

Relationen-Algebra und Persistenz – Teil I

Implementierungskonzepte für und Anforderungen an Attributdatentypen in SECONDO

Fabio Valdés

Lehrgebiet Datenbanksysteme für Neue Anwendungen

Fakultät für Mathematik und Informatik

FernUniversität in Hagen

9. Oktober 2015

Inhalt

- 1 Implementierung von Attributdatentypen
- 2 FLOBs
- 3 DbArrays
- 4 Beispiel: Klasse Polygon

Inhalt

- 1 Implementierung von Attributdatentypen
- 2 FLOBs
- 3 DbArrays
- 4 Beispiel: Klasse Polygon

Überblick

- Bisher möglich:
 - Verwaltung einfacher Datentypen
 - Erzeugen, Speichern und Löschen von Objekten
 - Anwendung von Operatoren
 - Umwandlung von/in Nested List

Überblick

- Bisher möglich:
 - Verwaltung einfacher Datentypen
 - Erzeugen, Speichern und Löschen von Objekten
 - Anwendung von Operatoren
 - Umwandlung von/in Nested List
- Gewünscht:
 - Verwendung als Attribut in Relationen

Anforderungen

- ableiten von der Klasse Attribute (direkt oder indirekt)

zu implementierende Funktionen

```
Klassenname ();  
static ListExpr Property();  
static const string BasicType();  
static bool CheckKind(ListExpr type, ListExpr& errorInfo);  
static bool checkType(ListExpr t);  
int NumOfFLOBs() const;  
Flob* GetFLOB(const int i);  
size_t Sizeof() const;  
int Compare(const Attribute *arg) const;  
Attribute* Clone() const;  
bool Adjacent(const Attribute *arg) const;  
size_t HashValue() const;  
void CopyFrom(const Attribute *arg);  
ListExpr ToListExpr(ListExpr typeInfo);  
bool ReadFrom(ListExpr LE, ListExpr typeInfo);
```

generischer Typkonstruktor

- Datei `GenericTC.h` inkludieren
- Verwendung:
`GenTC<Klassenname> Typkonstruktorname;`
- z.B.:
`GenTC<Disc> disc;`
- Vorteil: Vollständigkeit der Funktionen wird geprüft

Beispiel: Klasse Disc

Funktion Property

```
static ListExpr Disc::Property() {  
    return gentc::GenProperty("-> DATA",  
                               BasicType(),  
                               "(x y radius)",  
                               "(8.4 16.7 27)");  
}
```

Funktion BasicType

```
static string Disc::BasicType() {  
    return "disc";  
}
```


Beispiel: Klasse Disc

Funktion checkType

```
static const bool Disc::checkType(const ListExpr type) {  
    return listutils::isSymbol(type, BasicType());  
}
```

Funktion Sizeof

```
size_t Disc::Sizeof() const {  
    return sizeof(*this);  
}
```

Beispiel: Klasse Disc

Funktion ToListExpr

```
ListExpr Disc::ToListExpr(ListExpr typeInfo) {  
    if (!IsDefined()) {  
        return listutils::getUndefined();  
    }  
    return nl->ThreeElemList (nl->RealAtom(x),  
                             nl->RealAtom(y),  
                             nl->RealAtom(radius));  
}
```

Funktion ReadFrom

```
bool Disc::ReadFrom(ListExpr value, ListExpr typeInfo) {  
    if (listutils::isSymbolUndefined(value)) {  
        SetDefined(false);  
        return true;  
    }  
    if (!nl->HasLength(value, 3)) {  
        return false;  
    }  
    [...]  
    if (!listutils::isNumeric(X) || !listutils::isNumeric(Y) ||  
        !listutils::isNumeric(R)) {  
        return false;  
    }  
    [...]  
    if (r < 0) {  
        return false;  
    }  
    set(x, y, r);  
    return true;  
}
```

Inhalt

- 1 Implementierung von Attributdatentypen
- 2 FLOBs
- 3 DbArrays
- 4 Beispiel: Klasse Polygon

Motivation

- bisher nur Attribute mit fester Größe
 - primitive Datentypen (bool, int, real, string)
 - einfache geometrische Objekte (z.B. point, rect)
 - Zeitpunkte, -intervalle (instant, interval)
 - beliebige Kombinationen solcher Datentypen
- keine Zeiger oder Strukturen veränderlicher Größe

Motivation

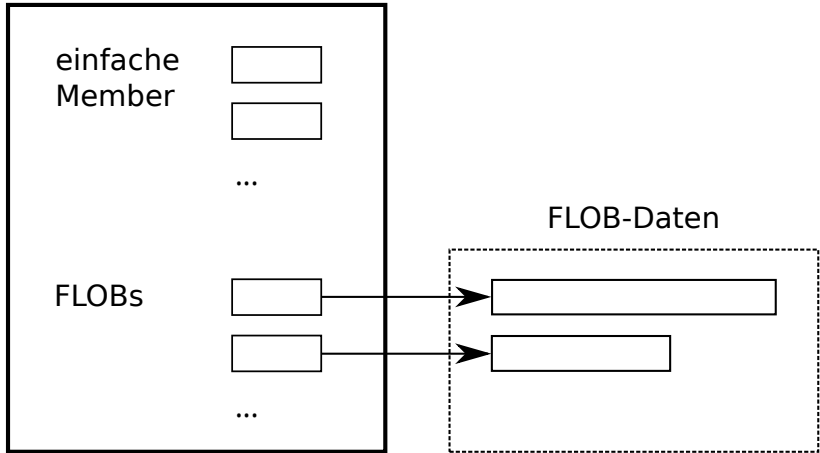
- bisher nur Attribute mit fester Größe
 - primitive Datentypen (bool, int, real, string)
 - einfache geometrische Objekte (z.B. point, rect)
 - Zeitpunkte, -intervalle (instant, interval)
 - beliebige Kombinationen solcher Datentypen
- keine Zeiger oder Strukturen veränderlicher Größe
- Ziel: Implementierung folgender (und weiterer) Datentypen
 - Texte beliebiger/variabler Länge (text)
 - komplexere geometrische Objekte (z.B. line, region, points)
 - zeitabhängige Datentypen (z.B. mpoint, mstring)

Eigenschaften

- FLOB = Faked Large Object
- speichert beliebige Datenmengen
- eingebauter Persistenzmechanismus
- unstrukturierte Speicherblöcke
- Zugriff mit Offset und Größe

Klasse mit FLOB-Attributen

root record



Funktionen

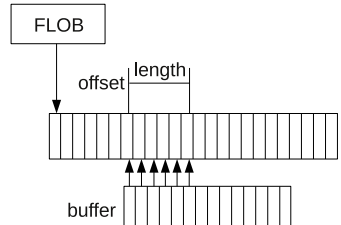
Klasse Flob

```
class Flob {  
    Flob(const SmiSize size);  
    SmiSize getSize() const;  
    bool write(const char* buffer,  
               const SmiSize length,  
               const SmiSize offset);  
    bool read(char* buffer,  
              const SmiSize length,  
              const SmiSize offset);  
    bool resize(const SmiSize newSize);  
    bool clean();  
    bool destroy();  
    [...]  
};
```

Funktionen

Klasse Flob

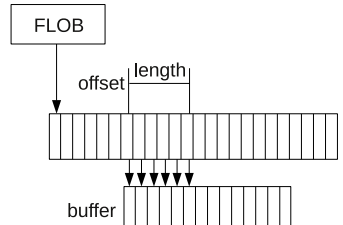
```
class Flob {  
    Flob(const SmiSize size);  
    SmiSize getSize() const;  
    bool write(const char* buffer,  
               const SmiSize length,  
               const SmiSize offset);  
    bool read(char* buffer,  
              const SmiSize length,  
              const SmiSize offset);  
    bool resize(const SmiSize newSize);  
    bool clean();  
    bool destroy();  
    [...]  
};
```



Funktionen

Klasse Flob

```
class Flob {  
    Flob(const SmiSize size);  
    SmiSize getSize() const;  
    bool write(const char* buffer,  
               const SmiSize length,  
               const SmiSize offset);  
    bool read(char* buffer,  
              const SmiSize length,  
              const SmiSize offset);  
    bool resize(const SmiSize newSize);  
    bool clean();  
    bool destroy();  
    [...]  
};
```



Eigenschaften

- Einschränkung: Objekt im Hauptspeicher darstellbar (für einige Operationen wie z.B. Sortieren)
- zusammenhängende Speicherung auf der Festplatte
- muss initialisiert werden, z.B.
`text(9);`
- Nachteil: keine komfortable Speicherung mehrerer gleichartiger Objekte

Inhalt

- 1 Implementierung von Attributdatentypen
- 2 FLOBs
- 3 DbArrays**
- 4 Beispiel: Klasse Polygon

Motivation

- dynamisches Array
- beliebig viele Einträge eines bestimmten Typs
- persistente Speicherung strukturierter Daten

Eigenschaften

- Template-Klasse `DbArray` abgeleitet von `Flob`
- Persistenzmechanismus
- Bedingungen an Array-Elemente:
 - keine Zeiger
 - keine FLOBs (also auch keine DbArrays)
- binäre Suche möglich bei sortierter Speicherung

Funktionen

Klasse DbArray

```
template<class DbArrayElement> class DbArray : public Flob {  
    DbArray(const int n);  
    int Size() const;  
    bool resize(const int newSize);  
    bool Get(const int index, DbArrayElement& elem) const;  
    bool Append(const DbArrayElement& elem);  
    bool Put(const int index, const DbArrayElement& elem);  
    bool Sort(int (*cmp)(const void *a, const void *b));  
    bool Find(const void *key,  
              int (*cmp)(const void *a, const void *b),  
              int& result) const;  
    bool TrimToSize();  
    bool clean();  
    bool Destroy();  
}
```


Funktionen

Klasse DbArray

```
template<class DbArrayElement> class DbArray : public Flob {  
    DbArray(const int n);  
    int Size() const;  
    bool resize(const int newSize);  
    bool Get(const int index, DbArrayElement& elem) const;  
    bool Append(const DbArrayElement& elem);  
    bool Put(const int index, const DbArrayElement& elem);  
    bool Sort(int (*cmp)(const void *a, const void *b));  
    bool Find(const void *key,  
              int (*cmp)(const void *a, const void *b),  
              int& result) const;  
    bool TrimToSize();  
    bool clean();  
    bool Destroy();  
}
```

Inhalt

- 1 Implementierung von Attributdatentypen
- 2 FLOBs
- 3 DbArrays
- 4 Beispiel: Klasse Polygon**

Beispiel

Klasse Polygon

```
class Polygon : public Attribute {  
  public:  
    Polygon() {}  
    Polygon(const int n,  
            const int *X = 0,  
            const int *Y = 0);  
    ~Polygon();  
  
    int NumOfFLOBs() const;  
    Flob *GetFLOB(const int i);  
    [...]  
  
  private:  
    DbArray<Vertex> vertices;  
    PolygonState state;  
};
```

Beispiel

Klasse Polygon

```
class Polygon : public Attribute {  
  public:  
  Polygon() {}  
  Polygon(const int n,  
          const int *X = 0,  
          const int *Y = 0);  
  ~Polygon();  
  
  int NumOfFLOBs() const;  
  Flob *GetFLOB(const int i);  
  [...]  
  
  private:  
  DbArray<Vertex> vertices;  
  PolygonState state;  
};
```

- abgeleitet von Attribute
- nur von einer Klasse erben lassen, sonst Probleme bei Typkonvertierung (cast)

Beispiel

Klasse Polygon

```
class Polygon : public Attribute {  
  public:  
    Polygon() {}  
    Polygon(const int n,  
            const int *X = 0,  
            const int *Y = 0);  
    ~Polygon();  
  
    int NumOfFLOBs() const;  
    Flob *GetFLOB(const int i);  
    [...]  
  
  private:  
    DbArray<Vertex> vertices;  
    PolygonState state;  
};
```

- einziges FLOB-Attribut der Klasse
- Datentyp `Vertex` hat konstante Größe (zwei Koordinaten)

Beispiel

Klasse Polygon

```
class Polygon : public Attribute {  
  public:  
    Polygon() {}  
    Polygon(const int n,  
            const int *X = 0,  
            const int *Y = 0);  
    ~Polygon();  
  
    int NumOfFLOBs() const;  
    Flob *GetFLOB(const int i);  
    [...]  
  
  private:  
    DbArray<Vertex> vertices;  
    PolygonState state;  
};
```

- Persistenzmechanismus verwendet
Standard-Konstruktor
- Fehler durch uninitialisierte Werte
- andere Konstruktoren verwenden

Beispiel

Klasse Polygon

```
class Polygon : public Attribute {
public:
    Polygon() {}
    Polygon(const int n,
            const int *X = 0,
            const int *Y = 0);

    ~Polygon();

    int NumOfFLOBs() const;
    Flob *GetFLOB(const int i);
    [...]

private:
    DbArray<Vertex> vertices;
    PolygonState state;
};
```

- Konstruktor erhält Größe und Koordinaten-Arrays
- FLOBs müssen initialisiert werden:

```
vertices(n);
```

- Anhängen der Koordinatenpaare:

```
for (int i=0; i<n; i++) {
    Vertex v(X[i], Y[i]);
    vertices.Append(v);
}
```

Beispiel

Klasse Polygon

```
class Polygon : public Attribute {
public:
    Polygon() {}
    Polygon(const int n,
            const int *X = 0,
            const int *Y = 0);
    ~Polygon();

    int NumOfFLOBs() const;
    Flob *GetFLOB(const int i);
    [...]

private:
    DbArray<Vertex> vertices;
    PolygonState state;
};
```

- gibt die Anzahl der FLOB-Member zurück

```
return 1;
```


Beispiel

Klasse Polygon

```
class Polygon : public Attribute {
public:
    Polygon() {}
    Polygon(const int n,
            const int *X = 0,
            const int *Y = 0);
    ~Polygon();

    int NumOfFLOBs() const;
    Flob *GetFLOB(const int i);
    [...]

private:
    DbArray<Vertex> vertices;
    PolygonState state;
};
```

- prüft Gültigkeit des Parameters
- gibt die Adresse des i -ten FLOBs zurück:

```
if (i == 0) {
    return &vertices;
}
return 0;
```

MARIO
066500

🍄×33

WORLD
1-4

TIME
224

THANK YOU MARIO!

BUT OUR PRINCESS IS IN
ANOTHER CASTLE!

