The Internals of the Monet Database

Bogdan Dumitriu   Lee Provoost

Department of Computer Science
University of Utrecht

June 6, 2005

Outline

1. Databases
   - Classical databases
   - Monet database
2. The MIL Language
   - The language
   - The BAT algebra
   - OQL to MIL translation
3. Running Monet

Databases
Classical databases

Most existing databases:

- OLTP oriented
- high performance on large # of small updates
- table data clustered by row on disk
- unsuited for query-intensive due to
  - a lot of unnecessary I/O

Databases
Classical databases

Most existing databases:

- OLTP oriented
- high performance on large # of small updates
- table data clustered by row on disk
- unsuited for query-intensive due to
  - a lot of unnecessary I/O

Databases
Classical databases

Most existing databases:

- OLTP oriented
- high performance on large # of small updates
- table data clustered by row on disk
- unsuited for query-intensive due to
  - a lot of unnecessary I/O

Databases

Classical databases

Most existing databases:

- OLTP oriented
- high performance on large # of small updates
- table data clustered by row on disk
- unsuited for query-intensive due to
  - a lot of unnecessary I/O

Databases
Vertical fragmentation

Keyword: vertical fragmentation

Id	Name	Postal Code	Date of Birth
1	John	2345 BP	17-09-1976
2	Jane	6146 TY	21-04-1959
3	Bob	8127 PR	04-04-1990

Id	Name
1	John
2	Jane
3	Bob

Id	Postal Code
1	2345 BP
2	6146 TY
3	8127 PR

Id	Date of Birth
1	17-09-1976
2	21-04-1959
3	04-04-1990

Databases

Monet goals

The goals of the Monet database:

1. primary: achieve high performance on query-intensive applications
2. support multiple logical data models
3. providing parallelism
4. extensibility to specific application domains

Databases

Monet goals

The goals of the Monet database:

1. primary: achieve high performance on query-intensive applications
2. support multiple logical data models
3. providing parallelism
4. extensibility to specific application domains

Databases
Monet model

The model employed by Monet:

- Decomposed Storage Model
- kernel of primitives on binary tables
- vertical fragmentation is explicit
- a simple, elegant and very flexible model
- downside: a lot of joining (partially solved)

Databases
Monet model

The model employed by Monet:

- Decomposed Storage Model
- kernel of primitives on binary tables
- vertical fragmentation is explicit
- a simple, elegant and very flexible model
- downside: a lot of joining (partially solved)

Databases

Monet model

The model employed by Monet:

- Decomposed Storage Model
- kernel of primitives on binary tables
- vertical fragmentation is explicit
- a simple, elegant and very flexible model
- downside: a lot of joining (partially solved)

Databases
Monet model

The model employed by Monet:

- Decomposed Storage Model
- kernel of primitives on binary tables
- vertical fragmentation is explicit
- a simple, elegant and very flexible model
- downside: a lot of joining (partially solved)

Databases

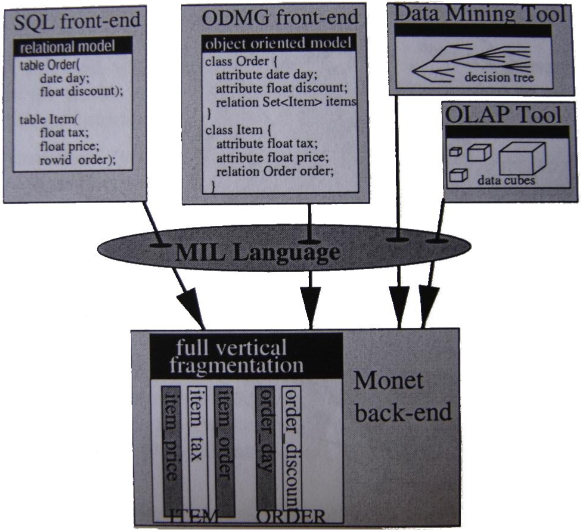
Monet and main memory

Monet is above all a main memory DBMS:

- shifts cost of processing from I/O to CPU cycles
- both its algorithms & its data structures are optimized for main memory access
- not a main memory-only DBMS, though:
  - uses OS-controlled virtual memory during operation
  - uses disk for long-term storage (naturally)

The MIL Language

Monet architecture

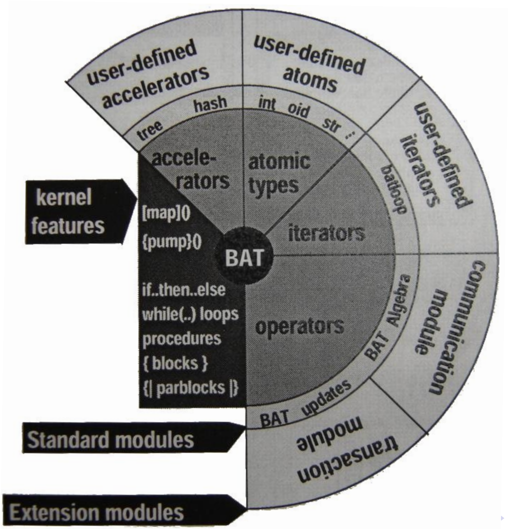


```
table Order(
      date day;
      float discount);

table Item(
      float tax;
      float price;
      rowid  order);
```

```
class Order {
  attribute date day;
  attribute float discount;
  relation Set<Item> items
}
class Item {
  attribute float tax;
  attribute float price;
  relation Order order;
}
```

The MIL Language
The language in a nutshell



The MIL Language

MIL types

1. $t \in \mathcal{A}_f \cup \mathcal{A}_v \Rightarrow t \in \mathcal{T}$

- atomic data types
- fixed: $\mathcal{A}_f = \{\mathtt{bit}, \mathtt{chr}, \mathtt{sht}, \mathtt{int}, \mathtt{lng}, \mathtt{flt}, \mathtt{dbl}, \mathtt{oid}\}$
- variable $\mathcal{A}_v = \{\mathtt{str}\}$

2. $T_1, T_2 \in \mathcal{T} \Rightarrow \mathsf{bat}[T_1, T_2] \in \mathcal{T}$

- the BAT (Binary Table) type
- each tuple in a BAT called a Binary Unit (BUN)
- left column - *head*
- right column - *tail*
- can be nested

The MIL Language
MIL types

1. $t \in \mathcal{A}_f \cup \mathcal{A}_v \Rightarrow t \in \mathcal{T}$

- atomic data types
- fixed: $\mathcal{A}_f = \{\mathtt{bit}, \mathtt{chr}, \mathtt{sht}, \mathtt{int}, \mathtt{lng}, \mathtt{flt}, \mathtt{dbl}, \mathtt{oid}\}$
- variable $\mathcal{A}_v = \{\mathtt{str}\}$

2. $T_1, T_2 \in \mathcal{T} \Rightarrow \mathsf{bat}[T_1, T_2] \in \mathcal{T}$

- the BAT (Binary Table) type
- each tuple in a BAT called a Binary Unit (BUN)
- left column - *head*
- right column - *tail*
- can be nested

The MIL Language

Ingmar's question

Ingmar:

On page 6 it is mentioned that MIL supports nested BATs. That sounds really interesting, but what are they (BATs within BATs, because that doesn't sound like a BAT anymore)?

The MIL Language

MIL features

Main features of the language:

- basic unit of execution: the operator
- operators can be overloaded, most are also polymorphic
- MIL is a dynamically typed language
- a procedural block-structured language (if-then-else, while-do, @)
- allows extension modules
- provides the usual operators ($=$, $\neq$, $<$, $>$, $\leq$, $\geq$, etc.)

The MIL Language
MIL features

Main features of the language:

- basic unit of execution: the operator
- operators can be overloaded, most are also polymorphic
- MIL is a dynamically typed language
- a procedural block-structured language (if-then-else, while-do, @)
- allows extension modules
- provides the usual operators (=, ≠, <, >, ≤, ≥, etc.)

The MIL Language

MIL features

Main features of the language:

- basic unit of execution: the operator
- operators can be overloaded, most are also polymorphic
- MIL is a dynamically typed language
- a procedural block-structured language (if-then-else, while-do, @)
- allows extension modules
- provides the usual operators ($=$, $\neq$, $<$, $>$, $\leq$, $\geq$, etc.)

The MIL Language

MIL features

Main features of the language:

- basic unit of execution: the operator
- operators can be overloaded, most are also polymorphic
- MIL is a dynamically typed language
- a procedural block-structured language (if-then-else, while-do, @)
- allows extension modules
- provides the usual operators ($=$, $\neq$, $<$, $>$, $\leq$, $\geq$, etc.)

The MIL Language

MIL features

Main features of the language:

- basic unit of execution: the operator
- operators can be overloaded, most are also polymorphic
- MIL is a dynamically typed language
- a procedural block-structured language (if-then-else, while-do, @)
- allows extension modules
- provides the usual operators ($=$, $\neq$, $<$, $>$, $\leq$, $\geq$, etc.)

The MIL Language

MIL features

Main features of the language:

- basic unit of execution: the operator
- operators can be overloaded, most are also polymorphic
- MIL is a dynamically typed language
- a procedural block-structured language (if-then-else, while-do, @)
- allows extension modules
- provides the usual operators ($=$, $\neq$, $<$, $>$, $\leq$, $\geq$, etc.)

The MIL Language
The BAT algebra

Core functionality of MIL offered by a BAT algebra of operators which:

- have an algebraic definition
- are free of side-effects
- take BATs and return BATs ⇒ closed algebra on BATs

The MIL Language

The mirror operator

Ingmar:

What's the use of the mirror operator? Why would you want a table with identical columns?

Possible uses:

- perform an set operation on a BAT (or on 2 BATs)
- perform a join operation. . .
- certainly many others

The MIL Language

The mirror operator

Id	Name
4	Jack
2	Bill
1	Bob
6	Clare

Id	Name
1	Daniels
2	Gates
3	Hope
4	Jones
5	Heart
6	James

↓ `mirror`

The MIL Language

The mirror operator

Id	Name
4	Jack
2	Bill
1	Bob
6	Clare

Id	Name
1	Daniels
2	Gates
3	Hope
4	Jones
5	Heart
6	James

↓ `mirror`

The MIL Language

The mirror operator

Id	Id
4	4
2	2
1	1
6	6

Id	Name
1	Daniels
2	Gates
3	Hope
4	Jones
5	Heart
6	James

$\xrightarrow[\textit{join}]{}$

Id	Id
4	Jones
2	Gates
1	Daniels
6	James

The MIL Language

Grouping operators

Id	Name
1	Bob
2	Amy
3	Bob
4	Clare
5	Clare
6	Bob

$\longrightarrow$

`unary group`

Id	GID
1	1
2	2
3	1
4	4
5	4
6	1

Id	Year-of-birth
1	1949
2	1948
3	1950
4	1948
5	1948
6	1950

$\longrightarrow$

`binary group`

Id	GID
1	1
2	2
3	3
4	4
5	4
6	3

The MIL Language

Grouping operators

Id	Name
1	Bob
2	Amy
3	Bob
4	Clare
5	Clare
6	Bob

⟶

`unary group`

Id	GID
1	1
2	2
3	1
4	4
5	4
6	1

Id	Year-of-birth
1	1949
2	1948
3	1950
4	1948
5	1948
6	1950

⟶

`binary group`

Id	GID
1	1
2	2
3	3
4	4
5	4
6	3

The MIL Language
Grouping operators

Id	Name
1	Bob
2	Amy
3	Bob
4	Clare
5	Clare
6	Bob

⟶
`unary group`

Id	GID
1	1
2	2
3	1
4	4
5	4
6	1

Id	Year-of-birth
1	1949
2	1948
3	1950
4	1948
5	1948
6	1950

⟶
`binary group`

Id	GID
1	1
2	2
3	3
4	4
5	4
6	3

The MIL Language
Split operator

Id	Name
1	Bob
2	Amy
3	Joe
4	Clare
5	Susan
6	Jeff

$\xrightarrow[\texttt{split}\ (n = 2)]{}$

Id	Id
1	3
4	6

The MIL Language

Fragment operator

Id	Name
1	Bob
2	Amy
3	Joe
4	Clare
5	Susan
6	Jeff

Id	Id
1	3
4	6

$\underset{\texttt{fragment}}{\longrightarrow}$

Id	BAT	
3	1	nil
	2	nil
	3	nil
6	4	nil
	5	nil
	6	nil

The MIL Language

Multi-join map

Let $vol(x, y, z) = x * y * z$ be the function for computing the volume of a parallelepiped.

Id	W
1	5
2	7
3	9
4	1

Id	H
1	2
2	6
3	1
4	7

Id	L
1	4
2	2
3	2
4	8

$$\xrightarrow[\texttt{[vol]}]{}$$

Id	Vol
1	40
2	98
3	18
4	56

The MIL Language

Pump

GID	Name
2	July
1	Ethan
2	Jill
1	Clare
3	Bob
4	Tim

$\xrightarrow[\texttt{count}]{}$

6

GID	Name
2	July
1	Ethan
2	Jill
1	Clare
3	Bob
4	Tim

GID	
1	
2	
4	
5	

$\xrightarrow[\{\texttt{count}\}]{}$

GID	Count
1	2
2	2
4	1
5	0

The MIL Language

Pump

GID	Name
2	July
1	Ethan
2	Jill
1	Clare
3	Bob
4	Tim

$\underset{\texttt{count}}{\longrightarrow}$

6

GID	Name
2	July
1	Ethan
2	Jill
1	Clare
3	Bob
4	Tim

GID	
1	
2	
4	
5	

$\underset{\{\texttt{count}\}}{\longrightarrow}$

GID	Count
1	2
2	2
4	1
5	0

The MIL Language

Peter's question

. . . which I hope answers Peter's request:

Peter:

Can you show an example of how the pump operator is used, and the results it creates?

Peter:

What is the *str s* in the `save`, `load` and `remove` operators?

The MIL Language

Peter's question

. . . which I hope answers Peter's request:

Peter:

Can you show an example of how the pump operator is used, and the results it creates?

Peter:

What is the *str s* in the `save`, `load` and `remove` operators?

The MIL Language

Peter's question

- A BAT Buffer Pool manages all known BATs
- It administers logical & physical names
- `bbpname (BAT[any,any], str s) : bit` can be used to name a BAT
- This name is global
- The *str s* refers to this logical name

The MIL Language

Laurence's question

Laurence:

Can you show, in a step-by-step fashion how the OQL query on page 109 is translated to MIL?

The MIL Language

OQL to MIL translation

Order class

```
class Order {
  attribute date day;
  attribute float discount;
  relation Set<Item> items;
}
```

Item class

```
class Item {
  attribute float price;
  attribute float tax;
  relation Order order inverse Order.items;
}
```

The MIL Language

OQL to MIL translation

Order class

```
class Order {
  attribute date day;
  attribute float discount;
  relation Set<Item> items;
}
```

Item class

```
class Item {
  attribute float price;
  attribute float tax;
  relation Order order inverse Order.items;
}
```

The MIL Language

OQL to MIL translation

The OQL query

```
SELECT year, sum(total)
FROM ( SELECT price * tax AS total,
              year(item.order.day) AS year
       FROM   item
       WHERE  order.discount BETWEEN 0.00 AND 0.06)
GROUP BY year
ORDER BY year
```

The MIL Language

OQL to MIL translation

The Order class:

order_day

oid	date
100	4/4/98
101	9/4/98
102	1/2/98
103	9/4/98
104	7/2/98
105	1/2/98

order_discount

oid	float
100	0.175
101	0.065
102	0.175
103	0.000
104	0.000
105	0.065

order_item

oid	oid
100	1000
100	1001
101	1002
101	1003
101	1004
102	1005
103	1006
103	1007
103	1008
104	1009
104	1010

The MIL Language

OQL to MIL translation

The Item class:

item_price

oid	float
1000	04.75
1001	11.50
1002	10.20
1003	75.00
1004	02.50
1005	92.80
1006	37.50
1007	14.25
1008	17.99
1009	22.33
1010	42.67

item_tax

oid	float
1000	0.10
1001	0.00
1002	0.00
1003	0.00
1004	0.00
1005	0.10
1006	0.10
1007	0.00
1008	0.00
1009	0.00
1010	0.10

item_order

oid	oid
1000	100
1001	100
1002	101
1003	101
1004	101
1005	102
1006	103
1007	103
1008	103
1009	104
1010	104

The MIL Language

OQL to MIL translation

```
ORD_NIL := select(order_discount, "between", 0.0, 0.6)
```

order_discount

oid	float
100	0.175
101	0.065
102	0.175
103	0.000
104	0.000
105	0.065

⟶

ORD_NIL

oid	oid
100	nil
102	nil
103	nil
104	nil

The MIL Language

OQL to MIL translation

```
ORD_SEL := ORD_NIL.mark(oid(0))
```

ORD_NIL

oid	oid
100	nil
102	nil
103	nil
104	nil

⟶

ORD_SEL

oid	oid
100	0
102	1
103	2
104	3

The MIL Language

OQL to MIL translation

```
SEL_DAY := join(ORD_SEL.reverse, order_day, "=")
```

ORD_SEL.reverse

oid	oid
0	100
1	102
2	103
3	104

order_day

oid	date
100	4/4/98
101	9/4/98
102	1/2/98
103	9/4/98
104	7/2/98
105	1/2/98

⟶

SEL_DAY

oid	date
0	4/4/98
1	1/2/98
2	9/4/98
3	7/2/98

The MIL Language

OQL to MIL translation

```
SEL_YEA := [year](SEL_DAY)
```

SEL_DAY

oid	date
0	4/4/98
1	1/2/98
2	9/4/98
3	7/2/98

⟶

SEL_YEA

oid	int
0	98
1	98
2	98
3	98

The MIL Language

OQL to MIL translation

```
GRP_SEL := group(SEL_YEA).reverse
```

SEL_YEA

oid	int
0	98
1	98
2	98
3	98

→

GRP_SEL

oid	oid
0	0
0	1
0	2
0	3

The MIL Language

OQL to MIL translation

```
GRP_GRP := unique(GRP_SEL.mirror)
```

GRP_SEL.mirror

oid	int
0	0
0	0
0	0
0	0

⟶

GRP_GRP

oid	oid
0	0

The MIL Language

OQL to MIL translation

```
GRP_YEA := join(GRP_GRP, SEL_YEA, "=")
```

GRP_GRP

oid	oid
0	0

SEL_YEA

oid	int
0	98
1	98
2	98
3	98

⟶

GRP_YEA

oid	int
0	98

The MIL Language
OQL to MIL translation

```
ITM_SEL := join(item_order, ORD_SEL, "=")
```

item_order

oid	oid
1000	100
1001	100
1002	101
1003	101
1004	101
1005	102
1006	103
1007	103
1008	103
1009	104
1010	104

ORD_SEL

oid	oid
100	0
102	1
103	2
104	3

→

ITM_SEL

oid	oid
1000	0
1001	0
1005	1
1006	2
1007	2
1008	2
1009	3
1010	3

The MIL Language

OQL to MIL translation

```
UNQ_ITM := ITM_SEL.mark(oid(0)).reverse
```

ITM_SEL

oid	oid
1000	0
1001	0
1005	1
1006	2
1007	2
1008	2
1009	3
1010	3

⟶

UNQ_ITM

oid	oid
0	1000
1	1001
2	1005
3	1006
4	1007
5	1008
6	1009
7	1010

The MIL Language
OQL to MIL translation

```
SEL_UNQ := ITM_SEL.reverse.mark(oid(0))
```

ITM_SEL

oid	oid
1000	0
1001	0
1005	1
1006	2
1007	2
1008	2
1009	3
1010	3

⟶

SEL_UNQ

oid	oid
0	0
0	1
1	2
2	3
2	4
2	5
3	6
3	7

The MIL Language

OQL to MIL translation

```
UNQ_PRI := join(UNQ_ITM, item_price, "=")
```

UNQ_ITM

oid	oid
0	1000
1	1001
2	1005
3	1006
4	1007
5	1008
6	1009
7	1010

item_price

oid	float
1000	04.75
1001	11.50
1002	10.20
1003	75.00
1004	02.50
1005	92.80
1006	37.50
1007	14.25
1008	17.99
1009	22.33
1010	42.67

⟶

UNQ_PRI

oid	float
0	04.75
1	11.50
2	92.80
3	37.50
4	14.25
5	17.99
6	22.33
7	42.67

The MIL Language

OQL to MIL translation

```
UNQ_TAX := join(UNQ_ITM, item_tax, "=")
```

UNQ_ITM

oid	oid
0	1000
1	1001
2	1005
3	1006
4	1007
5	1008
6	1009
7	1010

item_tax

oid	float
1000	0.10
1001	0.00
1002	0.00
1003	0.00
1004	0.00
1005	0.10
1006	0.10
1007	0.00
1008	0.00
1009	0.00
1010	0.10

→

UNQ_TAX

oid	float
0	0.10
1	0.00
2	0.10
3	0.10
4	0.00
5	0.00
6	0.00
7	0.10

The MIL Language

OQL to MIL translation

```
UNQ_TOT := [*](UNQ_PRI, UNQ_TAX)
```

UNQ_PRI

oid	float
0	04.75
1	11.50
2	92.80
3	37.50
4	14.25
5	17.99
6	22.33
7	42.67

UNQ_TAX

oid	float
0	0.10
1	0.00
2	0.10
3	0.10
4	0.00
5	0.00
6	0.00
7	0.10

$\longrightarrow$

UNQ_TOT

oid	float
0	005.225
1	011.500
2	102.080
3	041.250
4	014.250
5	017.990
6	022.330
7	046.937

The MIL Language
OQL to MIL translation

```
GRP_UNQ := join(GRP_SEL, SEL_UNQ, "=")
```

GRP_SEL

oid	oid
0	0
0	1
0	2
0	3

SEL_UNQ

oid	oid
0	0
0	1
1	2
2	3
2	4
2	5
3	6
3	7

⟶

GRP_UNQ

oid	oid
0	0
0	1
0	2
0	3
0	4
0	5
0	6
0	7

The MIL Language

OQL to MIL translation

```
GRP_TOT := join(GRP_UNQ, UNQ_TOT, "=")
```

GRP_UNQ

oid	oid
0	0
0	1
0	2
0	3
0	4
0	5
0	6
0	7

UNQ_TOT

oid	float
0	005.225
1	011.500
2	102.080
3	041.250
4	014.250
5	017.990
6	022.330
7	046.937

→

GRP_TOT

oid	float
0	005.225
0	011.500
0	102.080
0	041.250
0	014.250
0	017.990
0	022.330
0	046.937

The MIL Language
OQL to MIL translation

```
GRP_SUM := {sum}(GRP_TOT, GRP_GRP)
```

GRP_TOT

oid	float
0	005.225
0	011.500
0	102.080
0	041.250
0	014.250
0	017.990
0	022.330
0	046.937

GRP_GRP

oid	oid
0	0

→

GRP_SUM

oid	float
0	261.562

The MIL Language
OQL to MIL translation

```
table("1", GRP_YEA, GRP_SUM)
```

GRP_YEA

oid	int
0	98

GRP_SUM

oid	float
0	261.562

⟶

Result:

int	float
98	261.562

Running Monet

Running the server

Start the Monet server

```
Mserver
# Monet Database Server V4.6.2
# Copyright (c) 1993-2005, CWI. All rights reserved.
# Compiled for <arch>; dynamically linked.
# Visit http://monetdb.cwi.nl/ for further info.
MonetDB>
```

Allowing MIL clients

```
MonetDB>module(mapi);
MonetDB>listen(50000).fork();
```

Shutting down

```
MonetDB>quit();
```

Running Monet

Running the server

Start the Monet server

```
Mserver
# Monet Database Server V4.6.2
# Copyright (c) 1993-2005, CWI. All rights reserved.
# Compiled for <arch>; dynamically linked.
# Visit http://monetdb.cwi.nl/ for further info.
MonetDB>
```

Allowing MIL clients

```
MonetDB>module(mapi);
MonetDB>listen(50000).fork();
```

Shutting down

```
MonetDB>quit();
```

Running Monet

Running the server

Start the Monet server

```
Mserver
# Monet Database Server V4.6.2
# Copyright (c) 1993-2005, CWI. All rights reserved.
# Compiled for <arch>; dynamically linked.
# Visit http://monetdb.cwi.nl/ for further info.
MonetDB>
```

Allowing MIL clients

```
MonetDB>module(mapi);
MonetDB>listen(50000).fork();
```

Shutting down

```
MonetDB>quit();
```

Running Monet

Running the clients

Start the Mapi client

```
MapiClient
# Monet Database Server V4.6.2
# Copyright (c) 1993-2005, CWI. All rights reserved.
# Compiled for <arch>; dynamically linked.
# Visit http://monetdb.cwi.nl/ for further info.
mil>
```

Start Mknife

```
java -jar Mknife-1.6.2-1.jar
(choose MIL demo)
```

Running Monet

Running the clients

Start the Mapi client

```
MapiClient
# Monet Database Server V4.6.2
# Copyright (c) 1993-2005, CWI. All rights reserved.
# Compiled for <arch>; dynamically linked.
# Visit http://monetdb.cwi.nl/ for further info.
mil>
```

Start Mknife

```
java -jar Mknife-1.6.2-1.jar
(choose MIL demo)
```

Running Monet

Using the SQL front-end

Start the SQL front end

```
MonetDB>module(sql_server);
MonetDB>sql_server_start();
```

Use Mapi client

```
MapiClient -l sql
sql>
```

Use JDBC client

```
java -jar share/MonetDB/lib/MonetDB_JDBC.jar
-umonetdb
(use 'monetdb' as password)
```

Running Monet

Using the SQL front-end

Start the SQL front end

```
MonetDB>module(sql_server);
MonetDB>sql_server_start();
```

Use Mapi client

```
MapiClient -l sql
sql>
```

Use JDBC client

```
java -jar share/MonetDB/lib/MonetDB_JDBC.jar
-umonetdb
(use 'monetdb' as password)
```

Running Monet

Using the SQL front-end

Start the SQL front end

```
MonetDB>module(sql_server);
MonetDB>sql_server_start();
```

Use Mapi client

```
MapiClient -l sql
sql>
```

Use JDBC client

```
java -jar share/MonetDB/lib/MonetDB_JDBC.jar
-umonetdb
(use ’monetdb’ as password)
```