Relationen-Algebra und Persistenz – Teil II

Implementierungskonzepte für und Anforderungen an Tupelstromoperatoren in SECONDO

Fabio Valdés

Lehrgebiet Datenbanksysteme für Neue Anwendungen

Fakultät für Mathematik und Informatik

FernUniversität in Hagen

6. Oktober 2017

Inhalt

Inhalt

Grundlegende Datentypen

Tuple

```
            Name: tuple
       Signature: (IDENT x DATA)+ -> TUPLE
Example Type List: (tuple((Name string)(BevT int)))
        List Rep: (<attr1> ... <attrn>)
    Example List: ("Dortmund" 601)
```

Relation

```
            Name: rel
       Signature: TUPLE -> REL
Example Type List: (rel(tuple((Name string)(BevT int))))
        List Rep: (<tuple>*) where
                  <tuple> is (<attr1> ... <attrn>)
    Example List: (("Dortmund" 601)("Hagen" 187))
```

Grundlegende Operationen

```
     Name: feed
Signature: (rel t) -> (stream t)
   Syntax: _ feed
  Meaning: Produces a stream from a relation by
           scanning the relation tuple by tuple.
  Example: query Orte feed count
```

```
     Name: consume
Signature: (stream t) -> (rel t)
   Syntax: _ consume
  Meaning: Collects objects from a stream.
  Example: query Orte feed consume count
```

Grundlegende Operationen

```
     Name: filter
Signature: ((stream t) (map t bool)) -> (stream t)
   Syntax: _ filter [ fun ]
  Meaning: Only tuples fulfilling a certain condition
           are passed on to the output stream.
  Example: query Orte feed filter [.BevT > 500] count
```

```
     Name: attr
Signature: ((tuple ((x1 t1) ... (xn tn))) xi)  -> ti)
   Syntax: attr ( _ , _ )
  Meaning: Returns the value of an attribute at a given position.
  Example: query Orte feed filter
           [fun(t:STREAMELEM) attr(t, BevT) > 500] count
```

Grundlegende Operationen

```
     Name: filter
Signature: ((stream t) (map t bool)) -> (stream t)
   Syntax: _ filter [ fun ]
  Meaning: Only tuples fulfilling a certain condition
           are passed on to the output stream.
  Example: query Orte feed filter [.BevT > 500] count
```

```
     Name: attr
Signature: ((tuple ((x1 t1) ... (xn tn))) xi)  -> ti)
   Syntax: attr ( _ , _ )
  Meaning: Returns the value of an attribute at a given position.
  Example: query Orte feed filter
           [fun(t:STREAMELEM) attr(t, BevT) > 500] count
```

Type Mapping für den Operator `filter`

Anfrage

```
query Orte feed filter [.BevT > 500] consume
```

Type Mapping für den Operator `filter`

Anfrage

```
query Orte feed filter [.BevT > 500] consume
```

Eingabetyp

```
(stream (tuple ((Kennzeichen string) (Ort string)
                (Vorwahl string) (BevT int))))

  (map (tuple ((Kennzeichen string) (Ort string)
               (Vorwahl string) (BevT int))) bool))
```

Type Mapping für den Operator `filter`

Anfrage

```
query Orte feed filter [.BevT > 500] consume
```

Eingabetyp

```
(stream (tuple ((Kennzeichen string) (Ort string)
                (Vorwahl string) (BevT int))))

  (map (tuple ((Kennzeichen string) (Ort string)
               (Vorwahl string) (BevT int))) bool))
```

Type Mapping für den Operator `filter`

Anfrage

```
query Orte feed filter [.BevT > 500] consume
```

Eingabetyp

```
(stream (tuple ((Kennzeichen string) (Ort string)
                (Vorwahl string) (BevT int))))

  (map (tuple ((Kennzeichen string) (Ort string)
               (Vorwahl string) (BevT int))) bool))
```

Ergebnistyp

```
(stream (tuple ((Kennzeichen string) (Ort string)
                (Vorwahl string) (BevT int))))
```

Type Mapping für den Operator `filter`

Anfrage

```
query Orte feed filter [.BevT > 500] consume
```

Eingabetyp

```
(stream (tuple ((Kennzeichen string) (Ort string)
                (Vorwahl string) (BevT int))))

  (map (tuple ((Kennzeichen string) (Ort string)
               (Vorwahl string) (BevT int))) bool))
```

Ergebnistyp

```
(stream (tuple ((Kennzeichen string) (Ort string)
                (Vorwahl string) (BevT int))))
```

Inhalt

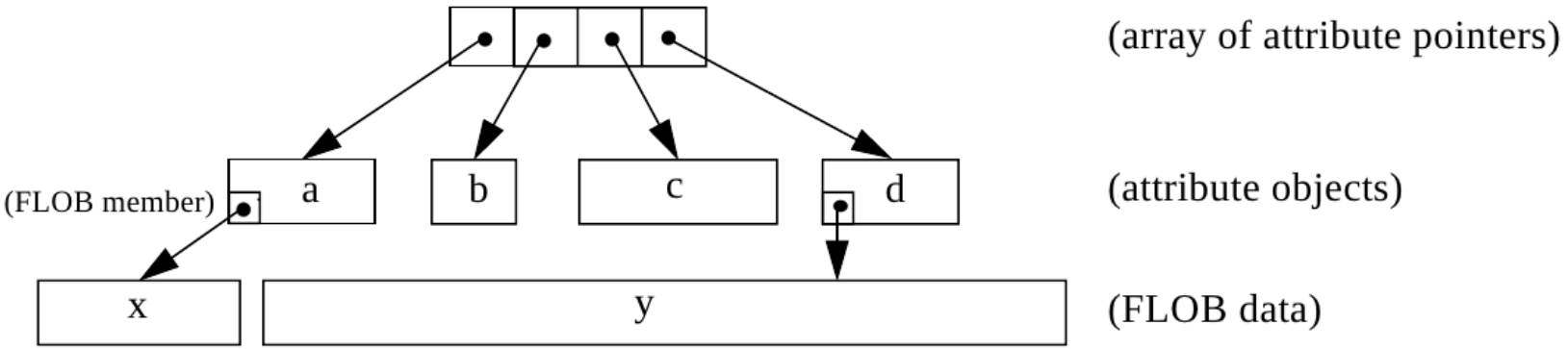
Motivation

- wichtige Operationen auf Tupeln:

`feed`	neue Tupel werden erzeugt
`filter`	Tupel werden unverändert weitergereicht (Selektion)
`consume`	Tupel werden verbraucht / gelöscht
`project`	Tupel werden „verändert“

Interne Darstellung

- Array von Zeigern auf Attributdatentyp-Objekte



Funktionen

Konstruktoren

```
TupleType(const ListExpr typeInfo)

Tuple(TupleType* tupleType)
Tuple(const ListExpr typeInfo)
```

Funktionen

Konstruktoren

```
TupleType(const ListExpr typeInfo)

Tuple(TupleType* tupleType)
Tuple(const ListExpr typeInfo)
```

Zugriff auf Attribute

```
Attribute* GetAttribute(int index)
void PutAttribute(int index, Attribute* attr)
void CopyAttribute(int sourceIndex, Tuple* source, int destIndex)
int GetNoAttributes()
```

Beispiel

Konstruktion eines Tupels

```
ListExpr tupleTypeList = nl->TwoElemList(
  nl->SymbolAtom(Tuple::BasicType()),
  nl->TwoElemList(
        nl->TwoElemList(nl->SymbolAtom("Ort"),
                        nl->SymbolAtom(CcString::BasicType())),
        nl->TwoElemList(nl->SymbolAtom("BevT"),
                        nl->SymbolAtom(CcInt::BasicType()))));
SecondoCatalog* sc = SecondoSystem::GetCatalog();
ListExpr numTupleTypeList = sc->NumericType(tupleTypeList);
TupleType *tupleType = new TupleType(numTupleTypeList);
Tuple *tuple = new Tuple(tupleType);
```

Beispiel

Konstruktion eines Tupels

```
ListExpr tupleTypeList = nl->TwoElemList(
  nl->SymbolAtom(Tuple::BasicType()),
  nl->TwoElemList(
        nl->TwoElemList(nl->SymbolAtom("Ort"),
                        nl->SymbolAtom(CcString::BasicType())),
        nl->TwoElemList(nl->SymbolAtom("BevT"),
                        nl->SymbolAtom(CcInt::BasicType()))));
SecondoCatalog* sc = SecondoSystem::GetCatalog();
ListExpr numTupleTypeList = sc->NumericType(tupleTypeList);
TupleType *tupleType = new TupleType(numTupleTypeList);
Tuple *tuple = new Tuple(tupleType);
```

Festlegung von Attributnamen und -typen

Beispiel

Konstruktion eines Tupels

```
ListExpr tupleTypeList = nl->TwoElemList(
  nl->SymbolAtom(Tuple::BasicType()),
  nl->TwoElemList(
        nl->TwoElemList(nl->SymbolAtom("Ort"),
                        nl->SymbolAtom(CcString::BasicType())),
        nl->TwoElemList(nl->SymbolAtom("BevT"),
                        nl->SymbolAtom(CcInt::BasicType()))));
SecondoCatalog* sc = SecondoSystem::GetCatalog();
ListExpr numTupleTypeList = sc->NumericType(tupleTypeList);
TupleType *tupleType = new TupleType(numTupleTypeList);
Tuple *tuple = new Tuple(tupleType);
```

Umwandlung in numerische Tupeltypliste

Beispiel

Konstruktion eines Tupels

```
ListExpr tupleTypeList = nl->TwoElemList(
  nl->SymbolAtom(Tuple::BasicType()),
  nl->TwoElemList(
        nl->TwoElemList(nl->SymbolAtom("Ort"),
                        nl->SymbolAtom(CcString::BasicType())),
        nl->TwoElemList(nl->SymbolAtom("BevT"),
                        nl->SymbolAtom(CcInt::BasicType()))));
SecondoCatalog* sc = SecondoSystem::GetCatalog();
ListExpr numTupleTypeList = sc->NumericType(tupleTypeList);
TupleType *tupleType = new TupleType(numTupleTypeList);
Tuple *tuple = new Tuple(tupleType);
```

Erzeugung eines neuen Tupeltyps

Beispiel

Konstruktion eines Tupels

```
ListExpr tupleTypeList = nl->TwoElemList(
  nl->SymbolAtom(Tuple::BasicType()),
  nl->TwoElemList(
        nl->TwoElemList(nl->SymbolAtom("Ort"),
                        nl->SymbolAtom(CcString::BasicType())),
        nl->TwoElemList(nl->SymbolAtom("BevT"),
                        nl->SymbolAtom(CcInt::BasicType()))));
SecondoCatalog* sc = SecondoSystem::GetCatalog();
ListExpr numTupleTypeList = sc->NumericType(tupleTypeList);
TupleType *tupleType = new TupleType(numTupleTypeList);
Tuple *tuple = new Tuple(tupleType);
```

Erzeugung eines neuen (leeren) Tupels

Beispiel

Operator `project`

```
     Name: project
Signature: ((stream (tuple ((a1 T1) ... (an Tn)))) (ai1 ... aik))
           -> (stream (tuple ((ai1 Ti1) ... (aik Tik))))
   Syntax: _ project [ list ]
  Meaning: Produces a projection tuple for
           each tuple of its input stream.
  Example: query Orte feed project[Ort, BevT] count
```

Beispiel

Value Mapping des Operators `project`

```
int Project(Word* args, Word& result, int message,
            Word& local, Supplier s) {
  switch (message) {
    case OPEN: {
      ListExpr resultType = GetTupleResultType(s);
      TupleType *tupleType = new TupleType(
                             nl->Second(resultType));
      local.addr = tupleType;
      qp->Open(args[0].addr);
      return 0;
    }
      [...]
  }
}
```

Beispiel

Value Mapping des Operators `project`

```
int Project(Word* args, Word& result, int message,
            Word& local, Supplier s) {
  switch (message) {
    case OPEN: {
      ListExpr resultType = GetTupleResultType(s);
      TupleType *tupleType = new TupleType(
                             nl->Second(resultType));
      local.addr = tupleType;
      qp->Open(args[0].addr);
      return 0;
    }
      [...]
  }
}
```

aus Operatorknoten `s`: Liste der Form

```
(stream ((3 0) ((Ort (1 3)) (BevT (1 0)))))
```

Beispiel

Value Mapping des Operators `project`

```
int Project(Word* args, Word& result, int message,
            Word& local, Supplier s) {
  switch (message) {
    case OPEN: {
      ListExpr resultType = GetTupleResultType(s);
      TupleType *tupleType = new TupleType(
                             nl->Second(resultType));
      local.addr = tupleType;
      qp->Open(args[0].addr);
      return 0;
    }
      [...]
  }
}
```

aus Attributliste `nl->Second(resultType)`: Objekt vom Typ `TupleType`

Beispiel

Value Mapping des Operators `project`

```
case REQUEST: {
    [...]
  TupleType *tupleType = (TupleType*)local.addr;
  Tuple *tuple = new Tuple(tupleType);
  int noAttrs = ((CcInt*)args[2].addr)->GetIntval();
  for (int i = 0; i < noAttrs; i++) {
      [...]
    index = ((CcInt*)elem2.addr)->GetIntval();
    tuple->CopyAttribute(index - 1, (Tuple*)elem1.addr, i);
  }
  ((Tuple*)elem1.addr)->DeleteIfAllowed();
    [...]
}
```

Beispiel

Value Mapping des Operators `project`

```
case REQUEST: {
    [...]
  TupleType *tupleType = (TupleType*)local.addr;
  Tuple *tuple = new Tuple(tupleType);
  int noAttrs = ((CcInt*)args[2].addr)->GetIntval();
  for (int i = 0; i < noAttrs; i++) {
      [...]
    index = ((CcInt*)elem2.addr)->GetIntval();
    tuple->CopyAttribute(index - 1, (Tuple*)elem1.addr, i);
  }
  ((Tuple*)elem1.addr)->DeleteIfAllowed();
    [...]
}
```

kopiert das (`index-1`)-te Attribut des Originaltupels an Position `i` des neuen Tupels `tuple`

Beispiel

Value Mapping des Operators `project`

```
case REQUEST: {
    [...]
  TupleType *tupleType = (TupleType*)local.addr;
  Tuple *tuple = new Tuple(tupleType);
  int noAttrs = ((CcInt*)args[2].addr)->GetIntval();
  for (int i = 0; i < noAttrs; i++) {
      [...]
    index = ((CcInt*)elem2.addr)->GetIntval();
    tuple->CopyAttribute(index - 1, (Tuple*)elem1.addr, i);
  }
  ((Tuple*)elem1.addr)->DeleteIfAllowed();
    [...]
}
```

löscht das `Tuple`-Objekt inklusive aller Attribute (unter Beachtung ggf. noch vorhandener Referenzen)

Beispiel

Value Mapping des Operators `project`

```
case CLOSE: {
  qp->Close(args[0].addr);
  if (local.addr) {
    ((TupleType*)local.addr)->DeleteIfAllowed();
    local.setAddr(0);
  }
  return 0;
}
```

Beispiel

Value Mapping des Operators `project`

```
case CLOSE: {
  qp->Close(args[0].addr);
  if (local.addr) {
    ((TupleType*)local.addr)->DeleteIfAllowed();
    local.setAddr(0);
  }
  return 0;
}
```

löscht das `TupleType`-Objekt (unter Beachtung ggf. noch vorhandener Referenzen)

Referenzzähler

- Ziel: Vermeidung von teuren Kopien, Speicherfehlern, Speicherlöchern
- Attributdatentypen
  - `Clone`: Tiefenkopie mit Referenz
  - `Copy`: weitere Referenz
  - `DeleteIfAllowed`: eine Referenz weniger, ggf. Löschung des Attributobjekts
- Tupel / Tupeltyp
  - `Clone`: Tiefenkopie mit Referenz
  - `IncReference`: Zählerinkrementierung um 1
  - `DeleteIfAllowed` Zählerdekrementierung um 1, ggf. Löschung des Tupels

Inhalt

Motivation

- Gewünscht: Weitergabe (interner) Argumente an Value Mappings zwecks
  - Übergabe von Attribut-Indizes
  - Verwendung von Default-Parametern

Lösung: APPEND

- Schlüsselwort `APPEND` im Type Mapping
- Rückgabe von

  `(APPEND <beliebigeListe> <Ergebnistypliste>)`

  statt wie bisher `<ErgebnistypListe>`

genaue Syntax

```
return nl->ThreeElemList(
                    nl->SymbolAtom(Symbol::APPEND()),
                    <beliebigeListe>,
                    <ErgebnistypListe>);
```

Beispiel

Type Mapping des Operators attr

```
ListExpr AttrTypeMap(ListExpr args) {
  if (nl->ListLength(args) != 2) {
    return listutils::typeError("two arguments expected");
  }
  ListExpr first = nl->First(args);
  if (!Tuple::checkType(first)) {
    return listutils::typeError
           ("first argument must be tuple(...)");
  }
  ListExpr second = nl->Second(args);
  if (!listutils::isSymbol(second)) {
    return listutils::typeError
           ("second argument must be an attribute name");
  }
```

Beispiel

Type Mapping des Operators attr

```
ListExpr AttrTypeMap(ListExpr args) {
  if (nl->ListLength(args) != 2) {
    return listutils::typeError("two arguments expected");
  }
  ListExpr first = nl->First(args);
  if (!Tuple::checkType(first)) {
    return listutils::typeError
           ("first argument must be tuple(...)");
  }
  ListExpr second = nl->Second(args);
  if (!listutils::isSymbol(second)) {
    return listutils::typeError
           ("second argument must be an attribute name");
  }
```

übliche Prüfungen

Beispiel

Type Mapping des Operators attr

```
  string name = nl->SymbolValue(second);
  ListExpr attrtype;
  int j = listutils::findAttribute
                    (nl->Second(first), name, attrtype);
  if (j == 0) {
    return listutils::typeError
          ("Attr name " + name + " not found in attribute list");
  }
  return nl->ThreeElemList(nl->SymbolAtom(Symbol::APPEND()),
                           nl->OneElemList(nl->IntAtom(j)),
                           attrtype);
}
```

Beispiel

Type Mapping des Operators attr

```
  string name = nl->SymbolValue(second);
  ListExpr attrtype;
  int j = listutils::findAttribute
                      (nl->Second(first), name, attrtype);
  if (j == 0) {
    return listutils::typeError
           ("Attr name " + name + " not found in attribute list");
  }
  return nl->ThreeElemList(nl->SymbolAtom(Symbol::APPEND()),
                           nl->OneElemList(nl->IntAtom(j)),
                           attrtype);
}
```

gibt den Datentyp `attrtype` und den Index `j` zum Attributnamen `name` in der Attributliste `nl->Second(first)` zurück

Beispiel

Type Mapping des Operators attr

```
  string name = nl->SymbolValue(second);
  ListExpr attrtype;
  int j = listutils::findAttribute
                      (nl->Second(first), name, attrtype);
  if (j == 0) {
    return listutils::typeError
          ("Attr name " + name + " not found in attribute list");
  }
  return nl->ThreeElemList(nl->SymbolAtom(Symbol::APPEND()),
                           nl->OneElemList(nl->IntAtom(j)),
                           attrtype);
}
```

sendet Fehlerbericht, falls `name` nicht in der Attributliste vorkommt

Beispiel

Type Mapping des Operators attr

```
  string name = nl->SymbolValue(second);
  ListExpr attrtype;
  int j = listutils::findAttribute
                      (nl->Second(first), name, attrtype);
  if (j == 0) {
    return listutils::typeError
          ("Attr name " + name + " not found in attribute list");
  }
  return nl->ThreeElemList(nl->SymbolAtom(Symbol::APPEND()),
                           nl->OneElemList(nl->IntAtom(j)),
                           attrtype);
}
```

gibt die Datentypbeschreibung `attrtype` zurück; angehängt ist eine Liste mit dem Attributindex

Beispiel

Type Mapping des Operators attr

```
  string name = nl->SymbolValue(second);
  ListExpr attrtype;
  int j = listutils::findAttribute
                      (nl->Second(first), name, attrtype);
  if (j == 0) {
    return listutils::typeError
          ("Attr name " + name + " not found in attribute list");
  }
  return nl->ThreeElemList(nl->SymbolAtom(Symbol::APPEND()),
                           nl->OneElemList(nl->IntAtom(j)),
                           attrtype);
}
```

gibt die Datentypbeschreibung `attrtype` zurück; angehängt ist eine Liste mit dem Attributindex

Beispiel

Type Mapping des Operators attr

```
  string name = nl->SymbolValue(second);
  ListExpr attrtype;
  int j = listutils::findAttribute
                      (nl->Second(first), name, attrtype);
  if (j == 0) {
    return listutils::typeError
          ("Attr name " + name + " not found in attribute list");
  }
  return nl->ThreeElemList(nl->SymbolAtom(Symbol::APPEND()),
                           nl->OneElemList(nl->IntAtom(j)),
                           attrtype);
}
```

gibt die Datentypbeschreibung `attrtype` zurück; angehängt ist eine Liste mit dem Attributindex

Beispiel

- Eingabe fürs Type Mapping:

```
((tuple ((Kennzeichen string) (Ort string) (Vorwahl string)
         (BevT int))) Vorwahl)
```

- Rückgabe des Type Mappings:

```
(APPEND (3) string)
```

Value Mapping für Operator `attr`

```
int Attr(Word* args, Word& result,
         int message, Word& local, Supplier s) {
  Tuple* tuple = (Tuple*)args[0].addr;
  int index = ((CcInt*)args[2].addr)->GetIntval();
  assert(1 <= index && index <= tuple->GetNoAttributes());
  result.setAddr(tuple->GetAttribute(index - 1));
  return 0;
}
```

Beispiel

- Eingabe fürs Type Mapping:

```
((tuple ((Kennzeichen string) (Ort string) (Vorwahl string)
         (BevT int))) Vorwahl)
```

- Rückgabe des Type Mappings:

```
(APPEND (3) string)
```

Value Mapping für Operator `attr`

```
int Attr(Word* args, Word& result,
         int message, Word& local, Supplier s) {
  Tuple* tuple = (Tuple*)args[0].addr;
  int index = ((CcInt*)args[2].addr)->GetIntval();
  assert(1 <= index && index <= tuple->GetNoAttributes());
  result.setAddr(tuple->GetAttribute(index - 1));
  return 0;
}
```

Beispiel

- Eingabe fürs Type Mapping:

```
((tuple ((Kennzeichen string) (Ort string) (Vorwahl string)
         (BevT int))) Vorwahl)
```

- Rückgabe des Type Mappings:

```
(APPEND (3) string)
```

Value Mapping für Operator `attr`

```
int Attr(Word* args, Word& result,
         int message, Word& local, Supplier s) {
  Tuple* tuple = (Tuple*)args[0].addr;
  int index = ((CcInt*)args[2].addr)->GetIntval();
  assert(1 <= index && index <= tuple->GetNoAttributes());
  result.setAddr(tuple->GetAttribute(index - 1));
  return 0;
}
```

Beispiel

- `args[1]` (Attributname) wird nicht mehr benötigt
- Type Mapping: `args[1]` $\mapsto$ `args[2]` (Attributindex)

Value Mapping für Operator `attr`

```
int Attr(Word* args, Word& result,
         int message, Word& local, Supplier s) {
  Tuple* tuple = (Tuple*)args[0].addr;
  int index = ((CcInt*)args[2].addr)->GetIntval();
  assert(1 <= index && index <= tuple->GetNoAttributes());
  result.setAddr(tuple->GetAttribute(index - 1));
  return 0;
}
```

Inhalt

Hinweise zur Fehlerbehebung

- Anzeige der Hilfe (in der TTY-Konsole)
  `help`
- Anzeige von Speicherfehlern durch
  `SecondoTTYBDB --valgrind`
  und Speicherlöchern (ungenutzter und nicht wieder freigegebener Speicher) durch
  `SecondoTTYBDB --valgrindlc`
- Fehlerquelle: wiederholtes Löschen von Zeigern, der Speicher könnte bereits anderweitig belegt sein;
  Empfehlung: nicht verwendete Zeiger auf 0 setzen
  - unter Linux: Setzen der Umgebungsvariable
    `MALLOC_CHECK=2` $\Rightarrow$ sofortiger Programmabbruch

MARIO 576400 ×47 WORLD 8-4 TIME 244

THANK YOU MARIO!

YOUR QUEST IS OVER.

WE PRESENT YOU A NEW QUEST.

PUSH BUTTON B

TO SELECT A WORLD