos funktionieren meist nur bei kompletter Äquivalenz, wie *grep*, das in einer Datei nach allen Vorkommen der gewünschten Zeichenfolge sucht. Bei *sed* ist die Äquivalenz auch einfach einzugeben, dabei ist es aber von Vorteil, zumindest den vorderen und hinteren Spaltentrenner mit anzugeben, oder gar alle möglichen:

```
$ grep -r '3,.*' Punktetabelle.csv
$ grep -r 'Marie' Punktetabelle.csv
$ sed -nr '/3,./p' Punktetabelle.csv
$ sed -nr '/Marie/p' Punktetabelle.csv
```

2.1.3. Projektion

Wenn nun einzelne Spalten ausgewählt werden, so wird die Projektion benötigt:

$\Pi_{Name, Punkte}(Punktetabelle)$

oder in SQL:

```
SELECT Name FROM Punktetabelle
```

Das klassische Unix-Kommando dazu ist *cut*, das es mit Hilfe der Option *-f* erlaubt, einzelne Felder zu extrahieren, Felder werden beginnend beim ersten durch Aufzählung mit Kommata bestimmt (1,3) und ganze Bereiche mit Bindestrich ausgewählt (1-3 entspricht Feldern eins bis drei), der Spaltentrenner wird durch die Option *-d* mitgeteilt (Standard: Leerzeichen).

```
$ cut -f2,3 -d, Punktetabelle.csv
```

Die Kommandos *awk* und *sed* erlauben die Projektion auch, ersterer Befehl einfach mit *print*, bei *sed* müssen explizit die Spaltentrenner angegeben werden:

```
$ awk -F, '{ print $2,$3 }' OFS=, Punktetabelle.csv
$ sed -nr 's/([^\,]*) ,([^\,]*) ,(.*)/\2\3/p'
```

2.1.4. Vereinigung

Die Vereinigung $\Pi_{ID}(Punktetabelle) \cup \Pi_{ID}(Zeittabelle)$ ist am einfachsten in der Unix-Shell zu realisieren, schließlich unterstützt fast jeder Befehl durch Eingabe mehrerer Dateien das Zusammenfügen dieser. Für das Zusammenfügen oder Konkatenieren drängt sich *cat* (concatenate) geradezu auf, dadurch definiert sich doch dieser, einfach alle Dateien der Reihe nach auflisten:

```
$ cat datei1 datei2
```

Und schon sind sie vereinigt, analog das Beispiel der oben gezeigten Projektion:

```
SELECT ID FROM Punktetabelle
UNION
SELECT ID FROM Zeittabelle
```

Das Beispiel erfordert vorher die Selektion, daher werden zwei anonyme Pipes verwendet, das sieht dann so aus:

```
$ cat <(cut -f1 -d, Punktetabelle.csv) \
      <(cut -f1 -d, Zeittabelle.csv)
```

2.1.5. Kreuzprodukt

Will man in der Shell das Kreuzprodukt $Punktetabelle \times Punktetabelle$ bilden, so geschieht das in SQL durch Auswahl mehrerer Tabellen:

```
SELECT * FROM Punktetabelle , Punktetabelle
```

In der Shell hilft einem auch hier *awk* weiter, diesmal mit Feldern. Zuerst werden alle Eingabezeilen (leeres Suchmuster) oder nur die gewünschten wie bisher durchgegangen und in dem Feld aufsteigend gespeichert. Anschließend, also im Schlussteil (bezeichnet durch END), kann mit den Feldern alles produziert werden, das Kreuzprodukt erfolgt durch die Ausgabe mit print in einer doppelten For-Schleife.

```
cat Punktetabelle | awk -F\| '
{
    lines[i++]=$0
}
END{
    for (i in lines)
        for (j in lines)
            print lines[i], lines[j]
}' OFS=,
```

2.1.6. Mengendifferenz

Um alle Mengenoperationen der Algebra abzudecken, wird auch noch die Differenz benötigt, geschrieben als $R - S$, zum Beispiel ergibt $\Pi_{ID}(Punktetabelle) - \Pi_{ID}(Zeittabelle)$ diejenigen Tupel, zu denen kein passender Eintrag in der Zeittabelle enthalten ist.

```
SELECT ID FROM Punktetabelle
EXCEPT
SELECT ID FROM Zeittabelle
```

In der Unix-Shell gibt der Befehl *comm* die Zeilen in drei Spalten aus, zuerst die nur der ersten Datei (1), dann die nur der Zweiten (2) und dann die aus beiden (3), durch Angabe der Zahlen, können die Spalten unterdrückt werden. Für den Vergleich müssen die Dateien aber sortiert sein. Analog zur Algebra entspricht folgender Befehl der Differenz:

```
$ comm -23 R S
```

Angewandt auf die Beispieltabellen:

```
$ comm -23 <(cut -f1 -d, Punktetabelle.csv | sort) \  
              <(cut -f1 -d, Zeittabelle.csv | sort)
```

2.1.7. Umbenennung

Um alle Ausdrücke der relationalen Algebra abzudecken, fehlt nun noch die Umbenennung der Tabelle $\rho_{t1}(Punktetabelle)$ und einzelner Spalten $\rho_{Nr \leftarrow ID}(Punktetabelle)$. Da in der Shell die Tabellen nichts anders als Datenströme sind, ist keine Unterscheidung der Namen notwendig, eine Möglichkeit, die Tabellen umzubenennen, besteht nur darin, eine temporäre Hilfstabelle mit neuem Namen anzulegen.

```
$ cp Punktetabelle.csv t1.csv
```

Auch einzelne Spalten können nur mit ihrer Nummer angesprochen werden (\$1,\$2, etc.), eine Umbenennung kann nur für die Ausgabe erfolgen, der Strom muss also vorher mit *awk* oder *sed* bearbeitet werden.

```
$ sed -r 's/ID/Nr/' Punktetabelle.csv  
NR,Name,Punkte  
1,Franz,50  
2,Alfred,10  
3,Marie,2  
$ cat Punktetabelle | awk -F\| ' \  
    NR==1{  
        print "Nr", $2, $3  
    }  
    NR>2{  
        print $0  
    }  
' OFS=,
```

2.1.8. Relationaler Verbund

Alle grundlegenden Operatoren der relationalen Algebra können auch mit einfachen Skripts auf textbasierten Datenbanken erfolgen, dennoch darf ein wichtiger Operator nicht fehlen, der relationale Verbund (Join), vor allem der natürliche Verbund (natural join), der Tabellen über Äquivalenz zusammengehöriger Attribute verknüpft:

$$R \bowtie S$$

oder im Beispielfall mit Verknüpfung über ID: $Punktetabelle \bowtie Zeittabelle$

Der analoge Fall in SQL:

```
SELECT *  
FROM Punktetabelle, Zeittabelle  
WHERE Punktetabelle.ID = Zeittabelle.ID
```

In Unix stehen für Equijoins jeglicher Art, bei denen jeweils ein Feld jeder Tabelle übereinstimmen sollen, das Kommando *join* zur Verfügung. Sollen zwei CSV-Dateien miteinander verknüpft werden, so müssen das Spaltentrennzeichen und die zu verknüpfenden Spalten (*-1 spalteA -2 spalteB*) angegeben werden, standardmäßig der Leerraum (White-space) sowie die jeweils erste Spalte, und, sofern die erste Zeile die Bezeichner enthält, müssen diese als solche mit *-header* deklariert sein.

```
$ join --header -t, -1 1 -2 1 Punktetabelle.csv Zeittabelle.csv
```

Zu beachten ist, dass im Ergebnis eine der verbundenen Spalten dann fehlt, also oben stehende Abfrage produziert folgendes Ergebnis:

ID	Name	Punkte	Zeit
1	Franz	50	44
2	Alfred	10	88
3	Marie	27	67

Die auszugebenden Spalten können auch hinter der Option *-o* explizit angegeben werden, *0* ist die verbundene Spalte, alle anderen mit der Spaltennummer der jeweiligen Tabelle, 2.3 meint die dritte Spalte der zweiten Tabelle. Also sollen die Spalten, für die die Join-Bedingung gilt, angezeigt werden und die zweite und dritte, so hilft folgender Befehl:

```
$ join -t, -o 0 1.2 1.3 tabelleA tabelleB
```

Damit können auch Semi-Joins $R \ltimes S$ und $R \rtimes S$ produziert werden, indem die Spalten angegeben sind, für einen linken Semi-Join sind das *-o 1.1 1.2 ... 1.n* und für den rechten *-o 2.1 2.2 ...*

Ein Join der Shell ist ein Sort-Join, er funktioniert (wie *comm* auch) nur auf sortierten Dateien, folglich muss oft eine Sortierung mit *sort* erfolgen, bevor gejoint werden kann. Das Kommando *sort* arbeitet mit Quicksort [18], also mit Durchschnittslaufzeit $O(n \log n)$, in schlechten Fällen auch $O(n^2)$. Dabei muss der Sortierfunktion noch das Trennzeichen (*-t,*) sowie die zu sortierenden Spalten als Feld angegeben werden, also *-k2,3* sortiert nach dem zweiten und dritten Feld. Erfolgt ein Join danach, so ist zu empfehlen, nach exakt einer Spalte zu sortieren *-k2,2*, da das Ergebnis sonst von der Länge des nachfolgenden Textes abhängt.

```
$ sort -t, -k2,2 tabelleA | join -1 2 - tabelleB
```

sort sortiert aber auch die Kopfzeile mit, also muss diese separat behandelt werden, am besten wird die erste Zeile mit *head* extrahiert, alle anderen mit *tail* und nach dem Sortieren können die Zeilen wieder zusammengefügt werden.

```
$ head -1 tabelleA > nurKopf
$ tail -n+2 tabelleA | sort -t, -k1,1 | cat nurKopf - > tabAmod
$ head -1 tabelleB > nurKopf2
$ tail -n+2 tabelleB | sort -t, -k1,1 | cat nurKopf2 - | \
  join -t, -1 1 -2 1 tabAmod -
```

Auch der Antijoin $R \supset S$ bzw. $R \supset S$ ist in der Unix-Shell mit der Option *-v1* und *-v2* implementiert, der die Zeilen der ersten Tabelle (bzw. der zweiten) ausgibt, zu denen kein Partner in der anderen Tabelle gefunden wurde. Damit nur die benötigten Spalten ausgegeben werden, empfiehlt es sich, die Spalten mit der Option *-o* noch explizit anzugeben:


```
$ join -t, -1 1 -2 1 -v1 -o 0 1.2 1.3 R.csv S.csv
```

Wenn zum Beispiel die Namen ausgegeben werden sollen, zu denen keine Zeit gemessen wurde, sieht das so aus:

```
$ join -t, -1 1 -2 1 -v1 -o 1.2 Punktetabelle.csv Zeittabelle.csv
```

Als einzige Join-Arten, die noch fehlen, verbleiben die äußeren Joins $R \bowtie S$, $R \ltimes S$ und $R \ltimes S$, die auch der Unix-Befehl mit der Option `-a1` und `-a2` erzeugt, wodurch alle Zeilen der ersten (analog der zweiten) Datei ausgegeben werden, auch solche mit fehlendem Join-Partner. Die fehlenden Werte, bei SQL die NullWerte, können mit `-e "Wert"` angegeben werden, sollen sie mit `"0"` aufgefüllt werden, dann mit `-e "0"`.

```
$ join -t, -a1 -a2 -1 2 -2 2 -o 0 1.1 2.1 -e "0" tabelleR tabelleS
$ join -t, -a1      -1 2 -2 2 -o 0 1.1 -e "0" tabelleR tabelleS
$ join -t,      -a2 -1 2 -2 2 -o 0 2.1 -e "0" tabelleR tabelleS
```

2.1.9. Gruppierung und Aggregation

Über die relationale Algebra hinaus, geht der Gamma-Operator, der die Werte gruppiert und Aggregatsfunktionen wie `max`, `min`, `sum` oder `avg` auf ihnen erlaubt. So gibt

$\gamma_{count(*)}(Punktetabelle)$ die Anzahl aller Teilnehmer aus. Aus dem Standard-Repertoire der Unix-Shell ist auch der Befehl `awk` nützlich: Dazu sollten die Dateien vorher nach den Feldern sortiert sein, der Trick nutzt die Sortierung aus, die Werte der Spalten, nach denen gruppiert wird, wird vermerkt, die Aggregation beginnt. Sobald sich ein Wert verändert, ist also zur nächsten Gruppe gesprungen worden, die aggregierten werden ausgegeben, die nächste Gruppe folgt, bis schließlich keine Zeilen mehr nachkommen, im END-Teil werden die letzten Aggregate ausgegeben.

```
SELECT spalte2, spalte3,
       max(spalte4), min(spalte4), count(*), avg(spalte4)
FROM   Tabelle
GROUP BY spalte2, spalte3
```

Die Abfrage in `awk` übersetzt benutzt temporäre Variablen für die Summe, die Anzahl, das Minimum und das Maximum, der Durchschnitt setzt sich später aus Summe und Anzahl zusammen. Zudem werden die Werte gespeichert, nach denen gruppiert wird. Ändern sich diese nicht, so werden die Werte der aktuellen Zeile in der Aggregation ergänzt, `count` wird inkrementiert, der entsprechende Wert zur Summe von `sum` addiert und nach größer und kleiner für `min` und `max` geschaut. Passen die Werte zum Gruppieren nicht überein, so werden die alten ausgegeben und die Aggregationsvariablen zurückgesetzt.

```
head -1 tmp.csv > tmp1.csv
tail -n+2 tmp.csv | sort -t\| -k2,2 | cat tmp1.csv - | awk -F\| '
    NR==1{print $2, $3,
        "max(S4)", "min(S4)", "count(*)", "avg(S4)"}
    }
    NR==2{g2=$2; g3=$3; count=1; max4=$4; min4=$4; sum4=$4}
    NR>2{
        if ( g2==$2 && g3==$3 ){
            count++; sum4+= $4;
            if (max4<$4)
                max4=$4;
        }
    }
    END{print count, sum4, max4, min4}
```



```
        if (min4>$4)
            min4=$4;
    } else {
        print g2,g3,
              max4,min4,sum4,count,sum4/count;
        g2=$2; g3=$3;
        count=1; max4=$4; min4=$4; sum4=$4
    }
}
END{ print g2,g3,max4,min4,sum4,count,sum4/count }
' OFS=\\
```

Eine einfachere Lösung bietet der Befehl *uniq -c*, sofern nur die Anzahl der Vorkommnisse gezählt werden sollen.

2.2. Performanzmessungen

Das vorherige Kapitel hat die Grundlagen erklärt, also wie die relationale Algebra, auf der die relationale Anfragesprache SQL basiert, auf textbasierte Datenbanken angewandt werden kann. Dieses Kapitel behandelt die Performanz solcher Abfragen, also wie schnell sie sich ausführen lassen, auch im Vergleich zu modernen relationalen Datenbanken.

2.2.1. TPC-H Benchmarks

Um die Leistungsfähigkeit von Datenbanken zu testen wurde im Jahr 1988 auf Initiative von Omri Serlin hin ein Konsortium namens Transaction Processing Performance Council (TPC) gegründet, an dem acht Firmen der IT-Branche beteiligt waren [20]. Das Ziel war es nicht, "die Funktionen und Operationen von Rechnern zu testen, [sondern] Transaktionen zu betrachten, wie sie allgemein in der Geschäftswelt üblich sind: Der Tausch von Gütern, Dienstleistungen und Geld"[23]. So wurde der erste Benchmark für Datenbanksysteme entwickelt, genannt TPC-A, der die maximalen Transaktionen pro Sekunde misst, wenn von verschiedenen Endgeräten darauf zugegriffen wird. Der Anwendungsbereich der TPC-A Benchmark ist die Online-Verarbeitung von Transaktionen, *Online Transaction Processing* (OLTP), wie sie in potentiellen Handelsunternehmen vorkommen, die Güter und Dienstleistungen gegen Geld tauschen. Sie "[zeichnen sich aus] durch relativ kurze Transaktionen, die im Allgemeinen nur auf ein eng begrenztes Datenvolumen zugreifen." [3, S. 711]

Der aktuellste Standard für OLTP-Anwendungen ist der TPC-H Benchmark, der die Leistung der Datenbank bei üblichen Anfragen misst, ohne dass die Datenbank zuvor darauf vorbereitet wird. Dazu sind 22 verschiedene Anfragen gegeben und eine Datenbasis, die mittels eines gegebenen Zufallsgenerators generiert wird, aber sich immer nach dem Handelsunternehmensschema aus acht Relationen richtet (vgl. Abb. 2.2).

Der Generator *DBGen* erzeugt die Datenbasis in verschiedenen Größen mit unterschiedlich vielen Tupeln in Abhängigkeit eines Faktors *SF* der ungefähr der Größe aller Daten in GB entspricht, *SF=1* steht für 1 GB, die möglichen Größen sind 1 GB, 10 GB, 30 GB, 100 GB, 300 GB, 1 000 GB, 3 000 GB, 1 0000 GB, 30 000 GB und 100 000 GB an Daten die per Zufall erstellt werden.

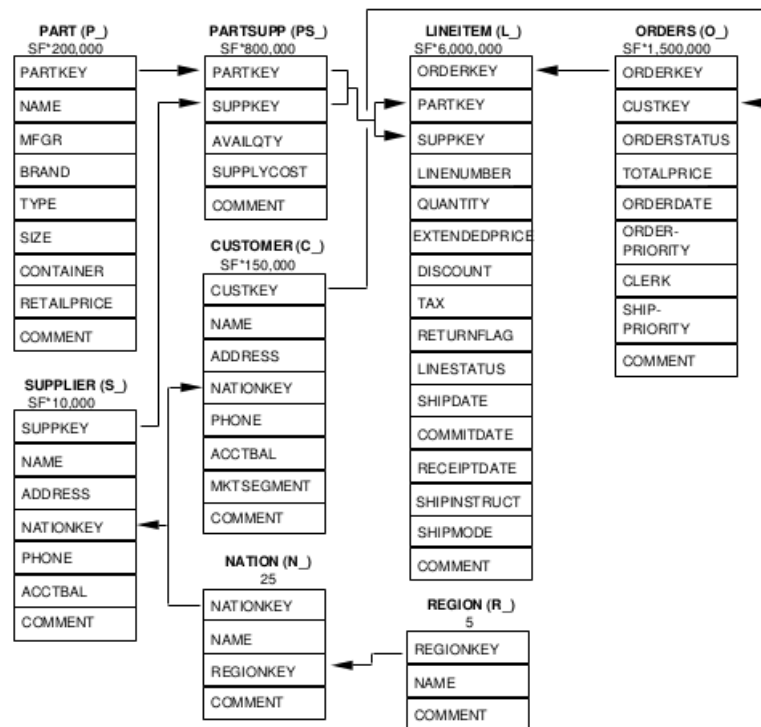


Abbildung 2.2.: TPC-H Schema [24]

2.2.2. Implementierung mit Shell-Skripten

Um nun die Leistungsfähigkeit der Shell-Skripte auf textbasierten Datenbanken zu testen, braucht es drei Werkzeuge: die Daten, die Skripte und natürlich Referenzwerten - die Skripte werden von Hand erzeugt, die Grundlage für die Datenbasis ist dieselbe wie für den TPC-H-Benchmark der Hyper-Schnittstelle¹, so lassen sich die Ergebnisse auch gut vergleichen.

Die Implementierung der SQL-Anfragen orientiert sich an dem vorgestellten Schema im letzten Unterkapitel, nachfolgend sei nur die vierte TPC-H-Anfrage vorgestellt, die Implementierungen der anderen erfolgen analog und sind im Anhang einzusehen. Die vierte Abfrage der Benchmark bewirkt Folgendes:

Mit Hilfe dieser Anfrage soll überprüft werden, wie gut das Auftragsprioritätensystem funktioniert. Zusätzlich liefert sie eine Einschätzung über die Zufriedenstellung der Kunden. Dazu zählt die Anfrage die Aufträge im dritten Quartal 1993, bei denen wenigstens eine Auftragsposition nach dem zugesagten Liefertermin zugestellt wurde. Die Ausgabeliste soll die Anzahl dieser Aufträge je Priorität sortiert in aufsteigender Reihenfolge enthalten.[3, S. 717]

In SQL ausgedrückt sieht das so aus:

¹<http://hyper-db.de/interface.html>

Table Name	Cardinality	Length (in bytes)	Typical Table
SUPPLIER	10,000	159	2
PART	200,000	155	30
PARTSUPP	800,000	144	110
CUSTOMER	150,000	179	26
ORDERS	1,500,000	104	149
LINEITEM	6,001,215	112	641
NATION	25	128	<1
REGION	5	124	<1
Total	8,661,245		956

Abbildung 2.3.: Größe der Relationen bei Faktor SF=1 [24]

```

select
    o_orderpriority ,
    count(*) as order_count
from
    orders
where
    o_orderdate >= date '1993-07-01'
    and o_orderdate < date '1993-10-01'
    and exists (
        select
            *
        from
            lineitem
        where
            l_orderkey = o_orderkey
            and l_commitdate < l_receiptdate
    )
group by
    o_orderpriority
order by
    o_orderpriority

```

Um die Anfragen in Skripte zu übersetzen, hilft die Orientierung am Abfrageplan (siehe Abb. 2.4). So erhält man einerseits den Ausdruck der relationalen Algebra dafür und eine Schritt-für-Schritt-Übersetzung ist möglich. Außerdem sind so die Ergebnisse besser vergleichbar. Die Tabellen liegen als CSV-Dateien mit Trennzeichen “|” in <tabellenname>.tbl vor, die Kopfzeilen analog in <tabellenname>.csv.

In diesem Fall müssen zuerst die Tabelle *orders* und *lineitem* nach den entsprechenden Tupeln gefiltert werden ($o_orderdate \geq \text{date '1993-07-01'}$ and $o_orderdate < \text{date '1993-10-01'}$ und $l_commitdate < l_receiptdate$), in der Shell geschieht dies durch Auswahl der Zeilen mittels *awk*.

```

sort -k1,1 -t\| orders.tbl | cat orders.csv - | awk -F\| '
    NR==1 || $5<"1993-10-01" && $5>="1993-07-01" {
        print $1"|"$6
    }
' > tmporder.csv

```

2. Bash statt SQL

```
sort -k1,1 -t\| lineitem.tbl | cat lineitem.csv - | awk -F\| '
    NR==1 || $12<$13{
        print $1
    }
' | uniq \|
```

Das anschließende *exists* in SQL wird durch einen linken äußeren Join verwirklicht. Im Gegensatz zu einem Join soll nur auf Existenz überprüft werden, deshalb hilft das Unix-Kommando *uniq* aus, um Duplikate zu eliminieren.

```
join --header -t\| -1 1 -2 1 tmporder.csv - > tmp.csv
```

Im nächsten Schritt folgt die Gruppierung (*group by*) auf die Spalte *o_orderpriority*, die einfach durch *uniq* erfolgen kann, bei komplizierteren Aggregatsfunktionen (*min*, *max*, *sum*, *count*) ist es oft einfacher mit *awk* zu hantieren.

```
head -1 tmp.csv > tmp1.csv
tail -n+2 tmp.csv | sort -t\| -k2,2 | cat tmp1.csv - | awk -F\| '
    NR==1{print $2, "order_count"}
    NR==2{g2=$2; count=1}
    NR>2{
        if( g2==$2 ){
            count++
        }else{
            print g2, count;
            g2=$2;count=1;
        }
    }
    END{print g2,count}
' OFS=\\|
```

Für den Fall der vierten TPC-H-Abfrage kann aber auch *uniq -c* benutzt werden, der neben der Gruppierung auch die Anzahl der Vorkommnisse aller Tupel mit ausgibt, dafür müssen wir uns zunächst auf die relevanten Spalten beschränken (mit *cut*) und alle Zeilen auch sortieren, damit der letztere Befehl ordentlich funktioniert.

```
cut -d\| -f2 | tail -n+2 | sort | uniq -c
```

Und schon ist das Skript fertig, beide Versionen liefern die gewünschte Ausgabe:

o_orderpriority order_count	
1-URGENT 10594	10594 1-URGENT
2-HIGH 10476	10476 2-HIGH
3-MEDIUM 10410	10410 3-MEDIUM
4-NOT SPECIFIED 10556	10556 4-NOT SPECIFIED
5-LOW 10487	10487 5-LOW

Die Implementierung aller weiteren TPC-H-Abfragen erfolgte analog.

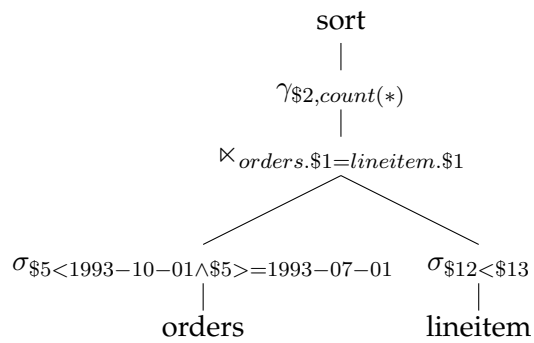
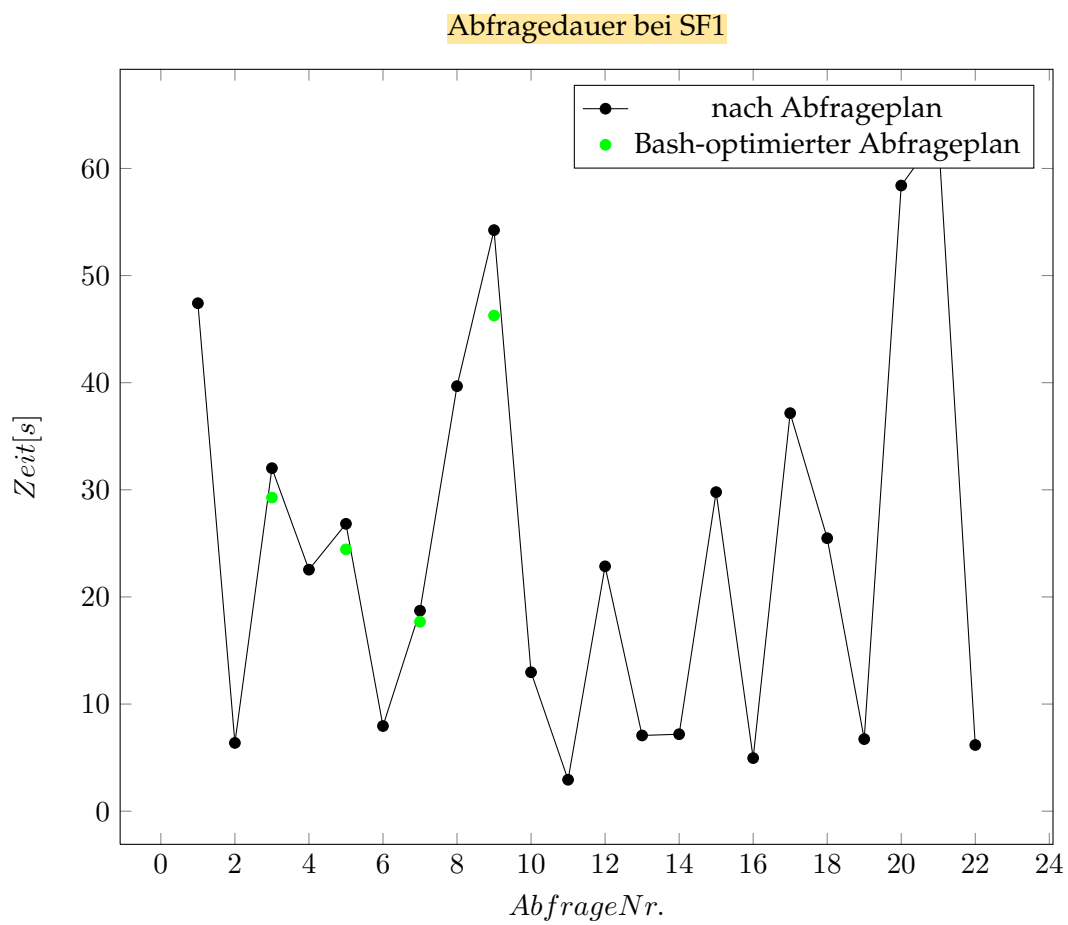
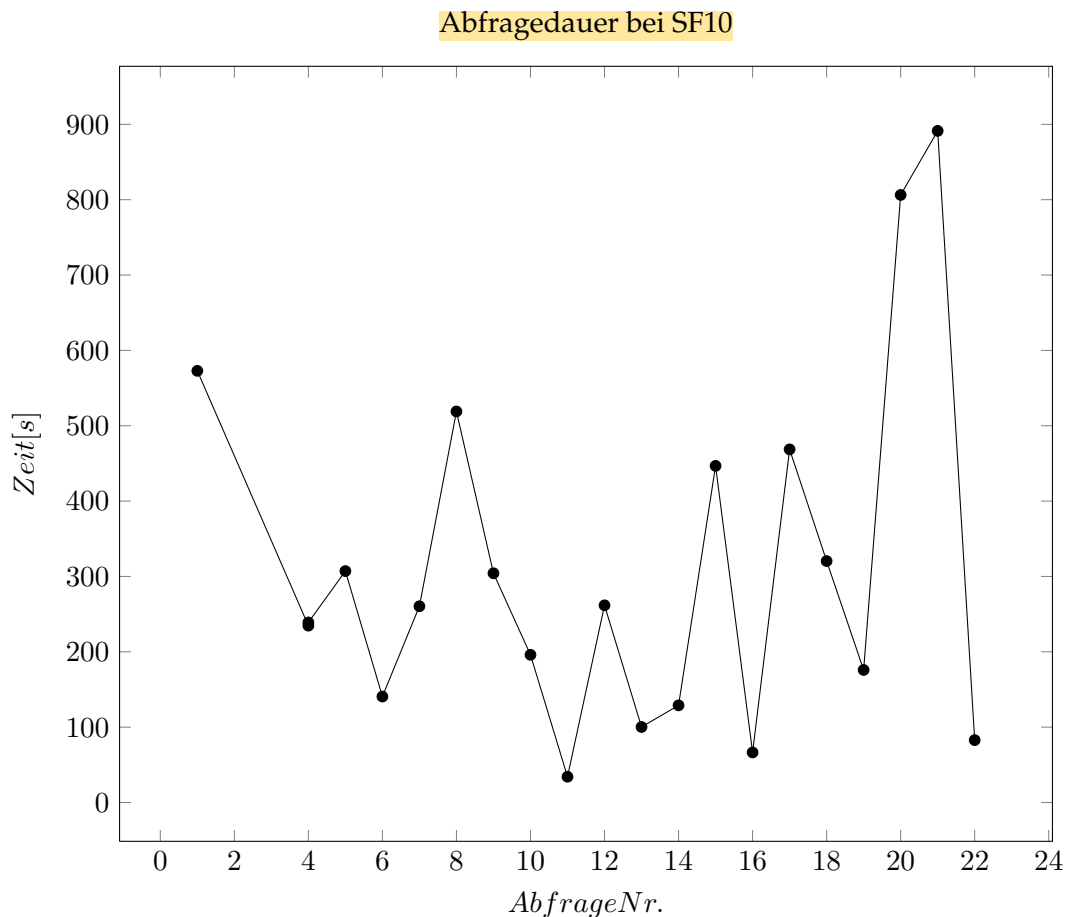


Abbildung 2.4.: Abfrageplan zu Nummer 4





2.2.3. Optimierung durch Parallelisierung

Ein kurzer Blick auf die laufenden Prozesse bestätigt einen Verdacht: die Abfragen laufen teils parallel, es werden so viele Prozesse gestartet wie Kommandos in einer Pipeline verwendet. Von den meisten heutigen Bash-Philosophen wird dies verschwiegen, daher lohnt sich ein Blick in das Buch von Pike und des awk-Erfinders Kernighan.

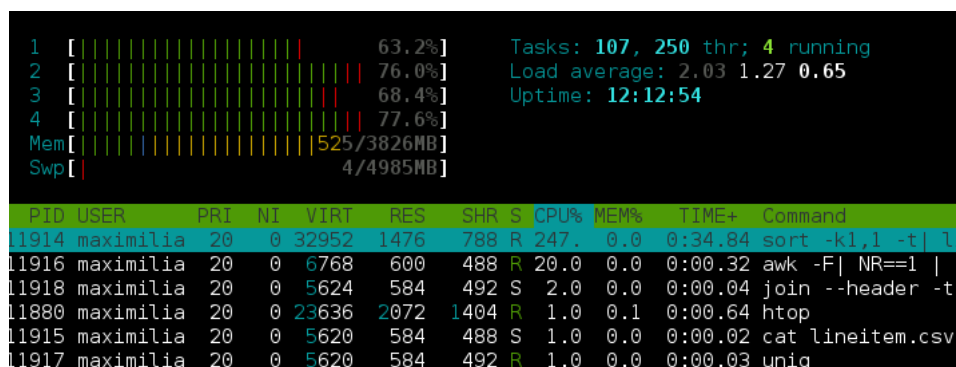


Abbildung 2.5.: Parallele Ausführung mittels Pipe verbundener Prozesse

Die Programme in der Pipeline werden in Wirklichkeit gleichzeitig ausgeführt, nicht nacheinander. Das bedeutet, da[ss] die Programme in einer Pipeline interaktiv arbeiten können; der Betriebssystem-Kern sorgt für die nötige Zuteilung von Rechenzeit und Synchronisation, damit alles funktioniert.[5, S. 34]

Das Ziel ist es also, eine möglichst hohe Durchlaufquote durch die Verwendung von Pipelines zu erzielen. Jedoch geht dies nur bis zu einem gewissen Grad, die zwangsweise Sortierung der Datenbestände vor Joins macht die Aufteilung in einen Kopf und einen Rumpf erforderlich, der Kopf bleibt unverändert, der Rumpf wird sortiert und alles in eine Datei umgeleitet oder mit *cat* wieder zusammengefügt.

```
head -1 tmpsns.csv > tmp1.csv
tail -n+2 tmpsns.csv | sort -t\\ -k2,2 >> tmp1.csv
head -1 join5.csv > tmp2.csv
tail -n+2 join5.csv | sort -t\\ -k2,2 | cat tmp2.csv - | \
join --header -t\\ -1 2 -2 2 tmp1.csv - | \
```

Versuche, die Kommandos in eine Pipeline zu packen, schlagen kläglich fehl. Jetzt wäre es doch praktisch, ein alternatives Konstrukt zu nutzen, sodass die Verarbeitung parallel ablaufen kann. Benannte Pipes sind die Lösung, sie können einfach erstellt werden.

```
$ mkfifo mypipe
```

Wichtig ist, dass die Prozesse, die in die Pipe hineinschreiben und aus ihr herauslesen, parallel ablaufen. Die Pipe blockiert so lange, bis aus ihr gelesen, bzw. in sie geschrieben worden ist.

Ein einfaches Beispiel, das bereits 13% Zeit einspart und zeitlich fast an eine Pipeline herankommt, zuerst über eine Pipeline verbunden:

```
$ sort orders.tbl | grep final benötigt 2,752 s
```

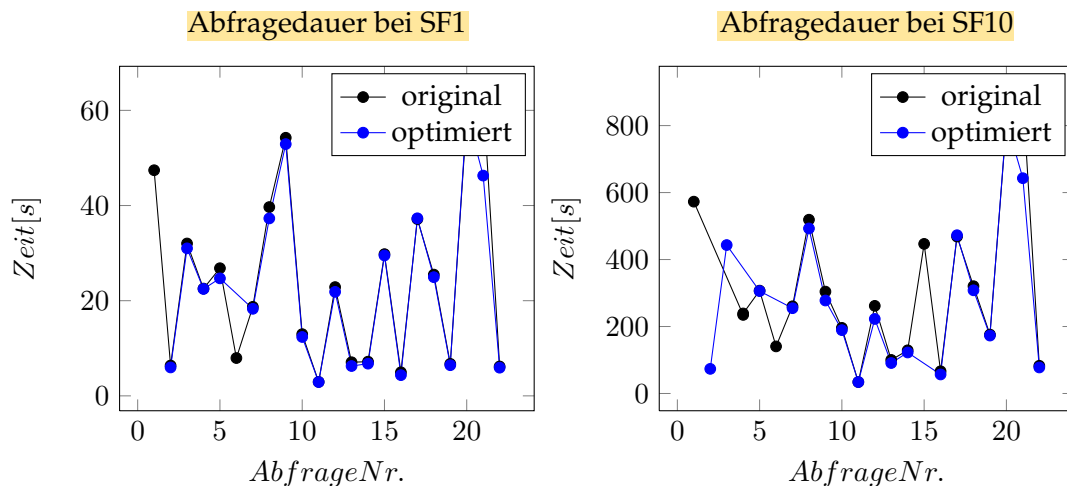
Ohne die Pipeline können die Prozesse über eine Datei kommunizieren oder über eine benannte Pipe:

<pre>#!/bin/bash sort \$1 > tmpfile grep final < tmpfile</pre>	<pre>#!/bin/bash mkfifo mypipe (sort \$1 > mypipe)& grep final < mypipe</pre>
--	---

Mit Umlenkung in eine Datei werden 3,165 s gebraucht, die zweite Variante schlägt die erste mit nur 2,760 s und liegt damit dicht an einer Pipeline. Darüberhinaus ist es von Vorteil, dass ein Prozess im Hintergrund und somit auch parallel läuft. Und synchronisiert sind sie auch, sie müssen nicht explizit auf ihren Vorgängerprozess warten. Solange in die Pipe noch nicht geschrieben ist, wird der andere Prozess blockiert. Lediglich die Reihenfolge ist wichtig. Auf diese Weise werden die TPC-H-Anfragen nun optimiert: Zuerst wird die Kopfzeile geschrieben, dann der Rumpf.

```
head -1 tmpsns.csv > tmp.csv &
tail -n+2 tmpsns.csv | sort -t\\ -k2,2 | cat tmp.csv - > tmp1.csv &
head -1 join5.csv > tmp2.csv &
tail -n+2 join5.csv | sort -t\\ -k2,2 | cat tmp2.csv - | \
join --header -t\\ -1 2 -2 2 tmp1.csv -
```

Analog dazu werden nun alle Anfragen optimiert und die neue Zeit gemessen.



Bei fast allen Abfragen konnten leichte Verbesserungen erzielt werden, bei den meisten zwar nur unerheblich, aber bei der vorletzten Abfrage ist deutlich zu sehen, was bisschen Parallelisierung bewirkt, sie läuft um 28 % schneller. Eine weitere Möglichkeit Parallelität zu erreichen ist die Prozesse mit *split* aufzuteilen. Dieser Befehl erzeugt aus einer Datei mehrere kleine aufsteigend benannte Dateien (beginnend bei *aa* bis *zz*, auch mit benannten Pipes kompatibel), auf denen die Anfragen laufen. Das Zusammenfügen passiert am Ende mit *cat*. Bei kleinen Anfragen dauert das Teilen länger als was die Parallelität einspart.

```
$ grep final lineitem.tbl
```

Das Skript nun mit geteilter Quelldatei und Ausführung der Prozesse im Hintergrund.

```
#!/bin/bash
mkfifo mini.aa mini.ab g1 g2
(split -n2 lineitem.tbl mini.)&
(grep final mini.aa > g1)&
(grep final mini.ab > g2)&
cat g1 g2

rm mini* g1 g2
```

Das Skript verdeutlicht die Parallelität durch Teilen, die getestete Ausführung benötigt jedoch 0,2 s länger (2,8 s statt 2,6 s).

Komplizierter wird der Syntax, wenn die Parallelisierung über GNU Parallel läuft. Dieses bietet zahlreiche Funktionen an, die aber über den Inhalt dieses Kapitels - einfache Skripte wie sie in der Wissenschaft vorkommen zu erstellen - hinaus gehen. Zusammen mit nachfolgendem Beispiel sei aber auf die Literatur verwiesen [14].

```
$ seq 4 | parallel "echo_{}"
1
2
3
4
```


3. Vergleich

Welche Ansätze existieren bereits, um Abfragen an CSV-Dateien auszuführen? Grob lassen sich diese in zwei Kategorien unterteilen: Programme, die mit ihrer eigenen Syntax SQL-ähnliche Funktionen entsprechen, und solche, die SQL-Abfragen ohne Datenbank auf Textdateien ausführen.

3.1. SQL-ähnlicher Syntax

3.1.1. Datamash

Ein wunderbares Programm für die Analyse auf Textdateien ist *datamash*,¹ das numerische Operationen auf Datensätze ausführen kann und dabei auch die meisten Operationen des γ -Operators beherrscht. Das Programm ist in C geschrieben, arbeitet ausschließlich als Filter für DSV-Dateien (Trennzeichen standardmäßig Tab) und erwartet in den Optionen die Spalten nach denen gruppiert wird und als Argument die auszuführende Aktion.

```
$ cat myexample
A      1
A      3
B      2
$ cat myexample | datamash sum 2
6
$ datamash -g1 sum 2
A      4
B      2
```

Um das Programm in die Skripte der TPC-H-Abfragen einzubinden, soll es Kopfzeilen erkennen und das Trennzeichen muss variabel bestimmt werden. Beides kann *datamash*, die Option *-headers* oder *-H* weist die erste Zeile als Kopf aus und mit *-field-separator=*, oder *-t*, wird das Komma „*,*“ als Separator bestimmt.

```
$ cat myexample
Name,Zahlen
A,1
A,3
B,2
$ datamash -H -t, -g1 sum 2
Name,sum(Zahlen)
A,4
B,2
```

Sieht das Gruppieren und anschließende Summieren, Zählen, Mittelwertbilden kompliziert aus, so dient jetzt eine einzige Anfrage dazu, die Option *-s* erspart einem sogar das vorherige Sortieren, nachfolgend die angepasste erste TPC-H Abfrage (erste Berechnung weiterhin mit *awk*).

¹<http://www.gnu.org/software/datamash/>

3. Vergleich

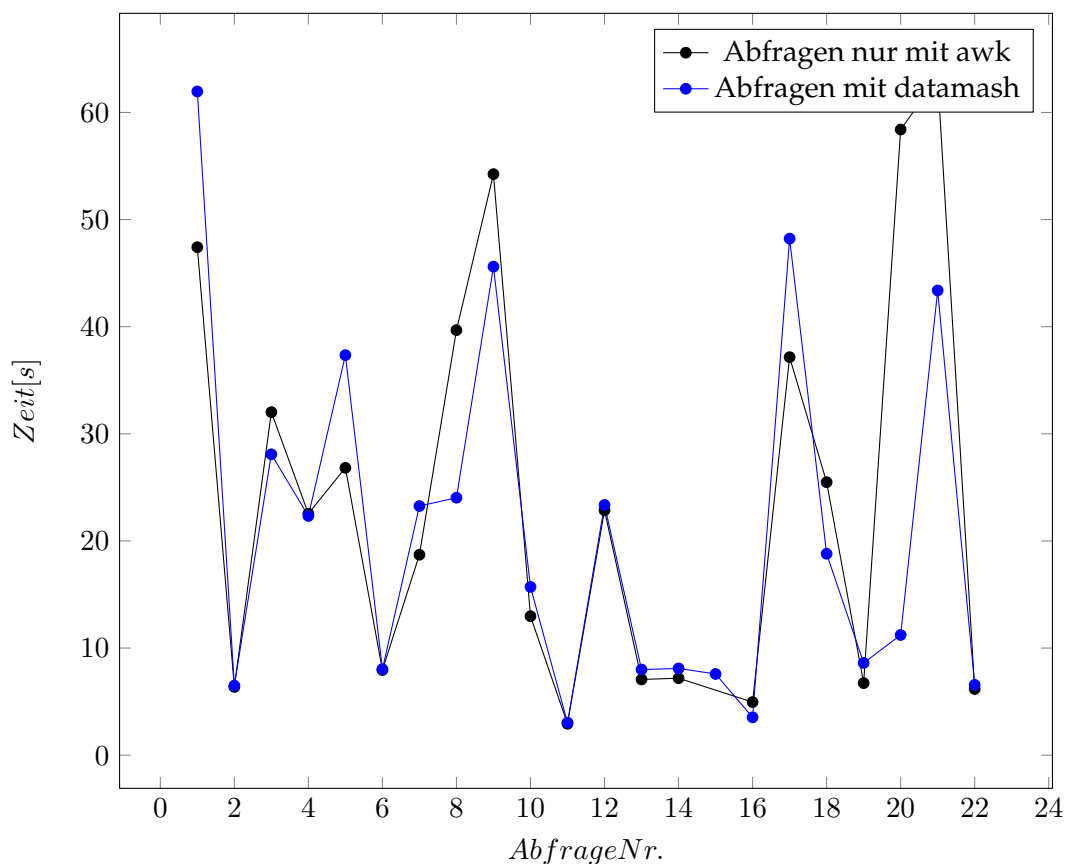
```
#!/bin/bash
#2014-09-01
cat lineitem.* | awk -F\| '
    NR==1{
        print $0 "sum_disc|sum_charge|"
    }
    NR>1 && $11<="1998-9-2" {
        suma=($6*(1.0-$7));
        sumb=($6*(1.0-$7)*(1.0+$8));
        print $0 suma "|" sumb "|"
    }
' > tmp.csv
head -1 tmp.csv > tmp1.csv
tail -n+2 tmp.csv | sort -t\| -k9,10 | cat tmp1.csv - |\
datamash -t\| -H -g9,10 sum 5 sum 6 sum 17 sum 18 mean 5 mean 6 mean 7 count 1
```

Zu beachten ist, dass datamash Dezimalzahlen im länderspezifischen Format ausgibt und auch solche in der Eingabe benötigt. Sind die Datensätze auf ein anderes Format ausgelegt, so sind meist "." und "," ganz einfach durch den Befehl *tr* zu vertauschen.

```
$ cat datei | tr '.,' ',.'
```

Auf diese Weise werden die Skripte mit datamash geschrieben und auch wieder die Zeit gemessen.

Abfragedauer bei SF1



In den meisten Fällen sind die umgeschriebenen Skripte langsamer, in manchen aber auch deutlich schneller. Der Vorteil an den Skripten, die ausschließlich awk nutzen, ist, dass Berechnungen und das Gruppieren in einem Schritt erfolgen kann. *datamash* unterstützt nur Aggregatsfunktionen und keine arithmetischen Ausdrücke, sodass für Berechnungen ein weiteres Programm benötigt wird. Der Vorteil von *datamash* liegt aber in der Einfachheit, mit der Aggregatsfunktionen eingegeben werden können. Auf der Seite zu *datamash* sind auch äquivalente Kommandos mit anderen Programmen aufgelistet.²

Fazit: *datamash* ist ein sehr schönes Programm, das einem die Arbeit erleichtern kann, der kompliziertere Umgang mit awk kann aber zu schnelleren Abfragen führen.

3.1.2. csvtool

Das in OCaml geschriebene *csvtool* ist mittlerweile in allen Linux-Paketverwaltungssystemen erhältlich und richtet sich an Nutzer mit komplizierteren CSV-Datensätzen.³ Im Grunde ersetzt das *csvtool* die klassischen Unix-Werkzeuge wie *grep*, *cut*, *head*, *tail*, *join*, *wc -l*, usw., jedoch kann es auch problemlos mit Spezialfällen hantieren, in denen sich ein Datensatz über mehrere Zeilen erstreckt. Die Selektion heißt dort *namedcol*.

```
$ csvtool namedcol ID,Name Punktetabelle.csv
1,Franz
2,Alfred
3,Marie
```

Für Joins ist *csvtool* nur bedingt geeignet, der Syntax ist alles andere als intuitiv.

```
$ csvtool join <Joinattribute> <Tabellennummer> Tabelle1 Tabelle2
```

Bei *Joinattribute* werden beginnend bei 0 die zu vergleichenden Spalten eingetragen, bei *Tabellennummer* als Liste die Spalten, die später sichtbar sein sollen. Dabei bezieht sich die Auswahl aber immer auf beide Tabellen, folglich müssen *Joinattribute* an der gleichen Position stehen.

```
$ csvtool join 1 2,3 Punktetabelle.csv Zeittabelle.csv
1,Franz,50,44,
2,Alfred,10,88,
3,Marie,27,67,
ID,Name,Punkte,Zeit,
```

Für relationale Algebra ist das Programm weniger gedacht als für einfach Vergleiche von CSV-Datensätzen. Für komplizierte Abfragen empfehlen sich mächtigere Programme, die in den meisten Fällen auch SQL-Ausdrücke beherrschen und deshalb noch vorgestellt werden.

3.1.3. Fsdb

Seit 1991 existiert *Fsdb* als eine textdateibasierte Datenbank (flat text database for shell scripting), die jedoch ihrem eigenen Datenformat gehorcht und Operationen ähnlich zu *datamash* beherrscht.⁴

²<http://www.gnu.org/software/datamash/alternatives/>

³<https://forge.ocamlcore.org/projects/csv/>

⁴<http://www.isi.edu/~johnh/SOFTWARE/FSDB/>

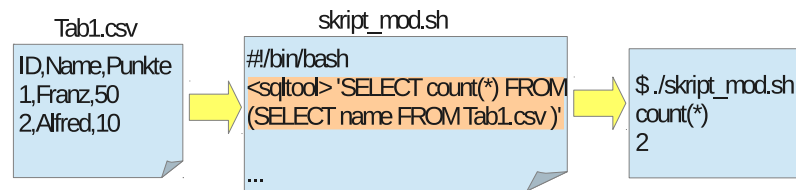


Abbildung 3.1.: Funktionsweise von SQL-Tools in der Kommandozeile

Die Textdateien enden auf *.fsdb*, am Anfang steht eine Kopfzeile, gekennzeichnet durch *"#fsdb"*, mit den Spaltenbezeichnern, anschließend die durch Leerräume separierten Werte. Das Feldtrennzeichen kann mit *-F* verändert werden, Kommentare beginnen mit *"#"*, mit dem Befehl **csv_to_db** können CSV-Dateien in das Format konvertiert werden.

```
$ cat Punktetabelle.fsdb
#ID Name Punkte
1 Franz 50
2 Alfred 10
3 Marie 27
$ cat Punktetabelle.fsdb | dbcol ID Name Punkte
#fsdb ID Name Punkte
1 Franz 50
2 Alfred 10
3 Marie 27
# | dbcol ID Name Punkte
```

Die Kommandos von *Fsdb* arbeiten als Filter, sie lesen also von der Standardeingabe Daten ein, die immer im entsprechenden Format stehen müssen. Insgesamt eine nette Idee, das Erlernen der vielen verschiedenen Kommandos nur eher unintuitiv.

3.2. Abfragen mit SQL

3.2.1. txt-sushi

Ein geniale Sammlung von Programmen, um auf CSV-Dateien mit SQL-Syntax zuzugreifen ist **txt-sushi**. Einmal installiert, so bietet es neben zahlreichen weiteren Funktionen, die die Ausgabe von Textdateien verschönern, auch ein Programm **tssql** an, mit dem sich in SQL formulierte Abfragen auf CSV-Dateien ausführen lassen.⁵

```
$ tssql 'select * from 'orders.csv''
O_Id,OrderDate,OrderPrice,Customer
1,2008/11/12,1000,Hansen
2,2008/10/23,1600,Nilsen
3,2008/09/02,700,Hansen
4,2008/09/03,300,Hansen
5,2008/08/30,2000,Jensen
6,2008/10/04,100,Nilsen
```

⁵<http://keithsheppard.name/txt-sushi/>

Das Programm arbeitet auch als Filter und kann von der Standardausgabe lesen, die Daten können über eine Pipe weitergereicht werden. Generell gilt, alle Daten, die eingelesen werden sollen, werden mit zwei Gravis (back tick, „`) geklammert.

```
$ cat orders.csv | tssql 'select * from `-'`
O_Id,OrderDate,OrderPrice,Customer
1,2008/11/12,1000,Hansen
2,2008/10/23,1600,Nilsen
...
```

Noch einfacher können die Tabellen gleich über Optionen eingebunden werden, dort werden auch Bezeichner für die Dateien definiert.

```
$ tssql -table orders orders.csv 'select * from orders'
```

Leider akzeptiert das Programm nur durch Komma separierte Dateien, wird ein anderes Symbol als Trennzeichen verwendet, so muss das über eine Umlenkung erfolgen.

```
$ tssql -table orders <(sed 's/;/,/g' orders_mit_Semikolon.csv) 'select * from
orders'
```

Probleme bereitet dies, wenn Kommata Bestandteil der Datensätze sind. Am einfachsten ist es, den Programmcode so umzuschreiben, sodass ein anderes Zeichen die Daten voneinander trennt. (Die Quelldateien finden sich auf Github, zu verändern ist die Datei *txt-sushi-master/Database/TxtSushi/FlatFile.hs*)⁶

So kann auch ein einfaches Skript geschrieben werden, um größere SQL-Abfragen gemäß dem TPC-H Schema einzugeben.

```
#!/bin/bash
TSSQL=/home/maximilian/Downloads/txt-sushi-master/tssql
$TSSQL -table lineitem <(cat lineitem.*) '
SELECT
    l_returnflag ,
    l_linestatus
FROM
    lineitem
GROUP BY
    l_returnflag ,
    l_linestatus
,
```

Bei der Ausführung selbst einfacher Abfragen wie die gerade beschriebenen stellt man schnell fest, dass der Arbeitsspeicher den limitierenden Faktor darstellt. Sobald größere Operationen durchgeführt werden, die über eine Selektion hinausgehen, so lädt das Haskell-Programm alle Datensätze in den Hauptspeicher. Das beschriebene Skript wird nach einiger Zeit durch das System beendet. Für große Datenmengen ist das Programm nicht geeignet, obwohl es in der Bedienung und von der Funktion einzigartig gut und praktisch ist.

3.2.2. csvfix

Die meisten Datenbanksysteme unterstützen den Import von CSV-Dateien, aber leider nicht alle. Besonders im mobilen Bereich, wo Anwender auf gewisse Datenbanken nur

⁶<https://github.com/keithshep/txt-sushi>

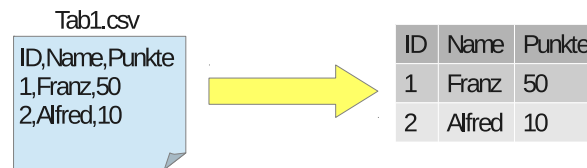


Abbildung 3.2.: Import von CSV

über eine SQL-Schnittstelle zugreifen können, sind SQL-Anweisungen zum Einfügen von Daten von Vorteil (s. Abb. 3.2). Dagegen hilft `csvfix`, das aus gegebenen CSV-Datensätze entsprechend viele INSERT-/UPDATE/- oder DELETE-Anweisungen generiert. Einfach den Tabellennamen, die Spaltenbezeichner und die Quelldatei angeben, schon werden die SQL-Anweisungen erzeugt, mit der Option `-ifn` wird die erste Zeile als Kopfzeile betrachtet.

```
$ cat Punktetabelle.csv
ID,Name,Punkte
1,Franz,50
2,Alfred,10
3,Marie,27
$ csvfix sql_insert -ifn -t punktetabelle -f 1:ID,2:Name,3:Punkte Punktetabelle.csv
INSERT INTO punktetabelle ( ID, Name, Punkte ) VALUES( '1', 'Franz', '50');
INSERT INTO punktetabelle ( ID, Name, Punkte ) VALUES( '2', 'Alfred', '10');
INSERT INTO punktetabelle ( ID, Name, Punkte ) VALUES( '3', 'Marie', '27');
```

Über die Option `-osep` kann noch das Feldtrennzeichen angegeben werden, sollen Felder ohne Anführungszeichen ausgegeben werden, so dient die Option `-nq` dazu.

```
$ csvfix sql_insert -ifn -osep , -t punktetabelle -nq 1,3 -f 1:ID,2:Name,3:
Punkte Punktetabelle.csv
INSERT INTO punktetabelle ( ID, Name, Punkte ) VALUES( 1, 'Franz', 50);
INSERT INTO punktetabelle ( ID, Name, Punkte ) VALUES( 2, 'Alfred', 10);
INSERT INTO punktetabelle ( ID, Name, Punkte ) VALUES( 3, 'Marie', 27);
```

Doch `csvfix` kann mehr, als nur SQL-Befehle generieren, ähnlich dem `csvtool` unterstützt es auch den Verbund mehrerer Tabellen und weitere arithmetische Operationen ähnlich zu `awk`.⁷

```
$ csvfix join -f 1:1 Zeittabelle.csv Punktetabelle.csv
"ID","Zeit","Name","Punkte"
"1","44","Franz","50"
"2","88","Alfred","10"
"3","67","Marie","27"
```

Äquivalenzen werden durch den Doppelpunkt beschrieben (*Feld von i : Feld von $i+1$*), in einem Schritt können mehrere Joins vollzogen werden, es werden einfach mehrere Quelldateien angegeben.

⁷<http://csvfix.byethost5.com/csvfix15/csvfix.html>

Generell gibt es für die meisten benötigten Kommandos ein Äquivalent in csvfix, der Vorteil liegt an den unterstützten Standards, ein Komma als Feldinhalt stellt kein Problem dar. Abgesehen von diesen Spezialfällen reichen auch die klassischen Kommandos aus.

3.2.3. csvkit

Die Funktionen von csvfix und txt-sushi kombiniert bietet das von der Literatur gepriesene csvkit [14]. Es unterstützt zwar auch einfache Operationen um CSV-Dateien auszugeben, ist aber primär auf das Anwenden von SQL-Abfragen auf CSV-Dateien ausgelegt und bietet an, bei größeren Datenmengen die Abfrage über eine Datenbankverbindung auszuführen.⁸

Zuerst einmal ist es fähig, eine Anweisung zum Erstellen von Datenbank-Schemate auszugeben und erkennt dabei selbständig den Typ. Über die Option *-i* ist die Anweisung sogar auf bestimmte Datenbanktypen zugeschnitten.

```
$ csvsql -i postgresql Punktetabelle.csv
CREATE TABLE "Punktetabelle" (
    "ID" INTEGER NOT NULL,
    "Name" VARCHAR(6) NOT NULL,
    "Punkte" INTEGER NOT NULL
);
```

Leider kann das Programm keine Operationen zum Einfügen direkt generieren, dafür dient csvfix. Dafür ist es möglich, über eine Datenbankanbindung direkt die CSV-Dateien zu importieren.

```
$ csvsql --db <Verbindung zur Datenbank> --insert Punktetabelle.csv
```

Aber nun folgt die Option *-query*, um eine SQL-Anweisung direkt auszuführen, mit *-d* kann auch das Trennzeichen bestimmt werden.

```
$ csvsql -d, --query 'select * from Punktetabelle' Punktetabelle.csv
ID,Name,Punkte
1,Franz,50
2,Alfred,10
3,Marie,27
```

Das Programm arbeitet nur auf CSV-Dateien, die auch eine Kopfzeile besitzen und auf die Dateiendung *.csv* hören, alles andere akzeptiert es nicht, auch die TPC-H Abfragen können nicht ohne deren Modifikation durchgeführt werden.

3.2.4. querycsv.py

Das in Python geschriebene Skript querycsv.py arbeitet ähnlich und ist flexibler in der Annahme von Dateinamen.⁹

```
$ alias querycsv='/home/maximilian/Downloads/querycsv-0.3.0.0/querycsv/querycsv.py'
$ querycsv -i Punktetabelle.csv 'Select * From Punktetabelle'
ID | Name | Punkte
=====
```

⁸<https://csvkit.readthedocs.org/en/latest/index.html>

⁹<https://pypi.python.org/pypi/querycsv>

```
1 | Franz | 50
2 | Alfred | 10
3 | Marie | 27
```

Dank eines CSV-Sniffers erkennt das Programm automatisch das Feldtrennzeichen, leider sind die möglichen Trennzeichen dadurch auch beschränkt, eine Pipe zum Trennen wird nicht unterstützt. Ohne Modifikation der CSV-Dateien ist auch dieses Programm für die TPC-H Abfragen nicht zu nutzen.

```
dialect = csv.Sniffer().sniff(open(infile, "rt").readline())
```

Die Endung der Dateien auf *.tbl* erkennt das Skript mühelos, der Name zuvor wird als Tabellenbezeichner verwendet, nur werden keine Daten ohne Kopfzeile akzeptiert, wie auch durch Pipe getrennte Dateien.

Fazit: Ein sehr schönes Skript, aber Mangel an Flexibilität.

3.2.5. gcsvsql

Ein ähnliches Programm wie csvsql, heißt gcsvsql, da es auf Groovy basiert, einer objekt-orientierten Skriptsprache und der Name csvsql schon vergeben ist.¹⁰ Die Abfrage wird einfach als SQL-Anweisung eingegeben, als Tabelle steht der Name der CSV-Datei.

```
$ gcsvsql "select *_from_Punktetabelle.csv"
```

Wegen der starren Syntax und einer seltenen Programmiersprache erweist sich die Anwendung des Programms als schwierig.

3.2.6. Mynodbcsv

Einen ähnlichen Ansatz wie txt-sushi nur noch optimierter und einen Schritt weiter geht das Konzept Mynodbcsv, das vor Kurzem veröffentlicht worden ist. Insgesamt geht es um das Problem, wie sehr große Datenmengen aus CSV-Dateien mit SQL-Anweisungen abgefragt werden, ohne dabei die kompletten Daten in eine Datenbank zu importieren und auch kein Datenbankschema vorher konfigurieren zu müssen.

Das Konzept analysiert anhand einer optimierten Abfrage zuerst die relevanten Felder einer CSV-Datei, anschließend lädt es nur die relevanten Daten in eine SQLite-Datenbank, worauf die Abfrage laufen kann. Somit geht dieses Konzept einen Mittelweg, es müssen nie alle Daten in eine Datenbank importiert werden, trotzdem erfolgt die Abfrage mittels eines Datenbanksystems. Für die genauen Messergebnisse sei auf die Veröffentlichung verwiesen.^[2]

3.2.7. shql

Ähnlich der geschriebenen Abfragen als TPC-H Skripte implementiert shql eine Datenbank, die ausschließlich auf den Unix-Kommandos basiert und SQL über awk-Funktionen interpretiert.¹¹

¹⁰<https://code.google.com/p/gcsvsql/>

¹¹<http://lorance.freeshell.org/shql/>

4. Parser mit Yacc und Lex

Wie bereits gesehen, Programme, die eine CSV-Datei parsen und diese in SQL importieren, gibt es bereits genügend, genauso wie die Möglichkeit da ist, auf Textdateien SQL-Abfragen laufen zu lassen. Außerdem ist auch analysiert worden, welche Bash-Konstrukte bestimmten SQL-Abfragen entsprechen. Darum nun der abschließende Teil der Arbeit - wie können Shell-Skripte geparkt und in SQL übersetzt werden.

4.1. Vorwissen zu Yacc und Lex

Der erste Versuch verwendet Yacc, ein Programm, das Anfang der 1970-er Jahre von Stephen Curtis Johnson unter dem Namen "yet another compiler compiler" für den Compilerbau erfunden und im GNU-Projekt durch Bison ersetzt worden ist. Die Grammatik wird in einer Art Backus-Naur-Form eingegeben, der Parsergenerator Yacc erzeugt dann automatisch ein C-Programm, dass von einer entsprechenden Eingabe liest.^[16]

Wie in Grammatiken üblich, existieren Terminalsymbole, die nicht mehr ersetzt werden, also den endgültigen Symbolen (bei einem Taschenrechner entspricht das den endgültigen Ziffern und Operatoren) und Nichtterminale, also Symbole, die durch Terminale nach bestimmten Regeln ersetzt werden. Um zu wissen, welche Terminalsymbole verwendet werden, sind zwei Schritte notwendig, zuerst die lexikalische Analyse (welche Symbole kommen vor), die durch die Funktion `yylex()` aufgerufen wird und ein Token mit dem gelesenen Symbol zurückgibt, und dem eigentlichen Parser, durch `yyparse()` aufgerufen, der anhand der Bausteine die Syntax analysiert und erkennt welche Grammatikregel angewendet wird. Die Yacc-Quelldatei folgt dabei folgendem Schema:

```
%{
C-Kopf
}%
Yacc-Deklarationen
%%
Grammatiken
%%
/*C-Funktionen , z.B. yylex() */

int main() {
    reset(queries);
    yyparse();
}
```

Für die lexikalische Analyse gibt es zwei Möglichkeiten, entweder man schreibt die Funktion manuell an das Ende oder man nutzt einfach das Programm Lex (oder in der GNU-Version flex), das einem die lexikalische Analyse übernimmt und die Funktion `yylex()` automatisch erzeugt. Dazu wird in einer Quelldatei `datei.lex` angegeben, welche Token

bei welchen Symbolen zurückgegeben werden:[16]

```
[0-9]+ { yylval = atoi(yytext);  
        return NUMBER; }
```

Die Kommandos *Lex* und *Yacc* erzeugen, in der Kommandozeile aufgerufen, beide jeweils eine C-Datei (lex.yy.c und y.tab.c), die zum fertigen Parser kombiniert werden.

```
yacc -d parser.y  
lex parser.lex  
cc -o meinparser lex.yy.c y.tab.c -ll -lm
```

Ein einfaches funktionierendes Beispiel findet sich dazu im Anhang.

4.2. Der Bash2SQL-Übersetzer mit Yacc

Das Ziel des Bash2SQL-Übersetzer ist es, ein erstes Erfolgserlebnis zu schaffen und zu zeigen, dass der Weg, ein Shell-Skript in eine SQL-Abfrage zu übersetzen, auch schaffbar ist. Darum behandelt dieser Parser die grundlegende Idee mit vereinfachter Grammatik, die komplette Bash-Grammatik kann erst mit dem nächsten Parser eingelesen werden. Folglich geht es darum, die Befehle richtig zu übersetzen, also sind zwei Punkte wichtig: Wie wird die SQL-Befehl intern repräsentiert und wie wird ein Kommando geparkt und übersetzt.

4.2.1. Der Quellcode

Zuerst zum späteren SQL-Befehl, er wird intern als ein einzelner struct abgespeichert, der in Abhängigkeit der eingelesenen Befehle verändert wird. Der struct selbst besitzt nur einige Attribute, ein Feld, um die selektierten Felder als Zahl zu speichern (analog zum Befehl cut und beginnend bei \$1, \$2, usw.) und einen Zeiger auf die nachfolgende Abfrage.

```
struct myquery{  
    char select[80];  
    char from[80];  
    char where[800];  
    char groupby[20];  
    char orderby[20];  
    char as[20];  
    int felder[MAXFIELDS];  
  
    struct myquery *next;  
}
```

Da durch das Skript allein nicht zu ermitteln ist, wieviele Spalten die gewählte Tabelle besitzt, wird vorher der Wert *MAXFIELDS* auf eine entsprechend hohe Zahl gesetzt, der die maximale Anzahl an Feldern definiert

4.2.2. Der Parser

Ganz am Anfang erfolgen die Yacc-Deklarationen, also welche Token existieren, und es wird definiert, welche Werte sie zurückgeben. Da nicht nur Strings, sondern manchmal auch Zahlen erwünscht sind, sollen beide Werte möglich sein, die Union-Definition in

Kombination mit `%type` macht es möglich und definiert den Rückgabewert `yylval` als Struktur mit verschiedenen Werten.

```
%union {
    long int4;
    int int2;
    float fp;
    char *str;
};

%type <str> WORD
%type <int4> NUMBER
%type <str> OPT
```

Die Typen zu `WORD`, `OPT` und `NUMBER` sind definiert (zu den Grammatiken, die gleich folgen), jetzt fehlen noch die Deklarationen der Token:

```
%token REDIR;
%token NUMBER ALNUM WORD OPT FSTDIN
%token PLUS MINUS TIMES DIVIDE POWER
%token OPT_D
%token CAT CUT GREP SORT
%token PIPE
%token LEFT_PARENTHESIS RIGHT_PARENTHESIS
%token END
```

Hinzu kann noch die Bindung und die Priorität der Token festgelegt werden, ersteres durch explizite Angabe, zweiteres implizit durch die Reihenfolge (das letzte bindet am stärksten). Außerdem definiert `%start` noch explizit das Startsymbol, also die erste anzuwendende Regel der Grammatik, im Folgenden *Input*:

```
%left PIPE
%left PLUS MINUS
%left TIMES DIVIDE
%left NEG
%right POWER
```

```
%start Input
```

Im dritten Teil können mit den Token Regeln für die Grammatik angegeben werden. Beim Parsen des Kommandos geht es zuerst um die richtige Interpretation der Shell-Kommandos, daher wird zuerst nur eine einfache Shell-Grammatik implementiert. Sie kann aus mehreren Zeilen bestehen und beherrscht auch die Konkatenation mehrerer Kommandos und die Umleitung in eine Datei.

```
Input:
    /* Empty */
    | Input Line
    ;

Line:
    END
    | Command END {
        ausgeben(&queries[crtq++]);
        reset(&queries[crtq]);
    }
    | Command PIPE Line
    | Command REDIR WORD END { strcpy(queries[crtq].as,$3);
```

```

        ausgeben(&queries [ crtq ++]);
        reset(&queries [ crtq ]) ;
    ;

```

Also wie sollen jetzt die Kommandos übersetzt und verbunden werden? Die erste Überlegung ist es, alles in einer SQL-Abfrage abzuhandeln, bis es nicht mehr geht, also ein *cat* *tabelle* füttert den From-Teil, ein über Pipe verbundenes *wc -l* füttert die Select-Klausel desselben structs mit einem *count(*)*, am Ende kann die Abfrage ausgegeben werden, also wie folgt:

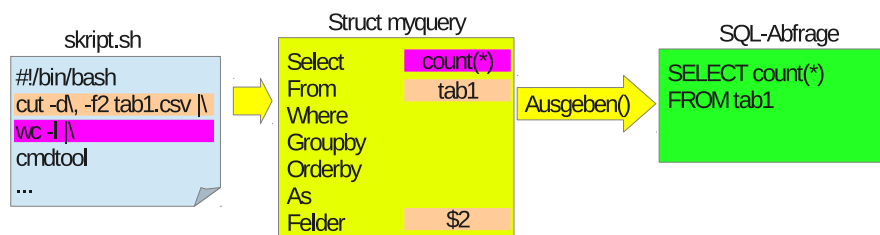


Abbildung 4.1.: Arbeitsweise des Yacc Bash2SQL-Parsers

Die Idee ist jetzt, für jedes Kommando eine Regel zu erstellen, die in Abhängigkeit des Kommandos (für jedes Kommando ein Token) die Abfrage anders modifiziert. Außerdem können zu den Kommandos noch Optionen angegeben werden, die beachtet werden müssen, sowie die Eingabedatei, die eingelesen werden soll. Eine Regel der Grammatik besteht immer zuerst aus dem Kommando und dann weiteren Parametern:

<Kommando> [Optionen] Datei [Optionen]

In dieser Form werden nun die Kommandos geparkt, vorläufig werden die Befehle *cat*, *cut*, *grep* und *sort* betrachtet.

Command:

```

    CAT
    | CAT WORD                                { reset(&queries [ crtq ]) ;
                                              strcpy ( queries [ crtq ]. from , $2) ; }
    | CUT OptionsCut WORD                     { strcpy ( queries [ crtq ]. from , $3) ; }

    | CUT OptionsCut
    | GREP WORD                                { zugrep(&queries [ crtq ], $2) ; }
    | GREP WORD WORD                           { strcpy ( queries [ crtq ]. from , $3) ;
                                              zugrep(&queries [ crtq ], $2) ; }

    | SORT OptionsSort
    ;

```

Die Aktionen werden entsprechend ausgeführt:

- **cat [dateiname]**: der Dateiname ist die gewünschte CSV-Datei und entspricht dem Tabellennamen, also kann der Name so in die From-Klausel übernommen werden (`strcpy(queries[crtq].from, $2)`)
- **cut Optionen [dateiname]**: wieder ist der Dateiname (sofern angegeben) die gewünschte Tabelle, die Optionen werden mit einer eigenen Regel extra geparkt

- **grep Muster [Dateiname]:** wieder analog, nur dass ein Muster angegeben ist, also ein Prädikat. Die Funktion `zugrep()` erzeugt eine Bedingung die in jeder Spalte nach Vorkommen des Musters sucht
- **sort Optionen:** eigene Regel für die Optionen

4.2.3. Der Lexer

Die Token liefert `lex`, der für die lexikalische Analyse zuständig ist. Der erhält die deklarierten Token und die Definition der Kommunikationsvariable `yylval` in den Anfangsdefinitionen:

```
%{
#include "y.tab.h"
extern YYSTYPE yylval;
#include <stdio.h>
#include <stdlib.h>
%}
```

Anschließend folgt eine Definition über die Arten von Symbolen, welche vorkommen können.

```
white          [ \t ]+

digit          [0-9]
integer        { digit }+
exponent       [eE][+-]?{ integer }

real           { integer } ( "." { integer } ) ? { exponent } ?

letter         [a-zA-Z0-9\.\,\|\ | ]+

options        -[a-zA-Z0-9\-\,\ | ]+ \
ende           [^\ \ ];
```

Mit `%%` wird der Beginn der syntaktischen Analyse gekennzeichnet, auch als `lex`-Regeln bekannt, die die Token liefern, also die Terminalsymbole für Yacc. Leerzeichen und Tabulatoren sollen ignoriert, bei Zahlen soll ein Integer-Wert zurückgegeben werden, also das Attribut `int4` von `yylval`, bei Zeichenketten ein String und für jeden Befehl soll ein eigenes Token produziert werden.

```
{ white }      { }
{ real }       {
                yylval.int4=atoi(yytext);
                return(NUMBER);
              }

">"           return(REDIR);
"|"           return(PIPE);
"cat"          return(CAT);
"cut"          return(CUT);
...

"\n"           return(END);
{ letter }     {
                yylval.str=strdup(yytext);
```

```
        return WORD;  
    }
```

Aus diesen Regeln generiert Lex nun die Funktion `yylex()`, die bei jedem Aufruf Zeichen einliest, bis sie ein Token zurückgibt. Im Zusammenspiel mit Yacc ruft dieser `yyparse()`, die `yylex()` benutzt und anhand der zurückgegebenen Token erkennt, ob die Eingabe richtig ist.

4.2.4. Bedienung des Bash2SQL-Übersetzers

Im Ordner `bash_parser_yacc` liegen die Dateien `bash2sql.lex` und `bash2sql.y`, also die Lexer-Regeln und die Yacc-Grammatik. Das beiliegende Makefile kompiliert aus den Dateien das Programm `kleinerParser` (s. Abb, 4.2), das aus einfachen Shell-Skripten SQL-Abfragen erzeugt.

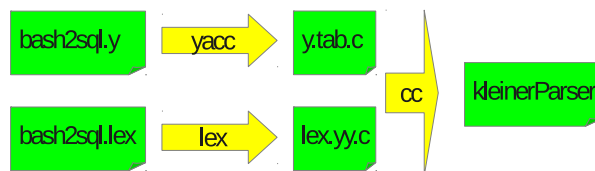


Abbildung 4.2.: Zusammenspiel von Lex und Yacc

Der Übersetzer arbeitet als einfacher Filter, er liest von der Standardeingabe und gibt auf der Standardausgabe das Ergebnis aus. Ein Befehl kann direkt übersetzt werden.

```
$ echo "cat_tabelle|_cut_-f1,2,3|_grep_Muster" | ./kleinerParser
```

Oder ein Befehl wird aus einer Datei gelesen.

```
$ cat abfrage.sh | ./kleinerParser
```

In beiden Fällen wird folgende Ausgabe erzeugt:

```
SELECT $1,$2,$3, FROM tabelle WHERE ($1 like '%Muster%'
      or $2 like '%Muster%' or $3 like '%Muster%'
      or false) GROUP BY ORDER BY AS
```

Da die Spaltennamen noch nicht eingelesen werden, müssen sie noch manuell ersetzt werden. Bis auf ein paar Schönheitsfehler ist die Abfrage in Ordnung.

5. Parser mit ANTLR

Der bisherige Parser unterstützt nur sehr wenige Befehle, implementiert noch die gesamte Shell-Grammatik und bedarf einiger Korrekturen um korrekte SQL-Syntax zu erzeugen. Dabei lohnt sich gleich der Umstieg auf einen mächtigeren Parser-Generator, der außerdem den Code in verschiedenen Programmiersprachen erzeugen kann, nämlich der Parser-Generator **ANTLR**.

5.1. Konfiguration

Der ANTLR-Parser basiert auf Java und erzeugt ohne Erweiterung nur Parser in dieser Sprache, soll der Parser für andere Sprachen Quelldateien ausspucken, so ist eine Bibliothek zu installieren.

5.1.1. Installieren der Bibliothek

Für einen Parser in C/C++ wird die Bibliothek *libantlr3c-3.4* ¹ benötigt, sie enthält alle benötigten Funktionen bzw. Klassen. Die Datei herunterladen, entpacken, konfigurieren und installieren. Dabei ist folgendermaßen vorzugehen:

```
tar -xzf libantlr3c-3.4.tar.gz
cd libantlr3c-3.4
./configure --enable-64bit
make
make install
```

Läuft das Programm auf einem 32-Bit-System, so ist der dritte Schritt durch *./configure* zu ersetzen. Anschließend ist die Bibliothek in */usr/local/lib* installiert.

5.1.2. Starten von ANTLR

Zuerst muss der Parsergenerator als *antlr-3.5.2-complete.jar* ¹ heruntergeladen werden, aber in der Version 3.5.2, da neuere Versionen noch nicht die Codegenerierung in andere Sprachen beherrschen. Für den folgenden Schritt wird angenommen, dass die jar-Datei in */Downloads* liegt. Bevor der Parser-Code generiert werden kann, muss der Klassenpfad für Java und der Pfad zur Bibliothek gesetzt werden, damit die Java Virtual Machine die Komponenten von ANTLR findet und die Bibliothek eingebunden ist.

```
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
export CLASSPATH=~/.Downloads/antlr-3.5.2-complete.jar:$CLASSPATH
```

¹erhältlich unter <http://www.antlr3.org/download/>

5.1.3. Vorwissen zu ANTLR

Im Gegensatz zu anderen Parsern ist ANTLR in einer objektorientierten Sprache geschrieben (wie schon erwähnt in Java), so realisiert er Lexer und Parser als zwei verschiedene Klassen. ANTLR ist mächtiger als Yacc mit Lex, da die lexikalische Analyse auch kontextsensitive Grammatiken unterstützt, LEX nur kontextfreie. Außerdem ist es in ANTLR möglich, die Regeln für die lexikalische und grammatikalische Analyse in eine Grammatik-Datei zu packen, können aber - dank Modularität - auf mehrere aufgeteilt werden. Die Dateien enden auf .g und folgen dem Schema:[17]

```
<Grammatik-Typ> grammar <name>;  
<Optionen>  
<Token-Definition>  
<Quellcode>  
<Grammatik-/Lexer-Regeln>
```

In den Bereich für den Quellcode können sowohl die Kopfzeilen und Deklarationen, wie auch die Funktionen angegeben werden, der Teil kombiniert also den zweiten und letzten Bereich des Schemas von Yacc, dennoch sind sie mit einem Schlüsselwort voneinander abzugrenzen:

```
@header  
{  
/* Parser-Kopf */  
}  
@members  
{  
/* Parser-Rumpf */  
}
```

Die Lexer-Definitionen erfolgen analog durch die Schlüsselworte `@lexer::header` und `@lexer::members`. Die anderen Teile stehen für Folgendes:

1. `<Grammatik-Typ>` entweder Lexer, Parser oder AST, fehlt die Angabe, so Lexer und Parser kombiniert
2. `<Optionen>` unter anderem die Zielsprache
3. `<Token-Definition>` und `<Quellcode>` sind analog zu Yacc und Lex
4. `<Grammatik-/Lexer-Regeln>` sind die gewünschten Regeln

Ein schönes und einfaches Beispiel findet sich auf der Homepage www.antlr.org, im Nachfolgenden werden nur die Zielsprachen C/C++ betrachtet.

5.2. Der Bash2SQL-Übersetzer mit ANTLR in C

Dieses Mal soll die komplette Bash-Grammatik geparkt werden können, die Abfragen sollen modularer und syntaktisch korrekt aufgebaut sein und mittels ANTLR für C generiert. Die Zielsprache ist C und das wird in den Optionen mitgeteilt und noch ein kleiner Bugfix im Kopf des Lexers:


```

options
{
    language=C;
}

tokens
{
    PIPE = ' | ';
}

@lexer::header
{
    /* ein Fix wenn Zustand _empty nicht deklariert*/
    #define _empty NULL
}

```

5.2.1. Neues Konzept

Bisher sind die Kommandos der Reihe nach geparkt und in eine SQL-Abfrage übersetzt worden, bis es nicht mehr geht, wenn ein Kommando unbekannt ist. Jetzt ist die Überlegung, dass jedes Kommando für sich allein steht und eine eigene SQL-Abfrage ergibt. Die Ausgabe erfolgt über den Standardfluss, kann also mit weiteren Kommandos durch eine Pipe verbunden werden. So können auch die einzelnen, bereits ausgeführten, SQL-Abfragen verbunden werden, indem das modifizierte Skript die Ausgabe durchreicht (s. Abb. 5.1).

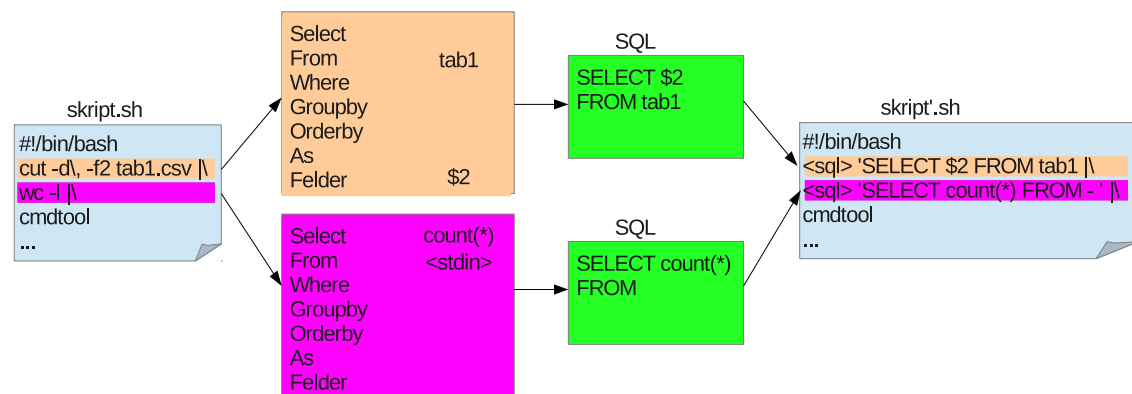


Abbildung 5.1.: Neues Konzept

Schöner ist es natürlich, wenn der Übersetzer gleich erkennt, wann er welche Abfragen zusammenfügen kann, sodass am Ende eine geschachtelte SQL-Abfrage entsteht, die auch optimiert werden kann. Das ist besser, als wenn jede Abfrage einzeln ausgewertet wird und die Kommunikation mit Hilfe der Shell erfolgt, die die Daten über den Strom schickt. Die vorgestellte Idee wird modifiziert, indem der Parser solange die Kommandos liest,

bis er am Ende ist oder ein ihm unbekanntes Kommando erscheint. Für jedes gelesene Kommando wird eine Abfrage erstellt, die die vorherige, falls existent, als Quelltable aufnimmt. Daraufhin entsteht eine geschachtelte SQL-Anweisung. Im Moment steht diese dann zwischen allen nicht erkannten Kommandos, sodass ein wieder funktionierendes Skript erzeugt wird (s. Abb. 5.2).

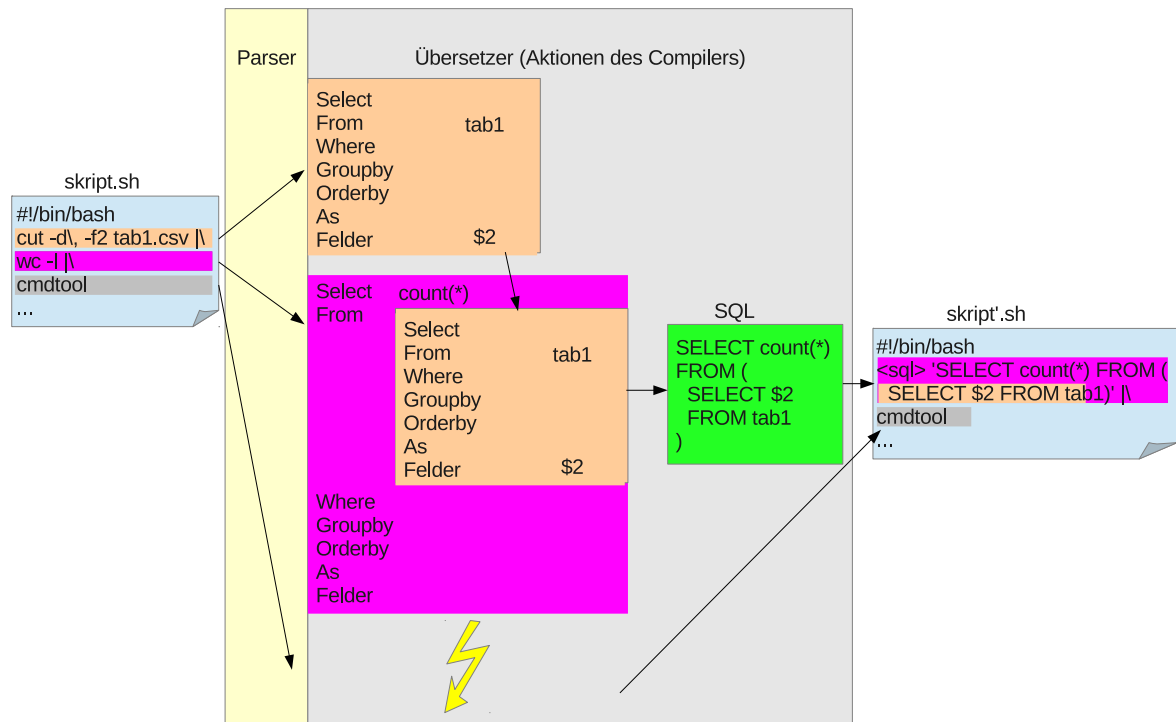


Abbildung 5.2.: Neues Konzept mit Schachtelung

5.2.2. C-Quellcode

Um wieder dem Schema einer Grammatikdatei zu folgen, werden zuerst die Definitionen des C-Quellcodes beschrieben. Im Grunde ist die Definition des SQL-Abfrage-Structs, die auch den gesamten Parser-Kopf bildet, identisch zu der des Parsers in Yacc, lediglich einige Erweiterungen um Joins und Vereinigungen zu bilden sind hinzugekommen.

```
@header
{
#ifdef MYHEAD
#define MYHEAD

#include <assert.h>
#include "SimpleBashSQLLexer.h"
#define MAXFIELDS 5
#define DELIMIT ' ,'

typedef struct myquery{
```

```

char select[10], from[30],
where[200], groupby[5],
orderby[5], as[20];
/*numerische Repraesentation der Felder*/
char felder[MAXFIELDS], groups[MAXFIELDS], orders[MAXFIELDS];
/*sprechende Namen */
char *fname[MAXFIELDS];
/* findet join statt? felder 1, felder2, tabelle1, tabelle2 */
char join, f1, f2, t2[20];
/* findet union statt? */
char sqlunion, sort, group, nrfields;
struct myquery *src, *src2;
} query;

#endif
}

```

Desweiteren werden noch, um den struct bearbeiten zu können, einige Funktionen benötigt, die sich im Detail im Anhang finden und für folgende Aufgaben zuständig sind:

- `int lookup(char* filename, char delimit, query *q)`: liest die erste Zeile einer Datei und nimmt die Werte als Spaltenbezeichner für die gewünschte Abfrage ²
- `int reset(query* abfrage)`: initialisiert einen struct, entspricht einem Konstruktor
- `int deletequeries(query* q)`: gibt Speicherplatz frei, entspricht Destruktor
- `int ausgeben(query* e)`: gibt einen struct query als SQL-Abfrage aus
- `query *makeunion(char* eingabe, query* q)` bzw. `query *makeunion_query(query* eingabe, query* q)`: bilden die Vereinigung aus mehreren Tabellen, im ersten Fall mittels eines Tabellennamens, im zweiten mittels einer weiteren Abfrage
- `int zugrep(char* pattern, query* abfrage)`: erzeugt aus einem Suchmuster das passende Prädikat für die Where-Klausel
- `int parseListe(char*e, char* feld)`: nimmt die Spaltenangaben, wie sie in der Shell üblich sind, und selektiert aus einem Feld nur die gewünschten
- `int opts<cmd>(char* e, query* q)`: parst die Optionen zu dem jeweiligen Kommando <cmd>

Damit der ANTLR-Parsergenerator laufen kann, muss die Main-Methode stimmen. Deren Aufbau wird zwar in verschiedenen Foren erklärt, nur leider sind die meisten Erklärungen veraltet und beziehen sich auf ältere Versionen des Parsergenerators. [13]

1. Im Gegensatz zu Yacc liest ein ANTLR-Parser die Eingabedatei als Operand ein: `argc == 2`, sonst breche ab
2. die Funktion, die den Lexer füttert, ist zu ersetzen durch: `antlr3FileStreamNew ((pANTLR3_UINT8)argv[1], ANTLR3_ENC_8BIT)`;

²das Teilen einer Zeichenkette erfolgt frei nach [21]

3. zuletzt noch das Startsymbol für die Grammatik festlegen, hier *file*: `parser->file(parser);`

Zusammengesetzt mit den übrigen Funktionen sieht das dann so aus:

```
int main(int argc, char * argv[])
{
    pANTLR3_INPUT_STREAM      input;
    pSimpleBashSQLLexer        lex;
    pANTLR3_COMMON_TOKEN_STREAM tokens;
    pSimpleBashSQLParser        parser;

    if(argc!=2){
        printf("\%s:_fehlender_Operand\n",argv[0]);
        return 1;
    }

    input = antlr3FileStreamNew (
        (pANTLR3_UINT8)argv[1], ANTLR3_ENC_8BIT);
    lex    = SimpleBashSQLLexerNew          (input);
    tokens = antlr3CommonTokenStreamSourceNew
        (ANTLR3_SIZE_HINT, TOKENSOURCE(lex));
    parser = SimpleBashSQLParserNew          (tokens);

    parser  ->file(parser);

    parser ->free(parser);
    tokens ->free(tokens);
    lex    ->free(lex);
    input  ->close(input);

    return 0;
}
```

Die Konfiguration ist fertig, jetzt kann mit den Grammatik-Regeln begonnen werden. Zuerst muss natürlich die Shell-Grammatik implementiert sein, praktischerweise existieren zahlreiche Bücher, die die Grammatik in Backus-Naur-Form beschrieben haben und sich somit leicht in ANTLR importieren lässt. [6]

5.2.3. Die Grammatik

Besonders interessant ist die Stelle, dir für die Ausgabe der SQL-Anweisung zuständig ist, die Regel *pipeline_cmd*. Sie erkennt die Konkatenation mehrerer Befehle (*command*) mittels einer Pipe. Die Kommunikation erfolgt über eine Referenz auf den letzten SQL-Abfrage-Struct *lastquery*, ist dieser *NULL*, so konnte das Kommando nicht geparkt werden und der Befehl soll wieder ausgegeben werden. Alternativ wird am Ende der SQL-Abfrage-Struct ausgegeben. Das Attribut *tobeprint* dient zum Erkennen, ob zuerst der Kopf einer Schleife ausgegeben werden soll.

```
pipeline_cmd      : c1=command
                  {
                      if (tobeprint && !lastquery && $c1.text->chars)
                          printf("\%s", $c1.text->chars);
                  }
                  (PIPE c2=command
                  {
```

```
        if (tobeprint && !lastquery)
            printf(" | %s", $c2.text->chars);
    }
)*
{
    if (lastquery)
        ausgeben(lastquery);
    printf("\n");
    tobeprint=1;
}
;
```

Auf diese Weise steigt man ab bis zur Regel *cmd*.

cmd returns [query *r]

Am Anfang wird Platz für einen neuen Query-Struct geschaffen, der als Quelle den vorherigen erhält (*lastquery*):

```
@init{
    /* neue Query r, vorherige Abfrage als FROM-Tabelle
       NULL, wenn erstes Kommando */
    char tmp=0; //zaehlt args
    r=(query*) malloc(sizeof(query));
    r->src=lastquery;
    reset(r);
}
```

Am Ende ist die aktuelle Query die letzte, also:

```
@after{
    lastquery=r;
}
```

Da das Parsen der Befehle und das Ausführen der zugehörigen Aktionen meistens sehr ähnlich abläuft, wird nachfolgend das Erkennen von *cut* vorgestellt.

Zuerst wird immer der Befehl geparkt, anschließend die Parameter. Der Zugriff auf den Inhalt eines Symbols erfolgt mit `\$SYMBOL.text->chars` als Zeichenkette (`char*`):

```
'cut'      ( OPTS
             { optscut($OPTS.text->chars,r); }
          | WORD
             { r=makeunion($WORD.text->chars,r); }
          | from_redir
             { /* fname!=NULL, wenn Dateiname */
               if ($from_redir.fname)
                   r=makeunion($from_redir.fname,r);
               else /* sonst subshell */
                   r=makeunion_query(lastquery,r);
             }
          | '- ' )+

```

Als Parameter können Optionen oder die anzuzeigende Datei angegeben werden. Optionen (das Lexer-Token *OPTS*) werden mit der Methode *optscut()* gleich analysiert und das Ergebnis in den Struct eingebunden. Für jedes Kommando existiert eine eigene Funktion, um die Optionen zu parsen, also die eine Zeichenkette nimmt und sie Buchstabe für Buchstabe abarbeitet.

```

/* zu cut */
int optscut(char* e, query* q){
    /*bis Ende erreicht*/
    while(++e!=0){
        switch (*e){
            case 'f':
                parseListe(e,q->felder);
                break;
        }
    }
    return 1;
}

```

Da mit der Option *-f* die auszugebenden Spalten bezeichnet sind, wird die Funktion `parseListe()` benötigt, die mithilfe einer solchen Kette die gewünschten Felder extrahiert, bei *cut* also die zu selektierenden Felder.

```

int parseListe(char* e, char* feld)
{
    int crtff=0;
    int a,b,ret;
    do{
        e++;
        ret=sscanf(e, "\\%d-\\%d",&a,&b);
        /* felder beginnen bei 0*/
        a--; b--;
        if (ret==0)/*error*/
            return -1;
        /* von - bis */
        else if (ret==2){
            for (;a<=b;a++)
                feld[crtff++]=feld[a];
        }/*einzelne Auswahl */else{
            feld[crtff++]=feld[a];
        }
        while ((*e>='0' && *e<='9') || *e=='-')
            e++;
    } while (*e==' ');
    feld[crtff]=-1;
    return 1;
}

```

Mit den vorgestellten Methoden und Regeln ist es nun möglich, komplette Bash-Skripte zu parsen und die unterstützten Kommandos zu interpretieren.

5.2.4. Lexer-Regeln

Der Vollständigkeit halber seien hier auch noch die Regeln zur syntaktischen Analyse beschrieben. Diese befinden sich bei ANTLR bei den übrigen Regeln, aus Konvention werden diese Symbole, die durch Terminale zu ersetzen sind, jedoch großgeschrieben.

```

/*-----
 * LEXER RULES
 *-----*/
OPTS          : ( '-' | '+' ) ( LETTER | DIGIT | '-' )+
               ;

```

```

NUMBER      : (DIGIT)+
;
WORD        : (LETTER|DIGIT)+;
ALCHAR      : (LETTER|NUMBER|SONDER)+;
WHITESPACE  : ( '\t' | ' ' | '\r' | '\u000C' | '\\\ ' '\n' | '"' )+
{
    $channel = HIDDEN;
}
;

fragment
DIGIT       : '0'..'9';
fragment
LETTER      : ('A'..'Z'|'a'..'z'|'.'|'|'_'|'|','|'\|'|'|';'|'$');
fragment
SONDER      : ('{' | '}' | '%' | '\'' | '"' | '-' | '=' | '/' | '!' );

```

5.2.5. Bedienung

Im Ordner `c_bash_parser_Yacc` liegen die Grammatikdatei `SimpleBashSQL.g` und ein Makefile. Beide reichen aus, um mit `make` den Parser zu erzeugen, wenn die Bibliothek richtig installiert und die Pfade richtig gesetzt sind.

ANTLR erzeugt eine Lexer- und eine Parser-Quelldatei `SimpleBashSQL{Lexer,Parser}.{h,c}`, die zu dem fertigen Compiler mittels `cc` kompiliert werden:

```

all: simplebashsql

simplebashsql: $(OBJ).g
    java org.antlr.Tool $^
    gcc -g -o $@ -I/usr/local/include/ -L/usr/local/lib/ -lantlr3c $(OBJ)
    Parser.c $(OBJ) Lexer.c

```

Das Programm nimmt den Dateinamen eines Skripts als Eingabeparameter und schickt das Ergebnis auf die Standardausgabe. So wird das Skript:

```

#!/bin/bash
cut -f1,2,3 vorlesungen.csv | sort -k2

```

übersetzt in:

```

#!/bin/bash
SELECT VorlNr,Titel,SWS FROM (
    SELECT VorlNr,Titel,SWS FROM vorlesungen.csv AS b
) AS b ORDER BY $1

```

5.3. Der Bash2SQL-Übersetzer mit ANTLR in C++

So schön die Sprache C auch ist, so ist die Speicherverwaltung mit ihr ziemlich anstrengend, vor allem da die Bezeichner von Tabellen oder Spalten unterschiedlich lang werden können, lohnt sich der Umstieg auf C++ mit impliziter Speicherverwaltung und einer Standardbibliothek mit `vector` und `string`, die einem das Leben erleichtern. Außerdem kann ein höheres Abstraktionsniveau erreicht werden, indem die Abfrage über eine eigene Klasse gekapselt wird und der Zugriff nur noch über Methoden erfolgt und Änderungen nicht mehr per Hand durchgeführt werden müssen.

5.3.1. Unterschiede: C vs. C++ mit ANTLR

Um ANTLR C++-Code generieren zu lassen, sind einige Änderungen nötig.

Die Unterstützung von C++ in ANTLR baut auf sogenannten Traits auf, also vorgefertigten Klassen die in ähnlicher Weise wiederverwendet werden, in dem vorliegenden Fall für den Lexer.

```
@lexer::traits {
    class SimpleBashSQLLexer;
    class SimpleBashSQLParser;
    typedef antlr3::Traits< SimpleBashSQLLexer, SimpleBashSQLParser >
        SimpleBashSQLLexerTraits;
    typedef SimpleBashSQLLexerTraits SimpleBashSQLParserTraits;
}
```

Die Header-Dateien, in denen die verwendeten Klassen wie *antlr3* definiert sind, müssen zum Kompilieren eingebunden werden. Die Dateien finden sich im *antlr3-master* (erhältlich bei github³) im Verzeichnis *antlr3-master/runtime/Cpp/include*.

Die Main-Methode muss natürlich auch noch verändert werden, die Funktionsweise ist aber identisch zum Parser in C, die einzulesende Datei wird als Operand mitgegeben.

```
int main(int argc, char * argv[])
{
    if(argc!=2){
        printf("\%s: _fehlender_Operand\n", argv[0]);
        return 1;
    }

    ANTLR_UINT8* fName = (ANTLR_UINT8*) argv[1];
    SimpleBashSQLLexer::InputStreamType input(fName, ANTLR_ENC_8BIT);
    SimpleBashSQLLexer lxr(&input); // TLexerNew is generated by ANTLR
    SimpleBashSQLParser::TokenStreamType tstream(ANTLR_SIZE_HINT, lxr.
        get_tokSource() );
    SimpleBashSQLParser psr(&tstream); // TParserNew is generated by ANTLR3
    psr.file();
    return 0;
}
```

Um in den Aktionen auf den Wert eines Symbols zugreifen zu können, dienen Ströme (Streams). Soll der Wert ausgegeben werden, so erfolgt der Zugriff über das Attribut *text*, zum Beispiel eine Ausgabe auf der Standardausgabe:

```
comment { std::cout << $comment.text; }
```

Soll der Inhalt in eine Zeichenkette überführt werden, so ist der Umweg über einen *stringstream* praktisch, da die Methode *str()* einen String zurückgibt.

Der Tokenstream funktioniert nur mit Nichtterminalsymbolen (Parserregeln), das heißt für jede Lexerregel, also die, die auf Terminalen arbeiten, muss eine Parserregel existieren. So besteht eine Zahl (*NUMBER*) aus Ziffern (*DIGIT*), dazu muss noch eine Parserregel *number* erstellt werden, um die zugehörige Zeichenkette, also die Zahlen, auszugeben:

```
number: NUMBER;
NUMBER    : (DIGIT)+
          ;
```

³<https://github.com/antlr/antlr3>


```
fragment
DIGIT      : '0'..'9';
```

Die letzte Regel ist als *fragment* gekennzeichnet, da Ziffern immer als Teil einer Zahl vorkommen sollen, keine Parserregel darf darauf aufbauen.

5.3.2. Die Klasse TheQuery

Für die erste fundamentale Änderung werden alle Operationen für eine Abfrage in der Klasse *TheQuery* gekapselt. Alle Veränderungen am Datenbestand erfolgen jetzt über Methoden, die interne Repräsentation interessiert den späteren Compiler nicht und das Praktische daran: die Klasse steht für sich allein und ist auch komplett austauschbar durch ein Substitut, das nur die entsprechenden Schnittstellen implementieren muss. Bevor die Methoden betrachtet werden, sollte sich der Aufbau der Klassen überlegt werden. Um einen praktikablen SQL-Standard zu erfüllen, orientiert sich der Aufbau an der DB2-Referenz von IBM, der schön modular beschrieben ist [7].

Interne Repräsentation

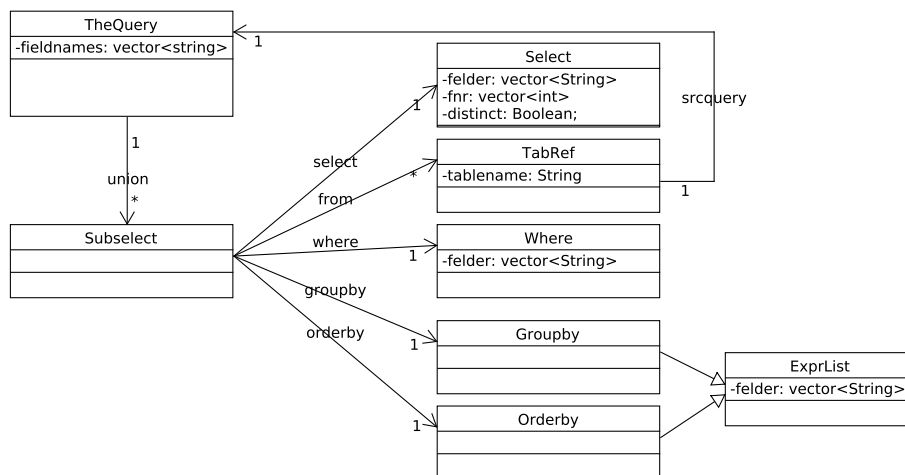


Abbildung 5.3.: Aufbau der Klasse TheQuery

Eine Klasse besteht aus einer Hauptklasse *TheQuery*, die auch alle Methoden für den Zugriff kapselt (s. Abb. 5.3). Sie verantwortet die Mengen aus mehreren eigentlichen Abfragen von *Subselect* durch Vereinigung (`vector<Subselect> union`) oder auch die im Moment nicht benötigte Schnittmenge oder Differenz. Die Klasse *Subselect* bildet die eigentliche Abfrage mit einer *Select*-Clause, einer Menge `vector<TabRef>` *from* als Referenztabelle und den Objekten für Prädikate (*Where*), Gruppierungen (*Groupby*) und Sortierungen (*Sortby*). Letztere beiden sind ähnlich als Vektor aus Feldern implementiert und erben noch von einer solchen Klasse *ExprList*. Die Prädikate der *Where*-Klausel sind durch Konjunktionen (*AND*) verknüpfte Aussagen und im Nachfolgenden wird eine konjunktive Normalform eingehalten, sodass eine Konjunktion aus Disjunktionen erstellt ist. In manchen Fällen

wird als neutrales Element *true* verwendet, wenn eine Aussage entfernt wird. Für Tabellen dient eine Klasse *TabRef* mit einem Tabellennamen oder einer Referenz auf eine komplette Abfrage als Quelle. Die Referenz ist geeigneter als eine Vererbung, da die Klasse auch die textuelle Ausgabe mit einer entsprechenden Klammerung steuert.

Darstellung der Felder

Eine Schwierigkeit stellt die interne Repräsentation der Felder dar, da die Spaltenbezeichner nicht aus den Skripten hervorgehen und nicht davon auszugehen ist, dass die Bezeichner aus einer Datei auszulesen sind. Deshalb werden vorübergehend die Feldbezeichner entsprechend zu *awk* gewählt (\$1,\$2,...), das Maximum an Spalten muss dann definiert werden, wie gewohnt in *MAXFIELDS*. Die gewählten Felder können direkt in die Select-Klausel eingefügt werden, auch als Zahlenvektor.

```
void Select::addSelect(std::vector<int> feldnr)
{
    fnr=feldnr;
    /*copies values into felder*/
    for (std::vector<int>::iterator it = fnr.begin();
         it!=fnr.end(); it++){
        std::stringstream stmp;
        stmp << "$" << *it;
        felder.push_back(stmp.str());
    }
}
```

Alternativ können auch einzelne Feldbezeichner oder SQL-Ausdrücke wie Aggregatsfunktionen als Zeichenkette direkt eingegeben werden.

```
void Select::addSelect(std::string name)
{
    felder.push_back(name);
}
```

Jedes Mal, wenn ein Tabellename in die Abfrage eingelesen wird, wird überprüft, ob eine passende Datei mit passender Kopfzeile vorhanden ist, und mit der Methode *getnamesfromfile()* werden die Spalten eingelesen, das Spaltentrennzeichen *DELIMIT* wird durch den Präprozessor definiert.

```
/*union more queries*/
void TheQuery::makeUnion(string src)
{
    /* table already defined? */
    if(queries[0]->notable())
        queries[0]->addTable(src);
    else
        queries.push_back(new Subselect(src));

    /*add column names*/
    getnamesfromfile(src,DELIMIT);
}
```

Die Methode *getnamesfromfile()* setzt die Spaltennamen in *fieldnames*, wenn noch nicht definiert, mit der Methode *lookup()*. Diese Methode liest die erste Spalte einer Datei, trennt die Zeile nach dem Trennzeichen und gibt eine Vektor der Klasse *String* zurück.

```

std::vector<std::string> TheQuery::lookup(std::string filename, char delimit)
{
    FILE *f;
    char * line = NULL;
    size_t len = 0;
    ssize_t read;
    char** ptr;
    std::vector<std::string> tmp;
    f=fopen(filename.c_str(), "r");
    if (!f)
        return std::vector<std::string>();
    if (read = getline(&line, &len, f) == -1)
        return std::vector<std::string>();
    fclose(f);

    tmp=str_split(line, delimit);
    /* header columns terminated with delimiter symbol */
    tmp.pop_back();
    return tmp;
}

```

Das Attribut *fieldnames* ist für das ganze Objekt einer Klasse *TheQuery* gültig, da bei jeder Teilabfrage einer Vereinigung auch die Spaltenbezeichner identisch sein müssen. Bei Bildung eines Kreuzproduktes werden die hinzukommenden Spalten einfach hinzugefügt. Wird die Abfrage in einer anderen verwendet, so werden auch die Spaltennamen übergeben.

```

void TheQuery::makeUnion(TheQuery* src)
{
    if (src==NULL)
        return;
    /* table already defined? */
    if (queries[0]->notable())
        queries[0]->addTable(src);
    else
        queries.push_back(new Subselect(src));
    /*get columnnames from file*/
    if (fieldnames.size()==0)
        fieldnames=src->getColumns();
}

```

Bevor eine Abfrage mit *print()* ausgegeben wird, werden alle Hilfsspaltenbezeichner, die durch Dollar gekennzeichnet sind, ersetzt. Dazu stellt jede Klasse eine Methode *replace_dollars()* zur Verfügung, die intern die Hilfsbezeichner ersetzt durch den passenden Spaltennamen aus *fieldnames*.

```

void TheQuery::print()
{
    /*only when a query is defined, should always be the case*/
    if (queries.size() > 0){
        queries[0]->replace_dollars(fieldnames);
        queries[0]->print();
    }
    /* union all queries */
    for (int i=1; i<queries.size(); i++){
        std::cout << " UNION ";
    }
}

```

```

        queries[i] -> replace_dollars(fieldnames);
        /*now all unions must have an unique id*/
        queries[i] -> print(i);
    }
}

```

Soweit steht die Klasse TheQuery und kann zur Erstellung der Abfragen genutzt werden, der Zugriff erfolgt nur über die Methoden von TheQuery, die auch im Anhang zu finden sind.

5.3.3. Der Parser näher betrachtet

Der Übersetzer bzw. Compiler besteht aus zwei Teilen, zuerst wird der Syntax analysiert und ein Syntaxbaum generiert, dann können die zugehörigen Aktionen ausgeführt werden (vgl. Abbildung 5.4). Dabei hat sich wenig verändert, die Syntaxanalyse funktioniert in C++ etwas flexibler, manche Lexerregeln können besser gestaltet werden.

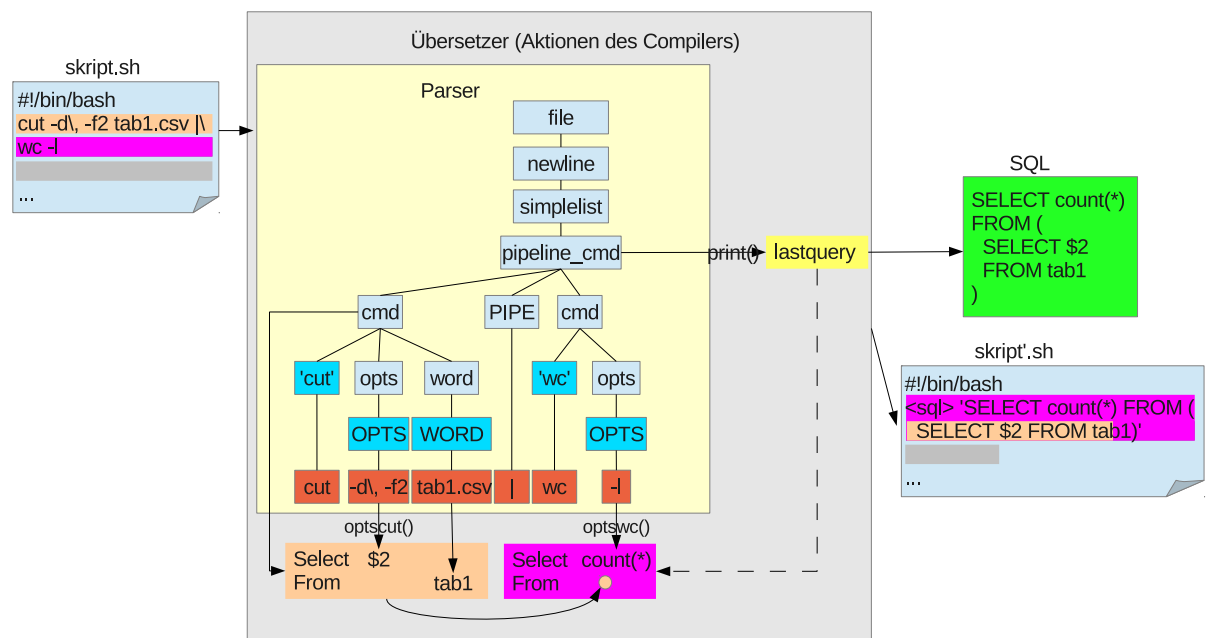


Abbildung 5.4.: Parsen und Erstellen eines Syntaxbaums, anschließend Übersetzen

Außerdem werden die Aktionen nun den Klassen angepasst, da die meiste Logik in `cmd` passiert, nun ein kleiner Abriss:

Die Regel übernimmt zuerst die alte Abfrage, der Inhalt aus der Pipeline, also die zuletzt in SQL übersetzte Abfrage, also `lastquery` wird in einer temporären Variablen `fromPipe` gespeichert. Eine Subshell kann Teil einer Abfrage sein, deshalb wird `lastquery` auf NULL gesetzt, damit geöffnete Subshells mit leerer Eingabe beginnen, also ohne einen Input. Anschließend kann eine neue SQL-Abfrage erzeugt werden, in die das zu parsende Kommando übersetzt wird. Alle weiteren Attribute dienen zur Bearbeitung bei bestimmten

Kommandos (z.B. zählen der Argumente bei join), immer wird ein stringstream benötigt, in den der Inhalt der Symbole geschrieben wird. Am Ende der Regel wird die gerade erzeugte Abfrage als aktuelle gesetzt, also in die globale Variable lastquery.

```
/* Rules for parsing bash command to SQL */
cmd returns [TheQuery *r]
@init{
    /* new TheQuery r,
    afterwards, set old query (lastquery) as table-reference
    set lastquery as null, so queries from subshell won't be
    included twice */
    TheQuery* fromPipe=lastquery;
    lastquery=NULL;
    r = new TheQuery();
    stringstream s;
    char buffer[80];
    int helpsize=0, join_on;
}
@after{
    /* query r is the most recent query */
    lastquery=r;
}
```

Das Kommando cut

Da das Übersetzen der meisten Kommandos ähnlich erfolgt, wird nachfolgend wieder *cut* vorgestellt (siehe auch Abb. 5.5).

```
:
    'cut'      ( opts
                {
                    s.str(""); s.clear();
                    s << $opts.text; s.getline(buffer,80);
                    optscut(buffer,r);
                }
    | word
                {
                    s.str(""); s.clear(); s << $word.text;
                    r->makeUnion(s.str());
                }
    | from_redir
                { /* fname!=NULL, wenn Dateiname */
                    if (!$from_redir.fname.empty())
                        r->makeUnion($from_redir.fname);
                }
    | ' _ '
    )+
    {r->makeUnion(fromPipe);}
```

Zuerst wird immer das Kommando abgefragt, anschließend werden die Parameter geparkt, also Optionen, die Eingabe oder die Umlenkung als Eingabe. Für jedes Kommando muss eine eigene Regel erstellt werden. Die Aktionen zu den Regeln sind recht schlicht, die Optionen, über das Symbol *opts*, das für alle Zeichenfolgen steht, die mit “-” eingeleitet werden, werden mit der zugehörigen Funktion *optscut()* geparkt und in die Abfrage übersetzt. Als Parameter kann eine Quelldatei stehen, die als Tabellennamen angenommen

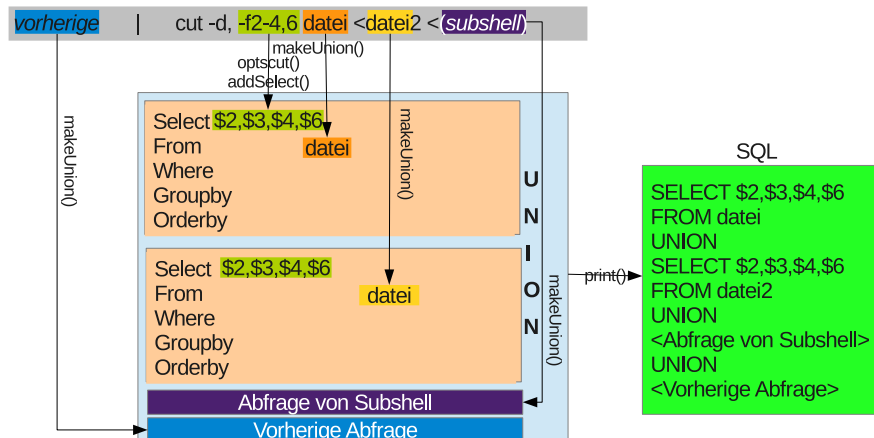


Abbildung 5.5.: Konvertieren des Befehls cut in SQL

wird. Dies passiert mit der Methode der Klasse TheQuery makeUnion(), die im Falle mehrerer Tabellen die Vereinigung darüber bildet. Am Ende wird die über die Pipeline erhaltene Abfrage berücksichtigt, auch sie wird mit makeunion() zur aktuellen Abfrage vereinigt.

Das Vereinigen von Tabellen mittels makeUnion() ist für fast alle Befehle identisch, auch das Parsen der Optionen ist meist sehr ähnlich, darum werden noch drei besondere Befehle hervorgehoben.

Das Kommando grep

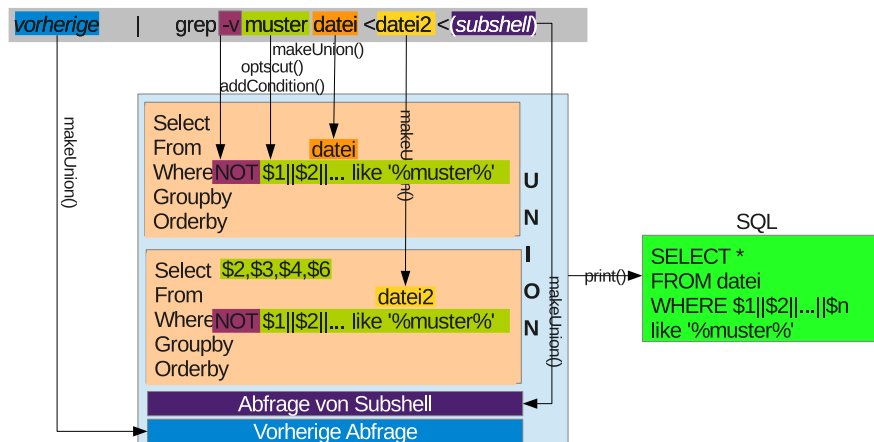


Abbildung 5.6.: Konvertieren des Befehls grep in SQL

Für den Befehl grep ist das angegebene Suchmuster mit allen Feldern zu vergleichen, daher werden zuerst alle Felder konkateniert (hier ist also die gesamte Anzahl an Spalten nötig) und danach wird mit like nach Vorkommen des Musters darin gesucht. Da in grep

eventuell auch das Spaltentrennzeichen mit angegeben ist, kann beim Verbinden der Felder auch das Zeichen berücksichtigt werden ($\$1 || | || \2). Der erzeugte Ausdruck ist dabei in alle aktuellen Abfragen einzubinden.

Das Kommando join

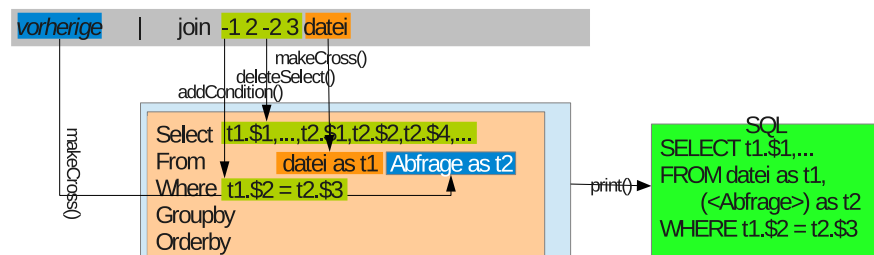


Abbildung 5.7.: Konvertieren des Befehls join in SQL

Den Verbund kann so in SQL übernommen werden, die zu verbindenden Spalten werden mit den Optionen `-1` und `-2` angegeben (ohne Angabe von Optionen wird über die jeweils erste Spalte verbunden), zu berücksichtigen ist noch die Reihenfolge der Tabellen, ob an erster oder zweiter Stelle, und dass die verbundene Spalte der zweiten Tabelle nicht im Ergebnis auftaucht. Um das Verbinden zu erleichtern, werden die beiden Tabellen noch mit einem Bezeichner versehen (`t1`, `t2`). Da keine Vereinigung sondern ein Join gebildet werden soll, erfolgt das Hinzufügen der Tabellen in die From-Klausel mittels `makeCross()`.

Ausdrücke der Sprache awk

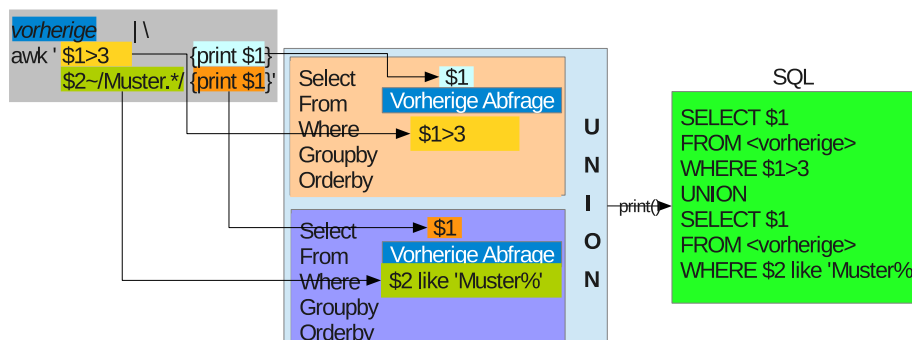


Abbildung 5.8.: Konvertieren des Befehls awk in SQL

Das Programm übersetzt auch einfache Konstrukte der Sprache `awk`. Da sie meist aus einem Muster und einer folgenden Anweisung bestehen, kann das Muster als Bedingung und die Anweisung in die Select-Klausel übernommen werden. Besteht ein Suchmuster

aus einem einfachen Vergleich wie `$2 == "Inhalt"`, so kann das Muster direkt in die Bedingung übernommen werden, lediglich die Kennzeichnung von Zeichenketten durch einfache Hochkommata und die Äquivalenz durch ein einfaches `"="` müssen angepasst werden (`$2 = 'Inhalt'`). Reguläre Ausdrücke sind ähnlich anzupassen (`$2 ~ /\.Muster.*\/`), da sie, identisch zu `grep`, dem *like*-Operator entsprechen (`$2 like '?Muster\%'`). Da nach jedem Muster eine andere Aktion folgt, sind diese auch als unabhängige Anfragen zu betrachten, für jedes Muster muss eine erneute Abfrage desselben Ursprungs mit identischen From-Klauseln erstellt werden. In die Select-Klausel wird jedes `print` übersetzt, denn nur da erfolgt eine Ausgabe. Im Moment werden nur Ausgaben der Felder unterstützt wie `print $1, $2`, kompliziertere arithmetische Ausdrücke und Ausgabe von Variablen müssten in eine entsprechende Anweisung mit *case* umgewandelt werden.

5.3.4. Bedienung

Im Ordner *plus_bash_parser_yacc* liegen die Grammatikdatei *SimpleBashSQL.g*, die Dateien *TheQuery.hpp* und *-.cpp*, eine dynamische Bibliothek und ein Makefile. Wie im vorherigen Fall wird der Compiler automatisch generiert, wenn die Konfiguration richtig erfolgt ist. Alternativ kann das *init*-Skript angepasst werden, das die Pfade richtig setzt.

```
$ . ./init
```

ANTLR erzeugt eine Lexer- und eine Parser-Quelldatei *SimpleBashSQL{Lexer,Parser}.{hpp,cpp}*, die zu dem fertigen Compiler mittels `g++` kompiliert werden.

```
simplebashsql: theparser thequery.o
               g++ -std=c++11 -g -o $@ $(CFLAGS) -lantlr3c $(OBJ)Parser.cpp $(OBJ)Lexer
               .cpp thequery.o
               rm thequery.o
```

Das Programm nimmt den Dateinamen eines Skripts als Eingabeparameter und schickt das Ergebnis auf die Standardausgabe. So wird das Skript:

```
$ cat testskript.sh
#!/bin/bash
cut -f1,2,3 vorlesungen.csv | sort -k2
```

übersetzt in:

```
$ ./simplebashsql testskript.sh
SELECT * FROM (SELECT $1,$2,$3 FROM vorlesungen.csv as t1 ) as t1 ORDER BY $2
```

Existiert darüber hinaus eine Datei *vorlesungen.csv* mit Kopfzeile, so erkennt das Programm gleich die Spaltenbezeichner und erzeugt sprechende Namen.

```
$ head -1 vorlesungen.csv
VorlNr,Titel,SWS,
$ ./simplebashsql testskript.sh
SELECT * FROM (SELECT VorlNr,Titel,SWS FROM vorlesungen.csv as t1 ) as t1 ORDER
BY Titel
```

5.3.5. Rückübersetzung einer TPC-H Abfrage

Um wieder zu den TPC-H Abfragen zurückzukehren, so genügen die definierten Regeln um die anfangs beschriebene vierte Abfrage wieder zurückzuübersetzen.


```

$ ./simplebashsql query4b
mkfifo tmporder.csv
WITH tmporder.csv AS (
SELECT $1,$6 FROM (SELECT * FROM orders.csv as t1 UNION SELECT * FROM (SELECT *
FROM orders.tbl as t1 ORDER BY $1,$1) as t1union1 ) as t1 WHERE (true)
AND ($5<'1993-10-01') AND ($5>='1993-07-01') )

WITH tmpline.csv AS (
SELECT $1 FROM (SELECT $1 FROM (SELECT * FROM lineitem.csv as t1 UNION SELECT *
FROM (SELECT * FROM lineitem.tbl as t1 ORDER BY $1,$1) as t1union1 ) as t1
WHERE (true) AND ($12<$13) ) as t1 GROUP BY $1 )

SELECT count(*),$2 FROM (SELECT * FROM (SELECT * FROM (SELECT $2 FROM (SELECT $1
,$2,$3,$4,$5,$6,$7,$8,$9 FROM tmporder.csv as t1, tmpline.csv as t2 WHERE (
t1.$1=t2.$10) ) as t1 ) as t1 ) as t1 ) as t1 GROUP BY $2
rm tmp*.csv

```

Im Moment erscheinen noch kryptische Spaltenbezeichner, denn die Dateien konnten noch nicht gefunden werden, der Übersetzer nimmt für jede Tabelle die definierten neun Spalten an. Für jede eingelesene Datei muss jetzt eine Datei mit Kopfzeilen definiert sein, für Abfragen innerhalb einer Vereinigung genügt die Existenz einer Datei mit Kopfzeilen (orders.csv gibt für orders.tbl mit die Spalten an). Leider erzeugt das Programm noch keine solche Datei für Hilfstabellen, die das Skript erzeugt. Folglich müssen für tmporder.csv und tmpline.csv auch Dateien hinterlegt werden (das ursprüngliche Skript ist ausgeführt worden um an die Dateien zu gelangen).

```

$ cat orders.csv
o_orderkey,o_custkey,o_orderstatus,o_totalprice,o_orderdate,o_orderpriority,
o_clerk,o_shippriority,o_comment,
$ cat orders.csv
o_orderkey,o_custkey,o_orderstatus,o_totalprice,o_orderdate,o_orderpriority,
o_clerk,o_shippriority,o_comment,
$ cat lineitem.csv
l_orderkey,l_partkey,l_suppkey,l_linenummer,l_quantity,l_extendedprice,
l_discount,l_tax,l_returnflag,l_linestatus,l_shipdate,l_commitdate,
l_receiptdate,l_shipinstruct,l_shipmode,l_comment,
$ cat tmporder.csv
o_orderkey,o_orderpriority,
$ cat tmpline.csv
l_orderkey,
$ ./simplebashsql query4b

mkfifo tmporder.csv
WITH tmporder.csv AS (
SELECT o_orderkey,o_orderpriority FROM (SELECT * FROM orders.csv as t1 UNION
SELECT * FROM (SELECT * FROM orders.tbl as t1 ORDER BY $1,$1) as t1union1 )
as t1 WHERE (true) AND (o_orderdate<'1993-10-01') AND (o_orderdate>=
1993-07-01') )

WITH tmpline.csv AS (
SELECT l_orderkey FROM (SELECT l_orderkey FROM (SELECT * FROM lineitem.csv as t1
UNION SELECT * FROM (SELECT * FROM lineitem.tbl as t1 ORDER BY $1,$1) as
t1union1 ) as t1 WHERE (true) AND (l_commitdate<l_receiptdate) ) as t1
GROUP BY l_orderkey )

```

```
SELECT count(*) , o_orderpriority FROM (SELECT * FROM (SELECT * FROM (SELECT
  o_orderpriority FROM (SELECT o_orderkey , o_orderpriority FROM tmporder.csv as
  t1 , tmpline.csv as t2 WHERE (t1.o_orderkey=t2.l_orderkey) ) as t1 ) as t1
) as t1 ) as t1 GROUP BY o_orderpriority
rm tmp*.csv
```

Diese Abfrage kann nun über eine SQL-Schnittstelle eingegeben werden, vorher müssen die Dateiendungen entfernt werden (.csv), die manche SQL-Inline-Tools jedoch benötigen, anschließend kann die Zeit gemessen werden.

Abfrage	Zeit
Query4 original	62.87 ms
Query4b Skript	21 910,00 ms
Query4b rückübersetzt	5 946.87 ms

6. Ausblick

Mit dem Compiler ist ein erster Grundstein gelegt, er beherrscht die Bash-Syntax bis auf Funktionen, kann grundlegende Kommandos parsen und in SQL übersetzen.

Grundsätzlich gilt auch für dieses Programm die Regel, ein Programm ist nie fertig, die beste Möglichkeit, es zu erweitern, ist, es an weitere Skripte, wie sie in der Wissenschaft vorkommen, anzupassen. Abgesehen davon, dass alle Befehle komplett implementiert werden sollen (am Besten natürlich alle möglichen Programme mit allen möglichen Optionen), was Schritt für Schritt um die benötigten gehen sollte, so gibt es drei wesentliche Änderungen die in Erwägung gezogen sollten.

6.1. Schleifen/Bedingungen

Bisher wurde nur mit nicht geschachtelten Schleifen experimentiert, da generell keine Skripte mit Schleifen oder Bedingungen analysiert wurden. Da eine Schleife immer als Ganzes eingelesen wird, führt dies zu Problemen, da zuerst die Kommandos im Rumpf bearbeitet werden, am Ende die Schleife selber. Bisher regelte dies eine globale Variable, in welcher Ebene sich das Kommando befindet (bisher nur als boolsche Variable, evtl. Ebenen-Zähler geschickter) und funktioniert für die getesteten Fälle mit nur einer for-Schleife, es gibt aber noch genug Platz für weitere Experimente.

6.2. Sprache awk separat

Im Moment erfolgt das Parsen von awk-Ausdrücken innerhalb der Bash-Grammatik, obwohl diese grundverschieden sind, so spielen in awk Leerräume keine Rolle, in Bash schon. Daher sollte überlegt werden, Teile an einen externen Parser zu übergeben, der dann auch eine Abfrage als Objekt der Klasse TheQuery zurückgibt.

6.3. Vor Übersetzen zusammenfügen

Im Moment werden zuerst die Kommandos und in Abhängigkeit dieses alle weiteren Parameter (Optionen, Eingaben, Subshells) eingelesen. Dies hat als Vorteil, dass die Anordnung gleich ist, Optionen können an beliebiger Stelle stehen, aber auch den Nachteil, dass für alle Kommandos ähnliche Regeln aufzustellen sind. Einfacher wäre es, die Regeln zu einer zusammenfassen und dann mit Rückgabetypen zu arbeiten (in Yacc dienen structs als Rückgabetypen). Funktional wäre dies aber noch keine Verbesserung, da die Implementierung nicht alle möglichen Optionen unterstützt. Die Funktion *getopts()* unterstützt auch Optionen, deren Parameter mit Leerzeichen dahinter stehen, dafür muss aber der Kommandoname miteingelesen werden. Das hieße den Befehl zuerst parsen und dann

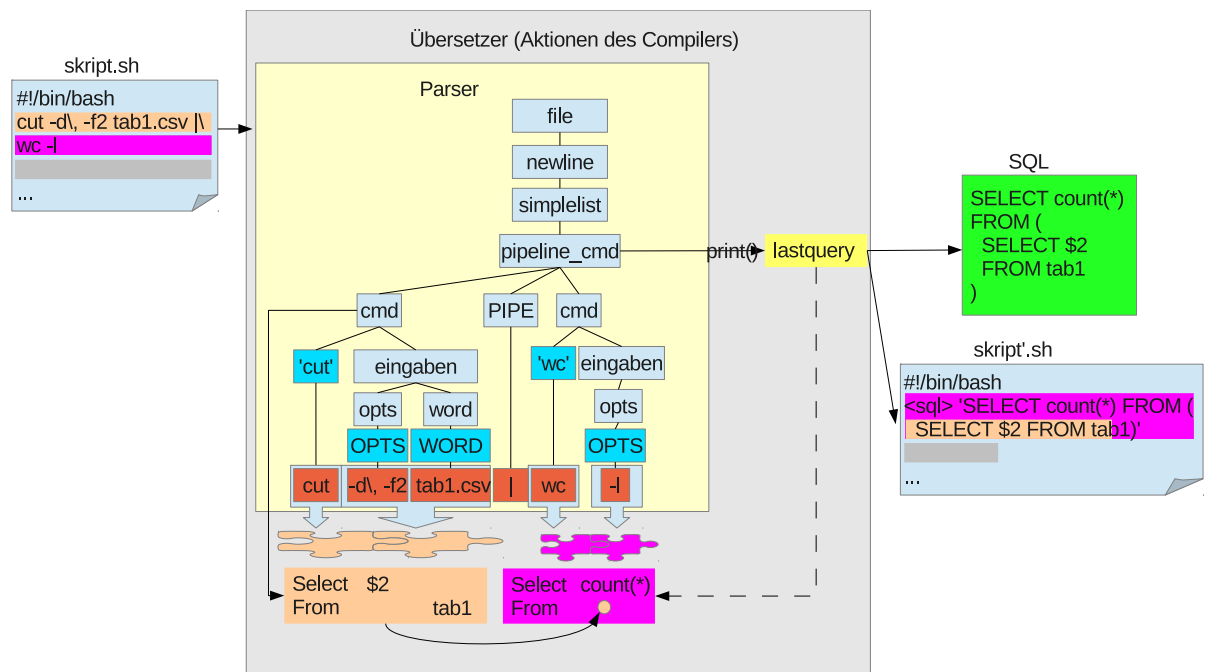


Abbildung 6.1.: Vorheriges Zusammenführen der Befehle

wieder zusammensetzen, die Logik zum Erkennen der Befehle stünde dann in einer Methode durch Auswahl via case und nicht mehr als Teil des Compilers.

6.4. Fazit

Insgesamt ist es gelungen einen Vergleich aufzustellen und man sieht: der Umstieg auf SQL lohnt sich. Es gibt einen Weg, die Abfragen zu übersetzen, und auch noch genug Platz für Ideen und Arbeit dort hineinzustecken, um alle möglichen Skripte übersetzen zu können. Diese Arbeit hat gezeigt, dass sich der Umstieg auf SQL lohnt, und will ermutigen, diesen Weg weiter zu verfolgen. Diese Arbeit hat gezeigt, dass sich der Umstieg auf SQL lohnt, und will ermutigen, diesen Weg weiter zu verfolgen.

Appendix

A. TPC-H-Abfragen

A.1. Abfragen

A.1.1. Abfrage 1

```
#!/bin/bash
#2014-08-25
cat lineitem.tbl | sort -t\| -k9,10 | cat lineitem.csv - |\
awk -F\| '
    NR==1{ count=0; sum5=sum6=sum17=sum18=sum7=0;
        print $9, $10, "sum_qty", "sum_base_price", "sum_disc_price", "
            sum_charge", "avg_qty", "avg_price", "avg_disc", "
            count_order"
    }
    NR==2{g9=$9; g10=$10;}
    NR>2{
        if( g9!=$9 || g10!=$10 ){
            if(count)
                print g9,g10,sum5,sum6,sum17,sum18,sum5/count ,
                    sum6/count ,sum7/count , count ;
            g9=$9; g10=$10; count=0;
            sum5=sum6=sum17=sum18=sum7=0}
    }
    NR>1 && $11<="1998-09-02" {
        count++; sum5+=$5; sum6+=$6;
        sum17+=$(6*(1.0-$7)); sum18+=$(6*(1.0-$7)*(1.0+$8));
        sum7+=$7;
    }END{print g9,g10,sum5,sum6,sum17,
        sum18, sum5/count , sum6/count , sum7/count , count;
    }
' OFS=\\|
```

A.1.2. Abfrage 2

```
#!/bin/bash
# 2014-08-28
## R >< N >< S
# R >< N (regionkey)
cut nation.csv -d\\ -f1-3 > tmp1.csv
cut nation.tbl -d\\ -f1-3 | sort -t\\ -k3,3 >> tmp1.csv
cut region.csv -d\\ -f1 > tmp2.csv
cat region.tbl | grep "EUROPE" | cut -d\\ -f1 | cat tmp2.csv - | \
join --header -t\\ -1 3 -2 1 tmp1.csv - > tmpnrn.csv

# >< S (nationkey)
cut supplier.csv -d\\ -f1-7 > tmp1.csv
cut supplier.tbl -d\\ -f1-7 | sort -t\\ -k4,4 >> tmp1.csv
head -1 tmpnrn.csv > tmp2.csv
tail -n+2 tmpnrn.csv | sort -t\\ -k2,2 | cat tmp2.csv - | \
join --header -t\\ -1 4 -2 2 tmp1.csv - > tmpnrns.csv

# PS >< RNS (suppkey)
cut partsupp.csv -d\\ -f1,2,4 > tmp1.csv
cut partsupp.tbl -d\\ -f1,2,4 | sort -t\\ -k2,2 >> tmp1.csv
head -1 tmpnrns.csv > tmp2.csv
tail -n+2 tmpnrns.csv | sort -t\\ -k2,2 | cat tmp2.csv - | \
join --header -t\\ -1 2 -2 2 tmp1.csv - > join1.csv
# group by partkey (2): ps_supplycost = select min(ps_supplycost) (3)
head -1 join1.csv > tmp1.csv
tail -n+2 join1.csv | sort -t\\ -k2,2 | cat tmp1.csv - | awk -F\\ '
    NR==1{print $2, "min("$3")"}
    NR==2{g2=$2; min=$3}
    NR>2{
        if( g2==$2 ){
            if(min>$3)
                min=$3
        }else{
            print g2, min;
            g2=$2; min=$3;
        }
    }
    END{print $2,min}
' OFS=\\ > join2.csv
# P >< PRNS (partkey)
#part: restrictions: p_size = 15; residuals: p_type like %BRASS
cut part.csv -d\\ -f1,2,3 > tmp1.csv
cat part.tbl | awk -F\\ '
    $6==15 && $5~/.*BRASS/ {print $1,$2,$3}' OFS=\\ | \
    sort -t\\ -k1,1 >> tmp1.csv
head -1 join2.csv > tmp2.csv
tail -n+2 join2.csv | sort -t\\ -k1,1 | cat tmp2.csv - | \
join --header -t\\ -1 1 -2 1 tmp1.csv - > join3.csv

# >< PS (partkey)
cut partsupp.csv -d\\ -f1,2,4 > tmp1.csv
cut partsupp.tbl -d\\ -f1,2,4 | sort -t\\ -k1,1 | cat tmp1.csv - | \
join --header -t\\ -1 1 -2 1 - join3.csv | \
awk -F\\ 'NR==1 || $3==$6 {print $0}' > join5.csv
```



```
# RNS <> P PS (suppkey)
head -1 tmpsns.csv > tmp1.csv
tail -n+2 tmpsns.csv | sort -t\\ -k2,2 >> tmp1.csv
head -1 join5.csv > tmp2.csv
tail -n+2 join5.csv | sort -t\\ -k2,2 | cat tmp2.csv - \\
join --header -t\\ -1 2 -2 2 tmp1.csv - \\
awk -F\\ '{print $6,$3,$9,$10,$13,$4,$5,$7}' OFS=\\ > join6.csv

#s_acctbal,s_name,s_name,s_partkey,s_mfgr,s_address,s_phone,s_comment
head -1 join6.csv
# sort s_acctbal desc,n_name,s_name,p_partkey
tail -n+2 join6.csv | sort -t\\ -k1,1 -snr
rm join{1,2,3,5,6}.csv tmp{1,2,rn,rns}.csv
```

A.1.3. Abfrage 3

```
#!/bin/bash
#2014-08-26
#C><O><L
#C><O (custkey) && o_orderdate<1995..
sort -t\\ -k2,2 orders.tbl | cat orders.csv - | awk -F\\ '
    NR==1 || $5<"1995-03-15"{print $1, $2, $5, $8}
' OFS=\\ > tmporder.csv
grep BUILDING customer.tbl | sort -t\\ -k1,1 | cut -d\\ -f1,2 customer.csv - |\\
join --header -t\\ -1 2 -2 1 tmporder.csv - > join1.csv

# >< L (orderkey) && l_shipdate>1995...
head -1 join1.csv > tmp1.csv
tail -n+2 join1.csv | sort -t\\ -k2,2 >> tmp1.csv
cat lineitem.tbl | sort -t\\ -k1,1 | cat lineitem.csv - | awk -F\\ '
    NR==1 || $11>"1995-03-15"{
        print $1,$6,$7
    }' OFS=\\ |\\
join --header -t\\ -1 2 -2 1 tmp1.csv - > output.csv

# gruppieren und sortieren
head -1 output.csv > tmp1.csv
tail -n+2 output.csv | sort -t\\ -k4,5 | sort -s -t\\ -k1,1 | cat tmp1.csv - |
awk -F\\ '
    NR==1{print $1,"sum(revenue)", $3, $4}
    NR==2{g1=$1; g3=$3; g4=$4; sum=($6*(1.0-$7))}
    NR>2{
        if(g1==$1 && g3==$3 && g4==$4 ){
            sum+=($6*(1.0-$7))
        }else{
            print g1, sum, g3, g4;
            g1=$1; g3=$3; g4=$4; sum=($6*(1.0-$7))
        }
    }
    END{print g1, sum, g3, g4}
' OFS=\\ > tmp3.csv
head -1 tmp3.csv
tail -n+2 tmp3.csv | sort -t\\ -k3,3 | sort -s -t\\ -k2,2 -nr | head -10

rm tmporder.csv tmp1.csv tmp3.csv output.csv
```

A.1.4. Abfrage 4

```
#!/bin/bash
#2014-08-25
sort -k1,1 -t\\ | orders.tbl | cat orders.csv - | awk -F\\ | '
    NR==1{
        print $1"|" $6
    }
    NR>1 && $5<"1993-10-01" && $5>="1993-07-01"{
        print $1"|" $6
    }
' > tmporder.csv

# and exists lineitem cdate<receiptdate
sort -k1,1 -t\\ | lineitem.tbl | cat lineitem.csv - | awk -F\\ | '
    NR==1 || $12<$13{
        print $1
    }
' | uniq \\
join --header -t\\ | -1 1 -2 1 tmporder.csv - > tmp.csv

# count mittels awk
# group by $2 (orderprio)
head -1 tmp.csv > tmp1.csv
tail -n+2 tmp.csv | sort -t\\ | -k2,2 | cat tmp1.csv - | awk -F\\ | '
    NR==1{print $2,"order_count"}
    NR==2{g2=$2; count=1}
    NR>2{
        if( g2==$2 ){
            count++
        }else{
            print g2, count;
            g2=$2;count=1;
        }
    }
    END{print g2,count}
' OFS=\\ | | cat

rm tmp*.csv
```

A.1.5. Abfrage 5

```
#!/bin/bash
# 2014-08-27
## R >< N >< C >< O >< L
# R >< N (regionkey)
cut nation.csv -d\\ -f1-3 > tmp1.csv
cut nation.tbl -d\\ -f1-3 | sort -t\\ -k3,3 >> tmp1.csv
cat region.tbl | grep "ASIA" | cut -d\\ -f1 | sort | cut -d\\ -f1 region.csv -
\\
join --header -t\\ -1 3 -2 1 tmp1.csv - > tmp1rnc.csv
# >< C (nationkey)
cut customer.csv -d\\ -f1,4 > tmp1.csv
cut customer.tbl -d\\ -f1,4 | sort -t\\ -k2,2 >> tmp1.csv
head -1 tmp1rnc.csv > tmp2.csv
tail -n+2 tmp1rnc.csv | sort -t\\ -k2,2 | cat tmp2.csv - \\
join --header -t\\ -1 2 -2 2 tmp1.csv - > tmp1rnc.csv
# >< O (custkey)
cut orders.csv -d\\ -f1,2,4 > tmp1.csv
cat orders.tbl | awk -F\\ '$5>="1994-01-01" && $5<"1995-01-01"{print $0}' | cut -
d\\ -f1,2,4 | sort -t\\ -k2,2 >> tmp1.csv
head -1 tmp1rnc.csv > tmp2.csv
tail -n+2 tmp1rnc.csv | sort -t\\ -k2,2 | cat tmp2.csv - \\
join --header -t\\ -1 2 -2 2 tmp1.csv - > join1.csv
# >< L (orderkey)
cut lineitem.csv -d\\ -f1,3,6,7 > tmp1.csv
cut lineitem.tbl -d\\ -f1,3,6,7 | sort -t\\ -k1,1 >> tmp1.csv
head -1 join1.csv > tmp2.csv
tail -n+2 join1.csv | sort -t\\ -k2,2 | cat tmp2.csv - \\
join --header -t\\ -1 1 -2 2 tmp1.csv - > join2.csv
# >< S (suppkey) && c_nationkey==s_nationkey
cut supplier.csv -d\\ -f1,4 > tmp1.csv
cut supplier.tbl -d\\ -f1,4 | sort -t\\ -k1,1 >> tmp1.csv
head -1 join2.csv > tmp2.csv
tail -n+2 join2.csv | sort -t\\ -k2,2 | cat tmp2.csv - \\
join --header -t\\ -1 1 -2 2 tmp1.csv - | awk -F\\ '
NR==1 || $2==S8{print $0}
' > join3.csv
# group by n_name, sum revenue
head -1 join3.csv > tmp1.csv
tail -n+2 join3.csv | sort -t\\ -k10,10 | cat tmp1.csv - | awk -F\\ '
NR==1{print $10,"sum(revenue)"}
NR==2{g10=$10; sum=$4*(1.0-$5)}
NR>2{
    if( g10==$10 ){
        sum+=$4*(1.0-$5)
    }else{
        print g10, sum;
        g10=$10;sum=$4*(1.0-$5);
    }
}
END{print g10,sum}
' OFS=\\ > join4.csv
head -1 join4.csv
tail -n+2 join4.csv | sort -t\\ -k2,2 -r
```

```
rm join {1,2,3,4}.csv tmp{1,2,rn,rnc}.csv
```

A.1.6. Abfrage 6

```
#!/bin/bash
#2014-08-27
cat lineitem.* | awk -F\| '
    NR==1{
        print "revenue"; sum=0;
    }
    NR>1 && $11>="1994-01-01" && $11<"1995-01-01" && $7>=0.05 && $7<=0.07 &&
        $5<24{
            sum+=$6*$7;
        }
    END{print sum}
,
```

A.1.7. Abfrage 7

```
#!/bin/bash
#2014-08-27
# N x N >< C >< O >< L >< S
# NxN (Germany, France)
sort -t\\ -k1,1 nation.tbl | cut nation.csv - -d\\ -f1,2 | awk -F\\ '
    NR==1{print "cust_n_key|cust_n_name|supp_n_key|supp_n_name"}
    NR>1 && ($2=="FRANCE" || $2=="GERMANY"){
        lines[i++]=$0
    }
    END{
        for (i in lines)
            for (j in lines)
                print lines[i] "|" lines[j]
    }
' | awk -F\\ 'NR==1 ||
    ($2=="FRANCE" && $4=="GERMANY") ||
    ($4=="FRANCE" && $2=="GERMANY"){print $0}'> tmpnation.csv

# >< C (nationkey)
cat customer.tbl | sort -t\\ -k4,4 | cut -d\\ -f1,4 customer.csv - |\\
join --header -t\\ -1 1 -2 2 tmpnation.csv - > join1.csv

# >< O (custkey)
head -1 join1.csv > tmp1.csv
tail -n+2 join1.csv | sort -t\\ -k5,5 >> tmp1.csv
sort -t\\ -k2,2 orders.tbl | cut -d\\ -f1,2 orders.csv - |\\
join --header -t\\ -1 5 -2 2 tmp1.csv - > join2.csv

# >< L (orderkey)
head -1 join2.csv > tmp1.csv
tail -n+2 join2.csv | sort -t\\ -k6,6 >> tmp1.csv
cat lineitem.tbl | awk -F\\ '
    $11>="1995-01-01" && $11 <="1996-12-31"{
        print $0
    }
' OFS=\\ | sort -t\\ -k1,1 | cut -d\\ -f1,3,6,7,11 lineitem.csv - |\\
join --header -t\\ -1 6 -2 1 tmp1.csv - > join3.csv

# >< S (nationkey, suppkey)
head -1 join3.csv > tmp1.csv
tail -n+2 join3.csv | sort -t\\ -k7,7 >> tmp1.csv
cat supplier.tbl | sort -t\\ -k1,1 | cut -d\\ -f1,4 supplier.csv - |\\
# -k5,5
join --header -t\\ -1 7 -2 1 tmp1.csv - | awk -F\\ '
    NR>1 && $6==$11{
        print $7, $5, substr($10,1,4), $8*(1.0-$9)
    }
' OFS=\\ > join4.csv
cat join4.csv | sort | awk -F\\ '
    BEGIN{print "supp_nation|cust_nation|l_year|revenue"}
    NR==1{g1=$1; g2=$2; g3=$3; sum=$4}
    NR>1{
        if ( g1==$1 && g2==$2 && g3==$3 ){
            sum+=$4
        }
    }
}
```

```
        } else {
            print g1,g2,g3,sum;
            g1=$1; g2=$2; g3=$3; sum=$4;
        }
    }
END{ print g1,g2,g3,sum}
' OFS=\\

rm tmp1.csv join {1,2,3}.csv tmpnation.csv
```


A.1.8. Abfrage 8

```
#!/bin/bash
#2014-08-28
# (R << N1) << ((P << L) << O << C)
# << S
# << N2

# N1 << R (regionkey)
head -1 region.csv | cut -d\| -f1 > tmpregion.csv
grep '|AMERICA|' region.tbl | cut -d\| -f1 | sort >> tmpregion.csv
cut -d\| -f1-3 nation.tbl | sort -t\| -k3,3 | cut nation.csv -d\| -f1,3 | tee
    tmpnation.csv | \
join --header -t\| -1 1 -2 2 tmpregion.csv - > tmpnr.csv

# P << L (partkey)
cat lineitem.tbl | sort -t\| -k2,2 | cut -d\| -f1,2,3,6,7 lineitem.csv - > tmp1.
    csv
cat part.tbl | grep 'ECONOMY ANODIZED STEEL' | cut -d\| -f1 | sort | cut part.csv
    -d\| -f1 | \
join --header -t\| -1 2 -2 1 tmp1.csv - > join2.csv

# << O (orderkey)
head -1 join2.csv > tmp1.csv
tail -n+2 join2.csv | sort -t\| -k2,2 >> tmp1.csv
cut orders.tbl -d\| -f1-5 | sort -t\| -k1,1 | cat orders.csv - | awk -F\| '
    NR==1 || ( $5>="1995-01-01" && $5<="1996-12-31"){
        print $1 "|" $2 "|" $5
    }
' | \
join --header -t\| -1 2 -2 1 tmp1.csv - > join3.csv

# << C (custkey)
head -1 join3.csv > tmp1.csv
tail -n+2 join3.csv | sort -t\| -k6,6 >> tmp1.csv
cut -d\| -f1-4 customer.tbl | sort -t\| -k1,1 | cut -d\| -f1,4 customer.csv - | \
join --header -t\| -1 6 -2 1 tmp1.csv - > join4.csv

# (N1<<R) << (nationkey)
head -1 join4.csv > tmp1.csv
tail -n+2 join4.csv | sort -t\| -k8,8 >> tmp1.csv
head -1 tmpnr.csv > tmp2.csv
tail -n+2 tmpnr.csv | sort -t\| -k2,2 | cat tmp2.csv - | \
join --header -t\| -1 8 -2 2 tmp1.csv - > join5.csv

# << S (suppkey)
head -1 join5.csv > tmp1.csv
tail -n+2 join5.csv | sort -t\| -k5,5 >> tmp1.csv
cut -d\| -f1-4 supplier.tbl | sort -t\| -k1,1 | cut -d\| -f1-4 supplier.csv - | \
join --header -t\| -1 5 -2 1 tmp1.csv - > join6.csv

# << N2 (s_nationkey==n2_nationkey)
head -1 join6.csv > tmp1.csv
tail -n+2 join6.csv | sort -t\| -k12,12 >> tmp1.csv
cut -d\| -f1,2 nation.tbl | sort -t\| -k1,1 | cut nation.csv -d\| -f1,2 | \
join --header -t\| -1 12 -2 1 tmp1.csv - | \
```

```
#extract(year from o_orderdate) as o_year,
#l_extendedprice * (1 - l_discount) as volume,
awk -F\\| '
    NR==1{print "nation|year|volume"}
    NR>1{print $13,substr($9,1,4),$7*(1.0-$8) }
' OFS=\\| > join7.csv

#group by $2 year; 10: $1 nation; $3 volume
head -1 join7.csv > tmp1.csv
tail -n+2 join7.csv | sort -t\\| -k2,2 | cat tmp1.csv - | awk -F\\| '
    NR==1{print $2,"mkt_share"}
    NR==2{g2=$2; sumt=$3; sumb=0}
    NR>2{
        if(g2==$2){
            sumt+=$3
        }else{
            print g2,sumb/sumt;
            g2=$2; sumt=$3; sumb=0;
        }
    }
    $1=="BRAZIL"{sumb+=$3}
    END{print g2, sumb/sumt}
' OFS=\\|

rm tmp*.csv join{2,3,4,5,6,7}.csv
```

A.1.9. Abfrage 9

```
#!/bin/bash
# (N >< S) >< (P >< PS)
# >< L
# >< O

#echo 'N >< S (nationkey)'
cut -d\| -f1,2 nation.tbl | sort -t\| -k1,1 | cut -d\| -f1,2 nation.csv -> tmp1.csv
cut -d\| -f1-4 supplier.tbl | sort -t\| -k4,4 | cut -d\| -f1,4 supplier.csv - \|
join --header -t\| -1 1 -2 2 tmp1.csv -> tmpsn.csv

# P >< PS (partkey)
cat part.tbl | grep 'green'| cut -d\| -f1 | sort -t\| -k1,1 | cut -d\| -f1 part.csv -> tmp1.csv
cat partsupp.tbl | sort -t\| -k1,1 | cut -d\| -f1,2,4 partsupp.csv - \|
join --header -t\| -1 1 -2 1 tmp1.csv -> tmppps.csv

# SN >< PPS (suppkey)
head -1 tmppps.csv > tmp1.csv
tail -n+2 tmppps.csv | sort -t\| -k2,2 >> tmp1.csv
head -1 tmpsn.csv > tmp2.csv
tail -n+2 tmpsn.csv | sort -t\| -k3,3 | cat tmp2.csv - \|
join --header -t\| -1 2 -2 3 tmp1.csv -> join1.csv

# >< L (suppkey, dann partkey gleich?)
cut lineitem.tbl -d\| -f1-7 | sort -t\| -k3,3 | cut -d\| -f1,2,3,5,6,7 lineitem.csv - \|
join --header -t\| -1 3 -2 1 - join1.csv | awk -F\| '
NR==1 || $3==$7 {print $0}' > join2.csv

# >< O (orderkey)
head -1 join2.csv > tmp1.csv
tail -n+2 join2.csv | sort -t\| -k2,2 >> tmp1.csv
cat orders.tbl | sort -t\| -k1,1 | cut -d\| -f1,5 orders.csv - \|
join --header -t\| -1 2 -2 1 tmp1.csv - | tee join3.csv \|

#_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
# extract year
awk -F\| '
NR==1{
    print $10 "|year|amount"
}
NR>1{
    sum=$5*(1.0-$6)-$8*$4
    print $10 "|" substr($11,1,4) "|" sum
}
'> join4.csv

# group by $6 (nation), year, sum_profit
head -1 join4.csv > tmp1.csv
tail -n+2 join4.csv | sort -r -t\| -k2,2 | sort -t\| -s -k1,1 | cat tmp1.csv -
| awk -F\| '
NR==1{print $1, $2, "sum_profit"}
NR==2{g1=$1; g2=$2; sum=$3}
```

```
NR>2{
    if (g1==$1 && g2==$2){
        sum+=$3
    } else {
        print g1,g2,sum;
        g1=$1; g2=$2; sum=$3;
    }
}
END{print g1,g2,sum}
' OFS=\\

rm tmp*.csv join {1,2,3,4}.csv
```

A.1.10. Abfrage 10

```

#!/bin/bash
#2014-08-28
# ((O >< C) >< N) >< L
# O >< C (custkey)
awk -F\| ' $5>="1993-10-01" && $5<"1994-01-01"{print $1 "|" $2}
' orders.tbl | sort -t\| -k2,2 | cut -d\| -f1,2 orders.csv - > tmp1.csv
cut -d\| -f1-8 customer.tbl | sort -t\| -k1,1 | cut -d\| -f1-6,8 customer.csv -
\|
join --header -t\| -1 2 -2 1 tmp1.csv - > join1.csv

# >< N (nationkey)
head -1 join1.csv > tmp1.csv
tail -n+2 join1.csv | sort -t\| -k5,5 >> tmp1.csv
cut -d\| -f1,2 nation.tbl | sort -t\| -k1,1 | cut -d\| -f1,2 nation.csv - \|
join --header -t\| -1 5 -2 1 tmp1.csv - > join2.csv

# >< L (orderkey)
head -1 join2.csv > tmp1.csv
tail -n+2 join2.csv | sort -t\| -k3,3 >> tmp1.csv
cut -d\| -f1,6,7 lineitem.csv > tmp2.csv
awk -F\| ' $9=="R"{print $1,$6,$7}' OFS=\| lineitem.tbl | sort -t\| -k1,1 | cat
tmp2.csv - \|
join --header -t\| -1 3 -2 1 tmp1.csv - > join3.csv

# group by, sum(revenue)
#c_custkey,c_name,c_acctbal,c_phone,n_name,c_address,c_comment
#sum(l_extendedprice * (1 - l_discount)) as revenue
head -1 join3.csv > tmp1.csv
tail -n+2 join3.csv | sort -r -t\| -k3,9 | cat tmp1.csv - | awk -F\| '
NR==1{print $3, $4, $7, $6, $9, $5, $8, "sum(revenue)"}
NR==2{g3=$3;g4=$4;g7=$7;g6=$6;g2=$2;g5=$5;g8=$8;sum=$10*(1.0-$11)}
NR>2{
    if (g3==$3 && g4==$4 && g7==$7 && g6==$6 &&
        g9==$9 && g5==$5 && g8==$8){
        sum+=$10*(1.0-$11)
    } else {
        print g3, g4, g7, g6, g9, g5, g8, sum;
        g3=$3;g4=$4;g7=$7;g6=$6;g9=$9;g5=$5;g8=$8;sum=$10*(1.0-
            $11);
    }
}
END{print g3, g4, g7, g6, g9, g5, g8, sum}
' OFS=\| > tmp.csv

head -1 tmp.csv;
tail -n+2 tmp.csv | sort -t\| -nrk8 | head -20

rm tmp*.csv join{1,2,3}.csv

```

A.1.11. Abfrage 11

```
#!/bin/bash
#2014-08-24
echo 'N >< S'
cut -d\\ -f1,4 supplier.csv > tmp1.csv
cut -d\\ -f1,4 supplier.tbl | sort -t\\ -k2,2 >> tmp1.csv
cat nation.tbl | grep 'GERMANY' | cut -d\\ -f1 | sort | cut -d\\ -f1 nation.csv
- \\
join --header -t\\ -1 2 -2 1 tmp1.csv - > tmpsn.csv

echo 'PS >< (partkey)'
head -1 tmpsn.csv | cut -d\\ -f2 > tmp2.csv
tail -n+2 tmpsn.csv | cut -d\\ -f2 | sort >> tmp2.csv
cut -d\\ -f1,2,3,4 partsupp.tbl | sort -t\\ -k2,2 | cut -d\\ -f1-4 partsupp.csv
- \\
join --header -t\\ -1 2 -2 1 - tmp2.csv > join.csv

#echo 'sum(ps_supplycost * ps_availqty) * 0.0001'
sum='awk -F\\ | '
    BEGIN{SUM=0}
    NR>1{SUM+=$3*$4}
    END{print SUM*0.0001}
' join.csv '
#echo $sum

# 'sum(ps_supplycost * ps_availqty); sum(ps_supplycost * ps_availqty)>$sum'
head -1 join.csv > tmp1.csv
tail -n+2 join.csv | sort -r -t\\ -k2,2 | cat tmp1.csv - | awk -F\\ -v min=$sum
,
    NR==1{print $2,"value"}
    NR==2{g2=$2;sum=$3*$4}
    NR>2{
        if(g2==$2 ){
            sum+=$3*$4
        } else {
            if (sum>min)
                printf ("%d\\%d\\n",g2, sum);
            g2=$2;sum=$3*$4;
        }
    }
    END{print g2, sum}
' OFS=\\ | > tmp.csv
head -1 tmp.csv
tail -n+2 tmp.csv | sort -t\\ -k2,2 -nr

rm tmp*.csv join.csv
```

A.1.12. Abfrage 12

```

#!/bin/bash
#2014-08-25
# O >< L (orderkey)
#      o_orderkey = l_orderkey
#      $15      and l_shipmode in ('MAIL', 'SHIP')
#      $12<$13   and l_commitdate < l_receiptdate
#      $11<12    and l_shipdate < l_commitdate
#      $13      and l_receiptdate >= date '1994-01-01'
#      $13      and l_receiptdate < date '1995-01-01'
sort -t\\ -k1,1 orders.tbl\\
cat orders.csv - | awk -F\\ | '
    NR==1{
        print $1 "|high_line_count|low_line_count"
    }
    NR>1{
        if ( $6=="1-URGENT" || $6=="2-HIGH" )
            print $1 "|1|0"
        else
            print $1 "|0|1"
    }
' > tmporder.csv

sort -t\\ -k1,1 lineitem.tbl |\\
cat lineitem.csv - | awk -F\\ | '
    NR==1{
        print $1 "|" $15
    }
    NR>1 && $13>="1994-01-01" && $13<"1995-01-01" && $12<$13 && $11<$12 && (
        $15=="MAIL" || $15=="SHIP"){
        print $1 "|" $15
    }
' |\\

join --header -t\\ -1 1 -2 1 tmporder.csv - > join.csv

#group by shipmode ($4), sum highline, lowline
head -1 join.csv > tmp1.csv
tail -n+2 join.csv | sort -t\\ -k4,4 | cat tmp1.csv - | awk -F\\ | '
    NR==1{print $4,"high_line_count", "low_line_count"}
    NR==2{g4=$4;sumhigh=$2;sumlow=$3}
    NR>2{
        if(g4==$4){
            sumlow+=$3; sumhigh+=$2
        } else {
            print g4, sumhigh, sumlow;
            g4=$4;sumhigh=$2;sumlow=$3;
        }
    }
    END{print g4, sumhigh, sumlow}
' OFS=\\

rm tmp*.csv

```

A.1.13. Abfrage 13

```
#!/bin/bash
#2014-08-26
echo 'c_count|custdist'
# C l>< O
# left outer join ^= join -a1 -a2 -1 2 -2 2 -o 0 1.1 2.1 -e "0" 1.txt 2.txt
#join -a1 -a2 -1 2 -2 2 -o 0 1.1 -e "0" 1.txt 2.txt
head -1 orders.csv | cut -d\| -f2 > tmp2.csv
cat orders.tbl |grep -v 'special.*requests' | cut -d\| -f2 | sort >> tmp2.csv
cut -d\| -f1,2 customer.tbl | sort -t\| -k1,1 | cut -d\| -f1,2 customer.csv - |\
join --header -t\| -1 1 -2 1 -a1 -o 0 2.1 -e "NULL" - tmp2.csv |\
awk -F\| '
    NR==1{print $1,"count"}
    NR==2{g1=$1; count=0}
    NR>2{
        if(g1!=$1){
            print g1, count;
            g1=$1;count=0;
        }
        $2!="NULL"{count++}
    END{print g1, count}
' OFS=\| | tail -n+2 | sort -t\| -k2,2 | awk -F\| '
    NR==1{g2=$2;count=1}
    NR>1{
        if(g2==$2){
            count++;
        }else{
            print g2, count;
            g2=$2;count=1;
        }
    }
    END{print g2, count}
' OFS=\| | sort -t\| -k2,2 -nr

rm tmp2.csv
```


A.1.14. Abfrage 14

```
#!/bin/bash
#2014-08-28
# P >< L (partkey)
cut lineitem.csv -d\| -f2,6,7 > tmp2.csv
cat lineitem.tbl | awk -F\| '
    $11>="1995-09-01" && $11<"1995-10-01"{
        print $2,$6,$7
    }
' OFS=\| | sort -t\| -k1,1 >> tmp2.csv

cut -d\| -f1-5 part.tbl | sort -t\| -k1,1 | cut -d\| -f1,5 part.csv - |\
join --header -t\| -1 1 -2 1 - tmp2.csv |\
awk -F\| '
    BEGIN{sum=0; sumpromo=0; print "promo_revenue"}
    {
        rev=$3*(1.0-$4);
        sum+=rev
    }
    $2~/PROMO.*/{sumpromo+=rev}
    END{print sumpromo/sum*100}
'
rm tmp2.csv
```

A.1.15. Abfrage 15

```
#!/bin/bash
#revenue.csv
cut lineitem.tbl -d\\ -f1-11 | sort -t\\ -k3,3 | cut lineitem.csv - -d\\ -f3
,6,7,11 | awk -F\\ '
    NR==1{
        print $1, "total_revenue"
    }
    NR==2{g1=$1; sum=0; i=0; max=0}
    NR>2{
        if (g1!=$1 && sum>0){
            line[i,0]=g1
            line[i++,1]=sum;
            sum=0;
        }
        g1=$1;
    }
    NR>1 && $4>="1996-01-01" && $4<"1996-04-01" {
        sum+=$(2*(1.0-$3));
        max=(max<sum)?sum:max;
    }
    END{
        line[i,0]=g1; line[i++,1]=sum;
        for (j=0; j<i; j++)
            if (line[j,1]==max)
                print line[j,0], line[j,1];
    }
' OFS=\\ > tmprev.csv
# S >< revenue.csv
sort -t\\ -k1,1 supplier.tbl |
cut -d\\ -f1,2,3,5 supplier.csv - |\\
join --header -t\\ -1 1 -2 1 - tmprev.csv
rm tmprev.csv
```

A.1.16. Abfrage 16

```

#!/bin/bash
# 2014-08-28
# PS >< P
# $4 p_brand <> 'Brand#45'
# $5      and p_type not like 'MEDIUM POLISHED%'
# $6      and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
cat part.tbl | sort -t\| -k1,1 | cat part.csv - | awk -F\| '
    NR==1{
        print $1,$4,$5,$6
    }
    NR>1 && $4!="Brand#45" && $5~/^MEDIUM POLISHED.*/ && ($6==49 || $6==14
        || $6==23 || $6==45 || $6==19 || $6==3 || $6==36 || $6==9){
        print $1,$4,$5,$6
    }
' OFS=\| > tmppart.csv
cat partsupp.tbl | sort -t\| -k1,1 | cut -d\| -f1-4 partsupp.csv - \|
join --header -t\| -1 1 -2 1 - tmppart.csv > join1.csv

# PS |> S (suppkey)
head -1 join1.csv | cut -d\| -f1,2,5,6,7,8 > tmp1.csv
tail -n+2 join1.csv | cut -d\| -f1,2,5,6,7,8 | sort -t\| -k2,2 >> tmp1.csv
sed -r '/Customer.*Complaints/ d' supplier.tbl | cut -d\| -f1 | sort | cut -d\|
    -f1 supplier.csv - \|
join --header -t\| -1 2 -2 1 tmp1.csv - > join2.csv

#group by p_brand, p_type, p_size; count(distinct ps_suppkey)
head -1 join2.csv > tmp1.csv
tail -n+2 join2.csv | sort | sort -s -t\| -k3,5 | cat tmp1.csv - | awk -F\| '
    NR==1{print $3,$4,$5, "uniqcount("$1)"}
    NR==2{g3=$3;g4=$4;g5=$5;c1=$1;count=1}
    NR>2{
        if(g3==$3 && g4==$4 && g5==$5){
            if(c1!= $1) count++;
            c1=$1;
        } else{
            print g3,g4,g5,count;
            g3=$3;g4=$4;g5=$5;c1=$1;count=1;
        }
    }
    END{print g3,g4,g5,count}
' OFS=\| > tmpout.csv

# supplier_cnt desc, p_brand, p_type, p_size
head -1 tmpout.csv; tail -n+2 tmpout.csv | sort -s -t\| -nr -k4,4

rm tmp*.csv join1.csv join2.csv

```

A.1.17. Abfrage 17

```
#!/bin/bash
# 2014-08-19
## L >< P (partkey)
cut -d\| -f1,4,7 part.tbl | grep 'Brand#23' | grep 'MED BOX' | sort -t\| -k1,1 |
    cat <(cut -d\| -f1,4,7 part.csv) - \|
join --header -t\| -1 1 -2 1 \
<(cut -d\| -f2,5,6 lineitem.tbl | sort -t\| -k1,1 \|
cat <(cut -d\| -f2,5,6 lineitem.csv) -) - | tee join1.csv \|

#avg(l_quantity) pro partkey
awk -F\| '
    NR==1{print $1,"avg("$2)"}
    NR==2{g1=$1;sum=$2;count=1}
    NR>2{
        if(g1==$1){
            sum+=$2;
            count++;
        }else{
            print g1,sum/count;
            g1=$1; sum=$2; count=1;
        }
    }
    END{print g1,sum/count}
' OFS=\| > tmpavgs.csv

#dann join mit join1
join --header -t\| -1 1 -2 1 join1.csv tmpavgs.csv \|
#greife die relevanten raus
awk -F\| '
    BEGIN{
        print "avg_yearly"
        sum=0
    }
    NR>1 && $2<$6*0.2 {sum+=$3}
    END{ print sum/7.0 }
,

rm tmpavgs.csv join1.csv
```

A.1.18. Abfrage 18

```
#!/bin/bash
# 2014-08-20
# group by l_orderkey, having sum quantity >300
cut -d\| -f1-5 lineitem.tbl | sort -t\| -k1,1 | cut -d\| -f1,5 lineitem.csv - |
  awk -F\| '
    NR==1{print $1,"sum("$2")"}
    NR==2{g1=$1;sum=$2}
    NR>2{
      if (g1==$1){
        sum+=$2;
      } else {
        if (sum>300)
          print g1,sum;
        g1=$1; sum=$2;
      }
    }
    END{print g1,sum}
  ' OFS=\| > tmp2.csv
# O >< L (orderkey)
sort -t\| -k1,1 orders.tbl | cut -d\| -f1-9 orders.csv - | \
join --header -t\| -1 1 -2 1 - tmp2.csv > join1.csv
# >< C (custkey)
#c_name,c_custkey,o_orderkey,o_orderdate,o_totalprice
head -1 join1.csv | cut -d\| -f1,2,4,5,10 > tmp2.csv
tail -n+2 join1.csv | cut -d\| -f1,2,4,5,10 | sort -t\| -k2,2 >> tmp2.csv
sort -t\| -k1,1 customer.tbl | cut -f1,2 customer.csv - | \
join --header -t\| -1 1 -2 2 -o 1.2 0 2.1 2.4 2.3 2.5 - tmp2.csv > join2.csv

head -1 join2.csv
tail -n+2 join2.csv | sort -t\| -k6 | sort -t\| -k5,5 -sr | head -100

rm tmp2.csv join1.csv join2.csv
```

A.1.19. Abfrage 19

```
#!/bin/bash
# 2014-08-28
#l_shipmode in ('AIR', 'AIR REG') and l_shipinstruct = 'DELIVER IN PERSON'
cat lineitem.tbl | awk -F\| '
    $14=="DELIVER_IN_PERSON" && ($15=="AIR" || $15=="AIR_REG"){
        suma=($6*(1.0-$7));
        print $2,$5,$6,$7
    }
' OFS=\| | sort -t\| -k1,1 | cat <(cut -d\| -f2,5,6,7 lineitem.csv) -> tmp1.csv
# P >< L (partkey)
cut -d\| -f1-9 part.tbl | grep -E "Brand#(12|23|34)" |\
grep -E "(SM|MED|LG)" | grep -E "(CASE|BOX|PACK|PKG|BAG)" |\
sort -t\| -k1,1 | cut -f1-9 part.csv - |\
join --header -t\| -1 1 -2 1 tmp1.csv - | awk -F\| '
    BEGIN{ print "revenue"; sum=0}
    $7=="Brand#12" &&
    ($10=="SM_CASE" || $10=="SM_BOX" ||
    $10=="SM_PACK" || $10=="SM_PKG") &&
    $2>=1 && $2<=11 &&
    $9<=5 && $9>=1{
        sum+=$3*(1.0-$4)
    }
    $7=="Brand#23" &&
    ($10=="MED_BAG" || $10=="MED_BOX" ||
    $10=="MED_PACK" || $10=="MED_PKG") &&
    $2>=10 && $2<=20 &&
    $9<=10 && $9>=1{
        sum+=$3*(1.0-$4)
    }
    $7=="Brand#34" &&
    ($10=="LG_CASE" || $10=="LG_BOX" ||
    $10=="LG_PACK" || $10=="LG_PKG") &&
    $2>=20 && $2<=30 &&
    $9<=15 && $9>=1{
        sum+=$3*(1.0-$4)
    }
    END{print sum}
,
# and p_brand = 'Brand#12' and p_container in ('SM CASE', 'SM BOX', 'SM PACK', '
SM PKG') and l_quantity >= 1 and l_quantity <= 1 + 10
# and p_brand = 'Brand#23' and p_container in ('MED BAG', 'MED BOX', 'MED PKG',
'MED PACK') and l_quantity >= 10 and l_quantity <= 10 + 10 and p_size between
1 and 10
# and p_brand = 'Brand#34' and p_container in ('LG CASE', 'LG BOX', 'LG PACK', '
LG PKG') and l_quantity >= 20 and l_quantity <= 20 + 10 and p_size between 1
and 15
rm tmp1.csv
```

A.1.20. Abfrage 20

```

#!/bin/bash
# 2014-08-20
# P >< PS (partkey)
cat partsupp.tbl | sort -t\| -k1,1 | cut -d\| -f1-5 partsupp.csv -> tmp1.csv
awk -F\| '$2~/^forest/{print $1}' part.tbl | sort -t\| -k1,1 | cut -d\| -f1 part
.csv - \|
join --header -t\| -1 1 -2 1 tmp1.csv -> join1.csv

#l_shipdate >= date '1994-01-01' and l_shipdate < date '1995-01-01'
# and group by
cat lineitem.tbl | sort -t\| -k2,3 | sort -t\| -k2,2 -s | cut -d\| -f2,3,5,11
lineitem.csv - | awk -F\| '
    NR==1{print $1,$2,"sum("$3)"}
    NR==2{g1=$1;g2=$2;sum=0}
    NR>2{
        if(g1!=$1 || g2!=$2){
            print g1,g2,sum;
            g1=$1; g2=$2; sum=0;
        }
    }
    NR>1 && $4>="1994-01-01" && $4<"1995-01-01"{
        sum+=$3;
    }
    END{print g1,g2,sum}
' OFS=\| \|
# L >< (l_partkey = ps_partkey)
join --header -t\| -1 1 -2 1 join1.csv - \|
# l_suppkey = ps_suppkey
awk -F\| 'NR==1 || ($2==$6 && $3>$7*0.5){print $2}' > join2.csv

# ' S >< (suppkey in...) '
head -1 join2.csv > tmp2.csv
tail -n+2 join2.csv | sort ->> tmp2.csv
cat supplier.tbl | sort -t\| -k1,1 \|
cut -d\| -f1-7 supplier.csv - \|
join --header -t\| -1 1 -2 1 - tmp2.csv > join3.csv

# S >< N (nationkey)
head -1 join3.csv > tmp1.csv
tail -n+2 join3.csv | sort -t\| -k4,4 >> tmp1.csv
awk -F\| 'NR==1 || $2=="CANADA"{print $1}' nation.* \|
join --header -t\| -1 4 -2 1 tmp1.csv - | cut -d\| -f3,4 > join4.csv

head -1 join4.csv
tail -n+2 join4.csv | sort -u

rm tmp*.csv join*.csv

```

A.1.21. Abfrage 21

```
#!/bin/bash
# 2014-08-25
## N <> S <> L1 <> O |> L
echo ' N <> S '
cut -d\| -f1,2,4 supplier.csv > tmp1.csv
cut -d\| -f1,2,4 supplier.tbl | sort -t\| -k3,3 >> tmp1.csv
cat nation.tbl | grep 'SAUDI ARABIA' | cut -d\| -f1 nation.csv - \|
join --header -t\| -1 3 -2 1 tmp1.csv - > tmpsn.csv

echo ' <> L1 (suppkey) '
#l1.l_receiptdate > l1.l_commitdate
head -1 tmpsn.csv > tmp2.csv
tail -n+2 tmpsn.csv | sort -t\| -k2,2 >> tmp2.csv
awk -F\| '$13>$12 {print $1,$2,$3}' OFS=\| lineitem.tbl \|
sort -t\| -k3,3 | cut -d\| -f1,3 lineitem.csv - | tee tmp1.csv \|
join --header -t\| -1 2 -2 2 - tmp2.csv > join2.csv

echo ' <> O (orderkey) '
#l1.l_receiptdate > l1.l_commitdate
head -1 join2.csv > tmp2.csv
tail -n+2 join2.csv | sort -t\| -k2,2 >> tmp2.csv
awk -F\| '$3=="F" {print $1}' orders.tbl | sort | cut -d\| -f1 orders.csv - \|
join --header -t\| -1 1 -2 2 - tmp2.csv > join3.csv

echo ' |> L3 (orderkey) '
#l3.l_suppkey <> l1.l_suppkey
#awk -F\| 'NR==1 || $13>$12 {print $1,$3}' OFS=\| lineitem.csv > tmp1.csv
head -1 tmp1.csv > tmp2.csv
tail -n+2 tmp1.csv | sort -t\| -k1,1 | cat tmp2.csv - \|
join --header -t\| -1 1 -2 1 join3.csv - \|
awk -F\| 'NR==1 || $2!=$5 {print $0}' \|
join --header -t\| -1 1 -2 1 -v1 join3.csv - | cut -d\| -f1-4 > join4.csv

echo ' <> L2 ( l2.l_orderkey = l1.l_orderkey and l2.l_suppkey <> l1.l_suppkey) '
# gibt es vom jeweiligen l_orderkey mind. 2? group by, join
cut lineitem.tbl -d\| -f1,3 | sort -t\| -k1,1 | cut lineitem.csv - -d\| -f1,3 |
awk -F\| '
    NR==1{print $1, "count(*)"}
    NR==2{g1=$1;count=1}
    NR>2{
        if(g1==$1){
            count++;
        } else {
            if(count>1)
                print g1,count;
            g1=$1;count=1;
        }
    }
    END{print g1,count}
' OFS=\| \|
join --header -t\| -1 1 -2 1 join4.csv - > join5.csv
# group by 4 count *
head -1 join5.csv > tmp1.csv
tail -n+2 join5.csv | sort -t\| -k3 | cat tmp1.csv - | awk -F\| '

```



```
NR==1{print $4, "count(*)"}
NR==2{g4=$4;count=1}
NR>2{
    if(g4==$4){
        count++;
    }else{
        print g4,count;
        g4=$4;count=1;
    }
}
END{print g4,count}
' OFS=\| > join6.csv

head -1 join6.csv
tail -n+2 join6.csv | sort -t\| -snr -k2,2 | head -99

rm tmp*.csv join*.csv
```

A.1.22. Abfrage 22

```
#!/bin/bash
# 2014-08-26
avg='awk -F\\ | 'BEGIN{sum=0; cnt=0; print "avg"}
NR>1 && $6>0 &&
(substr($5,1,2)==13||substr($5,1,2)==31||
 substr($5,1,2)==23||substr($5,1,2)==29||
 substr($5,1,2)==30||substr($5,1,2)==18||
 substr($5,1,2)==17){
    sum+=$6; cnt+=1
}
END{print sum/cnt}
' customer.* | tail -1'

#c_acctbal > select avg(c_acctbal) from...
awk -F\\ | -v avg=$avg '
NR==1{print $1,$6, "centrycode"}
NR>1 && $6>avg &&
(substr($5,1,2)==13||substr($5,1,2)==31||
 substr($5,1,2)==23||substr($5,1,2)==29||
 substr($5,1,2)==30||substr($5,1,2)==18||
 substr($5,1,2)==17){
    print $1,$6,substr($5,1,2)
}
' OFS=\\ | customer.* > tmpc.csv

# C |> O
head -1 tmpc.csv > tmp1.csv
tail -n+2 tmpc.csv | sort -t\\ | -k1,1 >> tmp1.csv
cut -d\\ | -f2 orders.tbl | sort | cat <(cut -d\\ | -f2 orders.csv) - |\\
join --header -t\\ | -v1 -o 0 1.2 1.3 tmp1.csv - > join.csv
# group by 3, count * sum 2
head -1 join.csv > tmp1.csv
tail -n+2 join.csv | sort -t\\ | -k3 | cat tmp1.csv - | awk -F\\ | '
NR==1{print $3, "count("$1)","sum("$2)"}
NR==2{g3=$3;count=1;sum=$2}
NR>2{
    if(g3==$3){
        count++;
        sum+=$2;
    }else{
        print g3,count,sum;
        g3=$3;sum=$2;count=1;
    }
}
END{print g3,count,sum}
' OFS=\\ | | tee tmpout.csv

rm tmp*.csv join.csv
```

A.2. Parallelisierte Abfragen

A.2.1. Abfrage 2

```
#!/bin/bash
# 2014-08-26
mkfifo tmp1.csv tmp2.csv tmp.csv
## R >< N >< S
# R >< N (regionkey)
cut nation.csv -d\| -f1-3 > tmp.csv &
cut nation.tbl -d\| -f1-3 | sort -t\| -k3,3 | cat tmp.csv - > tmp1.csv &
cat region.tbl | grep "EUROPE" | cut -d\| -f1 region.csv - \|
join --header -t\| -1 3 -2 1 tmp1.csv - > tmp1rn.csv

# >< S (nationkey)
cut supplier.tbl -d\| -f1-7 | sort -t\| -k4,4 | cut -d\| -f1-7 supplier.csv - >
tmp1.csv &
head -1 tmp1rn.csv > tmp2.csv &
tail -n+2 tmp1rn.csv | sort -t\| -k2,2 | cat tmp2.csv - \|
join --header -t\| -1 4 -2 2 tmp1.csv - > tmp1rns.csv

# PS >< RNS (suppkey)
cut partsupp.tbl -d\| -f1-4 | sort -t\| -k2,2 | cut -d\| -f1,2,4 partsupp.csv -
> tmp1.csv &
head -1 tmp1rns.csv > tmp2.csv &
tail -n+2 tmp1rns.csv | sort -t\| -k2,2 | cat tmp2.csv - \|
join --header -t\| -1 2 -2 2 tmp1.csv - > join1.csv

# group by partkey (2): ps_supplycost = select min(ps_supplycost) (3)
head -1 join1.csv > tmp1.csv &
tail -n+2 join1.csv | sort -t\| -k2,2 | cat tmp1.csv - | awk -F\| '
NR==1{print $2, "min("$3")"}
NR==2{g2=$2; min=$3}
NR>2{
    if( g2==$2 ){
        if(min>$3)
            min=$3
    } else {
        print g2, min;
        g2=$2; min=$3;
    }
}
END{print $2,min}
' OFS=\| > join2.csv

# P >< PRNS (partkey)
#part: restrictions: p_size = 15; residuals: p_type like %BRASS
cat part.tbl | awk -F\| '
$6==15 && $5~/.*BRASS/ {print $1,$2,$3}' OFS=\| \|
sort -t\| -k1,1 | cut -d\| -f1,2,3 part.csv - > tmp1.csv &
head -1 join2.csv > tmp2.csv &
tail -n+2 join2.csv | sort -t\| -k1,1 | cat tmp2.csv - \|
join --header -t\| -1 1 -2 1 tmp1.csv - > join3.csv

# >< PS (partkey)
cut partsupp.csv -d\| -f1,2,4 > tmp1.csv &
```

```
cut partsupp.tbl -d\\ -f1,2,4 | sort -t\\ -k1,1 | cat tmp1.csv - \\
join --header -t\\ -1 1 -2 1 - join3.csv \\
awk -F\\ 'NR==1 || $3==$6 {print $0}' > join5.csv

# RNS >< P PS (suppkey)
head -1 tmpsns.csv > tmp.csv &
tail -n+2 tmpsns.csv | sort -t\\ -k2,2 | cat tmp.csv - > tmp1.csv &
head -1 join5.csv > tmp2.csv &
tail -n+2 join5.csv | sort -t\\ -k2,2 | cat tmp2.csv - \\
join --header -t\\ -1 2 -2 2 tmp1.csv - \\
awk -F\\ '{print $6,$3,$9,$10,$13,$4,$5,$7}' OFS=\\ > join6.csv

#s_acctbal,s_name,s_name,s_partkey,s_mfgr,s_address,s_phone,s_comment
head -1 join6.csv
# sort s_acctbal desc,n_name,s_name,p_partkey
tail -n+2 join6.csv | sort -t\\ -k1,1 -snr
rm join{1,2,3,5,6}.csv tmp{,1,2,rn,rns}.csv
```

A.2.2. Abfrage 3

```
#!/bin/bash
#2014-08-26
#C<O><L
mkfifo tmporder.csv tmp1.csv tmp.csv

#C<O (custkey) && o_orderdate<1995..
(sort -t\\ -k2,2 orders.tbl | cat orders.csv - | awk -F\\ | '
    NR==1 || $5<"1995-03-15"{print $1, $2, $5, $8}
    ' OFS=\\ > tmporder.csv)&
grep BUILDING customer.tbl | sort -t\\ -k1,1 | cut -d\\ -f1,2 customer.csv - |\\
join --header -t\\ -1 2 -2 1 tmporder.csv - > join1.csv

# >< L (orderkey) && l_shipdate>1995...
(head -1 join1.csv > tmp.csv &
tail -n+2 join1.csv | sort -t\\ -k2,2 | cat tmp.csv - > tmp1.csv) &
cat lineitem.tbl | sort -t\\ -k1,1 | cat lineitem.csv - | awk -F\\ | '
    NR==1 || $11>"1995-03-15"{
        print $1,$6,$7
    }' OFS=\\ |\\
join --header -t\\ -1 2 -2 1 tmp1.csv - > output.csv

# gruppieren und sortieren
head -1 output.csv > tmp1.csv &
tail -n+2 output.csv | sort -t\\ -k4,5 | sort -s -t\\ -k1,1 | cat tmp1.csv - |
awk -F\\ | '
    NR==1{print $1,"sum(revenue)", $3, $4}
    NR==2{g1=$1; g3=$3; g4=$4; sum=($6*(1.0-$7))}
    NR>2{
        if(g1==$1 && g3==$3 && g4==$4 ){
            sum+=($6*(1.0-$7))
        } else{
            print g1, sum, g3, g4;
            g1=$1; g3=$3; g4=$4; sum=($6*(1.0-$7))
        }
    }
    END{print g1, sum, g3, g4}
    ' OFS=\\ > tmp3.csv
head -1 tmp3.csv
tail -n+2 tmp3.csv | sort -t\\ -k3,3 | sort -s -t\\ -k2,2 -nr | head -10

rm tmporder.csv tmp{,1,3}.csv output.csv
```

A.2.3. Abfrage 4a

```
#!/bin/bash
#2014-08-25
mkfifo tmporder.csv
(sort -k1,1 -t\\ | orders.tbl | cat orders.csv - | awk -F\\ | '
    NR==1{
        print $1"| " $6
    }
    NR>1 && $5<"1993-10-01" && $5>="1993-07-01" {
        print $1"| " $6
    }
' > tmporder.csv)&

# and exists lineitem cdate<receiptdate
sort -k1,1 -t\\ | lineitem.tbl | cat lineitem.csv - | awk -F\\ | '
    NR==1 || $12<$13{
        print $1
    }
' | uniq | \
join --header -t\\ | -1 1 -2 1 tmporder.csv - > tmp.csv

# count mittels awk
# group by $2 (orderprio)
head -1 tmp.csv > tmp1.csv
tail -n+2 tmp.csv | sort -t\\ | -k2,2 | cat tmp1.csv - | awk -F\\ | '
    NR==1{print $2, "order_count"}
    NR==2{g2=$2; count=1}
    NR>2{
        if( g2==$2 ){
            count++
        }else{
            print g2, count;
            g2=$2;count=1;
        }
    }
    END{print g2,count}
' OFS=\\

rm tmp*.csv
```

A.2.4. Abfrage 4b

```
#!/bin/bash
#2014-08-27
mkfifo tmporder.csv

(sort -k1,1 -t\| orders.tbl | cat orders.csv - | awk -F\| '
    NR==1{
        print $1"|" $6
    }
    NR>1 && $5<"1993-10-01" && $5>="1993-07-01"{
        print $1"|" $6
    }
' > tmporder.csv)&

sort -k1,1 -t\| lineitem.tbl | cat lineitem.csv - | awk -F\| '
    NR==1 || $12<$13{
        print $1
    }
' | uniq > tmpline.csv

join --header -t\| -1 1 -2 1 tmporder.csv tmpline.csv | tee tmp.csv \|

# count mittels uniq
cut -d\| -f2 | tail -n+2 | sort | uniq -c

rm tmp*.csv
```

A.2.5. Abfrage 5

```
#!/bin/bash
# 2014-08-27
## R >< N >< C >< O >< L
mkfifo tmp1.csv tmp2.csv tmp.csv
#echo 'R >< N (regionkey)'
cut nation.tbl -d\| -f1-3 | sort -t\| -k3,3 | cut -d\| -f1-3 nation.csv - >>
tmp1.csv &
cat region.tbl | grep "ASIA" | cut -d\| -f1 | sort | cut -d\| -f1 region.csv - \|
\
join --header -t\| -1 3 -2 1 tmp1.csv - > tmpn.csv
#echo '# >< C (nationkey)'
cut customer.csv -d\| -f1,4 > tmp.csv &
cut customer.tbl -d\| -f1,4 | sort -t\| -k2,2 | cat tmp.csv - > tmp1.csv &
head -1 tmpn.csv > tmp2.csv &
tail -n+2 tmpn.csv | sort -t\| -k2,2 | cat tmp2.csv - \|
\
join --header -t\| -1 2 -2 2 tmp1.csv - > tmpnrc.csv
#echo '# >< O (custkey)'
cat orders.tbl | awk -F\| '$5>="1994-01-01" && $5<"1995-01-01"{print $0}' | cut -
d\| -f1-4 | sort -t\| -k2,2 | cut -d\| -f1,2,4 orders.csv - > tmp1.csv &
head -1 tmpnrc.csv > tmp2.csv &
tail -n+2 tmpnrc.csv | sort -t\| -k2,2 | cat tmp2.csv - \|
\
join --header -t\| -1 2 -2 2 tmp1.csv - > join1.csv
#echo '# >< L (orderkey)'
cut lineitem.csv -d\| -f1,3,6,7 > tmp.csv &
cut lineitem.tbl -d\| -f1,3,6,7 | sort -t\| -k1,1 | cat tmp.csv - > tmp1.csv &
head -1 join1.csv > tmp2.csv &
tail -n+2 join1.csv | sort -t\| -k2,2 | cat tmp2.csv - \|
\
join --header -t\| -1 1 -2 2 tmp1.csv - > join2.csv
#echo '# >< S (suppkey) && c_nationkey=s_nationkey'
cut supplier.tbl -d\| -f1-4 | sort -t\| -k1,1 | cut -d\| -f1,4 supplier.csv - >
tmp1.csv &
head -1 join2.csv > tmp2.csv &
tail -n+2 join2.csv | sort -t\| -k2,2 | cat tmp2.csv - \|
\
join --header -t\| -1 1 -2 2 tmp1.csv - | awk -F\| '
NR==1 || $2==$8{print $0}
' > join3.csv
# group by n_name, sum revenue
head -1 join3.csv > tmp1.csv &
tail -n+2 join3.csv | sort -t\| -k10,10 | cat tmp1.csv - | awk -F\| '
NR==1{print $10,"sum(revenue)"}
NR==2{g10=$10; sum=$4*(1.0-$5)}
NR>2{
    if( g10==$10 ){
        sum+=$4*(1.0-$5)
    }else{
        print g10, sum;
        g10=$10;sum=$4*(1.0-$5);
    }
}
END{print g10,sum}
' OFS=\| > join4.csv
head -1 join4.csv
tail -n+2 join4.csv | sort -t\| -k2,2 -r
```



```
rm join {1,2,3,4}.csv tmp {,1,2,rn,rnc}.csv
```

A.2.6. Abfrage 7

```
#!/bin/bash
#2014-08-27
mkfifo tmp{,1,nation}.csv
# N x N >< C >< O >< L >< S
# NxN (Germany, France)
(sort -t\\ -k1,1 nation.tbl | cut -d\\ -f1,2 | awk -F\\ '
    NR==1{print "cust_n_key|cust_n_name|supp_n_key|supp_n_name"}
    NR>1 && ($2=="FRANCE" || $2=="GERMANY"){
        lines[i++]=$0
    }
    END{
        for (i in lines)
            for (j in lines)
                print lines[i] "|" lines[j]
    }
' | awk -F\\ 'NR==1 ||
    ($2=="FRANCE" && $4=="GERMANY") ||
    ($4=="FRANCE" && $2=="GERMANY"){print $0}'> tmpnation.csv)&

# >< C (nationkey)
cat customer.tbl | sort -t\\ -k4,4 | cut -d\\ -f1,4 customer.csv - |\\
join --header -t\\ -1 1 -2 2 tmpnation.csv - > join1.csv

# >< O (custkey)
(head -1 join1.csv > tmp.csv &
tail -n+2 join1.csv | sort -t\\ -k5,5 | cat tmp.csv - > tmp1.csv)&
sort -t\\ -k2,2 orders.tbl | cut -d\\ -f1,2 orders.csv - |\\
join --header -t\\ -1 5 -2 2 tmp1.csv - > join2.csv

# >< L (orderkey)
(head -1 join2.csv > tmp.csv &
tail -n+2 join2.csv | sort -t\\ -k6,6 | cat tmp.csv - > tmp1.csv)&
cat lineitem.tbl | awk -F\\ '
    $11>="1995-01-01" && $11<="1996-12-31"{
        print $0
    }
' OFS=\\ | sort -t\\ -k1,1 | cut -d\\ -f1,3,6,7,11 lineitem.csv - |\\
join --header -t\\ -1 6 -2 1 tmp1.csv - > join3.csv

# >< S (nationkey, suppkey)
(head -1 join3.csv > tmp.csv &
tail -n+2 join3.csv | sort -t\\ -k7,7 | cat tmp.csv - > tmp1.csv)&
cat supplier.tbl | sort -t\\ -k1,1 | cut -d\\ -f1,4 supplier.csv - |\\
# -k5,5
join --header -t\\ -1 7 -2 1 tmp1.csv - | awk -F\\ '
    NR>1 && $6==$11{
        print $7, $5, substr($10,1,4), $8*(1.0-$9)
    }
' OFS=\\ > join4.csv
cat join4.csv | sort | awk -F\\ '
    BEGIN{print "supp_nation|cust_nation|l_year|revenue"}
    NR==1{g1=$1; g2=$2; g3=$3; sum=$4}
    NR>1{
        if( g1==$1 && g2==$2 && g3==$3 ){
```

```
                sum+=$4
            } else {
                print g1,g2,g3,sum;
                g1=$1; g2=$2; g3=$3; sum=$4;
            }
        }
    END{ print g1,g2,g3,sum}
' OFS=\\

rm tmp1.csv join{1,2,3}.csv tmpnation.csv tmp.csv
```

A.2.7. Abfrage 8

```
#!/bin/bash
#2014-08-28
# (R >< N1) >< ((P >< L) >< O >< C)
# >< S
# >< N2
mkfifo tmp1.csv tmpregion.csv tmp.csv

# N1 >< R (regionkey)
(grep '!AMERICA!' region.tbl | cut -d\| -f1 | sort | cut -d\| -f1 region.csv ->
 tmpregion.csv&
cut -d\| -f1-3 nation.tbl | sort -t\| -k3,3 | cut nation.csv - -d\| -f1,3 | \
join --header -t\| -1 1 -2 2 tmpregion.csv -> tmpnr.csv)&

# P >< L (partkey)
cat lineitem.tbl | sort -t\| -k2,2 | cut -d\| -f1,2,3,6,7 lineitem.csv -> tmp1.
csv &
cat part.tbl | grep 'ECONOMY ANODIZED STEEL' | cut -d\| -f1 | sort | cut part.csv
- -d\| -f1 | \
join --header -t\| -1 2 -2 1 tmp1.csv -> join2.csv

# >< O (orderkey)
(head -1 join2.csv > tmp.csv &
tail -n+2 join2.csv | sort -t\| -k2,2 | cat tmp.csv -> tmp1.csv )&
cut orders.tbl -d\| -f1-5 | sort -t\| -k1,1 | cat orders.csv - | awk -F\| '
NR==1 || ( $5>="1995-01-01" && $5<="1996-12-31"){
    print $1 "|" $2 "|" $5
}
' | \
join --header -t\| -1 2 -2 1 tmp1.csv -> join3.csv

# >< C (custkey)
(head -1 join3.csv > tmp.csv &
tail -n+2 join3.csv | sort -t\| -k6,6 | cat tmp.csv -> tmp1.csv)&
cut -d\| -f1-4 customer.tbl | sort -t\| -k1,1 | cut -d\| -f1,4 customer.csv - | \
join --header -t\| -1 6 -2 1 tmp1.csv -> join4.csv

# (N1><R) >< (nationkey)
(head -1 join4.csv > tmp.csv &
tail -n+2 join4.csv | sort -t\| -k8,8 | cat tmp.csv -> tmp1.csv)&
head -1 tmpnr.csv > tmp2.csv
tail -n+2 tmpnr.csv | sort -t\| -k2,2 | cat tmp2.csv - | \
join --header -t\| -1 8 -2 2 tmp1.csv -> join5.csv

# >< S (suppkey)
(head -1 join5.csv > tmp.csv &
tail -n+2 join5.csv | sort -t\| -k5,5 | cat tmp.csv -> tmp1.csv)&
cut -d\| -f1-4 supplier.tbl | sort -t\| -k1,1 | cut -d\| -f1-4 supplier.csv - | \
join --header -t\| -1 5 -2 1 tmp1.csv -> join6.csv

# >< N2 (s_nationkey==n2_nationkey)
(head -1 join6.csv > tmp.csv&
tail -n+2 join6.csv | sort -t\| -k12,12 | cat tmp.csv -> tmp1.csv)&
cut -d\| -f1,2 nation.tbl | sort -t\| -k1,1 | cut nation.csv - -d\| -f1,2 | \
join --header -t\| -1 12 -2 1 tmp1.csv - | \
```

```
#extract(year from o_orderdate) as o_year,
#l_extendedprice * (1 - l_discount) as volume,
awk -F\| '
    NR==1{print "nation|year|volume"}
    NR>1{print $13,substr($9,1,4),$7*(1.0-$8) }
' OFS=\| > join7.csv

#group by $2 year; 10: $1 nation; $3 volume
head -1 join7.csv > tmp1.csv &
tail -n+2 join7.csv | sort -t\| -k2,2 | cat tmp1.csv - | awk -F\| '
    NR==1{print $2,"mkt_share"}
    NR==2{g2=$2; sumt=$3; sumb=0}
    NR>2{
        if (g2==$2){
            sumt+=$3
        } else {
            print g2,sumb/sumt;
            g2=$2; sumt=$3; sumb=0;
        }
    }
    $1=="BRAZIL"{sumb+=$3}
    END{print g2, sumb/sumt}
' OFS=\|

rm tmp*.csv join{2,3,4,5,6,7}.csv
```

A.2.8. Abfrage 9

```
#!/bin/bash
# (N >< S) >< (P >< PS)
# >< L
# >< O
mkfifo tmp1.csv tmp2.csv join1.csv tmp.csv
#echo 'N >< S (nationkey)'
(cut -d\| -f1,2 nation.tbl | sort -t\| -k1,1 | cut -d\| -f1,2 nation.csv - >
 tmp1.csv &
cut -d\| -f1-4 supplier.tbl | sort -t\| -k4,4 | cut -d\| -f1,4 supplier.csv - \|
join --header -t\| -1 1 -2 2 tmp1.csv - > tmpsn.csv)&

# P >< PS (partkey)
cat part.tbl | grep 'green' | cut -d\| -f1 | sort -t\| -k1,1 | cut -d\| -f1 part.
csv - > tmp2.csv &
cat partsupp.tbl | sort -t\| -k1,1 | cut -d\| -f1,2,4 partsupp.csv - \|
join --header -t\| -1 1 -2 1 tmp2.csv - > tmppps.csv

# SN >< PPS (suppkey)
(head -1 tmppps.csv > tmp.csv &
tail -n+2 tmppps.csv | sort -t\| -k2,2 | cat tmp.csv - > tmp1.csv)&
head -1 tmpsn.csv > tmp2.csv &
tail -n+2 tmpsn.csv | sort -t\| -k3,3 | cat tmp2.csv - \|
join --header -t\| -1 2 -2 3 tmp1.csv - > join1.csv &

# >< L (suppkey, dann partkey gleich?)
cut lineitem.tbl -d\| -f1-7 | sort -t\| -k3,3 | cut -d\| -f1,2,3,5,6,7 lineitem.
csv - \|
join --header -t\| -1 3 -2 1 - join1.csv | awk -F\| '
NR==1 || $3==$7 {print $0}' > join2.csv

# >< O (orderkey)
(head -1 join2.csv > tmp.csv &
tail -n+2 join2.csv | sort -t\| -k2,2 | cat tmp.csv - > tmp1.csv)&
cat orders.tbl | sort -t\| -k1,1 | cut -d\| -f1,5 orders.csv - \|
join --header -t\| -1 2 -2 1 tmp1.csv - | tee join3.csv \|

#_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
# extract year
awk -F\| '
NR==1{
    print $10 "|year|amount"
}
NR>1{
    sum=$5*(1.0-$6)-$8*$4
    print $10 "|" substr($11,1,4) "|" sum
}
'> join4.csv

# group by $6 (nation), year, sum_profit
head -1 join4.csv > tmp1.csv &
tail -n+2 join4.csv | sort -r -t\| -k2,2 | sort -t\| -s -k1,1 | cat tmp1.csv -
| awk -F\| '
NR==1{print $1, $2, "sum_profit"}
NR==2{g1=$1; g2=$2; sum=$3}
```

```
NR>2{
    if (g1==$1 && g2==$2) {
        sum+=$3
    } else {
        print g1,g2,sum;
        g1=$1; g2=$2; sum=$3;
    }
}
END{ print g1,g2,sum}
' OFS=\\

rm tmp*.csv join{1,2,3,4}.csv
```

A.2.9. Abfrage 10

```
#!/bin/bash
#2014-08-28
mkfifo tmp1.csv tmp2.csv tmp.csv
# ((O >< C) >< N) >< L
# O >< C (custkey)
awk -F\\| ' $5>="1993-10-01" && $5<"1994-01-01"{print $1 "|" $2}
' orders.tbl | sort -t\\| -k2,2 | cut -d\\| -f1,2 orders.csv -> tmp1.csv &
cut -d\\| -f1-8 customer.tbl | sort -t\\| -k1,1 | cut -d\\| -f1-6,8 customer.csv -
\\|
join --header -t\\| -1 2 -2 1 tmp1.csv -> join1.csv
# >< N (nationkey)
(head -1 join1.csv > tmp.csv &
tail -n+2 join1.csv | sort -t\\| -k5,5 | cat tmp.csv -> tmp1.csv)&
cut -d\\| -f1,2 nation.tbl | sort -t\\| -k1,1 | cut -d\\| -f1,2 nation.csv - |\\
join --header -t\\| -1 5 -2 1 tmp1.csv -> join2.csv

# >< L (orderkey)
(head -1 join2.csv > tmp.csv &
tail -n+2 join2.csv | sort -t\\| -k3,3 | cat tmp.csv -> tmp1.csv)&
cut -d\\| -f1,6,7 lineitem.csv > tmp2.csv &
awk -F\\| '$9=="R"{print $1,$6,$7}' OFS=\\| lineitem.tbl | sort -t\\| -k1,1 | cat
tmp2.csv - |\\
join --header -t\\| -1 3 -2 1 tmp1.csv -> join3.csv

# group by, sum(revenue)
#c_custkey,c_name,c_acctbal,c_phone,n_name,c_address,c_comment
#sum(l_extendedprice * (1 - l_discount)) as revenue
head -1 join3.csv > tmp1.csv &
tail -n+2 join3.csv | sort -r -t\\| -k3,9 | cat tmp1.csv - | awk -F\\| '
NR==1{print $3, $4, $7, $6, $9, $5, $8, "sum(revenue)"}
NR==2{g3=$3;g4=$4;g7=$7;g6=$6;g2=$2;g5=$5;g8=$8;sum=$10*(1.0-$11)}
NR>2{
    if(g3==$3 && g4==$4 && g7==$7 && g6==$6 &&
        g9==$9 && g5==$5 && g8==$8){
        sum+=$10*(1.0-$11)
    } else{
        print g3, g4, g7, g6, g9, g5, g8, sum;
        g3=$3;g4=$4;g7=$7;g6=$6;g9=$9;g5=$5;g8=$8;sum=$10*(1.0-
            $11);
    }
}
END{print g3, g4, g7, g6, g9, g5, g8, sum}
' OFS=\\| > join4.csv

head -1 join4.csv;
tail -n+2 join4.csv | sort -t\\| -nrk8 | head -20

rm tmp*.csv join{1,2,3,4}.csv
```


A.2.10. Abfrage 11

```
#!/bin/bash
#2014-08-28
mkfifo tmp1.csv tmp2.csv tmp.csv
echo 'N >< S'
(cut -d\| -f1,4 supplier.csv > tmp.csv &
cut -d\| -f1,4 supplier.tbl | sort -t\| -k2,2 | cat tmp.csv - > tmp1.csv)&
cat nation.tbl | grep 'GERMANY' | cut -d\| -f1 | sort | cut -d\| -f1 nation.csv
- \|
join --header -t\| -1 2 -2 1 tmp1.csv - > tmpsn.csv

echo 'PS >< (partkey)'
(head -1 tmpsn.csv | cut -d\| -f2 > tmp.csv&
tail -n+2 tmpsn.csv | cut -d\| -f2 | sort | cat tmp.csv - > tmp2.csv)&
cut -d\| -f1,2,3,4 partsupp.tbl | sort -t\| -k2,2 | cut -d\| -f1-4 partsupp.csv
- \|
join --header -t\| -1 2 -2 1 - tmp2.csv > join1.csv

#echo 'sum(ps_supplycost * ps_availqty) * 0.0001'
sum='awk -F\| '
    BEGIN{SUM=0}
    NR>1{SUM+=$3*$4}
    END{print SUM*0.0001}
' join1.csv '

# 'sum(ps_supplycost * ps_availqty); sum(ps_supplycost * ps_availqty)>$sum'
head -1 join1.csv > tmp1.csv &
tail -n+2 join1.csv | sort -r -t\| -k2,2 | cat tmp1.csv - | awk -F\| -v min=
$sum '
    NR==1{print $2, "value"}
    NR==2{g2=$2;sum=$3*$4}
    NR>2{
        if (g2==$2 ){
            sum+=$3*$4
        } else {
            if (sum>min)
                printf ("%d|%d\n", g2, sum);
            g2=$2;sum=$3*$4;
        }
    }
    END{print g2, sum}
' OFS=\| > join2.csv
head -1 join2.csv
tail -n+2 join2.csv | sort -t\| -k2,2 -nr

rm tmp*.csv join{1,2}.csv
```

A.2.11. Abfrage 12

```
#!/bin/bash
#2014-08-28
mkfifo tmporder.csv
# O >< L (orderkey)
#      o_orderkey = l_orderkey
# $15      and l_shipmode in ('MAIL', 'SHIP')
# $12<$13   and l_commitdate < l_receiptdate
# $11<12    and l_shipdate < l_commitdate
# $13   and l_receiptdate >= date '1994-01-01'
# $13   and l_receiptdate < date '1995-01-01'
(sort -t\\ -k1,1 orders.tbl\\
cat orders.csv - | awk -F\\ '
    NR==1{
        print $1 "|high_line_count|low_line_count"
    }
    NR>1{
        if ( $6=="1-URGENT" || $6=="2-HIGH" )
            print $1 "|1|0"
        else
            print $1 "|0|1"
    }
' > tmporder.csv)&

sort -t\\ -k1,1 lineitem.tbl \\
cat lineitem.csv - | awk -F\\ '
    NR==1{
        print $1 "|" $15
    }
    NR>1 && $13>="1994-01-01" && $13<"1995-01-01" && $12<$13 && $11<$12 && (
        $15=="MAIL" || $15=="SHIP"){
            print $1 "|" $15
        }
' \\

join --header -t\\ -1 1 -2 1 tmporder.csv - > join.csv

#group by shipmode ($4), sum highline, lowline
head -1 join.csv > tmp1.csv
tail -n+2 join.csv | sort -t\\ -k4,4 | cat tmp1.csv - | awk -F\\ '
    NR==1{print $4,"high_line_count", "low_line_count"}
    NR==2{g4=$4;sumhigh=$2;sumlow=$3}
    NR>2{
        if (g4==$4){
            sumlow+=$3; sumhigh+=$2
        } else {
            print g4, sumhigh, sumlow;
            g4=$4;sumhigh=$2;sumlow=$3;
        }
    }
    END{print g4, sumhigh, sumlow}
' OFS=\\

rm tmp*.csv
```

A.2.12. Abfrage 13

```
#!/bin/bash
#2014-08-26
echo 'c_count|custdist'
mkfifo tmp2.csv tmp.csv
# C |>< O
# left outer join ^= join -a1 -a2 -1 2 -2 2 -o 0 1.1 2.1 -e "0" 1.txt 2.txt
#join -a1 -a2 -1 2 -2 2 -o 0 1.1 -e "0" 1.txt 2.txt
(head -1 orders.csv | cut -d\| -f2 > tmp.csv &
cat orders.tbl |grep -v 'special.*requests' | cut -d\| -f2 | sort | cat tmp.csv
- > tmp2.csv)&
cut -d\| -f1,2 customer.tbl | sort -t\| -k1,1 | cut -d\| -f1,2 customer.csv - |\
join --header -t\| -1 1 -2 1 -a1 -o 0 2.1 -e "NULL" - tmp2.csv |\
awk -F\| '
    NR==1{print $1,"count"}
    NR==2{g1=$1; count=0}
    NR>2{
        if(g1!=$1){
            print g1, count;
            g1=$1;count=0;
        }
    }
    $2!="NULL"{count++}
    END{print g1, count}
' OFS=\| | tail -n+2 | sort -t\| -k2,2 | awk -F\| '
    NR==1{g2=$2;count=1}
    NR>1{
        if(g2==$2){
            count++;
        }else{
            print g2, count;
            g2=$2;count=1;
        }
    }
    END{print g2, count}
' OFS=\| | sort -t\| -k2,2 -nr

rm tmp2.csv tmp.csv
```

A.2.13. Abfrage 14

```
#!/bin/bash
#2014-08-28
mkfifo tmp2.csv
# P >< L (partkey)
(cat lineitem.tbl | awk -F\| '
    $11>="1995-09-01" && $11<"1995-10-01" {
        print $2,$6,$7
    }
' OFS=\| | sort -t\| -k1,1 | cat <(cut lineitem.csv -d\| -f2,6,7) -> tmp2.csv)&

cut -d\| -f1-5 part.tbl | sort -t\| -k1,1 | cut -d\| -f1,5 part.csv - |\
join --header -t\| -1 1 -2 1 - tmp2.csv |\
awk -F\| '
    BEGIN{sum=0; sumpromo=0; print "promo_revenue"}
    {
        rev=$3*(1.0-$4);
        sum+=rev}
    $2~/PROMO.*/{sumpromo+=rev}
    END{print sumpromo/sum*100}
,
rm tmp2.csv
```

A.2.14. Abfrage 15

```
#!/bin/bash
#revenue.csv
mkfifo tmprev.csv
(cut lineitem.tbl -d\| -f1-11 | sort -t\| -k3,3 | cut lineitem.csv - -d\| -f3
,6,7,11 | awk -F\| '
    NR==1{
        print $1,"total_revenue"
    }
    NR==2{g1=$1; sum=0; i=0; max=0}
    NR>2{
        if (g1!=$1 && sum>0){
            line[i,0]=g1
            line[i++,1]=sum;
            sum=0;
        }
        g1=$1;
    }
    NR>1 && $4>="1996-01-01" && $4<"1996-04-01" {
        sum+=$(2*(1.0-$3));
        max=(max<sum)?sum:max;
    }
    END{
        line[i,0]=g1;line[i++,1]=sum;
        for (j=0;j<i; j++)
            if (line[j,1]==max)
                print line[j,0], line[j,1];
    }
' OFS=\| > tmprev.csv) &
# S >< revenue.csv
sort -t\| -k1,1 supplier.tbl |
cut -d\| -f1,2,3,5 supplier.csv - \|
join --header -t\| -1 1 -2 1 - tmprev.csv
rm tmprev.csv
```

A.2.15. Abfrage 16

```
#!/bin/bash
# 2014-08-28
mkfifo tmp1.csv tmp.csv
# PS <> P
# $4 p_brand <> 'Brand#45'
# $5      and p_type not like 'MEDIUM POLISHED%'
# $6      and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
(cat part.tbl | sort -t\| -k1,1 | cat part.csv - | awk -F\| '
    NR==1{
        print $1,$4,$5,$6
    }
    NR>1 && $4!="Brand#45" && $5!~/^MEDIUM POLISHED.* / && ($6==49 || $6==14
        || $6==23 || $6==45 || $6==19 || $6==3 || $6==36 || $6==9){
        print $1,$4,$5,$6
    }
' OFS=\| > tmp1.csv)&
cat partsupp.tbl | sort -t\| -k1,1 | cut -d\| -f1-4 partsupp.csv - \|
join --header -t\| -1 1 -2 1 - tmp1.csv > join1.csv

# PS |> S (suppkey)
(head -1 join1.csv | cut -d\| -f1,2,5,6,7,8 > tmp.csv &
tail -n+2 join1.csv | cut -d\| -f1,2,5,6,7,8 | sort -t\| -k2,2 | cat tmp.csv -
> tmp1.csv)&
sed -r '/Customer.*Complaints/ d' supplier.tbl | cut -d\| -f1 | sort | cut -d\|
-f1 supplier.csv - \|
join --header -t\| -1 2 -2 1 tmp1.csv - > join2.csv

#group by p_brand, p_type, p_size; count(distinct ps_suppkey)
head -1 join2.csv > tmp1.csv &
tail -n+2 join2.csv | sort | sort -s -t\| -k3,5 | cat tmp1.csv - | awk -F\| '
    NR==1{print $3,$4,$5, "uniqcount("$1")"}
    NR==2{g3=$3;g4=$4;g5=$5;c1=$1;count=1}
    NR>2{
        if(g3==$3 && g4==$4 && g5==$5){
            if(c1!=$1) count++;
            c1=$1;
        } else{
            print g3,g4,g5,count;
            g3=$3;g4=$4;g5=$5;c1=$1;count=1;
        }
    }
    END{print g3,g4,g5,count}
' OFS=\| > tmpout.csv

# supplier_cnt desc, p_brand, p_type, p_size
head -1 tmpout.csv; tail -n+2 tmpout.csv | sort -s -t\| -nr -k4,4

rm tmp*.csv join1.csv join2.csv
```

A.2.16. Abfrage 17

```
#!/bin/bash
# 2014-08-28
## L >< P (partkey)
mkfifo tmp1.csv
(cut -d\| -f2,5,6 lineitem.tbl | sort -t\| -k1,1 | cat <(cut -d\| -f2,5,6
    lineitem.csv) - > tmp1.csv)&

cut -d\| -f1,4,7 part.tbl | grep 'Brand#23' | grep 'MED BOX' | sort -t\| -k1,1 |
    cat <(cut -d\| -f1,4,7 part.csv) - | \
join --header -t\| -1 1 -2 1 tmp1.csv - | tee join1.csv | \

#avg(l_quantity) pro partkey
awk -F\| '
    NR==1{print $1,"avg("$2)"}
    NR==2{g1=$1;sum=$2;count=1}
    NR>2{
        if(g1==$1){
            sum+=$2;
            count++;
        } else {
            print g1,sum/count;
            g1=$1; sum=$2; count=1;
        }
    }
    END{print g1,sum/count}
' OFS=\| > tmpavgs.csv

#dann join mit join1
join --header -t\| -1 1 -2 1 join1.csv tmpavgs.csv | \
#greife die relevanten raus
awk -F\| '
    BEGIN{
        print "avg_yearly"
        sum=0
    }
    NR>1 && $2<$6*0.2 {sum+=$3}
    END{ print sum/7.0 }
,
rm tmp1.csv join1.csv
```

A.2.17. Abfrage 18

```
#!/bin/bash
# 2014-08-28
mkfifo tmp2.csv tmp.csv
# group by l_orderkey, having sum quantity >300
(cut -d\\ -f1-5 lineitem.tbl | sort -t\\ -k1,1 | cut -d\\ -f1,5 lineitem.csv - |
  awk -F\\ '
    NR==1{print $1,"sum("$2)"}
    NR==2{g1=$1;sum=$2}
    NR>2{
      if(g1==$1){
        sum+=$2;
      } else {
        if(sum>300)
          print g1,sum;
        g1=$1; sum=$2;
      }
    }
  '
  END{print g1,sum}
' OFS=\\ > tmp2.csv)&
# O >> L (orderkey)
sort -t\\ -k1,1 orders.tbl | cut -d\\ -f1-9 orders.csv - |\\
join --header -t\\ -1 1 -2 1 - tmp2.csv > join1.csv
# >> C (custkey)
#c_name,c_custkey,o_orderkey,o_orderdate,o_totalprice
(head -1 join1.csv | cut -d\\ -f1,2,4,5,10 > tmp.csv&
tail -n+2 join1.csv | cut -d\\ -f1,2,4,5,10 | sort -t\\ -k2,2 | cat tmp.csv - >
  tmp2.csv)&
sort -t\\ -k1,1 customer.tbl | cut -f1,2 customer.csv - |\\
join --header -t\\ -1 1 -2 2 -o 1.2 0 2.1 2.4 2.3 2.5 - tmp2.csv > join2.csv

head -1 join2.csv
tail -n+2 join2.csv | sort -t\\ -k6 | sort -t\\ -k5,5 -sr | head -100

rm tmp.csv tmp2.csv join1.csv join2.csv
```


A.2.18. Abfrage 19

```

#!/bin/bash
# 2014-08-28
mkfifo tmp1.csv
#l_shipmode in ('AIR', 'AIR REG')and l_shipinstruct = 'DELIVER IN PERSON'
(cat lineitem.tbl | awk -F\| '
    $14=="DELIVER_IN_PERSON" && ($15=="AIR" || $15=="AIR_REG"){
        suma=($6*(1.0-$7));
        print $2,$5,$6,$7
    }
' OFS=\| | sort -t\| -k1,1 | cat <(cut -d\| -f2,5,6,7 lineitem.csv) -> tmp1.csv
)&
# P >< L (partkey)
cut -d\| -f1-9 part.tbl | grep -E "Brand#(12|23|34)" |\
grep -E "(SM|MED|LG)" | grep -E "(CASE|BOX|PACK|PKG|BAG)" |\
sort -t\| -k1,1 | cut -f1-9 part.csv - |\
join --header -t\| -1 1 -2 1 tmp1.csv - | awk -F\| '
    BEGIN{ print "revenue"; sum=0}
    $7=="Brand#12" &&
    ($10=="SM_CASE" || $10=="SM_BOX" ||
    $10=="SM_PACK" || $10=="SM_PKG") &&
    $2>=1 && $2<=11 &&
    $9<=5 && $9>=1{
        sum+=$3*(1.0-$4)
    }
    $7=="Brand#23" &&
    ($10=="MED_BAG" || $10=="MED_BOX" ||
    $10=="MED_PACK" || $10=="MED_PKG") &&
    $2>=10 && $2<=20 &&
    $9<=10 && $9>=1{
        sum+=$3*(1.0-$4)
    }
    $7=="Brand#34" &&
    ($10=="LG_CASE" || $10=="LG_BOX" ||
    $10=="LG_PACK" || $10=="LG_PKG") &&
    $2>=20 && $2<=30 &&
    $9<=15 && $9>=1{
        sum+=$3*(1.0-$4)
    }
    END{print sum}
,
# and p_brand = 'Brand#12' and p_container in ('SM CASE', 'SM BOX', 'SM PACK', '
SM PKG') and l_quantity >= 1 and l_quantity <= 1 + 10
# and p_brand = 'Brand#23'and p_container in ('MED BAG', 'MED BOX', 'MED PKG',
'MED PACK') and l_quantity >= 10 and l_quantity <= 10 + 10and p_size between
1 and 10
#and p_brand = 'Brand#34' and p_container in ('LG CASE', 'LG BOX', 'LG PACK', '
LG PKG' and l_quantity >= 20 and l_quantity <= 20 + 10and p_size between 1
and 15
rm tmp1.csv

```

A.2.19. Abfrage 20

```
#!/bin/bash
# 2014-08-28
mkfifo tmp{1,2}.csv
# P >< PS (partkey)
(cat partsupp.tbl | sort -t\\ -k1,1 | cut -d\\ -f1-5 partsupp.csv -> tmp1.csv)&
awk -F\\ ' $2~/^forest/{print $1}' part.tbl | sort -t\\ -k1,1 | cut -d\\ -f1 part
.csv - \\
join --header -t\\ -1 1 -2 1 tmp1.csv -> join1.csv

# l_shipdate >= date '1994-01-01' and l_shipdate < date '1995-01-01'
# and group by
cat lineitem.tbl | sort -t\\ -k2,3 | sort -t\\ -k2,2 -s | cut -d\\ -f2,3,5,11
lineitem.csv - | awk -F\\ '
    NR==1{print $1,$2,"sum("$3)"}
    NR==2{g1=$1;g2=$2;sum=0}
    NR>2{
        if(g1!=$1 || g2!=$2){
            print g1,g2,sum;
            g1=$1; g2=$2; sum=0;
        }
    }
    NR>1 && $4>="1994-01-01" && $4<"1995-01-01" {
        sum+=$3;
    }
    END{print g1,g2,sum}
' OFS=\\ \\
# L >< (l_partkey = ps_partkey)
join --header -t\\ -1 1 -2 1 join1.csv - \\
# l_suppkey = ps_suppkey
awk -F\\ 'NR==1 || ($2==$6 && $3>$7*0.5){print $2}' > join2.csv

# ' S >< (suppkey in...) '
(head -1 join2.csv > tmp1.csv&
tail -n+2 join2.csv | sort - | cat tmp1.csv -> tmp2.csv)&
cat supplier.tbl | sort -t\\ -k1,1 \\
cut -d\\ -f1-7 supplier.csv - \\
join --header -t\\ -1 1 -2 1 - tmp2.csv > join3.csv

# S >< N (nationkey)
(head -1 join3.csv > tmp2.csv &
tail -n+2 join3.csv | sort -t\\ -k4,4 | cat tmp2.csv -> tmp1.csv)&
awk -F\\ 'NR==1 || $2=="CANADA"{print $1}' nation.* \\
join --header -t\\ -1 4 -2 1 tmp1.csv - | cut -d\\ -f3,4 > join4.csv

head -1 join4.csv
tail -n+2 join4.csv | sort -u

rm tmp*.csv join*.csv
```

A.2.20. Abfrage 21

```
#!/bin/bash
# 2014-08-28
mkfifo tmp1.csv tmp2.csv join4.csv tmp.csv
## N >< S >< L1 >< O |> L
echo ' N >< S '
(cut -d\| -f1,2,4 supplier.csv > tmp.csv &
cut -d\| -f1,2,4 supplier.tbl | sort -t\| -k3,3 | cat tmp.csv - > tmp1.csv)&
cat nation.tbl | grep 'SAUDI ARABIA' | cut -d\| -f1 nation.csv - \|
join --header -t\| -1 3 -2 1 tmp1.csv - > tmpsn.csv

echo ' >< L1 (suppkey) '
#l1.l_receiptdate > l1.l_commitdate
(head -1 tmpsn.csv > tmp.csv &
tail -n+2 tmpsn.csv | sort -t\| -k2,2 | cat tmp.csv - > tmp2.csv)&
awk -F\| '$13>$12 {print $1,$2,$3}' OFS=\| lineitem.tbl \|
sort -t\| -k3,3 | cut -d\| -f1,3 lineitem.csv - | tee tmp1.csv \|
join --header -t\| -1 2 -2 2 - tmp2.csv > join2.csv

echo ' >< O (orderkey) '
#l1.l_receiptdate > l1.l_commitdate
(head -1 join2.csv > tmp.csv &
tail -n+2 join2.csv | sort -t\| -k2,2 | cat tmp.csv - > tmp2.csv)&
awk -F\| '$3=="F" {print $1}' orders.tbl | sort | cut -d\| -f1 orders.csv - \|
join --header -t\| -1 1 -2 2 - tmp2.csv > join3.csv

echo ' |> L3 (orderkey) '
#l3.l_suppkey <> l1.l_suppkey
#awk -F\| 'NR==1 || $13>$12 {print $1,$3}' OFS=\| lineitem.csv > tmp1.csv
(head -1 tmp1.csv > tmp2.csv &
tail -n+2 tmp1.csv | sort -t\| -k1,1 | cat tmp2.csv - \|
join --header -t\| -1 1 -2 1 join3.csv - \|
awk -F\| 'NR==1 || $2!=$5 {print $0}' \|
join --header -t\| -1 1 -2 1 -v1 join3.csv - | cut -d\| -f1-4 > join4.csv)&

echo ' >< L2 ( l2.l_orderkey = l1.l_orderkey and l2.l_suppkey <> l1.l_suppkey) '
# gibt es vom jeweiligen l_orderkey mind. 2? group by, join
cut lineitem.tbl -d\| -f1,3 | sort -t\| -k1,1 | cut lineitem.csv - -d\| -f1,3 |
awk -F\| '
NR==1{print $1, "count(*)"}
NR==2{g1=$1;count=1}
NR>2{
    if(g1==$1){
        count++;
    } else {
        if(count>1)
            print g1,count;
        g1=$1;count=1;
    }
}
END{print g1,count}
' OFS=\| \|
join --header -t\| -1 1 -2 1 join4.csv - > join5.csv
# group by 4 count *
head -1 join5.csv > tmp1.csv &
```

A. TPC-H-Abfragen

```
tail -n+2 join5.csv | sort -t\\ | -k3 | cat tmp1.csv - | awk -F\\ | '
    NR==1{print $4, "count(*)"}
    NR==2{g4=$4;count=1}
    NR>2{
        if(g4==$4){
            count++;
        }else{
            print g4,count;
            g4=$4;count=1;
        }
    }
    END{print g4,count}
' OFS=\\ | > join6.csv

head -1 join6.csv
tail -n+2 join6.csv | sort -t\\ | -snr -k2,2 | head -99

rm tmp*.csv join*.csv
```

A.2.21. Abfrage 22

```

#!/bin/bash
# 2014-08-28
mkfifo tmp1.csv tmp.csv

avg='awk -F\\| 'BEGIN{sum=0; cnt=0; print "avg"}
      NR>1 && $6>0 &&
      ( substr($5,1,2)==13||substr($5,1,2)==31||
      substr($5,1,2)==23||substr($5,1,2)==29||
      substr($5,1,2)==30||substr($5,1,2)==18||
      substr($5,1,2)==17){
          sum+=$6; cnt+=1
      }
      END{ print sum/cnt}
' customer.* | tail -1'

#c_acctbal > select avg(c_acctbal) from...
(awk -F\\| -v avg=$avg '
      NR==1{ print $1,$6, "centrycode" }
      NR>1 && $6>avg &&
      ( substr($5,1,2)==13||substr($5,1,2)==31||
      substr($5,1,2)==23||substr($5,1,2)==29||
      substr($5,1,2)==30||substr($5,1,2)==18||
      substr($5,1,2)==17){
          print $1,$6,substr($5,1,2)
      }
' OFS=\\| customer.* > tmpc.csv

# O |> C
head -1 tmpc.csv > tmp.csv&
tail -n+2 tmpc.csv | sort -t\\| -k1,1 | cat tmp.csv - > tmp1.csv)&
cut -d\\| -f2 orders.tbl | sort | cat <(cut -d\\| -f2 orders.csv) - |\\
join --header -t\\| -v1 -o 0 1.2 1.3 tmp1.csv - > join.csv
# group by 3, count * sum 2
head -1 join.csv > tmp1.csv &
tail -n+2 join.csv | sort -t\\| -k3 | cat tmp1.csv - | awk -F\\| '
      NR==1{ print $3, "count("$1)","sum("$2)"}
      NR==2{ g3=$3; count=1; sum=$2 }
      NR>2{
          if (g3==$3){
              count++;
              sum+=$2;
          } else {
              print g3,count,sum;
              g3=$3;sum=$2;count=1;
          }
      }
      END{ print g3,count,sum}
' OFS=\\| | tee tmpout.csv

rm tmp*.csv join.csv

```


A.3. Die Ergebnisse

A.3.1. SF=1

query1	real 0m47.420s	user 0m39.370s	sys 0m2.744s
query10	real 0m12.971s	user 0m13.549s	sys 0m0.560s
query10p	real 0m12.374s	user 0m13.657s	sys 0m0.592s
query11	real 0m2.934s	user 0m3.300s	sys 0m0.112s
query11p	real 0m2.906s	user 0m3.284s	sys 0m0.096s
query12	real 0m22.859s	user 0m53.267s	sys 0m2.420s
query12p	real 0m21.855s	user 0m55.831s	sys 0m2.536s
query13	real 0m7.068s	user 0m8.089s	sys 0m0.256s
query13p	real 0m6.271s	user 0m8.601s	sys 0m0.252s
query14	real 0m7.185s	user 0m7.160s	sys 0m0.652s
query14p	real 0m6.767s	user 0m7.336s	sys 0m0.608s
query15p	real 0m29.505s	user 0m35.518s	sys 0m1.080s
query16	real 0m4.957s	user 0m5.692s	sys 0m0.432s
query16p	real 0m4.354s	user 0m6.104s	sys 0m0.496s
query17	real 0m37.156s	user 0m1.764s	sys 0m0.024s
query17p	real 0m37.317s	user 0m38.866s	sys 0m0.588s
query18	real 0m25.484s	user 0m34.946s	sys 0m0.972s
query18p	real 0m24.947s	user 0m35.770s	sys 0m0.892s
query19	real 0m6.728s	user 0m6.788s	sys 0m0.584s
query19p	real 0m6.443s	user 0m6.852s	sys 0m0.576s
query2	real 0m6.372s	user 0m6.540s	sys 0m0.320s
query20	real 0m58.405s	user 1m2.956s	sys 0m4.456s
query20p	real 0m59.412s	user 1m4.336s	sys 0m4.656s
query21	real 1m3.225s	user 1m11.084s	sys 0m1.308s
query21p	real 0m46.289s	user 1m16.401s	sys 0m1.228s
query22	real 0m6.178s	user 0m6.540s	sys 0m0.120s
query22p	real 0m5.921s	user 0m6.616s	sys 0m0.084s
query2p	real 0m5.974s	user 0m6.860s	sys 0m0.284s
query3	real 0m32.022s	user 0m43.503s	sys 0m3.428s
query3.alt	real 0m29.276s	user 0m41.879s	sys 0m3.320s
query3p	real 0m31.000s	user 0m43.975s	sys 0m3.352s
query4a	real 0m20.535s	user 0m54.887s	sys 0m2.548s
query4b	real 0m21.910s	user 0m53.483s	sys 0m2.380s
query5	real 0m26.827s	user 0m26.562s	sys 0m1.004s
query5p	real 0m24.696s	user 0m26.982s	sys 0m0.888s
query5pb	real 0m24.441s	user 0m26.634s	sys 0m0.796s
query6	real 0m7.949s	user 0m7.716s	sys 0m0.556s
query7	real 0m18.713s	user 0m30.002s	sys 0m1.580s
query7.alt	real 0m17.680s	user 0m24.814s	sys 0m0.940s
query7p	real 0m18.294s	user 0m30.270s	sys 0m1.540s
query8	real 0m39.677s	user 0m41.495s	sys 0m2.888s
query8p	real 0m37.294s	user 0m44.331s	sys 0m2.744s
query9	real 0m54.243s	user 1m9.584s	sys 0m2.580s
query9.alt	real 0m46.270s	user 0m49.831s	sys 0m1.848s
query9p	real 0m52.910s	user 1m10.184s	sys 0m2.480s

A.3.2. SF=10

query1	real 9m32.857s	user 9m31.970s	sys 0m49.739s
query10	real 3m15.974s	user 3m24.642s	sys 0m8.759s
query10p	real 3m9.187s	user 3m26.610s	sys 0m9.261s
query11	real 0m34.194s	user 0m37.878s	sys 0m1.686s
query11p	real 0m34.220s	user 0m37.766s	sys 0m1.640s
query12	real 4m21.685s	user 8m54.478s	sys 0m26.273s
query12p	real 3m42.580s	user 9m1.861s	sys 0m27.183s
query13	real 1m40.206s	user 1m53.250s	sys 0m4.347s
query13p	real 1m30.957s	user 1m57.208s	sys 0m4.726s
query14	real 2m8.932s	user 2m9.678s	sys 0m12.308s
query14p	real 2m2.453s	user 2m10.616s	sys 0m12.022s
query15	real 7m26.731s	user 8m7.402s	sys 0m27.885s
query16	real 1m6.396s	user 1m11.125s	sys 0m6.569s
query16p	real 0m56.808s	user 1m13.359s	sys 0m7.201s
query17	real 7m48.630s	user 0m22.020s	sys 0m0.754s
query17p	real 7m52.637s	user 8m6.840s	sys 0m9.326s
query18	real 5m20.389s	user 6m47.595s	sys 0m20.564s
query18p	real 5m8.145s	user 6m57.919s	sys 0m20.536s
query19	real 2m55.907s	user 2m59.125s	sys 0m12.261s
query19p	real 2m52.960s	user 2m58.968s	sys 0m12.193s
query20	real 13m26.291s	user 13m18.047s	sys 1m30.816s
query20p	real 13m32.589s	user 13m27.336s	sys 1m31.657s
query21	real 14m51.314s	user 16m17.387s	sys 0m29.451s
query21p	real 10m42.894s	user 16m17.977s	sys 0m28.387s
query22	real 1m22.748s	user 1m27.505s	sys 0m2.001s
query22p	real 1m17.706s	user 1m27.889s	sys 0m1.971s
query2p	real 1m13.534s	user 1m22.012s	sys 0m4.298s
query3p	real 7m23.247s	user 9m18.016s	sys 0m59.532s
query4a	real 3m54.677s	user 9m26.891s	sys 0m28.586s
query4b	real 3m58.975s	user 8m46.921s	sys 0m27.085s
query5b	real 5m7.112s	user 5m26.370s	sys 0m17.300s
query5p	real 5m5.788s	user 5m24.228s	sys 0m17.527s
query6	real 2m20.614s	user 2m16.706s	sys 0m10.894s
query7	real 4m20.438s	user 5m55.684s	sys 0m33.536s
query7p	real 4m14.706s	user 5m57.255s	sys 0m33.023s
query8	real 8m38.944s	user 8m35.325s	sys 0m49.757s
query8p	real 8m12.842s	user 8m44.417s	sys 0m49.566s
query9	real 15m4.204s	user 17m10.921s	sys 0m52.472s
query9p	real 14m37.647s	user 17m4.015s	sys 0m51.840s

B. Vergleich

B.1. Datamash Abfragen

B.1.1. Abfrage 1

```
#!/bin/bash
#2014-08-18
cat lineitem.* | awk -F\| '
    NR==1{
        print $0 "sum_disc|sum_charge|"
    }
    NR>1 && $11<="1998-9-2" {
        suma=($6*(1.0-$7));
        sumb=($6*(1.0-$7)*(1.0+$8));
        print $0 suma "|" sumb "|"
    }
'| tee tmp.csv | tr '.,' ',.' | datamash --sort -t\| -H -g9,10 sum 5 sum 6 sum
17 sum 18 mean 5 mean 6 mean 7 count 1
```

B.1.2. Abfrage 2

```
#!/bin/bash
# 2014-08-20
## R >< N >< S
# R >< N (regionkey)
cut nation.csv -d\\ -f1-3 > tmp1.csv
cut nation.tbl -d\\ -f1-3 | sort -t\\ -k3,3 >> tmp1.csv
cut region.csv -d\\ -f1 > tmp2.csv
cat region.tbl | grep "EUROPE" | cut -d\\ -f1 | sort >> tmp2.csv
join --header -t\\ -1 3 -2 1 tmp1.csv tmp2.csv > tmpprn.csv

# >< S (nationkey)
cut supplier.csv -d\\ -f1-7 > tmp1.csv
cut supplier.tbl -d\\ -f1-7 | sort -t\\ -k4,4 >> tmp1.csv
head -1 tmpprn.csv > tmp2.csv
tail -n+2 tmpprn.csv | sort -t\\ -k2,2 >> tmp2.csv
join --header -t\\ -1 4 -2 2 tmp1.csv tmp2.csv > tmpprns.csv

# PS >< RNS (suppkey)
cut partsupp.csv -d\\ -f1,2,4 > tmp1.csv
cut partsupp.tbl -d\\ -f1,2,4 | sort -t\\ -k2,2 >> tmp1.csv
head -1 tmpprns.csv > tmp2.csv
tail -n+2 tmpprns.csv | sort -t\\ -k2,2 >> tmp2.csv
join --header -t\\ -1 2 -2 2 tmp1.csv tmp2.csv | tee join1.csv | \
# group by partkey: ps_supplycost = select min(ps_supplycost)
tr ',.' '.,' | datamash -H -s -t\\ -g2 min 3 | tr ',.' '.,' > join2.csv

# P >< PRNS (partkey)
#part: restrictions: p_size = 15; residuals: p_type like %BRASS
cut part.csv -d\\ -f1,2,3 > tmp1.csv
cat part.tbl | awk -F\\ '
    $6==15 && $5~/.*BRASS/ {print $1,$2,$3}' OFS=\\ | \
    sort -t\\ -k1,1 >> tmp1.csv
head -1 join2.csv > tmp2.csv
tail -n+2 join2.csv | sort -t\\ -k1,1 >> tmp2.csv
join --header -t\\ -1 1 -2 1 tmp1.csv tmp2.csv > join3.csv

# >< PS (partkey)
cut partsupp.csv -d\\ -f1,2,4 > tmp1.csv
cut partsupp.tbl -d\\ -f1,2,4 | sort -t\\ -k1,1 >> tmp1.csv
join --header -t\\ -1 1 -2 1 tmp1.csv join3.csv | tee join4.csv | \
awk -F\\ 'NR==1 || $3==$6 {print $0}' > join5.csv

# RNS >< P PS (suppkey)
head -1 tmpprns.csv > tmp1.csv
tail -n+2 tmpprns.csv | sort -t\\ -k2,2 >> tmp1.csv
head -1 join5.csv > tmp2.csv
tail -n+2 join5.csv | sort -t\\ -k2,2 >> tmp2.csv
join --header -t\\ -1 2 -2 2 tmp1.csv tmp2.csv | \
awk -F\\ '{print $6,$3,$9,$10,$12,$4,$5,$7}' OFS=\\ > join6.csv

#s_acctbal,s_name,s_name,s_partkey,s_mfgr,s_address,s_phone,s_comment
head -1 join6.csv
tail -n+2 join6.csv | sort -t\\ -k1,1 -nr
```

```
rm join {1,2,3,4,5,6}.csv tmp{1,2,rn,rns}.csv
```

B.1.3. Abfrage 3

```
#!/bin/bash
sort -t\\| -k1,1 orders.tbl | cat orders.csv - | awk -F\\| '
    NR==1{
        print $1, $2, $5, $8
    }
    NR>1 && $5<"1995-03-15"{
        print $1, $2, $5, $8
    }
' OFS=\\| > tmporder.csv

sort -t\\| -k1,1 lineitem.tbl | cat lineitem.csv - | awk -F\\| '
    NR==1{
        print $1, "revenue"
    }
    NR>1 && $11>"1995-03-15"{
        suma=($6*(1.0-$7));
        print $1, suma
    }
' OFS=\\| > tmpline.csv

join --header -t\\| -1 1 -2 1 tmporder.csv tmpline.csv > tmplo.csv

# >< customer (custkey)
head -1 tmplo.csv > tmp1.csv
cat customer.csv > tmp2.csv
tail -n+2 tmplo.csv | sort -t\\| -k2,2 >> tmp1.csv
grep BUILDING customer.tbl | sort -t\\| -k1,1 >> tmp2.csv
join --header -t\\| -1 2 -2 1 -o 1.1 1.5 1.3 1.4 tmp1.csv tmp2.csv > output.csv

# gruppieren und sortieren
cat output.csv | tr '.,' ',.' | datamash -H -t\\| -g1,3,4 sum 2 > tmp3.csv
head -1 tmp3.csv; sort -t\\| -k2 tmp3.csv | sort -t\\| -k4 -nr | head -10

rm tmpline.csv tmporder.csv tmplo.csv tmp1.csv tmp2.csv tmp3.csv output.csv
```

B.1.4. Abfrage 4

```
#!/bin/bash
sort -k1,1 -t\\ | orders.tbl | cat orders.csv - | awk -F\\ | '
    NR==1{
        print $1"|" $6
    }
    NR>1 && $5<"1993-10-01" && $5>="1993-07-01"{
        print $1"|" $6
    }
' > tmporder.csv

sort -k1,1 -t\\ | lineitem.tbl | cat lineitem.csv - | awk -F\\ | '
    NR==1 || $12<$13{
        print $1
    }
' | uniq > tmpline.csv

join --header -t\\ | -1 1 -2 1 tmporder.csv tmpline.csv | \
tr '.,' ',.' | datamash -H -s -t\\ | -g2 count 2

rm tmpline.csv tmporder.csv
```

B.1.5. Abfrage 5

```
#!/bin/bash
# 2014-08-20
## R >< N >< C >< O >< L
# R >< N (regionkey)
cut nation.csv -d\\ -f1-3 > tmp1.csv
cut nation.tbl -d\\ -f1-3 | sort -t\\ -k3,3 >> tmp1.csv
cut region.csv -d\\ -f1 > tmp2.csv
cat region.tbl | grep "ASIA" | cut -d\\ -f1 | sort >> tmp2.csv
join --header -t\\ -1 3 -2 1 tmp1.csv tmp2.csv > tmprn.csv

# >< C (nationkey)
cut customer.csv -d\\ -f1-7 > tmp1.csv
cut customer.tbl -d\\ -f1-7 | sort -t\\ -k4,4 >> tmp1.csv
head -1 tmprn.csv > tmp2.csv
tail -n+2 tmprn.csv | sort -t\\ -k2,2 >> tmp2.csv
join --header -t\\ -1 4 -2 2 tmp1.csv tmp2.csv > tmprnc.csv

# >< O (custkey)
cut orders.csv -d\\ -f1,2,4 > tmp1.csv
cat orders.tbl | awk -F\\ ' $5>="1994-01-01" && $5<"1995-01-01"{print $0}' | cut -
d\\ -f1,2,4 | sort -t\\ -k2,2 >> tmp1.csv
head -1 tmprnc.csv > tmp2.csv
tail -n+2 tmprnc.csv | sort -t\\ -k2,2 >> tmp2.csv
join --header -t\\ -1 2 -2 2 tmp1.csv tmp2.csv > join1.csv

# >< L (orderkey)
cat lineitem.csv | awk -F\\ '{print $1,$3,"revenue"}' OFS=\\ > tmp1.csv
cat lineitem.tbl | awk -F\\ '{
    rev=$6*(1.0-$7); print $1,$3,rev
}' OFS=\\ | sort -t\\ -k1,1 >> tmp1.csv
head -1 join1.csv > tmp2.csv
tail -n+2 join1.csv | sort -t\\ -k2,2 >> tmp2.csv
join --header -t\\ -1 1 -2 2 tmp1.csv tmp2.csv > join2.csv

# >< S (suppkey) && c_nationkey==s_nationkey
cut supplier.csv -d\\ -f1-7 > tmp1.csv
cut supplier.tbl -d\\ -f1-7 | sort -t\\ -k1,1 >> tmp1.csv
head -1 join2.csv > tmp2.csv
tail -n+2 join2.csv | sort -t\\ -k2,2 >> tmp2.csv
join --header -t\\ -1 1 -2 2 tmp1.csv tmp2.csv | awk -F\\ '
NR==1 || $4==$12{print $0}
' > join3.csv

# nationkey==nationkey
cat join3.csv | tr '.,' ',.' | datamash -H -t\\ -s -g19 sum 9 > join4.csv
head -1 join4.csv
tail -n+2 join4.csv | sort -t\\ -k2,2 -nr

rm join{1,2,3,4}.csv tmp{1,2,rn,rnc}.csv
```

B.1.6. Abfrage 6

```
#!/bin/bash
cat lineitem.* | awk -F\| '
    NR==1{
        print $1 "|revenue"
    }
    NR>1 && $11>="1994-01-01" && $11<"1995-01-01" && $7>=0.05 && $7<=0.07 &&
    $5<24{
        suma=$6*$7;
        print $1 "|" suma
    }
'| tr '.,' '.,' | datamash -H -t\| sum 2
```

B.1.7. Abfrage 7

```
#!/bin/bash
# supplier mit nation1 und customer nation2
sort -t\\ | -k1,1 nation.tbl | cut nation.csv - -d\\ | -f1,2 | awk -F\\ | '
    NR==1{
        print $0
    }
    NR>1 && ($2=="FRANCE" || $2=="GERMANY"){
        print $0
    }
' > tmpnation.csv
#header
awk -F\\ | 'NR==1{print $1"|" $4}' supplier.csv > tmp2.csv
awk -F\\ | 'NR==1{print $1"|" $4}' customer.csv > tmp3.csv
#rest
awk -F\\ | '{print $1"|" $4}' supplier.tbl | sort -t\\ | -k2,2 >> tmp2.csv
awk -F\\ | '{print $1"|" $4}' customer.tbl | sort -t\\ | -k2,2 >> tmp3.csv
join --header -t\\ | -1 2 -2 1 tmp2.csv tmpnation.csv > suppnation.csv
join --header -t\\ | -1 2 -2 1 tmp3.csv tmpnation.csv > custnation.csv

#s_suppkey=l_suppkey
#l_shipdate between 95-01-01 and 96-12-31
#l_orderkey, l_suppkey, l_shipdate, volume
cat lineitem.* | awk -F\\ | '
    NR==1{
        print $1 "|" $3 "|" $11 "|volume"
    }
    NR>1 && $11>="1995-01-01" && $11 <="1996-12-31"{
        suma=($6*(1.0-$7));
        print $1 "|" $3 "|" $11 "|" suma
    }
' > tmpline.csv
head -1 tmpline.csv > tmp1.csv
tail -n+1 tmpline.csv | sort -t\\ | -k2,2 >> tmp1.csv
head -1 suppnation.csv > tmp2.csv
tail -n+1 suppnation.csv | sort -t\\ | -k2,2 >> tmp2.csv
join --header -t\\ | -1 2 -2 2 tmp1.csv tmp2.csv > join1.csv

#orders.csv mit join1: o_orderkey = l_orderkey
head -1 join1.csv > tmp1.csv
tail -n+1 join1.csv | sort -t\\ | -k2,2 >> tmp1.csv
cut -d\\ | -f1,2 orders.csv > tmp2.csv
sort -t\\ | -k1,1 orders.tbl | cut -d\\ | -f1,2 >> tmp2.csv
join --header -t\\ | -1 1 -2 2 tmp2.csv tmp1.csv > join2.csv

# o_custkey=c_custkey | join aus s,n1,l,o zu join aus c,n2
head -1 custnation.csv > tmp1.csv
tail -n+1 custnation.csv | sort -t\\ | -k2,2 >> tmp1.csv
head -1 join2.csv > tmp2.csv
tail -n+1 join2.csv | sort -t\\ | -k2,2 >> tmp2.csv
join --header -t\\ | -1 2 -2 2 tmp1.csv tmp2.csv | awk -F\\ | '
    NR==1{
        print "supp_nation", "cust_nation", "l_year", "revenue"
    }
'
```



```
NR>1 && (($9=="FRANCE" && $3=="GERMANY") || ($3=="FRANCE" && $9=="
    GERMANY")){
    print $9, $3, substr($6,1,4), $7
}
' OFS=\\ | tr ',.' '.,' | datamash -H -s -t\\ | -g1,2,3 sum 4

rm tmp1.csv tmp2.csv custnation.csv suppnation.csv join1.csv join2.
csv
```

B.1.8. Abfrage 8

```
#!/bin/bash
## O >< C >< N1 >< R
# N1 >< R (regionkey)
head -1 region.csv | cut -d\| -f1 > tmpregion.csv
grep '|AMERICA|' region.tbl | cut -d\| -f1 | sort >> tmpregion.csv
head -1 nation.csv | cut -d\| -f1,3 > tmpnation.csv
cat nation.tbl | cut -d\| -f1,3 | sort -t\| -k2,2 >> tmpnation.csv
join --header -t\| -1 1 -2 2 tmpregion.csv tmpnation.csv > tmpnr.csv

# C >< (nationkey)
head -1 customer.csv | cut -d\| -f1,4 > tmp1.csv
cat customer.tbl | cut -d\| -f1,4 | sort -t\| -k2,2 >> tmp1.csv
head -1 tmpnr.csv > tmp2.csv
tail -n+2 tmpnr.csv | sort -t\| -k2,2 >> tmp2.csv
join --header -t\| -1 2 -2 2 tmp1.csv tmp2.csv > tmpocn1r.csv

# O >< (custkey)
cat orders.* | awk -F\| '
    NR==1{
        print $1 "|" $2 "|year"
    }
    NR>1 && ( $5>="1995-01-01" && $5<="1996-12-31"){
        print $1 "|" $2 "|" substr($5,1,4)
    }
' > tmporder.csv
head -1 tmporder.csv > tmp1.csv
tail -n+2 tmporder.csv | sort -t\| -k2,2 >> tmp1.csv
head -1 tmpocn1r.csv > tmp2.csv
tail -n+2 tmpocn1r.csv | sort -t\| -k2,2 >> tmp2.csv
join --header -t\| -1 2 -2 2 tmp1.csv tmp2.csv > join1.csv

## P >< L >< S >< N2 (nationkey)
# S >< N2
head -1 supplier.csv | cut -d\| -f1,4 > tmp2.csv
cat supplier.tbl | cut -d\| -f1,4 | sort -t\| -k2,2 >> tmp2.csv
cut -d\| -f1,2 nation.csv > tmp1.csv
cut -d\| -f1,2 nation.tbl | sort -t\| -k1,1 >> tmp1.csv
join --header -t\| -1 1 -2 2 tmp1.csv tmp2.csv > tmpsn.csv

# L >< P (partkey)
cat lineitem.* | awk -F\| '
    NR==1{
        print $1 "|" $2 "|" $3 "|volume"
    }
    NR>1 && $11>="1995-01-01" && $11<="1996-12-31">{
        suma=($6*(1.0-$7))
        print $1 "|" $2 "|" $3 "|" suma
    }
' > tmpline.csv
head -1 tmpline.csv > tmp1.csv
tail -n+2 tmpline.csv | sort -t\| -k2,2 >> tmp1.csv
head -1 part.csv | cut -d\| -f1 > tmp2.csv
cat part.tbl | grep 'ECONOMY ANODIZED STEEL'| cut -d\| -f1 | sort >> tmp2.csv
join --header -t\| -1 2 -2 1 tmp1.csv tmp2.csv > join2.csv
```

```
# L >< O... (orderkey)
head -1 join2.csv > tmp1.csv
tail -n+2 join2.csv | sort -t\| -k2,2 >> tmp1.csv
head -1 join1.csv > tmp2.csv
tail -n+2 join1.csv | sort -t\| -k2,2 >> tmp2.csv
join --header -t\| -1 2 -2 2 tmp1.csv tmp2.csv > join3.csv

# L >< S (suppkey)
head -1 join3.csv > tmp1.csv
tail -n+2 join3.csv | sort -t\| -k3,3 >> tmp1.csv
head -1 tmpsn.csv > tmp2.csv
tail -n+2 tmpsn.csv | sort -t\| -k3,3 >> tmp2.csv
join --header -t\| -1 3 -2 3 tmp1.csv tmp2.csv > output.csv

cat output.csv | tr ',.' '.,' | datamash -H -s -t\| -g6,10 sum 4 | awk -F\| '
    NR==1{
        print $0 "|volume_g"
    }
    NR>1 && $2=="BRAZIL"{
        print $0 "|" $3
    }
    NR>1 && $2!="BRAZIL"{
        print $0 "|" 0
    }
' | datamash -H -s -t\| -g1 sum 3 sum 4 | awk -F\| '
    NR==1{
        "o_year|mkt_share"
    }
    NR>1{
        print $1 "|" $3/$2
    }
,

rm tmp*.csv join1.csv join2.csv output.csv
```

B.1.9. Abfrage 9

```
#!/bin/bash
echo 'N >< S (nationkey)'
head -1 supplier.csv | cut -d\| -f1,4 > tmp2.csv
cat supplier.tbl | cut -d\| -f1,4 | sort -t\| -k2,2 >> tmp2.csv
cut -d\| -f1,2 nation.csv > tmp1.csv
cut -d\| -f1,2 nation.tbl | sort -t\| -k1,1 >> tmp1.csv
join --header -t\| -1 1 -2 2 tmp1.csv tmp2.csv > tmpsn.csv

echo 'L >< P (partkey)'
head -1 lineitem.csv | cut -d\| -f1,2,3,5,6,7 > tmp1.csv
cat lineitem.tbl | cut -d\| -f1,2,3,5,6,7 | sort -t\| -k2,2 >> tmp1.csv
head -1 part.csv | cut -d\| -f1 > tmp2.csv
cat part.tbl | grep 'green' | cut -d\| -f1 | sort -t\| -k1,1 >> tmp2.csv
join --header -t\| -1 2 -2 1 tmp1.csv tmp2.csv > join1.csv

echo 'join1 >< PS (partkey && supkey)'
#_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount
sort -t\| -k1,1 partsupp.tbl | cut partsupp.csv - -d\| -f1,2,3,4 \|
join --header -t\| -1 1 -2 1 join1.csv - | awk -F\| '
    NR==1{
        print $2 "|" $3 "|amount"
    }
    NR>1 && $3==$7{
        sum=$5*(1.0-$6)-$9*$4
        print $2 "|" $3 "|" sum
    }
'> join2.csv

echo 'join2 >< O (orderkey)'
head -1 join2.csv > tmp1.csv
tail -n+2 join2.csv | sort -t\| -k1,1 >> tmp1.csv
cut -d\| -f1,5 orders.csv > tmp2.csv
cut -d\| -f1,5 orders.tbl | sort -t\| -k1,1 >> tmp2.csv
join --header -t\| -1 1 -2 1 tmp1.csv tmp2.csv > join3.csv

echo 'join3 >< SN (supkey)'
head -1 join3.csv > tmp1.csv
tail -n+2 join3.csv | sort -t\| -k2,2 >> tmp1.csv
head -1 tmpsn.csv > tmp2.csv
tail -n+2 tmpsn.csv | sort -t\| -k3,3 >> tmp2.csv
join --header -t\| -1 2 -2 3 tmp1.csv tmp2.csv \|
awk -F\| '
    NR==1{
        print $6 "|year|" $3
    }
    NR>1{
        print $6 "|" substr($4,1,4) "|" $3
    }
' \|
tr '.,' '._' | datamash -H -s -t\| -g1,2 sum 3
#sort -t\| -k2,2 -r | sort -t\| -k1,1

rm tmp*.csv join1.csv join2.csv join3.csv
```

B.1.10. Abfrage 10

```
#!/bin/bash
## N >< C >< O >< L
echo 'O >< L (orderkey)'
cut -d\| -f1,2 orders.csv > tmporder.csv
awk -F\| '
    $5>="1993-10-01" && $5<"1994-01-01"{
        print $1 "|" $2
    }
' orders.tbl | sort -t\| -k1,1 >> tmporder.csv

awk -F\| '{print $1 "|revenue"}' lineitem.csv > tmp1.csv
awk -F\| '$9=="R"{
    suma=($6*(1.0-$7));
    print $1 "|" suma
}' lineitem.tbl | sort -t\| -k1,1 >> tmp1.csv

join --header -t\| -1 1 -2 1 tmporder.csv tmp1.csv > tmp2.csv

echo 'C >< (custkey)'
cat customer.csv > tmp1.csv
cut -d\| -f8 customer.tbl | sort -t\| -k1,1 >> tmp1.csv
head -1 tmp2.csv > tmp3.csv
tail -n+2 tmp2.csv | sort -t\| -k2,2 >> tmp3.csv
join --header -t\| -1 1 -2 2 tmp1.csv tmp3.csv > join1.csv

echo 'N >< (nationkey)'
head -1 join1.csv > tmp2.csv
tail -n+2 join1.csv | sort -t\| -k4,4 >> tmp2.csv
cut -d\| -f1,2 nation.csv > tmp1.csv
cut -d\| -f1,2 nation.tbl | sort -t\| -k1,1 >> tmp1.csv
join --header -t\| -1 1 -2 4 tmp1.csv tmp2.csv | \
tr ' ' '\t' | datamash -H -s -t\| -g3,4,7,6,2,5,8 sum 11 > tmp.csv

head -1 tmp.csv;
tail -n+2 tmp.csv | sort -t\| -nrk8 | head -20

rm tmp*.csv join1.csv
```

B.1.11. Abfrage 11

```
#!/bin/bash
echo 'N >< S'
head -1 supplier.csv | cut -d\| -f1,4 > tmp1.csv
cat supplier.tbl | cut -d\| -f1,4 | sort -t\| -k2,2 >> tmp1.csv
head -1 nation.csv | cut -d\| -f1 > tmp2.csv
cat nation.tbl | grep 'GERMANY' | cut -d\| -f1 | sort >> tmp2.csv
join --header -t\| -1 2 -2 1 tmp1.csv tmp2.csv > tmpsn.csv

echo 'PS >< (partkey)'
head -1 partsupp.csv | cut -d\| -f1,2,3,4 > tmp1.csv
cat partsupp.tbl | cut -d\| -f1,2,3,4 | sort -t\| -k2,2 >> tmp1.csv
head -1 tmpsn.csv | cut -d\| -f2 > tmp2.csv
tail -n+2 tmpsn.csv | cut -d\| -f2 | sort >> tmp2.csv
join --header -t\| -1 2 -2 1 tmp1.csv tmp2.csv > join.csv

#echo 'sum(ps_supplycost * ps_availqty) * 0.0001'
sum='awk -F\| '
    BEGIN{SUM=0}
    NR>1{SUM+=$3*$4}
    END{print SUM*0.0001}
' join.csv '
#echo $sum

# 'sum(ps_supplycost * ps_availqty); sum(ps_supplycost * ps_availqty)>$sum'
awk -F\| '
    NR==1{
        print $2"|value"
    }
    NR>1 {
        mul=$3*$4
        print $2"|mul"
    }
' join.csv | tr ',.' '.,' | datamash -H -s -t\| -g1 sum 2 | awk -F\| "
    NR==1||\ $2>$sum{
        print_\ $0
    }
" > tmpfile.csv
head -1 tmpfile.csv
tail -n+2 tmpfile.csv | sort -t\| -k2,2 -nr

rm tmp*.csv join.csv
```

B.1.12. Abfrage 12

```

#!/bin/bash
## N >< C >< O >< L
# O >< L (orderkey)
#       o_orderkey = l_orderkey
# $15      and l_shipmode in ('MAIL', 'SHIP')
# $12<$13   and l_commitdate < l_receiptdate
# $11<12     and l_shipdate < l_commitdate
# $13      and l_receiptdate >= date '1994-01-01'
# $13      and l_receiptdate < date '1995-01-01'
sort -t\\ | -k1,1 orders.tbl | \
cat orders.csv - | awk -F\\ | '
    NR==1{
        print $1 "|high_line_count|low_line_count"
    }
    NR>1{
        if ( $6=="1-URGENT" || $6=="2-HIGH" )
            print $1 "|1|0"
        else
            print $1 "|0|1"
    }
' > tmporder.csv

sort -t\\ | -k1,1 lineitem.tbl | \
cat lineitem.csv - | awk -F\\ | '
    NR==1{
        print $1 "|" $15
    }
    NR>1 && $13>="1994-01-01" && $13<"1995-01-01" && $12<$13 && $11<$12 && (
        $15=="MAIL" || $15=="SHIP"){
        print $1 "|" $15
    }
' > tmpline.csv

join --header -t\\ | -1 1 -2 1 tmporder.csv tmpline.csv | \
tr '.,' '.,.' | datamash -H -s -t\\ | -g4 sum 2 sum 3
rm tmp*.csv

```

B.1.13. Abfrage 13

```
#!/bin/bash
# C |>< O
# left outer join ^= join -a1 -a2 -1 2 -2 2 -o 0 1.1 2.1 -e "0" 1.txt 2.txt
#join -a1 -a2 -1 2 -2 2 -o 0 1.1 -e "0" 1.txt 2.txt
cut -d\\| -f1,2 customer.csv > tmp1.csv
cut -d\\| -f1,2 customer.tbl | sort -t\\| -k1,1 >> tmp1.csv
head -1 orders.csv | cut -d\\| -f2 > tmp2.csv
cat orders.tbl |grep -v 'special.*requests' | cut -d\\| -f2 | sort >> tmp2.csv
join --header -t\\| -1 1 -2 1 -a1 -o 0 2.1 -e "NULL" tmp1.csv tmp2.csv | tee join
.csv |\\
sed 's/|[0-9][0-9]*|1/' | sed 's/|NULL/|0/' | datamash -H -s -t\\| -g1 sum 2 |
datamash -H -s -t\\| -g2 count 1 > tmp.csv

head -1 tmp.csv; tail -n+2 tmp.csv | sort -nr | sort -nrk2,2 -t\\|

rm tmp*.csv join.csv
```


B.1.14. Abfrage 14

```
#!/bin/bash
# P >< L (partkey)

cat lineitem.* | awk -F\| '
    NR==1{
        print $2 "|revenue"
    }
    NR>1 && $11>="1995-09-01" && $11<"1995-10-01" {
        suma=($6*(1.0-$7))*100;
        print $2 "|" suma
    }
' > tmp1.csv

head -1 tmp1.csv > tmp2.csv
tail -n+2 tmp1.csv | sort -t\| -k1,1 >> tmp2.csv

cut -d\| -f1,5 tmp2.csv > tmp1.csv
cut -d\| -f1,5 tmp2.csv | sort -t\| -k1,1 >> tmp1.csv
join --header -t\| -1 1 -2 1 tmp1.csv tmp2.csv | \
awk -F\| '
    BEGIN{sum=0; sumpromo=0; print "promo_revenue"}
    {sum+=$3}
    $2~/PROMO.*/{sumpromo+=$3}
    END{print sumpromo/sum*100}
'
rm tmp*.csv
```

B.1.15. Abfrage 15

```
#!/bin/bash
#revenue.csv
cat lineitem.* | awk -F\| '
    NR==1{
        print "supplier_no|total_revenue"
    }
    NR>1 && $11>="1996-01-01" && $11<"1996-04-01" {
        suma=($6*(1.0-$7));
        print $3 "|" suma
    }
' | tr '.,' ',.' | datamash -s -H -t\| -g1 sum 2 > revenue.csv
max='datamash -H -t\| max 2 < revenue.csv | tail -n+2'
# S >< revenue.csv
head -1 revenue.csv > tmp2.csv
grep "|$max" revenue.csv | sort -t\| -k1,1 >> tmp2.csv
sort -t\| -k1,1 supplier.tbl |
cut -d\| -f1,2,3,5 supplier.csv - \|
join --header -t\| -1 1 -2 1 - tmp2.csv
#rm tmp2.csv revenue.csv
```

B.1.16. Abfrage 16

```

#!/bin/bash
# 2014-08-19
# PS >< P
# $4 p_brand <> 'Brand#45'
# $5      and p_type not like 'MEDIUM POLISHED%'
# $6      and p_size in (49, 14, 23, 45, 19, 3, 36, 9)
sort -t\\ -k1,1 part.tbl | cat part.csv - | awk -F\\ '
    NR==1{
        print $1,$4,$5,$6
    }
    NR>1 && $4!="Brand#45" && $5!~/^MEDIUM POLISHED.* / && ($6==49 || $6==14
        || $6==23 || $6==45 || $6==19 || $6==3 || $6==36 || $6==9){
        print $1,$4,$5,$6
    }
' OFS=\\ > tmppart.csv
sort -t\\ -k1,1 partsupp.tbl | cut -d\\ -f1-4 partsupp.csv - > tmpsupp.csv
join --header -t\\ -1 1 -2 1 tmpsupp.csv tmppart.csv > join1.csv

# PS |> S (suppkey)
head -1 join1.csv | cut -d\\ -f1,2,5,6,7,8 > tmp1.csv
tail -n+2 join1.csv | cut -d\\ -f1,2,5,6,7,8 | sort -t\\ -k2,2 >> tmp1.csv
cut -d\\ -f1 supplier.csv > tmp2.csv
sed -r '/Customer.*Complaints/ d' supplier.tbl | cut -d\\ -f1 | sort >> tmp2.csv
join --header -t\\ -1 2 -2 1 tmp1.csv tmp2.csv > join2.csv

#group by p_brand, p_type, p_size; count(distinct ps_suppkey)
cat join2.csv | datamash -H -s -t\\ -g3,4,5 countunique 1 > tmpout.csv

# supplier_cnt desc, p_brand, p_type, p_size
head -1 tmpout.csv; tail -n+2 tmpout.csv | sort | sort -t\\ -nr -k4,4

rm tmp*.csv join1.csv join2.csv

```

B.1.17. Abfrage 17

```
#!/bin/bash
# 2014-08-19
## L >< P (partkey)
head -1 lineitem.csv | cut -d\| -f2,5,6 > tmp1.csv
cut -d\| -f2,5,6 lineitem.tbl | sort -t\| -k1,1 >> tmp1.csv

head -1 part.csv | cut -d\| -f1,4,7 > tmp2.csv
cut -d\| -f1,4,7 part.tbl | sort -t\| -k1,1 >> tmp2.csv
join --header -t\| -1 1 -2 1 tmp1.csv tmp2.csv > join1.csv

#avg(l_quantity) pro partkey
tr '.,' ' ' < join1.csv | datamash -H -s -t\| -g1 mean 2 | tr '.,' ' ' >
    tmpavgs.csv
#dann join mit join1
join --header -t\| -1 1 -2 1 join1.csv tmpavgs.csv | tee join2.csv | \
#greife die relevanten raus
awk -F\| '
    NR==1{
        print $0
    }
    NR>1 && $4=="Brand#23" && $5=="MED_BOX" && $2<$6*0.2 {
        print $0
    }
' | tee join3.csv | \
awk -F\| '
    BEGIN{
        print "avg_yearly"
        sum=0
    }
    NR>1{ sum+=$3}
    END{ print sum/7.0 }
,
rm tmp*.csv join1.csv join2.csv join3.csv
```

B.1.18. Abfrage 18

```
#!/bin/bash
# 2014-08-20
# group by l_orderkey, having sum qunatity >300
cat lineitem.* | tr '.,' ' ,' | datamash -H -t\\ -g1 -s sum 5 | awk -F\\ | '
    NR==1 || $2>300 { print $0 } ' > tmp2.csv
# O >< L (orderkey)
sort -t\\ -k1,1 orders.tbl | cut -d\\ -f1-9 orders.csv - > tmp1.csv
join --header -t\\ -1 1 -2 1 tmp1.csv tmp2.csv > join1.csv

#c_name,c_custkey,o_orderkey,o_orderdate,o_totalprice
head -1 join1.csv | cut -d\\ -f1,2,4,5,10 > tmp2.csv
tail -n+2 join1.csv | cut -d\\ -f1,2,4,5,10 | sort -t\\ -k2,2 >> tmp2.csv
sort -t\\ -k1,1 customer.tbl | cut -f1,2 customer.csv - > tmp1.csv
join --header -t\\ -1 1 -2 2 -o 1.2 0 2.1 2.3 2.2 2.4 2.5 tmp1.csv tmp2.csv >
    join2.csv

head -1 join2.csv
tail -n+2 join2.csv | sort -t\\ -k6 | sort -t\\ -k4,4 -r | head -100

rm tmp*.csv join1.csv join2.csv
```

B.1.19. Abfrage 19

```
#!/bin/bash
# 2014-08-20
#l_shipmode in ('AIR', 'AIR REG')and l_shipinstruct = 'DELIVER IN PERSON'
cat lineitem.* | awk -F\| '
    NR==1{
        print $2 "|" $5"|revenue"
    }
    NR>1 && $14=="DELIVER_IN_PERSON" && ($15=="AIR" || $15=="AIR_REG"){
        suma=($6*(1.0-$7));
        print $2 "|" $5"| suma
    }
' > tmp1.csv

# P <> L (partkey)
head -1 tmp1.csv > tmp1.csv
tail -n+2 tmp1.csv | sort -t\| -k1,1 >> tmp1.csv
cat part.csv > tmp2.csv
cut -d\| -f1-9 part.tbl | sort -t\| -k1,1 >> tmp2.csv
join --header -t\| -1 1 -2 1 tmp1.csv tmp2.csv | tee join.csv | awk -F\| '
    BEGIN{ print "revenue"; sum=0}
    $6=="Brand#12" &&
    ($9=="SM_CASE" || $9=="SM_BOX" ||
    $9=="SM_PACK" || $9=="SM_PKG") &&
    $2>=1 && $2<=11 &&
    $8<=5 && $8>=1{
        sum+=$3
    }
    $6=="Brand#23" &&
    ($9=="MED_BAG" || $9=="MED_BOX" ||
    $9=="MED_PACK" || $9=="MED_PKG") &&
    $2>=10 && $2<=20 &&
    $8<=10 && $8>=1{
        sum+=$3
    }
    $6=="Brand#34" &&
    ($9=="LG_CASE" || $9=="LG_BOX" ||
    $9=="LG_PACK" || $9=="LG_PKG") &&
    $2>=20 && $2<=30 &&
    $8<=15 && $8>=1{
        sum+=$3
    }
    END{print sum}
'
# and p_brand = 'Brand#12' and p_container in ('SM CASE', 'SM BOX', 'SM PACK', '
SM PKG') and l_quantity >= 1 and l_quantity <= 1 + 10
# and p_brand = 'Brand#23'and p_container in ('MED BAG', 'MED BOX', 'MED PKG',
'MED PACK') and l_quantity >= 10 and l_quantity <= 10 + 10and p_size between
1 and 10
#and p_brand = 'Brand#34' and p_container in ('LG CASE', 'LG BOX', 'LG PACK', '
LG PKG' and l_quantity >= 20 and l_quantity <= 20 + 10and p_size between 1
and 15
rm tmp*.csv join.csv
```

B.1.20. Abfrage 20

```
#!/bin/bash
# 2014-08-20
# P >< PS (partkey)
cat partsupp.tbl | sort -t\\ | -k1,1 | cut -d\\ | -f1-5 partsupp.csv - > tmp1.csv
cut -d\\ | -f1 part.csv > tmp2.csv
awk -F\\ | '$2~/^forest/{print $1}' part.tbl | sort -t\\ | -k1,1 >> tmp2.csv
join --header -t\\ | -1 1 -2 1 tmp1.csv tmp2.csv > join1.csv

#l_shipdate >= date '1994-01-01' and l_shipdate < date '1995-01-01'
cat lineitem.* | awk -F\\ | '
    NR==1{
        print $2,$3,$5
    }
    NR>1 && $11>="1994-01-01" && $11<"1995-01-01"{
        print $2,$3,$5
    }
' OFS=\\ | | tr '.,' '.,.' | datamash -H -s -t\\ | -g1,2 sum 3 |\\
# L >< (l_partkey = ps_partkey)
join --header -t\\ | -1 1 -2 1 join1.csv - | tr '.,' '.,.' |\\
# l_suppkey = ps_suppkey
awk -F\\ | 'NR==1 || ($2==$6 && $3>$7*0.5){print $2}' > join2.csv

# S >< (suppkey in ...) '
head -1 join2.csv > tmp2.csv
tail -n+2 join2.csv | sort - >> tmp2.csv
cat supplier.tbl | sort -t\\ | -k1,1 |\\
cut -d\\ | -f1-7 supplier.csv - > tmp1.csv
join --header -t\\ | -1 1 -2 1 tmp1.csv tmp2.csv > join3.csv

# S >< N (nationkey)
head -1 join3.csv > tmp1.csv
tail -n+2 join3.csv | sort -t\\ | -k4,4 >> tmp1.csv
awk -F\\ | 'NR==1 || $2=="CANADA"{print $1}' nation.* |\\
join --header -t\\ | -1 4 -2 1 tmp1.csv - | cut -d\\ | -f3,4 > join4.csv

head -1 join4.csv
tail -n+2 join4.csv | sort -u

rm tmp*.csv join*.csv
```

B.1.21. Abfrage 21

```
#!/bin/bash
## N >< S >< L1 >< O |> L
echo ' N >< S' 1>&2
cut -d\\ -f1,2,4 supplier.csv > tmp1.csv
cut -d\\ -f1,2,4 supplier.tbl | sort -t\\ -k3,3 >> tmp1.csv
cut -d\\ -f1 nation.csv > tmp2.csv
cat nation.tbl | grep 'SAUDI ARABIA' | cut -d\\ -f1 >> tmp2.csv
join --header -t\\ -1 3 -2 1 tmp1.csv tmp2.csv > tmpsn.csv

echo ' >< L1 (suppkey)' 1>&2
#l1.l_receiptdate > l1.l_commitdate
awk -F\\ 'NR==1 || $13>$12 {print $1,$3}' OFS=\\ lineitem.* > tmp1.csv
head -1 tmp1.csv > tmppli.csv
tail -n+2 tmp1.csv | sort -t\\ -k2,2 >> tmp1.csv
head -1 tmpsn.csv > tmp2.csv
tail -n+2 tmpsn.csv | sort -t\\ -k2,2 >> tmp2.csv
join --header -t\\ -1 2 -2 2 tmp1.csv tmp2.csv > join2.csv

echo ' >< O (orderkey)' 1>&2
#l1.l_receiptdate > l1.l_commitdate
head -1 orders.csv | cut -d\\ -f1 > tmpA.csv
awk -F\\ 'NR==1 && $3=="F" {print $1}' orders.tbl | sort >> tmpA.csv
head -1 join2.csv > tmpB.csv
tail -n+2 join2.csv | sort -t\\ -k2,2 >> tmpB.csv
join --header -t\\ -1 1 -2 2 tmpA.csv tmpB.csv > join3.csv

echo ' |> L3 (orderkey)' 1>&2
#l3.l_suppkey <> l1.l_suppkey
#awk -F\\ 'NR==1 || $13>$12 {print $1,$3}' OFS=\\ lineitem.csv > tmp1.csv
head -1 tmp1.csv > tmp2.csv
tail -n+2 tmp1.csv | sort -t\\ -k1,1 >> tmp2.csv
join --header -t\\ -1 1 -2 1 join3.csv tmp2.csv | \
  awk -F\\ 'NR==1 || $2!=$5 {print $0}' | \
  join --header -t\\ -1 1 -2 1 -v1 join3.csv - | cut -d\\ -f1-4 > join4.csv

echo ' >< L2 ( l2.l_orderkey = l1.l_orderkey and l2.l_suppkey <> l1.l_suppkey)'
1>&2
cut lineitem.* -d\\ -f1,3 | datamash -H -s -t\\ -g1 count 2 | awk -F\\ '
  NR==1 || $2>1 {print $1}' > tmp2.csv
join --header -t\\ -1 1 -2 1 join4.csv tmp2.csv | tee join5.csv | \
  datamash -H -s -t\\ -g4 count 1 > join6.csv

head -1 join6.csv
tail -n+2 join6.csv | sort -t\\ -snr -k2,2 | head -99

rm tmp*.csv join*.csv
```


B.1.22. Abfrage 22

```

#!/bin/bash
# 2014-08-20
avg='awk -F\\| 'BEGIN{sum=0; cnt=0; print "avg"}
NR>1 && $6>0 &&
( substr($5,1,2)==13||substr($5,1,2)==31||
substr($5,1,2)==23||substr($5,1,2)==29||
substr($5,1,2)==30||substr($5,1,2)==18||
substr($5,1,2)==17){
    sum+=$6; cnt+=1
}
END{print sum/cnt}
' customer.* | tail -1'

#c_acctbal > select avg(c_acctbal) from...
awk -F\\| -v avg=$avg '
NR==1{print $1,$6, "cntrycode"}
NR>1 && $6>avg &&
( substr($5,1,2)==13||substr($5,1,2)==31||
substr($5,1,2)==23||substr($5,1,2)==29||
substr($5,1,2)==30||substr($5,1,2)==18||
substr($5,1,2)==17){
    print $1,$6,substr($5,1,2)
}
' OFS=\\| customer.* > tmpc.csv

# O |> C
head -1 tmpc.csv > tmp1.csv
tail -n+2 tmpc.csv | sort -t\\| -k1,1 >> tmp1.csv
cut -d\\| -f2 orders.csv > tmp2.csv
cut -d\\| -f2 orders.tbl | sort >>tmp2.csv
join --header -t\\| -v1 -o 0 1.2 1.3 tmp1.csv tmp2.csv |\\
# group by
tr '.,' '.,.' | datamash -t\\| -s -H -g3 count 1 sum 2

rm tmp*.csv

```

B.1.23. Ergebnisse

query1	real 1m1.962s	user 1m5.244s	sys 0m3.312s
query10	real 0m15.708s	user 0m15.545s	sys 0m0.604s
query11	real 0m3.037s	user 0m2.980s	sys 0m0.164s
query12	real 0m23.359s	user 0m52.883s	sys 0m2.216s
query13	real 0m7.992s	user 0m9.929s	sys 0m0.340s
query14	real 0m8.099s	user 0m7.896s	sys 0m0.564s
query15	real 0m7.581s	user 0m7.416s	sys 0m0.588s
query16	real 0m3.532s	user 0m6.360s	sys 0m0.292s
query17	real 0m48.223s	user 0m49.755s	sys 0m2.332s
query18	real 0m18.802s	user 0m22.997s	sys 0m3.308s
query19	real 0m8.611s	user 0m8.573s	sys 0m0.716s
query2	real 0m6.494s	user 0m6.396s	sys 0m0.348s
query20	real 0m11.221s	user 0m12.149s	sys 0m0.992s
query21	real 0m43.384s	user 0m44.259s	sys 0m1.136s
query22	real 0m6.565s	user 0m6.496s	sys 0m0.120s
query3	real 0m28.086s	user 0m57.104s	sys 0m2.344s
query4	real 0m22.332s	user 0m52.435s	sys 0m2.384s
query5	real 0m37.344s	user 0m37.010s	sys 0m1.404s
query6	real 0m8.024s	user 0m7.864s	sys 0m0.532s
query7	real 0m23.258s	user 0m27.546s	sys 0m1.280s
query8	real 0m24.036s	user 0m23.249s	sys 0m1.140s
query9	real 0m45.607s	user 0m49.027s	sys 0m2.116s

Abbildung B.1.: Gemessene Zeiten der Abfragen mit SF10

C. Der Bash2SQL-Compiler

C.1. Yacc und Lex

C.1.1. Makefile

```
all: kleinerParser

kleinerParser: lex.yy.c y.tab.c
               cc -o $@ lex.yy.c y.tab.c -ll -lm

y.tab.c: bash2sql.y
          yacc -d $^

lex.yy.c: bash2sql.lex
          lex $^
```

C.1.2. bash2sql.lex

```
%{
#include "y.tab.h"
extern YYSTYPE yylval;
#include <stdio.h>
#include <stdlib.h>

%}

white          [ \t]+

digit          [0-9]
integer        {digit}+
exponent       [eE][+-]?{integer}

real           {integer}("."{integer})?{exponent}?

letter         [a-zA-Z0-9\.\_\|\|]+

options        -[a-zA-Z0-9\-\_\,]+
ende           [^\|];
%%

" _ _ "        return FSTDIN;

{white}        { /* We ignore white characters */ }

{real}         {
                yylval.int4=atoi(yytext);
                return (NUMBER);
            }
```

```
    }

">"          return(REDIR);

"| "         return(PIPE);
"cat"        return(CAT);
"cut"        return(CUT);
"grep"       return(GREP);
"sort"       return(SORT);
"join"       return(JOIN);
"d"          return(OPT_D);
"+"          return(PLUS);
"-"          return(MINUS);
"*"          return(TIMES);
"/"          return(DIVIDE);

"^"          return(POWER);

"("          return(LEFT_PARENTHESIS);
")"          return(RIGHT_PARENTHESIS);

"\n"         return(END);

{ letter }   {
                yyval.str=strdup(yytext);
                return WORD;
            }
```

C.1.3. bash2sql.y

```
%{
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>

#define MAXFIELDS 20

struct myquery{
    char select[80];
    char from[80];
    char where[800];
    char groupby[20];
    char orderby[20];
    char as[20];
    int  felder[MAXFIELDS];

    struct myquery *next;
} queries[10];
typedef struct myquery query;

int crtq=0;
```

```

int reset(struct myquery* abfrage){
    int i;
    for(i=0; i<20;i++){
        abfrage->felder[i]=i+1;
    }
}

int zugrep(struct myquery* abfrage, char* pattern){
    int i=0; char buffer[3];
    strcat(abfrage->where, "(");
    while(abfrage->felder[i]!=-1 && i<MAXFIELDS){
        sprintf(buffer, "%d", abfrage->felder[i++]);
        strcat(abfrage->where, buffer);
        strcat(abfrage->where, "_like_\'\\\'");
        strcat(abfrage->where, pattern);
        strcat(abfrage->where, "\\%\'_or_");
    }

    strcat(abfrage->where, "false");
}

int ausgeben(struct myquery* e)
{
    int i=0; char buffer[3];
    while(e->felder[i]!=-1 && i<MAXFIELDS){
        sprintf(buffer, "%d", e->felder[i++]);
        strcat(e->select, buffer);
    }
    return
    printf("SELECT_%s_FROM_%s_WHERE_%s_GROUP_BY_%s_ORDER_BY_%s_AS_%s\n",
        e->select, e->from, e->where, e->groupby, e->orderby, e->as);
}

int optionZuListe(char* e, int* feld){
    int crtfl=0;
    int a,b,ret;
    do{
        e++;
        ret=sscanf(e, "%d-%d",&a,&b);
        /* felder beginnen bei 0 */
        a--; b--;
        if (ret==0)/*error*/
            return -1;
        /* von - bis */
        else if (ret==2){
            for (;a<=b;a++){
                feld[crtfl++]=feld[a];
            }/*einzelne Auswahl */else{
                feld[crtfl++]=feld[a];
            }
        }

        while ((*e>='0' && *e<='9') || *e=='-')
            e++;
    } while (*e==' ');
    feld[crtfl]=-1;
}

```

```

        return 1;
    }

    %}

%union {
    long int4;           /* Constant integer value */
    int int2;
    float fp;           /* Constant floating point value */
    char *str;          /* Ptr to constant string (strings are malloc'd)
                        */
};

%type <str> WORD
%type <int4> NUMBER
%type <str> OPT

%token  REDIR;
%token  NUMBER ALNUM WORD OPT FSTDIN
%token  PLUS    MINUS    TIMES    DIVIDE    POWER
%token  OPT_D
%token  CAT CUT GREP SORT JOIN
%token  PIPE
%token  LEFT_PARENTHESIS    RIGHT_PARENTHESIS
%token  END

%left  PIPE
%left  PLUS    MINUS
%left  TIMES    DIVIDE
%left  NEG
%right POWER

%start Input
%%

Input:
    /* Empty */
    | Input Line
    ;

Line:
    END
    | Command END
    { /*printf(" fertig\n");*/
      ausgeben(&queries[crtq++]);
      reset(&queries[crtq]);}
    | Command PIPE Line
    { /*printf(" pipe\n");*/ }
    | Command REDIR WORD END
    { strcpy(queries[crtq].as,$3);
      ausgeben(&queries[crtq++]);
      reset(&queries[crtq]);}
    ;

Command:
    CAT
    | CAT WORD
    { reset(&queries[crtq]);
      strcpy(queries[crtq].from,$2);}

```

```

| CUT OptionsCut WORD          { strcpy(queries[crtq].from,$3); }

| CUT OptionsCut
| GREP WORD                    { zugrep(&queries[crtq],$2); }
| GREP WORD WORD                { strcpy(queries[crtq].from,$3);
                                zugrep(&queries[crtq],$2); }

| SORT OptionsSort
;

OptionsSort:
;

OptionsCut:
OptionsCut OptionsCut
| MINUS WORD { char* e;
               for(e=$2; *e!=0; e++){
                 switch(*e){
                   case 'f':
                     optionZuListe(e,queries[crtq].felder);
                     break;
                 }
               }
               /*printf("erkannt :%c:voila\n",*e);*/
             }
;

%%

int yyerror(char *s) {
    printf("%s\n",s);
}

int main() {
    reset(queries);
    yyparse();
}

```

C.2. ANTLR mit C

C.2.1. Makefile

```

OBJ = SimpleBashSQL

all: simplebashsql

simplebashsql: $(OBJ).g
               java org.antlr.Tool $^
               gcc -g -o $@ -I/usr/local/include/ -L/usr/local/lib/ -lantlr3c $(OBJ)
               Parser.c $(OBJ)Lexer.c

```

C.2.2. SimpleBashSQL.g

```

grammar SimpleBashSQL;

options
{

```

```
    language=C;
}

tokens
{
    PIPE = '|';
}

@lexer::header
{
    /* ein Fix wenn Zustand _empty nicht deklariert*/
    #define _empty NULL
}

@header
{
    #ifndef MYHEAD
    #define MYHEAD

    #include <assert.h>
    #include "SimpleBashSQLLexer.h"
    #define MAXFIELDS 5
    #define DELIMIT ','

    typedef struct myquery{
        char select[10], from[30],
            where[200], groupby[5],
            orderby[5], as[20];
        /*numerische Repraesentation der Felder*/
        char felder[MAXFIELDS], groups[MAXFIELDS], orders[MAXFIELDS];
        /*sprechende Namen */
        char *fname[MAXFIELDS];
        /* findet join statt? felder 1, felder2, tabelle1, tabelle2 */
        char join, f1,f2,t2[20];
        /* findet union statt? */
        char sqlunion,sort,group,nrfields;
        struct myquery *src, *src2;
    } query;

    #endif
}

@members
{
    query *lastquery=NULL;
    char tobeprint=1;

    /*-----
    * Methoden fuer die SQL-Abfrage
    *-----*/

    /*str_split fuer die lookup der Spaltenbezeichner*/
    char** str_split(char* a_str, const char a_delim)
    {
        char** result    = 0;
        size_t count     = 0;
        char* tmp        = a_str;
    }
```



```
char* last_comma = 0;
char delim[2];
delim[0] = a_delim;
delim[1] = 0;

/* Count how many elements will be extracted. */
while (*tmp)
{
    if (a_delim == *tmp)
    {
        count++;
        last_comma = tmp;
    }
    tmp++;
}

/* Add space for trailing token. */
count += last_comma < (a_str + strlen(a_str) - 1);

/* Add space for terminating null string so caller
   knows where the list of returned strings ends. */
count++;

result = malloc(sizeof(char*) * count);

if (result)
{
    size_t idx = 0;
    char* token = strtok(a_str, delim);

    while (token)
    {
        assert(idx < count);
        *(result + idx++) = strdup(token);
        token = strtok(0, delim);
    }
    assert(idx == count - 1);
    *(result + idx) = 0;
}

return result;
}

/*nimmt die erste Zeile einer Datei als Spaltenbezeichner
  gibt Feld mit Strings der Namen zurueck*/
int lookup(char* filename, char delimit, query *q)
{
    FILE *f;
    char * line = NULL;
    size_t len = 0;
    ssize_t read;
    char **ptr, **forfree;
    /* beginne mit den neuen Feldern */
    char i=q->nrfields;

    f=fopen(filename, "r");
```

```
    if (!f)
        return -1;
    if ( read = getline(&line , &len , f) == -1 )
        return -1;
    fclose(f);

    ptr=str_split(line , delimiter);
    forfree=ptr;
    while(*ptr && i<MAXFIELDS)
        q->fname[i++]=*ptr++;

    q->nrfields=i;
    free(forfree);
    return 1;
}

/** setzt die Abfrage auf Standardwerte */
int reset(query* abfrage){
    int i;
    if (!abfrage)
        return -1;

    if (abfrage->src!=NULL){
        /*wenn Felder durch SRC festgelegt sind*/
        for (i=0; i<MAXFIELDS;i++){
            abfrage->felder[i]=abfrage->src->felder[i];
            abfrage->orders[i]=abfrage->src->felder[i];
            abfrage->groups[i]=abfrage->src->felder[i];
            if (abfrage->src->felder[i]>=0)
                abfrage->fname[i]=
                    abfrage->src->fname[abfrage->src
                        ->felder[i]];
        }
        abfrage->nrfields=abfrage->src->nrfields;
    } else {
        /*sonst*/
        abfrage->nrfields=0;
        for (i=0; i<MAXFIELDS;i++){
            abfrage->felder[i]=i;
            abfrage->orders[i]=i;
            abfrage->groups[i]=i;
            abfrage->fname[i]=(char*) malloc(4* sizeof(char));
            sprintf(abfrage->fname[i], "%d", i);
        }
    }
    strcpy(abfrage->select, "");
    strcpy(abfrage->from, "");
    strcpy(abfrage->where, "");
    strcpy(abfrage->groupby, "");
    strcpy(abfrage->orderby, "");
    strcpy(abfrage->as, "");
    abfrage->join=0;
    abfrage->f1=1;
    abfrage->f2=1;
    abfrage->sqlunion=0;
    abfrage->sort=0;
```

```

        abfrage->group=0;
    }

    /*gibt den Speicher wieder frei*/
    int deletequeries(query* q)
    {
        if(q->src)
            deletequeries(q->src);
        if(q->src2)
            deletequeries(q->src);
        free(q);
    }

    /** gibt die Abfrage in SQL aus */
    int ausgeben(query* e)
    {
        int i=0;
        if (e==NULL)
            return -1;

        if (strlen(e->as))
            printf("WITH_\\%s_\\AS_\\",e->as);

        /*##### SELECT #####*/
        printf("SELECT_\\%s",e->select);
        /* select relevanten felder, wenn kein * in select */
        if (! strchr(e->select, '*')) {
            while(e->felder[i+1] != -1 && i+1 < MAXFIELDS)
                printf("\\%s, ", e->fname[e->felder[i+1]]);
            printf("\\%s_\\", e->fname[e->felder[i]]);
        }

        /*##### FROM #####*/
        printf("FROM_\\%s_\\", e->from);
        /* bei Join und union: Quelle2 abfragen */
        if (e->src2 != NULL) {
            printf("\\n\\t");
            ausgeben(e->src2);
            printf("_\\");
        }
        /*findet ein join oder eine union statt?*/
        if (e->join)
            printf("AS_a_\\JOIN_\\%s_\\", e->t2);
        if (e->sqlunion)
            printf("UNION_\\");

        /* rekursiv alle Unterabfragen */
        if (e->src != NULL) {
            printf("\\n\\t");
            ausgeben(e->src);
            printf("_\\");
        }
        printf("AS_b");

        /*nur bei join, Joinbedingung*/
        if (e->join)

```

```
        printf("_on_a.$\\%d_=_b.$\\%d",e->f1,e->f2);

/*##### REST, bei Bedarf #####*/
if(strlen(e->where)!=0)
    printf("_WHERE_\\%s",
        e-> where);

if(e->group){
    i=0;
    printf("_GROUP_BY_\\%s",e->groupby);
    while(e->groups[i+1]!=-1 && i+1<MAXFIELDS)
        printf("$\\%d,",e->groups[i++]);
    printf("$\\%d_",e->groups[i]);
}

if(e->sort){
    i=0;
    printf("_ORDER_BY_");
    while(e->orders[i+1]!=-1 && i+1<MAXFIELDS)
        printf("$\\%d,",e->orders[i++]);
    printf("$\\%d_\\%s",e->orders[i],e->orderby);
}

if(strlen(e->as))
    printf(")");

printf("\\n");

return 1;
}

/*
 * Methoden zu den Bash-Kommandos
 */
/* vereint zwei Tabellen */
query *makeunion(char* eingabe, query* q)
{
    query* neu;
    /* erstes Argument? => gleich rein */
    if(strlen(q->from)==0 && q->src==NULL){
        strcpy(q->from, eingabe);
        lookup(eingabe, DELIMIT, q);
        return q;
    }
    /* sonst: mache union */
    neu=(query*) malloc(sizeof(query));
    neu->src=q;
    reset(neu);
    neu->sqlunion=1;
    strcpy(neu->from, eingabe);
    return neu;
}
/* vereint zwei Tabellen, SQL-Anfrage als Eingabe */
query *makeunion_query(query* eingabe, query* q)
{
    query* neu;
```

```

/* erstes Argument? => gleich rein */
if (strlen(q->from)==0 && q->src==NULL) {
    q->src=eingabe;
    return q;
}
/* sonst: mache union*/
neu=(query*) malloc(sizeof(query));
neu->src=q;
reset(neu);
neu->sqlunion=1;
neu->src2=eingabe;
return neu;
}

/** zu grep: parse pattern fuer grep*/
int zugrep(char* pattern, query* abfrage )
{
    int i=0; char buffer[3];
    strcat(abfrage->where, "(");
    while(abfrage->felder[i]!=-1 && i<MAXFIELDS){
        sprintf(buffer, "$%d", abfrage->felder[i++]);
        strcat(abfrage->where, buffer);
        strcat(abfrage->where, "_like_'\%'");
        strcat(abfrage->where, pattern);
        strcat(abfrage->where, "%'\_or_");
    }
    strcat(abfrage->where, "false)");
}

/** parst eine in Optionen angegebene Liste wie d1,2,4-6
    in ein Feld zu 1,2,4,5,6 */
int parseListe(char*e, char* feld)
{
    int crtff=0;
    int a,b,ret;
    do{
        e++;
        ret=sscanf(e, "%d-%d", &a, &b);
        /* felder beginnen bei 0*/
        a--; b--;
        if (ret==0)/*error*/
            return -1;
        /* von - bis */
        else if (ret==2){
            for (;a<=b;a++)
                feld[crtff++]=feld[a];
        }/*einzelne Auswahl */else{
            feld[crtff++]=feld[a];
        }
        while ((*e>='0' && *e<='9') || *e=='-')
            e++;
    } while (*e==' , ');
    feld[crtff]=-1;
    return 1;
}

```

```
/* zu sort */
int optssort(char* e, query* q){
    /*bis Ende erreicht*/
    while(++e!=0){
        switch (*e){
            case 'k':
                parseListe(e,q->orders);
                break;
            case 'u':
                strcat(q->select, "distinct");
                break;
            case 'r':
                strcat(q->orderby, "desc");
                break;
        }
    }
    return 1;
}

/* zu cut */
int optscut(char* e, query* q){
    /*bis Ende erreicht*/
    while(++e!=0){
        switch (*e){
            case 'f':
                parseListe(e,q->felder);
                break;
        }
    }
    return 1;
}

/* zu grep */
int optsgrep(char* e, query* q){
    /*bis Ende erreicht*/
    while(++e!=0){
        switch (*e){
            case 'v':
                strcat(q->where, "NOT_");
                break;
            case 'c':
                strcpy(q->select, "count(*)");
                break;
        }
    }
    return 1;
}

/* zu join */
int optsjoin(char* e, query* q){
    /*bis Ende erreicht*/
    while(++e!=0){
        switch (*e){
            case 'f':
                parseListe(e,q->felder);
                break;
        }
    }
}
```

```

        }
    }
    return 1;
}

/* zu uniq */
int optsuniq(char* e, query* q){
    /*bis Ende erreicht*/
    while(*++e!=0){
        switch (*e){
            case 'c':
                strcat(q->select, "count(*)");
                break;
        }
    }
    return 1;
}

/* zu cut */
int optswc(char* e, query* q){
    /*bis Ende erreicht*/
    while(*++e!=0){
        switch (*e){
            case 'l':
                break;
        }
    }
    return 1;
}

/*-----
 * Main
 *-----*/
int main(int argc, char * argv[])
{
    pANTLR3_INPUT_STREAM      input;
    pSimpleBashSQLLexer        lex;
    pANTLR3_COMMON_TOKEN_STREAM tokens;
    pSimpleBashSQLParser        parser;

    if (argc!=2){
        printf("\%s: _fehlender_Operand\n", argv[0]);
        return 1;
    }

    input = antlr3FileStreamNew ((pANTLR3_UINT8)argv[1],
        ANTLR3_ENC_8BIT);
    lex = SimpleBashSQLLexerNew (input);
    tokens = antlr3CommonTokenStreamSourceNew (ANTLR3_SIZE_HINT,
        TOKENSOURCE(lex));
    parser = SimpleBashSQLParserNew (tokens);

    parser ->file(parser);

    // Must manually clean up
    //

```

```

        parser ->free(parser);
        tokens ->free(tokens);
        lex    ->free(lex);
        input  ->close(input);

        return 0;
    }
}

/*-----
 *  PARSER RULES
 *-----*/

/* BASH-Grammar */
file: newline*;

newline: ('\n')* simple_list ('\n' | ';' | '&')
        | comment { printf("\%s\n", $comment.text->chars); };

comment: '#' (NUMBER)* (ALCHAR)* (WORD)* ('\n');

simple_list
@init{
lastquery=NULL;
}
: ('!')? pipeline_cmd (('&&' | '||') pipeline_cmd)*;

pipeline_cmd
: c1=command
    {
        if(tobeprint && !lastquery && $c1.text->chars)
            printf("\%s", $c1.text->chars);
    }
(PPIPE c2=command
    {
        if(tobeprint && !lastquery)
            printf("_|\_%s", $c2.text->chars);
    }
)*
{
    if(lastquery)
        ausgeben(lastquery);
    printf("\n");
    tobeprint=1;
}

;

command: cmd (to_redir | other_redir)* | shell_cmd (redirection)*;
subshell: '(' compound_list ')';

compound_list: simple_list ( ('\n' | ';' | '&') ('\n')* simple_list)*;

redirection: to_redir | from_redir | other_redir ;

to_redir: ('>' | '>>') WORD {
    if(lastquery)

```



```

        strcpy(lastquery->as,$WORD.text->chars);
    };

from_redir returns[ char* fname]
    : ( '<' | '<<' | '<<<' ) WORD {fname=$WORD.text->chars;}
    | '<' subshell {fname=NULL;};

other_redir:
    NUMBER '>' WORD
  | NUMBER '<' WORD
  | NUMBER '>>' WORD
  | NUMBER '<<' WORD
  | '<&' NUMBER
  | NUMBER '<&' NUMBER
  | '>&' NUMBER
  | NUMBER '>&' NUMBER
  | '<&' WORD
  | NUMBER '<&' WORD
  | '>&' WORD
  | NUMBER '>&' WORD
  | '<<-' WORD
  | NUMBER '<<-' WORD
  | '>&' '-'
  | NUMBER '>&' '-'
  | '<&' '-'
  | NUMBER '<&' '-'
  | '&>' WORD
  | NUMBER '<>' WORD
  | '<>' WORD
  | '>|' WORD
  | NUMBER '>|' WORD
;

shell_cmd:  for_command
    | case_command
    | 'while' compound_list 'do' compound_list 'done'
    | 'until' compound_list 'do' compound_list 'done'
    | if_command
    | subshell
    | group_command
    | function_def;

for_command:  for_header
    { printf("\%s", $for_header.text->chars); }
    compound_list ( 'done' | '}' )
    { printf("done\n"); tobeprint=0; }
;

for_header:  'for' WORD 'in' WORD* ALCHAR* ( '\n' | ';' ) ( 'do' | '{' );

if_command:  'if' compound_list 'then' compound_list ( 'elif' compound_list ) * '
    else' compound_list 'fi';

group_command :  '{' compound_list '}';

function_def :  ( 'function' ) * WORD '(' ')' ( '\n' | ';' ) group_command
;

```

```

case_command: 'case' WORD ('\n' | ';') 'in' ('\n' | ';') case_clause* 'esac';

case_clause: pattern ')' compound_list ';;';

/* Regeln um Kommandos in SQL zu parsen */
cmd returns [query *r]
@init{
    /* neue Query r, vorherige Abfrage als FROM-Tabelle
       NULL, wenn erstes Kommando */
    char tmp=0; //zaehlt args
    r=(query*) malloc(sizeof(query));
    r->src=lastquery;
    reset(r);
}
@after{
    lastquery=r;
}

:
    'cut'      ( OPTS
                { optscut($OPTS.text->chars,r); }
    | WORD
                { r=makeunion($WORD.text->chars,r); }
    | from_redir
                { /* fname!=NULL, wenn Dateiname */
                  if ($from_redir.fname)
                      r=makeunion($from_redir.fname,r);
                  else /* sonst subshell */
                      r=makeunion_query(lastquery,r);
                }
    | '$_' )+

    'cat'      ( OPTS
                { r=makeunion($WORD.text->chars,r); }
    | from_redir
                { /* fname!=NULL, wenn Dateiname */
                  if ($from_redir.fname)
                      r=makeunion($from_redir.fname,r);
                  else /* sonst subshell */
                      r=makeunion_query(lastquery,r);
                }
    | '$_' )*

    'grep'     ( o1=OPTS { optsgrep($o1.text->chars,r); } )*
                p=WORD   { zugrep($p.text->chars,r); }
                ( o2=OPTS { optsgrep($o2.text->chars,r); }
    | f=WORD    { r=makeunion($f.text->chars,r); }
    | from_redir
                { /* fname!=NULL, wenn Dateiname */
                  if ($from_redir.fname)
                      r=makeunion($from_redir.fname,r);
                  else /* sonst subshell */
                      r=makeunion_query(lastquery,r);
                }
    | '$_' )*

```

```

| 'sort'      ( OPTS { optssort($OPTS.text->chars,r); }
| WORD { r=makeunion($WORD.text->chars,r); }
| from_redir
    { /* fname!=NULL, wenn Dateiname */
    if ($from_redir.fname)
        r=makeunion($from_redir.fname,r);
    else /* sonst subshell */
        r=makeunion_query(lastquery,r);
    }
    | '─' )* { r->sort=1; }
| 'join'      (( '-1' n1=NUMBER { r->f1=atoi($n1.text->chars); } )
| ('-2' n2=NUMBER { r->f2=atoi($n2.text->chars); } )
| OPTS
| WORD {
    if(tmp++==0) /* erstes Argument*/
        strcpy(r->from,$WORD.text->chars);
    else
        strcpy(r->t2,$WORD.text->chars);
    }
| '─' {
    if (tmp++==0){ /*erstes Arg? => swap*/
        r->src2=r->src; r->src=NULL;
    }
    }
| from_redir
    { /* fname!=NULL, wenn Dateiname */
    if ($from_redir.fname)
        r=makeunion($from_redir.fname,r);
    else /* sonst subshell */
        r=makeunion_query(lastquery,r);
    }
    )+ { r->join=1; }
| 'uniq'      ( OPTS
    { optsuniq($OPTS.text->chars,r); }
| WORD
    { r=makeunion($WORD.text->chars,r); }
| from_redir
    { /* fname!=NULL, wenn Dateiname */
    if ($from_redir.fname)
        r=makeunion($from_redir.fname,r);
    else /* sonst subshell */
        r=makeunion_query(lastquery,r);
    }
    | '─' )*
    { r->group=1; }
| 'wc_l'      ( OPTS
    { optswc($OPTS.text->chars,r); }
| WORD
    { r=makeunion($WORD.text->chars,r); }
| from_redir
    { /* fname!=NULL, wenn Dateiname */
    if ($from_redir.fname)
        r=makeunion($from_redir.fname,r);
    else /* sonst subshell */
        r=makeunion_query(lastquery,r);
    }
    | '─' )*
    { strcpy(r->select,"count(*)─"); }

```

```

| 'awk' OPTS* '\'' awkbody[r] '\'' OPTS*
| words {
    if (lastquery != NULL)
        ausgeben (lastquery);
    r=NULL;
}
;

words: ALCHAR* WORD*;

assignment: (ALCHAR|WORD) '=' (ALCHAR|WORD);

word
    : '"' ALCHAR* '"'
    | '\'' ALCHAR* '\''
;

awkbody[query* r]: (pattern
                    { strcat (r->where, $pattern.text->chars); }
                    )?
    '{' ( 'print' NUMBER* (',' NUMBER)* ) * WORD* NUMBER*
    '}'
;

pattern: (WORD|NUMBER) ('<' | '>' | '=' ) ('=' )? (WORD|NUMBER)
        (('&&' | '||' )
        (WORD|NUMBER) ('<' | '>' | '=' ) ('=' )? (WORD|NUMBER)
        ) *
;

/*-----
* LEXER RULES
*-----*/

OPTS
    : ( '-' | '+' ) (LETTER|DIGIT| '-' )+
    ;

NUMBER
    : (DIGIT)+
    ;

WORD
    : (LETTER|DIGIT)+;

ALCHAR
    : (LETTER|NUMBER|SONDER)+;

WHITESPACE
    : ( '\t' | '\_ ' | '\r' | '\u000C' | '\\ ' '\n' | '"' )+
    {
        $channel = HIDDEN;
    }
    ;

fragment
DIGIT
    : '0' .. '9' ;

fragment
LETTER
    : ( 'A' .. 'Z' | 'a' .. 'z' | '.' | '_' | ',' | '\' | ' ' | ';' | '$' );

```

```
fragment
SONDER          : ( '{' | '}' | '%' | '\'' | '"' | '-' | '=' | '/' | '!' );
```

C.3. ANTLR mit C++

C.3.1. Makefile

```
OBJ = SimpleBashSQL
CFLAGS=-I/home/maximilian/Downloads/antlr3-master/runtime/Cpp/include
all: simplebashsql

simplebashsql: theparser thequery.o
    g++ -std=c++11 -g -o @$ $(CFLAGS) -lantlr3c $(OBJ) Parser.cpp $(OBJ) Lexer
    .cpp thequery.o
    rm thequery.o

theparser: $(OBJ).g
    java org.antlr.Tool $<

thequery.o: thequery.cpp thequery.hpp
    g++ -std=c++11 -g -c -o @$ $(CFLAGS) thequery.cpp

clean:
    rm -f $(OBJ:=Lexer.cpp) $(OBJ:=Lexer.hpp) $(OBJ:=Parser.cpp) $(OBJ:=
    Parser.hpp) simplebashsql

.PHONY: clean all
```

C.3.2. Query.hpp

```
#include <vector>
#include <string>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#define DELIMIT ' ,'
#define MAXFIELDS 9

class TheQuery;

/* references for a table, the used kinds of tables*/
class Tabref
{
public:
    Tabref(std::string);
    Tabref(TheQuery*);
    void print();

private:
    std::string tablename;
    TheQuery* srcquery;
};

/* container for Expressions*/
class ExprList
{
```

```
public:
    ExprList();
    ~ExprList();
    void add(std::string);
    void add(std::vector<int>);
    void add(std::vector<std::string>);
    void replace_dollars(std::vector<std::string>);

protected:
    std::vector<std::string>felder;
};

class Groupby: public ExprList
{
public:
    void print();
};

class Orderby: public ExprList
{
public:
    Orderby();
    void print();
    void setDesc();
private:
    bool desc;
};

/*where conditions, concatenated using AND*/
class Where
{
public:
    Where();
    ~Where();
    void add(std::string);
    void print();
    void replace_dollars(std::vector<std::string>);
private:
    /* conditions for the where-stm, linked with and */
    std::vector<std::string> felder;
};

/* container for select statements*/
class Select
{
public:
    Select();
    ~Select();
    void print();
    void print(std::string);

    void addSelect(std::vector<int>);
    void addSelect(std::string);
    void deleteSelect(int);
    void setDistinct();
    void replace_dollars(std::vector<std::string>);
};
```

```

        int getColumnCount();
        std::vector<std::string> getColumns();
        std::string getColumn(int);
    private:
        std::vector<std::string> felder;
        std::vector<int> fnr;
        bool distinct;
};

/*a full select stm*/
class Subselect
{
    public:
        Subselect();
        Subselect(std::string);
        Subselect(TheQuery*);
        ~Subselect();

        void print();
        void print(int);
        void addSelect(std::vector<int>);
        void addSelect(std::string);
        void deleteSelect(int);
        void addTable(std::string);
        void addTable(TheQuery*);
        void addCondition(std::string);
        void addGroup(std::vector<int>);
        void addOrder(std::vector<int>);
        void addGroup(std::vector<std::string>);
        void addOrder(std::vector<std::string>);
        void setDesc();
        void setDistinct();

        bool notable();
        void replace_dollars(std::vector<std::string>);

        int getColumnCount();
        std::vector<std::string> getColumns();
        std::string getColumn(int);

    private:
        Select select;
        std::vector<Tabref*> from;
        Where where;
        Groupby groupby;
        Orderby orderby;
};

/*the fullselect stm, also using union (or except...) */
class TheQuery
{
    public:
        TheQuery();
        TheQuery(TheQuery*);
        TheQuery(std::string);

```

```
~TheQuery();
void makeUnion(std::string);
void makeUnion(TheQuery*);
void makeCross(std::string);
void makeCross(TheQuery*);
void makeJoin();
void addSelect(std::vector<int>);
void addSelect(std::string);
void deleteSelect(int);
/*add table should be makeCross (cross join)*/
void addCondition(std::string);
void addGroup(std::vector<int>);
void addOrder(std::vector<int>);
void addGroup(std::vector<std::string>);
void addOrder(std::vector<std::string>);
void setDistinct();
void setDesc();

int getColumnsCount();
std::vector<std::string> getColumns();
std::string getColumn(int);

std::vector<std::string> lookup(std::string, char);
std::vector<std::string> str_split(char*, const char);
void getnamesfromfile(std::string, char);

void print();
private:
/* when union more queries */
std::vector<Subselect*> queries;
std::vector<std::string> fieldnames;
};
```

C.3.3. Query.cpp

```
#include "thequery.hpp"
#include <iostream>
#include <sstream>
#include <string>
using namespace std;

/*replaces each appearance of oldStr with newStr in str*/
void replace_inline(std::string& str, const std::string& oldStr, const std::string& newStr)
{
    size_t pos = 0;
    while((pos = str.find(oldStr, pos)) != std::string::npos)
    {
        str.replace(pos, oldStr.length(), newStr);
        pos += newStr.length();
    }
}

Tabref::Tabref(string name)
{
    tablename=name;
```



```
        srcquery=NULL;
    }

    Tabref::Tabref(TheQuery* mysrc)
    {
        tablename;
        srcquery=mysrc;
    }

    void Tabref::print()
    {
        if(!srcquery)
            std::cout << tablename;

        if(srcquery){
            std::cout << "(";
            srcquery->print();
            std::cout << ")";
        }
    }

    ExprList::ExprList()
    {
    }

    ExprList::~ExprList()
    {
    }

    void ExprList::add(std::string value)
    {
        felder.push_back(value);
    }

    void ExprList::add(std::vector<int> values)
    {
        /*copies values into felder*/
        for (std::vector<int>::iterator it = values.begin();
             it!=values.end(); it++){
            std::stringstream stmp;
            stmp << "$" << *it;
            felder.push_back(stmp.str());
        }
    }

    void ExprList::add(std::vector<string> values)
    {
        /*copies values into felder*/
        for (std::vector<string>::iterator it = values.begin();
             it!=values.end(); it++){
            felder.push_back(*it);
        }
    }

    void ExprList::replace_dollars(std::vector<std::string> fieldnames)
```

```
{
    /*nothing to do?*/
    if(fieldnames.size() <=0)
        return;
    /*for every field, look for $1, $2, ...*/
    for(int i=0; i<felder.size(); i++)
        for(int j=fieldnames.size()-1; j>=0; j--)
            replace_inline(felder[i],
                           "$"+std::to_string(j+1),fieldnames[j]);
}

Orderby::Orderby()
{
    desc=false;
}

void Orderby::setDesc() {
    desc=true;
}

void Orderby::print()
{
    std::vector<std::string>::iterator it = felder.begin();
    /*begin with first, so at the end no comma will appear*/
    if(it!=felder.end())
        std::cout << "_ORDER_BY_" << *it++;
    if(desc)
        std::cout << "DESC_" << *it++;

    while(it != felder.end()){
        std::cout << "," << *it++;
        if(desc)
            std::cout << "DESC_" << *it++;
    }
}

void Groupby::print()
{
    std::vector<std::string>::iterator it = felder.begin();
    /*begin with first, so at the end no comma will appear*/
    if(it!=felder.end())
        std::cout << "_GROUP_BY_" << *it++;
    while(it != felder.end())
        std::cout << "," << *it++;
}

Where::Where()
{
}

Where::~~Where()
{
}

void Where::add(std::string name)
{
}
```

```

        felder.push_back(name);
    }

void Where::replace_dollars(std::vector<std::string> fieldnames)
{
    /*nothing to do?*/
    if(fieldnames.size() <=0)
        return;
    /*for every fields for every name,
    start replacing with the highest to $1, so $10 comes before $1*/
    for(int i=0; i<felder.size(); i++)
        for(int j=fieldnames.size()-1; j>=0; j--)
            replace_inline(felder[i],
                           "$"+std::to_string(j+1),fieldnames[j]);
}

void Where::print()
{
    if(felder.size() >0)
        std::cout << "_WHERE_" << felder[0];
    /* link with AND*/
    for(int i=1; i<felder.size(); i++)
        std::cout << ")_AND_" << felder[i];
    if(felder.size() >0)
        std::cout << "_";
}

Select::Select()
{
    distinct=false;
    felder.clear();
}

Select::~~Select()
{
}

void Select::addSelect(std::vector<int> feldnr)
{
    fnr=feldnr;
    /*copies values into felder*/
    for (std::vector<int>::iterator it = fnr.begin();
         it!=fnr.end(); it++){
        std::stringstream stmp;
        stmp << "$" << *it;
        felder.push_back(stmp.str());
    }
}

void Select::addSelect(std::string name)
{
    felder.push_back(name);
}

/* deletes a certain field, specified by nr, starting with 1 */
void Select::deleteSelect(int nr)

```

```
{
    if (felder.size() >= nr)
        felder.erase(felder.begin() + nr - 1);
}

void Select::setDistinct()
{
    distinct = true;
}

int Select::getColumnCount()
{
    if (felder.size() > 0)
        return felder.size();
    else
        return MAXFIELDS;
}

/** replaces the dollar values in a select
    $1 is for the first column name, so start with $1*/
void Select::replace_dollars(std::vector<std::string> fieldnames)
{
    /*nothing to do?*/
    if (fieldnames.size() <= 0)
        return;
    /*for every fields for every name, start replace with $1*/
    for (int i = 0; i < felder.size(); i++)
        for (int j = fieldnames.size() - 1; j >= 0; j--)
            replace_inline(felder[i],
                           "$" + std::to_string(j + 1), fieldnames[j]);
}

std::vector<std::string> Select::getColumns()
{
    return felder;
}

/** return Column starting by 1 so c in (1,...,n) */
std::string Select::getColumn(int c)
{
    if (felder.size() < c)
        return "*";
    return felder.at(c - 1);
}

void Select::print()
{
    std::cout << "SELECT_";
    if (distinct == true)
        std::cout << "DISTINCT_";
    std::vector<string>::iterator it = felder.begin();
    /*begin print select-stm with first, at the end no comma will appear*/
    if (it != felder.end())
        std::cout << *it++;
    /*if no field selected, select "*" */
}
```

```

        else
            std::cout << "*";

        while(it != felder.end()){
            std::cout << ',' << *it++;
        }
    }
    /*print it, but use the tablename before*/
    void Select::print(std::string name)
    {
        std::cout << "SELECT_";
        if(distinct==true)
            std::cout << "DISTINCT_";
        std::vector<string>::iterator it = felder.begin();
        /*begin print select-stm with first, at the end no comma will appear*/
        if(it!=felder.end())
            std::cout << name << "." <<*it++;
        /*if no field selected, select "*" */
        else
            std::cout << name << ".*";

        while(it != felder.end()){
            std::cout << ',' << name << "." << *it++;
        }
    }

    Subselect::Subselect()
    {
        from.clear();
    }

    Subselect::Subselect(string src)
    {
        from.push_back(new Tabref(src));
    }

    Subselect::Subselect(TheQuery* src)
    {
        from.push_back(new Tabref(src));
    }

    void Subselect::addSelect(std::vector<int> feldnr)
    {
        select.addSelect(feldnr);
    }

    void Subselect::addSelect(std::string name)
    {
        select.addSelect(name);
    }

    void Subselect::deleteSelect(int nr)
    {
        select.deleteSelect(nr);
    }

```

```
}

void Subselect::addTable(std::string name)
{
    from.push_back(new Tabref(name));
}

void Subselect::addTable(TheQuery* src)
{
    from.push_back(new Tabref(src));
}

void Subselect::addCondition(std::string name)
{
    where.add(name);
}

void Subselect::addGroup(std::vector<int> groups)
{
    groupby.add(groups);
}

void Subselect::addOrder(std::vector<int> orders)
{
    orderby.add(orders);
}

void Subselect::addGroup(std::vector<string> groups)
{
    groupby.add(groups);
}

void Subselect::addOrder(std::vector<string> orders)
{
    orderby.add(orders);
}

/*is table empty?*/
bool Subselect::notable()
{
    return from.size()==0;
}

int Subselect::getColumnsCount()
{
    return select.getColumnsCount();
}

void Subselect::setDistinct()
{
    select.setDistinct();
}

void Subselect::setDesc()
{
    orderby.setDesc();
}
```

```

}

void Subselect::replace_dollars(std::vector<std::string> fieldnames)
{
    select.replace_dollars(fieldnames);
    where.replace_dollars(fieldnames);
    groupby.replace_dollars(fieldnames);
    orderby.replace_dollars(fieldnames);
}

std::vector<std::string> Subselect::getColumns()
{
    return select.getColumns();
}

std::string Subselect::getColumn(int c)
{
    return select.getColumn(c);
}

void Subselect::print()
{
    select.print();

    int i;
    std::cout << "FROM ";
    if(0<from.size())
        from[0]->print();
    for(i=1; i<from.size(); i++){
        std::cout << "as_t" << i << ", ";
        from[i]->print();
    }
    std::cout << "as_t" << i << " ";

    where.print();
    groupby.print();
    orderby.print();
}
/** print with an extra identifier*/
void Subselect::print(int uid)
{
    select.print();

    int i;
    std::cout << "FROM ";
    if(0<from.size())
        from[0]->print();
    for(i=1; i<from.size(); i++){
        std::cout << "as_t" << i << "union" << uid << ", ";
        from[i]->print();
    }
    std::cout << "as_t" << i << "union" << uid << " ";

    where.print();
    groupby.print();
    orderby.print();
}

```

```
}
TheQuery::TheQuery()
{
    queries.push_back(new Subselect());
}

TheQuery::TheQuery(TheQuery* src)
{
    queries.push_back(new Subselect());

    if(src!=NULL)
        makeUnion(src);
}

TheQuery::TheQuery(string src)
{
    queries.push_back(new Subselect());
    makeUnion(src);
}

TheQuery::~~TheQuery()
{
}

/*union more queries*/
void TheQuery::makeUnion(string src)
{
    /* table already defined? */
    if(queries[0]->notable())
        queries[0]->addTable(src);
    else
        queries.push_back(new Subselect(src));

    /*add column names*/
    getnamesfromfile(src,DELIMIT);
}

void TheQuery::makeUnion(TheQuery* src)
{
    if(src==NULL)
        return;
    /* table already defined? */
    if(queries[0]->notable())
        queries[0]->addTable(src);
    else
        queries.push_back(new Subselect(src));
    /*get columnnames from file*/
    if(fieldnames.size()==0)
        fieldnames=src->getColumns();
}

/* add one more table*/
void TheQuery::makeCross(string src)
{
    queries[0]->addTable(src);
    /*concatenate the current vector of fields with the one from the table
    column names read with lookup
```



```
        only if already names exist*/
std::vector<std::string> tmp = lookup(src, DELIMIT);
if (fieldnames.size() <= 0)
    fieldnames=tmp;
else
    fieldnames.insert(fieldnames.end(),
                      tmp.begin(), tmp.end());
}

void TheQuery::makeCross(TheQuery* src)
{
    queries[0]->addTable(src);
    /*concatenate the current vector of fields with the one from src
    only if already names exist*/
    std::vector<std::string> tmp=src->getColumns();
    if (fieldnames.size() <= 0)
        fieldnames=src->getColumns();
    else
        fieldnames.insert(fieldnames.end(),
                          tmp.begin(), tmp.end());
}

void TheQuery::makeJoin()
{
}

void TheQuery::addSelect(std::vector<int> feldnr)
{
    for (int i=0; i<queries.size(); i++)
        queries[i]->addSelect(feldnr);
}

void TheQuery::addSelect(std::string name)
{
    for (int i=0; i<queries.size(); i++)
        queries[i]->addSelect(name);
}

void TheQuery::deleteSelect(int nr)
{
    for (int i=0; i<queries.size(); i++)
        queries[i]->deleteSelect(nr);
}

void TheQuery::addCondition(std::string name)
{
    queries[0]->addCondition(name);
}

void TheQuery::addGroup(std::vector<int> groups)
{
    queries[0]->addGroup(groups);
}

void TheQuery::addOrder(std::vector<int> orders)
```

```
{
    queries[0] -> addOrder(orders);
}

void TheQuery::addGroup(std::vector<string> groups)
{
    queries[0] -> addGroup(groups);
}

void TheQuery::addOrder(std::vector<string> orders)
{
    queries[0] -> addOrder(orders);
}

void TheQuery::setDistinct()
{
    queries[0] -> setDistinct();
}

void TheQuery::setDesc()
{
    queries[0] -> setDesc();
}

int TheQuery::getColumnsCount()
{
    if(getColumns().size() <= 0)
        return MAXFIELDS;
    return getColumns().size();
}

/** takes a c-string and a delimiting character, returns the list of strings*/
std::vector<std::string> TheQuery::str_split(char* a_str, const char a_delim)
{
    char delim[2];
    std::vector<std::string> r2;
    delim[0] = a_delim;
    delim[1] = 0;
    /* token is the first string detected */
    char* token = strtok(a_str, delim);
    while (token)
    {
        r2.push_back(std::string(token));
        token = strtok(0, delim);
    }
    return r2;
}

/** reads the first line of a file and takes it as header, columns separated by
    delimiter, returns a vector of column names
    the line should be terminated with the delimiting character */
std::vector<std::string> TheQuery::lookup(std::string filename, char delimiter)
{
    FILE *f;
    char * line = NULL;
    size_t len = 0;
```

```

        ssize_t read;
        char** ptr;
        std::vector<std::string> tmp;
        f=fopen(filename.c_str(),"r");
        if(!f)
            return std::vector<std::string>();
        if( read = getline(&line, &len, f) ==-1)
            return std::vector<std::string>();
        fclose(f);

        tmp=str_split(line,delimiter);
        /* header columns terminated with delimiter symbol */
        tmp.pop_back();
        return tmp;
    }

void TheQuery::getnamesfromfile(std::string filename, char delimiter)
{
    std::vector<std::string> tmp;
    /*only replace the old one, when a newer exists*/
    if((tmp=lookup(filename, delimiter)).size()>0){
        fieldnames=tmp;
    }
}

/* returns the selected columns
   all columns of each subselect should be named identically, queries[0].
   getColumns() == queries[x].getColumns()*/
std::vector<std::string> TheQuery::getColumns()
{
    for(int i; i<queries.size();i++)
        queries[i]->replace_dollars(fieldnames);
    /* if select *, all columns are needed */
    if(queries[0]->getColumns().size()!=0)
        return queries[0]->getColumns();
    else
        return fieldnames;
}

/*returns name of the column c, does not matter if from queries[0] or [1] */
std::string TheQuery::getColumn(int c)
{
    return queries[0]->getColumn(c);
}

void TheQuery::print()
{
    /*only when a query is defined, should always be the case*/
    if(queries.size()>0){
        queries[0]->replace_dollars(fieldnames);
        queries[0]->print();
    }
    /* union all queries */
    for(int i=1; i<queries.size(); i++){
        std::cout << "UNION ";
        queries[i]->replace_dollars(fieldnames);
    }
}

```

```

        /*now all unions must have an unique id*/
        queries[i]->print(i);
    }
}
/*
int main()
{
    std::string s("muh");
    string ss = "exampletable";
    TheQuery* myq = new TheQuery();
    TheQuery* myq2 = new TheQuery();
    std::vector<int> mygroup(1,1);
    myq->makeUnion(ss);
    myq2->makeUnion(ss);
    TheQuery* myq3 = new TheQuery(myq2);
    myq2->addSelect(mygroup);
    myq2->print();
    std::cout << std::endl;
    myq3->print();
    std::cout << std::endl;
    myq2->addGroup(mygroup);
    myq2->addOrder(mygroup);
    myq2->addCondition(" true");
    myq2->makeCross(" table2");
    myq2->print();

    std::cout << endl;
    myq->getColumns();
    std::cout << myq2->getColumn(1) <<endl;
}*/

```

C.3.4. SimpleBashSQL.g

grammar SimpleBashSQL;

options

```

{
    language=Cpp;
}

```

tokens

```

{
    PIPE = '|';
}

```

@lexer::traits {

```

    class SimpleBashSQLLexer;
    class SimpleBashSQLParser;
    typedef antlr3::Traits< SimpleBashSQLLexer, SimpleBashSQLParser >
        SimpleBashSQLLexerTraits;
    typedef SimpleBashSQLLexerTraits SimpleBashSQLParserTraits;
}

```

@header

```

{

```

```

#ifndef MYHEAD
#define MYHEAD
#include "SimpleBashSQLLexer.hpp"
#include <assert.h>
#include <iostream>
#include "thequery.hpp"
using namespace std;

#endif
}

@members
{
    TheQuery *lastquery=NULL;
    char tobeprint=1;
    /*when it should work as a bashscript, a sqltool is needed*/
    string sqltool="";
}

/*
 * Methods for Bash commands
 */

/** grep: parse pattern for grep as a condition for WHERE
    the columns will be linked with '||' */
int zugrep(std::string pattern, TheQuery *abfrage)
{
    stringstream condition;
    condition << "$1";

    for(int i=2; i<=abfrage->getColumnsCount(); i++){
        condition << "||_" << i;
    }
    condition << "_like_" << pattern << "%";
    abfrage->addCondition(condition.str());
}

/** parses a list from bash options like d1,2,4-6
    into a vector like 1,2,4,5,6 returns the vector */
std::vector<int> parseListe(char*e)
{
    std::vector<int> mylist;
    int a,b,ret;
    do{
        e++;
        ret=sscanf(e, "%d-%d", &a, &b);
        /* a is first field, b the last */
        if(ret==0)/*error*/
            return mylist;
        /* all columns from a till b */
        else if(ret==2){
            for(;a<=b;a++){
                mylist.push_back(a);
            }
        } else{
            /*only one selected*/
            mylist.push_back(a);
        }
    }
}

```

```
        while ((*e>='0' && *e<='9') || *e=='-')
            e++;
    } while (*e==' , ');

    return mylist;
}

/* sort */
int optssort(char* e, TheQuery* q){
    /*until last character*/
    while(++e!=0){
        switch (*e){
            case 'k':
                q->addOrder(parseListe(e));
                break;
            case 'u':
                q->setDistinct();
                break;
            case 'r':
                //strcat(q->orderby," desc");
                break;
        }
    }
    return 1;
}

/* cut */
int optscut(char* e, TheQuery* q){
    /*until last character*/
    while(++e!=0){
        switch (*e){
            case 'f':
                q->addSelect(parseListe(e));
                break;
        }
    }
    return 1;
}

/* grep */
int optsgrep(char* e, TheQuery* q){
    /*until last character*/
    while(++e!=0){
        switch (*e){
            case 'v':
                //strcat(q->where, "NOT ");
                break;
            case 'c':
                q->addSelect("count(*)");
                break;
        }
    }
    return 1;
}

/* join */
```

```

int optsjoin(char* e, TheQuery* q){
    /*until last character*/
    /*while(++e!=0){
        switch (*e){
            case 'f':
                parseListe(e,q->felder);
                break;
        }
    }*/
    return 1;
}

/* uniq */
int optsuniq(char* e, TheQuery* q){
    /*until last character*/
    while(++e!=0){
        switch (*e){
            case 'c':
                q->addSelect("count(*)");
                break;
        }
    }
    return 1;
}

/* cut */
int optswc(char* e, TheQuery* q){
    /*until last character*/
    while(++e!=0){
        switch (*e){
            case 'l':
                break;
        }
    }
    return 1;
}

/*replaces each appearance of oldStr with newStr in str*/
void replace_inline_grammar(std::string& str, const std::string& oldStr,
    const std::string& newStr)
{
    size_t pos = 0;
    while((pos = str.find(oldStr, pos)) != std::string::npos)
    {
        str.replace(pos, oldStr.length(), newStr);
        pos += newStr.length();
    }
}

void printhelp()
{
    cout << "_Usage:" << endl;
    cout << "_simplebashsql_scriptname_[sqltool]" << endl;
}

/*
* Main

```

```
*/
int main(int argc, char * argv[])
{
    if(argc<2){
        printf("\%s:_missing_operand\n",argv[0]);
        printhelp();
        return 1;
    }else if(argc>2){
        /*second parameter is the sqltool*/
        sqltool = argv[2];
    }

    ANTLR_UINT8* fName = (ANTLR_UINT8*) argv[1];
    SimpleBashSQLLexer::InputStreamType input(fName, ANTLR_ENC_8BIT)
    ;
    SimpleBashSQLLexer lxr(&input); // TLexerNew is generated by
    ANTLR
    SimpleBashSQLParser::TokenStreamType tstream(ANTLR_SIZE_HINT,
        lxr.get_tokSource() );
    SimpleBashSQLParser psr(&tstream); // TParserNew is generated by
    ANTLR3
    psr.file();
    return 0;
}

}

/*
 * PARSER RULES
 */

/* BASH-Grammar */
file: newline*;

newline: ('\n')* simple_list[false] ('\n' | ';' | '&')
    | comment { cout << $comment.text; };

comment: '#' alchar* ('\n')*
    | '#!/bin/bash' ('\n')*;

/*list of pipeline, connected by && ||, when Subshell print nothing*/
simple_list[bool isSub]
@init{
lastquery=NULL;
}
: ('!')? pipeline_cmd[isSub] (('&&' | '||') pipeline_cmd[isSub])*;

/*list of the real cmds :)
 here at the end, the query will be printed*/
pipeline_cmd[bool isSub]: c1=command
{
    /*only if: the lastquery could not be translated &&
    in bashmode && should be printed */
    if(tobeprint && !lastquery && sqltool != "")
        cout << $c1.text;
}
(PIPE c2=command
```



```

        {
            if(tobeprint && !lastquery && sqltool != "")
                cout << "└┐" << $c2.text;
        }
    )*
    {
        if(!isSub && lastquery)
            lastquery->print();
        printf("\n");
        tobeprint=1;
    }
;

command: cmd (other_redir|to_redir)* | shell_cmd (redirection)*;
subshell: '(' compound_list[true] ')';

compound_list[bool isSubshell]
    : simple_list[isSubshell]
    ( ('\n' | ';' | '&') ('\n')* simple_list[isSubshell])*;

redirection: from_redir | other_redir ;

to_redir: ('>'| '>>') word {
    /* on normal sql mode */
    if(sqltool==""){
        std::cout << "WITH└" << $word.text << "┐AS└(" << endl;
        lastquery->print();
        std::cout << "└)" << endl;
        lastquery=NULL;
        tobeprint=0;
    }else{
        /* on bash mode use sqltool */
        std::cout << sqltool << "└";
        lastquery->print();
        std::cout << "└┐" << $word.text << endl;
        lastquery=NULL;
        tobeprint=0;
    }
    /*TODO hashmap with lastqueries */
};

/*returns the name when inputfilename exists, otherwise subshell*/
from_redir returns[string fname]
    : ('<'| '<<'| '<<<') word
        {
            stringstream s; s << $word.text;
            fname=s.str();
        }
    | '<' subshell
        {
            fname="";
        }
;

other_redir:
    number '>' word

```

```
| number '<' word
| number '>>' word
| number '<<' word
| '<&' number
| number '<&' number
| '>&' number
| number '>&' number
| '<&' word
| number '<&' word
| '>&' word
| number '>&' word
| '<<-' word
| number '<<-' word
| '>&' '-'
| number '>&' '-'
| '<&' '-'
| number '<&' '-'
| '&>' word
| number '<>' word
| '<>' word
| '>|' word
| number '>|' word
;

shell_cmd:      for_command
                | case_command
                | 'while' compound_list[true] 'do' compound_list[true] 'done'
                | 'until' compound_list[true] 'do' compound_list[true] 'done'
                | if_command
                | subshell
                | group_command
                | function_def;

for_command:    for_header
                {cout << $for_header.text << endl;}
                compound_list[true] ('done' | '}')
                {printf("done\n");tobeprint=0;}
;
for_header:     'for' word 'in' word* ('\n'|';') ('do' | '{');

if_command:     'if' compound_list[true] 'then' compound_list[true] ('elif'
    compound_list[true])* 'else' compound_list[true] 'fi';

group_command : '{' compound_list[true] '}';

function_def :  ('function')* word '(' ')' ('\n' | ';') group_command
;
case_command:   'case' word ('\n' | ';') 'in' ('\n' | ';') case_clause* 'esac';

case_clause:    pattern ')' compound_list[true] ';;';

/* Rules for parsing bash command to SQL */
cmd returns [TheQuery *r]
@init{
    /* new TheQuery r,
    afterwards, set old query (lastquery) as table-reference
```

```

    set lastquery as null, so queries from subshell will won't be
    included twice */
    TheQuery* fromPipe=lastquery;
    lastquery=NULL;
    r = new TheQuery();
    stringstream s;
    char buffer[80];
    int helpsize=0, join_on;
}
@after{
    /* query r is the most recent query */
    lastquery=r;
}

:
    'cut'      ( opts
                {
                    s.str(""); s.clear();
                    s << $opts.text; s.getline(buffer,80);
                    optscut(buffer,r);
                }
                | word
                {
                    s.str(""); s.clear(); s << $word.text;
                    r->makeUnion(s.str());
                }
                | from_redir
                { /* fname!=NULL, wenn Dateiname */
                  if (!$from_redir.fname.empty())
                      r->makeUnion($from_redir.fname);
                  else /* sonst subshell */
                      r->makeUnion(lastquery);
                }
                | ' _ '
            )+
            {r->makeUnion(fromPipe);}

    'cat'      ( opts
                | word
                {
                    s.str(""); s.clear(); s << $word.text;
                    r->makeUnion(s.str());
                }
                | from_redir
                { /* fname!=NULL, wenn Dateiname */
                  if (!$from_redir.fname.empty())
                      r->makeUnion($from_redir.fname);
                  else /* sonst subshell */
                      r->makeUnion(lastquery);
                }
                | ' _ '
            )*
            {r->makeUnion(fromPipe);}

    'grep'     ( o1=opts
                {
                    s.str(""); s.clear();

```

```
        s << $o1.text; s.getline(buffer,80);
        optsgrep(buffer,r);
    }
)*
p=word
{
    r->makeUnion(fromPipe);
    s.str(""); s.clear(); s << $p.text;
    zugrep(s.str(),r);
}
( o2=opts
{
    s.str(""); s.clear();
    s << $o2.text; s.getline(buffer,80);
    optsgrep(buffer,r);
}
| f=word
{
    s.str(""); s.clear(); s << $f.text;
    r->makeUnion(s.str());
}
| from_redir
{ /* fname!=NULL, wenn Dateiname */
  if (!$from_redir.fname.empty())
    r->makeUnion($from_redir.fname);
  else /* sonst subshell */
    r->makeUnion(lastquery);
}
| '_u'
)*
{ /* r->makeUnion(lastquery); */ }

| 'sort'
( opts
{
    s.str(""); s.clear();
    s << $opts.text; s.getline(buffer,80);
    optssort(buffer,r);
}
| word
{
    s.str(""); s.clear(); s << $word.text;
    r->makeUnion(s.str());
}
| from_redir
{ /* fname!=NULL, if filename specified */
  if (!$from_redir.fname.empty())
    r->makeUnion($from_redir.fname);
  else /* sonst subshell */
    r->makeUnion(lastquery);
}
)*
{ r->makeUnion(fromPipe); }

| 'head'
( opts /*TODO: should select top... */
| '-1' | '-2'
| word
```

```

        {
            s.str(""); s.clear(); s << $word.text;
            r->makeUnion(s.str());
        }
    | from_redir
      { /* fname!=NULL, wenn Dateiname */
        if (!$from_redir.fname.empty())
            r->makeUnion($from_redir.fname);
        else /* sonst subshell */
            r->makeUnion(lastquery);
      }
    | '─'
  )*
  {r->makeUnion(fromPipe);}

| 'tail'
  ( opts /* like a reverse select top */
  | '─1' | '─2'
  | word
    {
        s.str(""); s.clear(); s << $word.text;
        r->makeUnion(s.str());
    }
  | from_redir
    { /* fname!=NULL, wenn Dateiname */
      if (!$from_redir.fname.empty())
          r->makeUnion($from_redir.fname);
      else /* sonst subshell */
          r->makeUnion(lastquery);
    }
  | '─'
  )*
  {r->makeUnion(fromPipe);}

| 'join' (( '─1' n1=number)
| ( '─2' n2=number)
| OPTS
| word
  {
      s.str(""); s.clear();
      s << $word.text; r->makeCross(s.str());
      /* helpsize to detect first field of 2nd table*/
      helpsize= helpsize==0 ?
          r->getColumnsCount() : helpsize;
  }
  | '─'
  {
      r->makeCross(fromPipe);
      /* helpsize to detect first field of 2nd table*/
      helpsize= helpsize==0 ?
          r->getColumnsCount() : helpsize;
  }
  | from_redir
    { /* fname!=NULL, wenn Dateiname */
      if (!$from_redir.fname.empty())

```

```
        r->makeUnion($from_redir.fname);
    else /* sonst subshell */
        r->makeCross(lastquery);
    /* helpsize to detect first field of 2nd table*/
    helpsize= helpsize==0 ?
        r->getColumnsCount() : helpsize;
    }
)+
{
    s.str(""); s.clear(); s << $n2.text;
    /*get relative column number*/
    join_on=stoi(s.str())+helpsize;
    s.str(""); s.clear();
    s<< "t1.\$" <<$n1.text << "=t2.\$" << join_on;
    r->addCondition(s.str());
    /*print the relevant columns*/
    int max=r->getColumnsCount();
    for(int i=1; i<=max; i++)
        r->addSelect("$"+std::to_string(i));
    r->deleteSelect(join_on);
}

| 'tee'      word /* TODO add opts, to a file */
{
    delete r;
    r=fromPipe;
}

| 'uniq'     ( opts
{ s.str(""); s.clear();
s << $opts.text; s.getline(buffer,80);
optsuniq(buffer,r);
}
| word
{ s.str(""); s.clear(); s << $word.text;
r->makeUnion(s.str());
}
| from_redir
{ /* fname!=NULL, wenn Dateiname */
if (!$from_redir.fname.empty())
    r->makeUnion($from_redir.fname);
else /* sonst subshell */
    r->makeUnion(lastquery);
}
| '─' )*
{
    r->makeUnion(fromPipe);
    /* uniq is realized by groups, so group
f.a. x elem select
begin with $1*/
for (int i=1; i<=fromPipe->getColumnsCount(); i++)
    r->addSelect("$"+std::to_string(i));
r->addGroup(fromPipe->getColumns());
}

| 'wc─l'
```

```

( opts
    {
        s.str(""); s.clear();
        s << $opts.text; s.getline(buffer,80);
        optswc(buffer,r);
    }
| word
    { s.str(""); s.clear(); s << $word.text;
      r->makeUnion(s.str());
    }
| from_redir
    { /* fname!=NULL, wenn Dateiname */
      if (!$from_redir.fname.empty())
          r->makeUnion($from_redir.fname);
      else /* sonst subshell */
          r->makeUnion(lastquery);
    }
| '[_]' )*
{
    r->addSelect("count(*)");
    r->makeUnion(fromPipe);
}

| 'awk' OPTS* '\n' ('\n')*
    (awkbody[r] ('\n')*)+ '\n' (assignment|OPTS)*

/*and data?*/
(word
    {
        s.str(""); s.clear(); s << $word.text;
        r->makeUnion(s.str());
    }
| from_redir
    { /* fname!=NULL, wenn Dateiname */
      if (!$from_redir.fname.empty())
          r->makeUnion($from_redir.fname);
      else /* sonst subshell */
          r->makeUnion(lastquery);
    }
| '[_]'
)*
{
    r->makeUnion(fromPipe);
}

/*default, cmd not implemented */
| word*
    {
        if (fromPipe!=NULL)
            fromPipe->print();
        delete r;
        r=NULL;
    }
;

```

assignment: word '=' word;

```

awkbody[TheQuery* r]
@init{
    stringstream s;
}
: ( a1=awkpattern[r]
{
    r->addCondition($a1.cond);
}

(( '&&' | '||' ) a2=awkpattern[r]
{
    r->addCondition($a2.cond);
}
)*
)?
'{' ('\\n')* (awkcmds[r] ('\\n')*)+ '}'
;

```

```

awkpattern[TheQuery* r] returns [string cond]
@init{
    stringstream s;
}
@after{
    cond=s.str();
    /*contains NR? not so in sql*/
    if(!cond.find("NR"))
        cond="true";
    /*contains NF? not so in sql*/
    if(!cond.find("NF"))
        cond="true";
    /*replaces all appearances of " by ' for sql string*/
    replace_inline_grammar(cond,"\\\"", "\"");
    replace_inline_grammar(cond,"==", "=");
    /* x ~/ pattern / ==> x like 'pattern' */
    replace_inline_grammar(cond,"~/", "like");
    replace_inline_grammar(cond,"/", "\"");
    /*awk regex to sql regex*/
    replace_inline_grammar(cond,".*", "%");
    replace_inline_grammar(cond,".", "?");
}

: awkpatternparsed
{
    s << $awkpatternparsed.text;
}
;

```

```

awkpatternparsed
: (w1=word|n1=number) ('!='| '< '| '> '| '== '| '<=' '| '>=' ) (w2=word|n2=number)
| (word|number) '~/' word '/'
;

```

```

awwkpattern[TheQuery* r] returns [string cond]
@init{
    stringstream s;
}

```



```

}
@after {
    cond=s.str();
    /*contains NR? not so in sql*/
    if(!cond.find("NR"))
        cond="true";
    /*contains NF? not so in sql*/
    if(!cond.find("NF"))
        cond="true";
    /*replaces all appearances of " by ' for sql string*/
    replace_inline_grammar(cond,"\"", "'");
}

:(w1=word|n1=number) ( '==' ) (w2=word|n2=number)
{
    s <<$w1.text<<$n1.text<<"="<<$w2.text<<$n2.text;
}
| (w1=word|n1=number) '<' (w2=word|n2=number)
{
    s <<$w1.text<<$n1.text<<"<"<<$w2.text<<$n2.text;
}
| (w1=word|n1=number) '<=' (w2=word|n2=number)
{
    s <<$w1.text<<$n1.text<<"<="<<$w2.text<<$n2.text;
}
| (w1=word|n1=number) '>' (w2=word|n2=number)
{
    s <<$w1.text<<$n1.text<<">"<<$w2.text<<$n2.text;
}
| (w1=word|n1=number) '>=' (w2=word|n2=number)
{
    s <<$w1.text<<$n1.text<<">="<<$w2.text<<$n2.text;
}
| (w1=word|n1=number) '!=' (w2=word|n2=number)
{
    s <<$w1.text<<$n1.text<<"<"<<$w2.text<<$n2.text;
}
| (w1=word|n1=number) '~/' awkregex '/'
{
    s<<$w1.text<<$n1.text<<"_like_"<<$awkregex.ret<<"'";
}
;

awkcmts[TheQuery* r]
: ( 'print' (awklist[r])
  | assignment
  ) ( ';' ) *
;

awklist[TheQuery* r]
@init{
    stringstream s;
}

: w1=word
{
    s<<$w1.text;
    r->addSelect(s.str());
}

```

```

    }
    ( ' , ' w2=word
    {
        s.str(""); s.clear(); s<<$w1.text;
        r->addSelect(s.str());
    } ) *
;

awkregex returns [string ret]
@init{
    stringstream s;
}
@after{
    ret=s.str();
}

: (word
    {s<<$word.text;}
| ' .* '
    {s<<'\\%';}
| ' . '
    {s<<'?';}
) *
;

pattern:
;

/*-----
 * LEXER RULES
 *-----*/
/* convert each lexer token to an equal parser rule for output text
   text can only be obtained from a parser rule, like e.g. stringstream s;
   s<<$number.text; */
number: NUMBER;
word: WORD| ' ' WORD ' ';
opts: OPTS;
alchar: ALCHAR;

/*Lexer rules */

NUMBER      : (DIGIT)+
;

WORD        : (LETTER|DIGIT) (LETTER|DIGIT| '-' ) *;

OPTS        : ( '-' | '+' ) (LETTER|DIGIT| '-' ) +
;

ALCHAR      : (LETTER|DIGIT|SONDER) +;

WHITESPACE  : ( '\\t' | '\\_ ' | '\\r' | '\\u000C' | '\\ ' ( '\\n' ) + ) +
{
    $channel = HIDDEN;
}
;

```

```
COMMENT      :  '#' ~( '\n' ) *  '\n'
              {
                  $channel = HIDDEN;
              }
              ;

fragment
DIGIT        :  '0' .. '9';

fragment
LETTER       :  (ALPHA|ESCAPED| '.' | '_' | ',' | '$' | '*' );

fragment
ALPHA        :  ( 'A' .. 'Z' | 'a' .. 'z' );

fragment
ESCAPED      :  '\\' ( ';' | ' ' );

fragment
SONDER       :  ( '%' | '\\' | '\'' | '!' );
```


Literaturverzeichnis

- [1] IBM FORTRAN Program Products for OS and the CMS Component of VM/370 General Information. IBM, 1972.
- [2] Stanislaw Adaszewski. *Mynodbcsv: Lightweight Zero-Config Database Solution for Handling Very Large CSV Files*. Plos One, 2014.
- [3] André Eickler Alfons Kemper. *Datenbanksysteme - Eine Einführung*. Oldenburg Verlag, 2011.
- [4] Detlef Borchers. *Heinz Nixdorf Museumsforum zeigt Hollerithmaschine*. Heise Zeitschriften Verlag, 10.05.2007.
- [5] Robert Pike Brian W. Kernighan. *Der Unix-Werkzeugkasten*. Carl Hanser Verlag, 1987.
- [6] Bill Rosenblatt Cameron Newham. *Learning the bash Shell*. O'Reilly Media, Inc., 1998.
- [7] Don Chamberlin. *A Complete Guide to DB2 Universal Database*. Morgan Kaufmann Publishers, inc., 1998.
- [8] Edgar F. Codd. A relational model of data for large shared data banks. 1970.
- [9] Glenn Fowler. *cql - A Flat File Database Query Language*. AT&T Bell Laboratories, 1994.
- [10] Jürgen Gulbins. *UNIX*. Springer-Verlag, 1985.
- [11] Harry Henderson. *A to Z of computer scientists*. Facts On File, 2003.
- [12] David R. Hoffman. *Effective Database Design for Geoscience Professionals*. AT&T Bell Laboratories, 1994.
- [13] Jimi Idle. *Using the ANTLR3 C Target*. Atlassian, 2008.
- [14] Jeroen Janssens. *Data Science at the Command Line*. O'Reilly, 2014.
- [15] Jeremy O. Baum Marketa J. Zvelebil. *Understanding Bioinformatics*. Garland Science, 2008.
- [16] Eike Meinders. *Der Parser-Generator Yacc*. 2004.
- [17] Terrence Parr. *The Definitive ANTLR Reference*. The Pragmatic Bookshelf, 2007.
- [18] Robert Sedgewick. *Computer Science 226: Data Structures and Algorithms*. Princeton University, 2002.
- [19] Y. Shafranovich. *Common Format and MIME Type for Comma-Separated Values (CSV) Files*. Network Working Group, 2005.

- [20] Kim Shanley. *History and Overview of the TPC*. TPC, 02.1998.
- [21] Stackoverflow. *Split string with delimiters in C*. Stackoverflow, 2012.
- [22] Statista. *Prognose zum Volumen der jährlich generierten digitalen Datenmenge weltweit in den Jahren 2005 bis 2020 (in Exabyte)*. Statista, 2014.
- [23] TPC. *About the TPC*. TPC, 2014.
- [24] Transaction Processing Performance Council (TPC). *TPC BENCHMARK H*. Transaction Processing Performance Council (TPC), 2013.