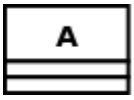
Implémentation en Java

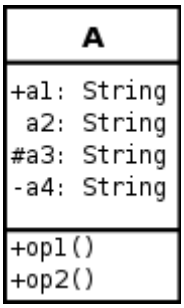
Classe

Parfois, la génération automatique de code produit, pour chaque classe, un constructeur et une méthode finalize comme ci-dessous. Rappelons que cette méthode est invoquée par le *ramasse miettes* lorsque celui-ci constate que l'objet n'est plus référencé. Pour des raisons de simplification, nous ne ferons plus figurer ces opérations dans les sections suivantes.




```
public class A {
 public A() {
  ...
 }
 protected void finalize() throws Throwable {
  super.finalize();
  ...
 }
}
```

Classe avec attributs et opérations



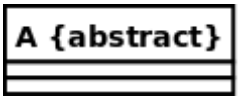

```
public class A {
 public    String a1;
 package   String a2;
 protected String a3;
 private   String a4;
 public void op1() {
  ...
 }
 public void op2() {
```

...

}

}

Classe abstraite

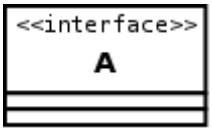


public abstract class A {

...

}

Interface

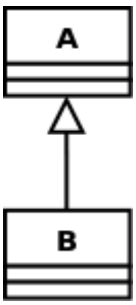


public interface A {

...

}

Héritage simple



public class A {

...

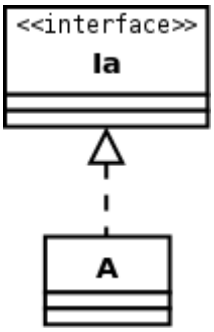
}

public class B extends A {

...

}

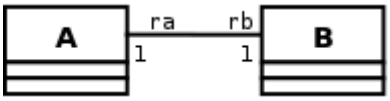
Réalisation d'une interface par une classe




```
public interface Ia {
  ...
}


public class A implements Ia {
  ...
}
```

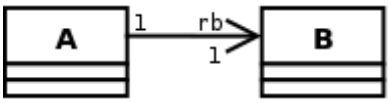
Association bidirectionnelle 1 vers 1




```
public class A {
  private B rb;
  public void addB( B b ) {
    if( b != null ){
      if ( b.getA() != null ) {   // si b est déjà connecté à un autre A
        b.getA().setB(null);    // cet autre A doit se déconnecter
      }
      this.setB( b );
      b.setA( this );
    }
  }
  public B getB() { return( rb ); }
  public void setB( B b ) { this.rb=b; }
}
```

```
public class B {
  private A ra;
  public void addA( A a ) {
    if( a != null ) {
      if (a.getB() != null) {   // si a est déjà connecté à un autre B
        a.getB().setA( null );  // cet autre B doit se déconnecter
      }
      this.setA( a );
      a.setB( this );
    }
  }
  public void setA(A a){ this.ra=a; }
  public A getA(){ return(ra); }
}
```

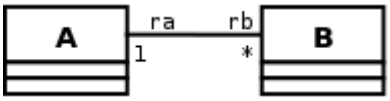
Association unidirectionnelle 1 vers 1




```
public class A {
  private B rb;
  public void addB( B b ) {
    if( b != null ) {
      this.rb=b;
    }
  }
}


public class B {
 ... // La classe B ne connaît pas l'existence de la classe A
}
```

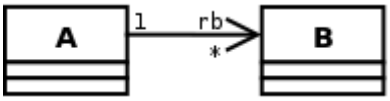
Association bidirectionnelle 1 vers N




```
public class A {
  private ArrayList <B> rb;
  public A() { rb = new  ArrayList<B>(); }
  public ArrayList <B> getArray() {return(rb);}
  public void remove(B b){rb.remove(b);}
  public void addB(B b){
    if( !rb.contains(b) ){
      if (b.getA()!=null) b.getA().remove(b);
      b.setA(this);
      rb.add(b);
    }
  }
}

public class B {
  private  A ra;
  public B() {}
  public A getA() { return (ra); }
  public void setA(A a){ this.ra=a; }
  public void addA(A a){
    if( a != null ) {
      if( !a.getArray().contains(this)) {
        if (ra != null) ra.remove(this);
        this.setA(a);
        ra.getArray().add(this);
      }
    }
  }
}
```

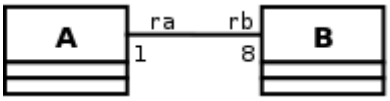
Association unidirectionnelle 1 vers plusieurs




```
public class A {
  private ArrayList <B> rb;
  public A() { rb = new  ArrayList<B>(); }
  public void addB(B b){
    if( !rb.contains( b ) ) {
      rb.add(b);
    }
  }
}

public class B {
 ... // B ne connaît pas l'existence de A
}
```

Association 1 vers N

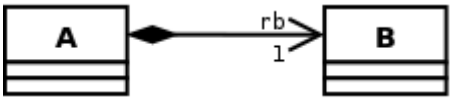


Dans ce cas, il faut utiliser un tableau plutôt qu'un vecteur. La dimension du tableau étant données par la cardinalité de la terminaison d'association.

Agrégations

Les agrégations s'implémentent comme les associations.

Composition



Une composition peut s'implémenter comme une association unidirectionnelle.

-

Implémentation en SQL

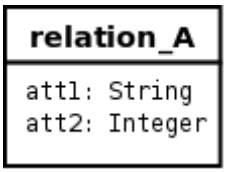
Introduction

Il est possible de traduire un diagramme de classe en modèle relationnel. Bien entendu, les méthodes des classes ne sont pas traduites. Aujourd'hui, lors de la conception de base de données, il devient de plus en plus courant d'utiliser la modélisation UML plutôt que le traditionnel modèle entités-associations.

Cependant, à moins d'avoir respecté une méthodologie adaptée, la correspondance entre le modèle objet et le modèle relationnel n'est pas une tâche facile. En effet, elle ne peut que rarement être complète puisque l'expressivité d'un diagramme de classes est bien plus grande que celle d'un schéma relationnel. Par exemple, comment représenter dans un schéma relationnel des notions comme la navigabilité ou la composition ? Toutefois, de nombreux AGL (Atelier de Génie Logiciel) comportent maintenant des fonctionnalités de traduction en SQL qui peuvent aider le développeur dans cette tâche.

Classe avec attributs

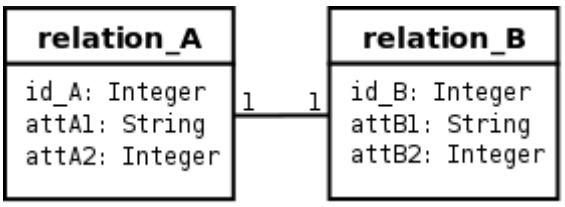
Chaque classe devient une relation. Les attributs de la classe deviennent des attributs de la relation. Si la classe possède un identifiant, il devient la clé primaire de la relation, sinon, il faut ajouter une clé primaire arbitraire.




```
create table relation_A (
   num_relation_A integer primary key,
   att1 text,
   att2 integer);
```

Association 1 vers 1

Pour représenter une association 1 vers 1 entre deux relation, la clé primaire de l'une des relations doit figurer comme clé étrangère dans l'autre relation.




```
create table relation_A (
   id_A integer primary key,
   attA1 text,
   attA2 integer);
```

create table relation_B (

id_B integer primary key,

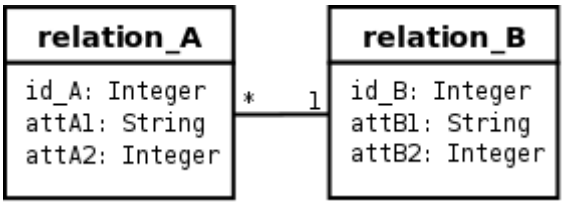
num_A integer references relation_A,

attB1 text,

attB2 integer);

Association 1 vers plusieurs

Pour représenter une association 1 vers plusieurs, on procède comme pour une association 1 vers 1, excepté que c'est forcément la relation du côté plusieurs qui reçoit comme clé étrangère la clé primaire de la relation du côté 1.



create table relation_A (

id_A integer primary key,

num_B integer references relation_B,

attA1 text,

attA2 integer);

create table relation_B (

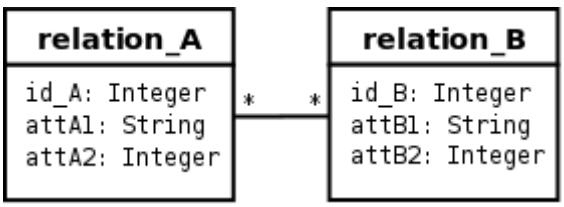
id_B integer primary key,

attB1 text,

attB2 integer);

Association plusieurs vers plusieurs

Pour représenter une association du type plusieurs vers plusieurs, il faut introduire une nouvelle relation dont les attributs sont les clés primaires des relations en association et dont la clé primaire est la concaténation de ces deux attributs.



create table relation_A (

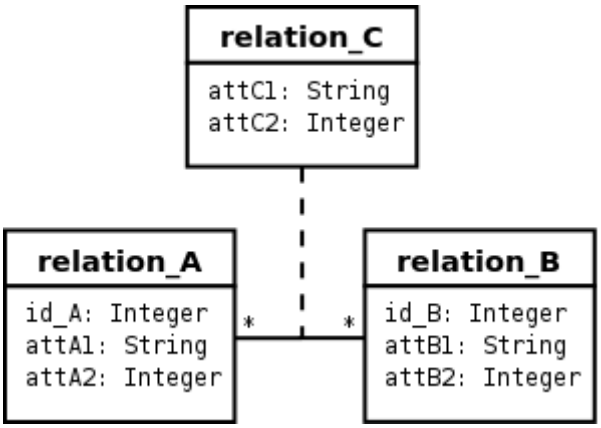
```
    id_A integer primary key,
    attA1 text,
    attA2 integer);

create table relation_B (
    id_B integer primary key,
    attB1 text,
    attB2 integer);

create table relation_A_B (
    num_A integer references relation_A,
    num_B integer references relation_B,
    primary key (num_A, num_B));
```

Classe-association plusieurs vers plusieurs

Le cas est proche de celui d'une association plusieurs vers plusieurs, les attributs de la classe-association étant ajoutés à la troisième relation qui représente, cette fois ci, la classe-association elle-même.




```
create table relation_A (
    id_A integer primary key,
    attA1 text,
    attA2 integer);

create table relation_B (
    id_B integer primary key,
```

attB1 text,

attB2 integer);

create table relation_C (

num_A integer references relation_A,

num_B integer references relation_B,

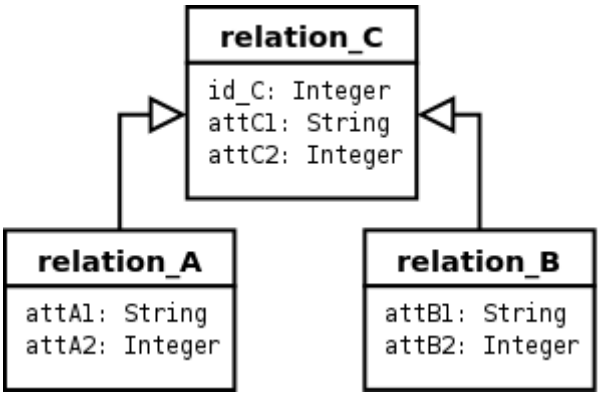
attC1 text,

attC2 integer,

primary key (num_A, num_B));

Héritage

Les relations correspondant aux sous-classes ont comme clé étrangère et primaire la clé de la relation correspondant à la classe parente. Un attribut type est ajouté dans la relation correspondant à la classe parente. Cet attribut permet de savoir si les informations d'un tuple de la relation correspondant à la classe parente peuvent être complétées par un tuple de l'une des relations correspondant à une sous-classe, et, le cas échéant, de quelle relation il s'agit. Ainsi, dans cette solution, un objet peut avoir ses attributs répartis dans plusieurs relations. Il faut donc opérer des jointures pour reconstituer un objet. L'attribut type de la relation correspondant à la classe parente doit indiquer quelles jointures faire.



create table relation_C (

id_C integer primary key,

attC1 text,

attC2 integer,

type text);

create table relation_A (

id_A references relation_C,

attA1 text,

```
    attA2 integer,
    primary key (id_A));

create table relation_B (
    id_B references relation_C,
    attB1 text,
    attB2 integer,
    primary key (id_B));
```

Une alternative à cette représentation est de ne créer qu'une seule table par arborescence d'héritage. Cette table doit contenir tous les attributs de toutes les classes de l'arborescence plus l'attribut type dont nous avons parlé ci-dessus. L'inconvénient de cette solution est qu'elle implique que les tuples contiennent de nombreuses valeurs nulles.

```
create table relation_ABC (
    id_C integer primary key,
    attC1 text, attC2 integer,
    attA1 text, attA2 integer,
    attB1 text, attB2 integer,
    type text);
```