



---

**Übungen zur Vorlesung  
“Database Systems and Modern CPU Architecture”  
Sommersemester 2009**

Jan Rittinger ([jan.rittinger@uni-tuebingen.de](mailto:jan.rittinger@uni-tuebingen.de))

**7. Übungsblatt**

Ausgabe: 17. Juni 2009 · Besprechung: 24. Juni 2009

**Aufgabe 1: MonetDB**

Das MonetDB-System (entwickelt am CWI, Amsterdam) wurde in der Vorlesung bereits mehrfach erwähnt. Die meisten der Konzepte, die wir in der Vorlesung kennen gelernt haben (bzw. kennen lernen werden), finden sich in MonetDB wieder.

**Installieren Sie MonetDB4 auf Ihrem System.**

MonetDB wird als Open Source-Software über <http://www.monetdb.cwi.nl/> vertrieben. Für gängige Rechnerplattformen finden sich auf den „Download“-Seiten Binär- oder Quellpakete.

Für die weiteren Übungen wird MonetDB in der Version 4 benötigt. Windows Benutzer sollten daher **MonetDB4-XQuery installieren**.

Einige Hinweise zur Source Installation mit Hilfe des Super Source Tarball (z. B. `.tar.bz2`):

- Anleitungen zur Installation des Super Source Tarball finden Sie in der Datei README. Mit dem Befehl `./monetdb-install.sh --enable-xquery` installieren Sie die für die Übung benötigte Version 4 von MonetDB.
- Wenn Sie MonetDB auf einer Unix-/Linux-Plattform installieren, empfiehlt es sich, das `readline`-Paket Ihrer Distribution vorher zu installieren (achten Sie ggf. darauf, dass Sie die entsprechende *Entwicklerversion*, z. B. `readline-devel` installieren). Die Verwendung von `readline` erleichtert später die Handhabung der MonetDB-Kommandozeile erheblich.

Wie auch andere Datenbanksysteme ist MonetDB als Client/Server-Anwendung konzipiert. Einen MonetDB-Serverprozess starten Sie mit dem Kommando<sup>1</sup>

---

<sup>1</sup>Geben Sie nur den unterschlingelten Teil ein. Der Rest ist die Begrüßungsmeldung von MonetDB, bzw. symbolisiert Ihren Shell-Prompt \$.

```

$ Mserver
# MonetDB Server v4.31.0
# based on GDK v1.31.0
# Copyright (c) 1993-July 2008, CWI. All rights reserved.
# Copyright (c) August 2008-2009, MonetDB B.V.. All rights reserved.
# Compiled for i386-apple-darwin9.7.0/64bit with 32bit OIDs; dynamically linked.
# Visit http://monetdb.cwi.nl/ for further information.
MonetDB> █

```

Sie befinden sich jetzt in der MonetDB *Server-Konsole*. Sie können hier Kommandos der MonetDB-Algebra MIL („MonetDB Interpreter Language“) absetzen, oder den Server-Prozess mit `quit()`; beenden.

```

MonetDB> print (42);
[ 42 ]
MonetDB> var foo := bat (int, str);
MonetDB> foo.insert (1, "abc");
MonetDB> foo.insert (2, "def");
MonetDB> print (foo);
#-----#
# h      t          # name
# int   str         # type
#-----#
[ 1,     "abc"     ]
[ 2,     "def"     ]
MonetDB> var bar := bat (int, str);
MonetDB> bar.insert (2, "xyz");
MonetDB> print (bar);
#-----#
# h      t          # name
# int   str         # type
#-----#
[ 2,     "xyz"     ]
MonetDB> print (join (reverse (foo), bar));
#-----#
# t      h          # name
# str   str         # type
#-----#
[ "def", "xyz"     ]
MonetDB> quit ();
$ █

```

Um einen MonetDB-Server für Clients zugänglich zu machen, müssen Sie in der Server-Konsole das *Mapi-Protokoll* starten. Das geschieht wie folgt (Sie laden damit das `mapi`-Modul und starten anschließend den „Mapi-Listener“):

```

MonetDB> module (mapi);
MonetDB> mil_start ();

```

Als Client können Sie jetzt in einem weiteren Terminal das Kommando `MapiClient` verwenden. Obwohl `MapiClient` sich ganz ähnlich „anfühlt“ wie die Server-Konsole, werden Sie schnell feststellen, dass `MapiClient` wesentlich komfortabler zu bedienen ist:

```

$ mclient -lmil
mil> █

```

Für Ihre ersten Schritte mit MonetDB werden die folgenden Informationen für Sie hilfreich sein:

- Eine Sprachbeschreibung von MIL finden Sie auf der MonetDB-Webseite unter „Server, Version 4, Documentation, MIL Guide“.
- Über das Kommando `help ()` erhalten Sie eine Online-Hilfe (einschließlich der Signatur) zu allen MIL-Kommandos, z. B.

```
MonetDB> help("reverse");
COMMAND:  reverse(BAT[any::1,any::2]) : BAT[any::2,any::1]
MODULE:   bat
COMPILED: by adm on Fri Jun 12 15:19:42 2009
Returns the reverse view of a BAT (head is tail and tail is head).
BEWARE:  no copying is involved; input and output refer to the same object!
MonetDB> █
```

- Jedes Kommando gehört zu einem bestimmten *Modul*. Eine Liste aller geladenen Module erhalten Sie mit der Funktion `loaded ()`, eine Auflistung der darin enthaltenen Kommandos mit `sigs (modulname);`:

```
mil> loaded ();
#-----#
# module      usage_count    # name
# str         int            # type
#-----#
[ "algebra",  1              ]
[ "arith",    1              ]
...
mil> sigs ("bat");
...
[ "reverse(BAT[any::1,any::2]) : BAT[any::2,any::1]" ]
...
mil> █
```

- Auf der Kurswebseite finden Sie eine Beispieldatei mit einem kleinen MIL-Programm zum Einstieg. Zum Ausführen solcher Programme können Sie den Server oder Client mit der Datei als Argument starten oder während des Betriebs das Kommando `source ()` verwenden.<sup>2</sup>

## Aufgabe 2: Decomposed Storage Model

(Abgabe)

Ein grundlegendes Konzept in MonetDB ist die ausschliessliche Verwendung *zweispaltiger* Tabellen („BATs“, „binary association tables“). Trotz dieser Einschränkung ist das Datenmodell ausreichend, um auch Anfragen für mehrspaltige Tabellen (z. B. SQL) korrekt und effizient zu implementieren.

Für diese Aufgabe wollen wir Daten aus dem TPC-H<sup>3</sup> Benchmark verwenden. Einen Datengenerator dazu finden Sie auf <http://www.tpc.org/tpch/default.asp>, einen fertig generierten Beispiel-Datensatz auf der Kurswebseite. Zweckmäßigerweise wird eine  $n$ -spaltige Tabelle in MonetDB auf  $n$  BATs verteilt, die jeweils eine `void`-Spalte als „Head“ tragen (vgl. Vorlesungsfolien 11 und 12). Ein MIL-Programm, das TPC-H-Daten in MonetDB-BATs lädt finden Sie auf der Kurswebseite.

**Implementieren Sie auf dieser Grundlage die folgenden SQL-Anfragen in MIL:**

<sup>2</sup>Achten Sie dabei darauf, `source ()` mit einem *absoluten* Pfad aufzurufen. Ansonsten wird MonetDB Ihr MIL-Programm nicht finden.

<sup>3</sup>Transaction Processing Council, <http://www.tpc.org/>

1. 

```
SELECT partkey, suppkey, quantity
FROM lineitem
```
2. 

```
SELECT partkey, suppkey, quantity
FROM lineitem
WHERE quantity < 5
ORDER BY quantity
```
3. 

```
SELECT partkey, suppkey, quantity, shipdate
FROM lineitem
WHERE quantity < 5 AND shipdate < '1992-06-01'
```
4. 

```
SELECT l.partkey, p.name, l.quantity
FROM lineitem l, part p
WHERE l.partkey = p.partkey
```
5. 

```
SELECT l.partkey, l.suppkey, l.quantity * ps.supplycost
FROM lineitem l, partsupp ps
WHERE l.partkey = ps.partkey
AND l.suppkey = ps.suppkey
```
6. 

```
SELECT partkey, AVG(quantity)
FROM lineitem
GROUP BY partkey
```
7. 

```
SELECT partkey, suppkey, quantity, shipdate
FROM lineitem
ORDER BY quantity, shipdate
```

**Hinweise:**

- Die notwendigen Datentypen für die Arbeit mit Datumsfeldern stellt das Modul `monetime` zur Verfügung. Das MIL-Programm auf der Kurswebseite bindet `monetime` bereits ein.
- Für eine Implementation der obigen Sortierprädikate könnten Ihnen Routinen aus den Modulen `malalgebra` oder `xtables` hilfreich sein.
- Sollte die Installation von MonetDB bei Ihnen wider Erwarten nicht erfolgreich gewesen sein, so können Sie sich zu den obigen Aufgaben dennoch überlegen, wie eine Implementation mittels nur zweispaltiger Tabellen aussehen könnte.