Charles Veillon SA
Département Informatique
Méthodes & Système d'Information

Guide de la traduction d'un diagramme de classes en une structure de base de données relationnelle

Table des matières

1. Introduction

Ce document traite de la traduction d'un CLD (*Class Diagram*, diagramme de classes) en DDL (*Data Definition Language*, langage de définition de données), qui est un sous-ensemble de SQL (*Structured Query Language*, langage structuré d'interrogation).

Il présente la manière de traduire les classes, les associations et les attributs dans des tables, des colonnes et des clauses `NOT NULL`, `UNIQUE`, `PRIMARY KEY` et `FOREIGN KEY`. De plus, il examine la navigabilité des associations pour proposer la création d'index. Par contre, les clauses `CHECK` ne sont pas examinées.

Il a pour but de donner les règles à appliquer et de décrire le résultat à obtenir, pas de définir une marche à suivre pour effectuer la traduction ou une norme de dénomination pour les objets SQL.

Du moins dans cette version, il ne traite pas des associations n-aires, ni des hiérarchies de super-classes et sous-classes, ni des classes identifiées au travers d'un rôle, ni de l'optimisation du schéma physique par éclatement de tables ou dénormalisation. Il ne cherche à couvrir, pour l'instant, que les cas les plus courants.

Il ne parle pas de la génération DDL de Rational Rose. Il n'en est ni une explication, ni une critique.

Le DDL-SQL utilisé est celui d'Oracle7 et tend à correspondre à la norme SQL92.

2. Rappels

Tout d'abord quelques rappels qui nous seront nécessaires pour la suite.

2.1 Terminologie

On dira qu'un objet de modélisation UML (classe, attribut, association, rôle, ...) *est traduit par* ou *est représenté par* le ou les objets SQL correspondants (table, colonne, contrainte, ...). On dira également qu'un objet SQL est l'*image* (sous-entendu l'image *physique*) de l'objet de modélisation UML qu'il représente.

2.2 Identifiant d'une classe

En UML, l'identité, le fait d'être distinct de n'importe quel autre objet de la classe, est une propriété intrinsèque d'un objet. Donc, dans un CLD, les classes ne possèdent pas nécessairement un identifiant, elles peuvent cependant en posséder un ou même plusieurs.

Par contre, dans le modèle relationnel, il est indispensable qu'une relation possède (au moins) un identifiant (*master key* dans l'article de E. F. Codd, `PRIMARY KEY` en SQL).

Pour traduire un diagramme de classes en DDL, nous supposerons que chaque classe possède (au moins) un identifiant (implicite ou explicite). Si elle en possède plusieurs, l'un d'entre eux est choisi comme identifiant *principal*. Les autres sont des identifiants *secondaires*.

2.3 Contrainte de base UML

Un *objet* (*Object*) est une instance d'une classe, et un *lien* (*Link*) est une instance d'une association.

Un lien relie obligatoirement des objets existants. Par conséquent, tous les objets reliés doivent exister au plus tard à la création d'un lien. Un lien doit disparaître au plus tard lorsque l'un des objets reliés disparaît.

2.4 Rôle et multiplicité

Un rôle est l'extrémité d'une association. Une association binaire possède 2 rôles, une association ternaire en possède 3, ...

Dans une association, une classe reliée *joue* ou *assume* un rôle. Le rôle joué est celui qui est adjacent à la classe. Dans l'exemple ci-dessous, Classe1 assume le Role1 et Classe2 joue le Role2.

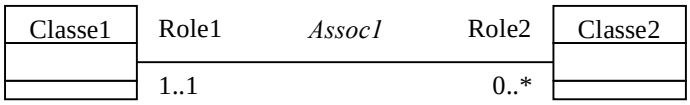


Figure 1 : Association, rôles et multiplicités en UML

La *multiplicité* d'un rôle détermine le nombre d'objets de la classe qui assume le rôle pouvant être reliés, par des liens de l'association, à *un objet donné* de la classe qui assume l'autre rôle. Une multiplicité est dénotée par deux valeurs, le minimum et le maximum de la multiplicité.

Classiquement, le *minimum* de la multiplicité prend l'une des valeurs 0 ou 1. Dans l'exemple, le minimum de la multiplicité de Role1 est 1, celui de Role2 est 0.

Si le minimum de la multiplicité est 1 (1..1 ou 1..*), chaque objet de la classe qui assume l'autre rôle de l'association doit obligatoirement être lié à un objet de la classe qui assume le rôle. Si le minimum est 0, l'existence d'un lien est facultative. Dans l'exemple, chaque objet de Classe2 est obligatoirement lié à (au moins) un objet de Classe1. Par contre, il peut exister des objets de Classe1 qui ne sont liés à aucun objet de Classe2.

Classiquement, le *maximum* de la multiplicité prend l'une des valeurs 1 ou *, cette dernière signifie « plusieurs ». Dans l'exemple, le maximum de la multiplicité de Role1 est 1, celui de Role2 est *.

Si le maximum est 1 (0..1 ou 1..1), chaque objet de la classe qui assume l'autre rôle de l'association est lié au plus à un objet de la classe qui assume le rôle. Si le maximum est * (0..* ou 1..*), il peut y avoir plusieurs objets. Dans l'exemple, chaque objet de Classe2 est lié au plus à un objet de Classe1. Par contre, un objet de Classe1 peut être lié à plusieurs objets de Classe2.

En résumé, pour un objet de Classe1 qui assume le Role1, il peut y avoir de zéro à plusieurs (0..*) objets de Classe2 qui sont liés au travers de Assoc1. Il doit y avoir exactement un (1..1) objet de Classe1 relié, au travers de Assoc1, à un objet donné de Classe2.

Il est à noter que Rational Rose utilise souvent le terme de *cardinalité* (*Cardinality*) pour parler de la multiplicité. La multiplicité correspond à la notion de connectivité dans Ida et de cardinalité dans Merise, mais la signification n'en est pas exactement la même. C'est ce qui fait que les multiplicités de UML sont « inversées » par rapport aux connectivités de Ida.

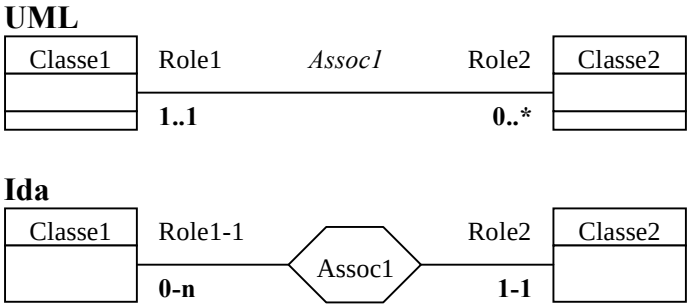


Figure 2 : Multiplicités en UML et connectivités en Ida

2.5 Identifiant d'une association

Un identifiant (d'une classe ou d'une association) est un ensemble *minimal* d'attributs et de rôles qui *détermine* (qui distingue de manière unique) un objet (de la classe) ou un lien (de l'association).

Un lien est déterminé par les objets qu'il relie, ce qui signifie qu'il n'y a qu'un seul lien (de la même association) entre des objets donnés. Une association est donc déterminée par l'ensemble de ses rôles. Toutefois, si l'ensemble des rôles n'est pas minimal, il ne forme pas un identifiant, il le contient.

Si un rôle possède une multiplicité dont le maximum est 1, *l'autre rôle* identifie l'association à lui tout seul. Dans l'exemple de la Figure 1, Role2 identifie Assoc1, parce que le maximum de la multiplicité de Role1 est 1. Il est important de noter que c'est Role1 (multiplicité 1..1) qui a

une multiplicité dont le maximum est 1, et que c'est Role2 (multiplicité 0..*) qui identifie l'association.

Si l'association possède plusieurs rôles avec une multiplicité dont le maximum est 1, elle possède plusieurs identifiants. Il sera alors nécessaire de choisir l'identifiant principal parmi eux. Les autres seront des identifiants secondaires.

Si toutes les multiplicités de l'association ont un maximum * (association 0..* / 0..*), l'identifiant de l'association est formé de l'ensemble des rôles.

2.6 Traduction littérale ou optimisée

La traduction *littérale* (ou « brute », voir le Cours Bases de Données) est une traduction où chaque classe et chaque association est traduite par une table indépendante. Elle produit plus de tables qu'il n'est nécessaire, mais elle est plus simple à comprendre. Elle est aussi une bonne base pour expliquer et comprendre la traduction optimisée.

Cette traduction ne permet pas d'imposer l'existence d'un lien, exprimée par un minimum de multiplicité 1 (1..1 ou 1..*).

L'image d'une association avec un rôle de maximum de multiplicité de 1 (0..1 ou 1..1) possède le même identifiant (l'image du rôle identifiant) que l'image de l'autre classe reliée. Par la traduction littérale, nous obtenons souvent plusieurs tables avec le même identifiant, l'image de la classe et les images de ces associations.

La traduction *optimisée* est obtenue en collant ensemble les tables qui possèdent le même identifiant. Elle réduit le nombre de tables et permet d'imposer l'existence d'un lien exprimée par une multiplicité 1..1, mais ne permet pas, non plus, d'imposer l'existence d'un lien exprimée par une multiplicité 1..*.

Lorsque le collage est possible, il n'est pas obligatoire. Dans certains cas, il n'est pas judicieux de coller une association dans la classe, par exemple si les populations sont très différentes (une classe qui n'est que très rarement liée par l'association).

2.7 Renommage de colonnes

Deux cas introduisent des ambiguïtés quant aux noms des colonnes d'une table lors de la traduction :

- Les associations réflexives
- Plusieurs associations entre les deux mêmes classes

Une association *réflexive* (on dit aussi *cyclique*) est une association dont les deux rôles sont assumés par la même classe. Il est alors indispensable de nommer les rôles.

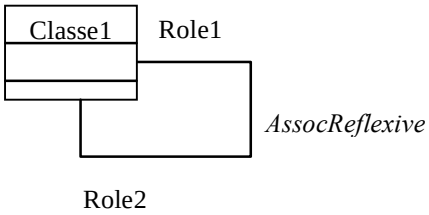


Figure 3 : Association réflexive

Les images de chaque rôle sont identiques, elles se traduisent par les mêmes noms. Il est nécessaire de renommer les colonnes, images des rôles, pour lever l'ambiguïté. Pour cela, on utilise en général le nom du rôle, soit pour suffixer le nom de la colonne, soit pour s'y substituer. Ce problème se produit aussi bien dans la traduction littérale que dans la traduction optimisée.

Si deux classes données sont reliées par plusieurs associations, il est nécessaire de nommer les associations et/ou les rôles.

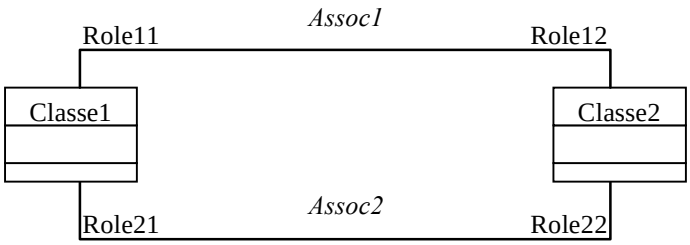


Figure 4 : Plusieurs associations entre les mêmes classes

Les images des deux associations contiennent des noms identiques de colonnes. En traduction optimisée, si les deux associations sont collées dans la même classe, elles donneront lieu à des ambiguïtés de dénomination. Il est nécessaire de renommer les colonnes qui représentent des rôles. Pour cela, on utilise en général le nom du rôle, à défaut le nom de l'association.

2.8 Collage de tables

Nous ne considérons ici que le collage de tables qui possèdent le même identifiant. Si les deux tables originales sont normalisées (en 3ème forme normale, 3FN), le résultat reste normalisé.

Coller une table dans une autre consiste à supprimer la première et à introduire ses colonnes dans la seconde.

Tables originales :

```
CREATE TABLE Table1 (
Ident <datatype>,
Attr1 <datatype>,
PRIMARY KEY (Ident)) ;

CREATE TABLE Table2 (
Ident  <datatype>,
ForKey <datatype>,
Attr2  <datatype>,
FOREIGN KEY (Ident)  REFERENCES Table1   (Ident),
FOREIGN KEY (ForKey) REFERENCES ForTable (ForKey),
PRIMARY KEY (Ident)) ;
```

Table résultat du collage de Table2 dans Table1 :

```
CREATE TABLE Table1 (
Ident  <datatype>,
Attr1  <datatype>,
ForKey <datatype>,
Attr2  <datatype>,
FOREIGN KEY (ForKey) REFERENCES ForTable (ForKey),
PRIMARY KEY (Ident)) ;
```

Figure 5 : Collage de deux tables

On remarque que :

- Il ne subsiste qu'un exemplaire de la `PRIMARY KEY`,
- Les autres attributs des deux tables sont tous présents dans le résultat,
- La clause `FOREIGN KEY`, qui porte sur la table conservée, a disparu,
- Les autres clauses `FOREIGN KEY` subsistent.

2.9 PRIMARY KEY

Une clause `PRIMARY KEY` implique la clause `NOT NULL` sur chacune de ses colonnes et une clause `UNIQUE` sur l'ensemble de ses colonnes. Ces dernières clauses sont redondantes, il est donc inutile de les spécifier.

2.10 Index en Oracle7

La discussion qui suit ne se préoccupe que des index nécessaires pour permettre d'atteindre les objets au travers des liens d'une association.

Un index est un moyen d'améliorer les performances de requêtes d'interrogation de la base de données. Sa présence ou son absence ne modifie ni la signification, ni la structure des données.

Un index est utilisé pour retrouver, sans devoir balayer toute la table, des lignes qui possèdent des valeurs déterminées ou pour accéder à une table dans un ordre déterminé. Toutefois, il a un coût en espace disque et en temps de mise à jour.

L'ordre des colonnes dans l'index a une importance. Supposons qu'on a défini un index sur les colonnes A et B d'une table, dans cet ordre. Oracle7 utilise l'index entier lorsque la requête précise les valeurs de A et de B. Si la requête ne fixe que la valeur de A, Oracle7 peut utiliser la partie haute de l'index pour accéder aux lignes de la table qui ont une valeur fixée de A, et, de plus, dans l'ordre des valeurs de B. Par contre, cet index est inutilisable pour accéder aux lignes qui possèdent une valeur fixée de B si on ne précise pas la valeur de A.

Il est donc redondant, inutile, et même nuisible, de définir un index qui est une partie haute d'un autre index. Par exemple, si un index est défini sur les colonnes A, B et C d'une table, un autre index sur les colonnes A et B est inutile.

Il faut relever que les clauses `UNIQUE` et `PRIMARY KEY` impliquent la création d'un index en Oracle7. Il est donc superflu, et même nuisible, de créer un index sur une colonne, ou un ensemble de colonnes *dans le même ordre*, qui portent une clause `UNIQUE` ou `PRIMARY KEY`. Par contre, nous verrons plus loin qu'il peut être nécessaire de créer un index sur le même ensemble de colonnes qu'une `PRIMARY KEY`, mais *dans un ordre différent*.

Selon la le contenu des index impliqués par des clauses `UNIQUE` et `PRIMARY KEY`, il peut être nécessaire de créer des index supplémentaires pour permettre, lorsque c'est nécessaire, de parcourir une association dans les deux sens. UML dit qu'un rôle d'une association est *navigable*.

3. Principes de la traduction

3.1 Attribut

Chaque attribut est traduit par une colonne dans une table.

Un attribut, s'il fait partie d'un identifiant, possède généralement plusieurs images. S'il n'en fait pas partie, il n'en possède qu'une.

Bien que cette classification ne fasse pas partie de UML, un attribut peut être obligatoire ou facultatif.

Un attribut *obligatoire* est un attribut qui doit posséder une valeur pour chaque objet de la classe. Par exemple, chaque personne a un nom.

Un attribut *facultatif* est un attribut qui peut ne posséder aucune valeur pour certains objets de la classe. Par exemple, certaines personnes ne possèdent pas de nom d'alliance (tous les célibataires).

Un attribut obligatoire possède la clause `NOT NULL`, contrairement à un attribut facultatif.

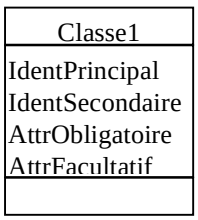
```
AttrObligatoire <datatype> NOT NULL,
AttrFacultatif  <datatype>,
```

Figure 6 : Traduction d'un attribut obligatoire ou facultatif

3.2 Classe

Il arrive parfois que, sous certaines conditions, on décide de ne pas traduire directement une classe dans le schéma de base de données (par l'utilisation de IV-Domaine, par exemple). Ces cas mis à part, chaque classe est traduite par une table, chacun de ses attributs constitue une colonne de la table.

L'image de l'identifiant principal de la classe porte la clause `PRIMARY KEY`. Les images des éventuels identifiants secondaires portent les clauses `NOT NULL` et `UNIQUE`.




```
CREATE TABLE Classe1 (
IdentPrincipal  <datatype>,
IdentSecondaire <datatype> NOT NULL,
AttrObligatoire <datatype> NOT NULL,
AttrFacultatif  <datatype>,
PRIMARY KEY (IdentPrincipal),
UNIQUE (IdentSecondaire)) ;
```

Figure 7 : Traduction d'une classe

3.3 Rôle

Un rôle d'une association est traduit par :

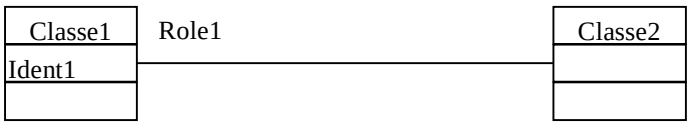
- une copie de l'image de l'identifiant principal de la classe qui assume le rôle,
- une contrainte référentielle sur la table qui représente cette classe.

Puisqu'un rôle est représenté par la copie de l'image d'un identifiant, un attribut qui fait partie d'un identifiant peut avoir plusieurs images, contrairement aux autres attributs.

Comme un lien ne peut relier que des objets qui existent, la ou les colonnes de l'image du rôle doivent porter la clause `NOT NULL`.

La contrainte référentielle `FOREIGN KEY` porte sur la ou les colonnes de l'image du rôle et référence obligatoirement toute la `PRIMARY KEY` de l'image de la classe qui assume le rôle. En SQL92, l'option `MATCH FULL` doit être précisée, mais elle ne fait pas partie du SQL de Oracle7.

Lorsque plusieurs rôles assumés par la même classe ont leurs images dans la même table, les colonnes de l'image des rôles doivent être renommées.




```
Ident1 <datatype> NOT NULL,
FOREIGN KEY (Ident1) REFERENCES Classe1 (Ident1),
ou
Ident1Role1 <datatype> NOT NULL,
FOREIGN KEY (Ident1Role1) REFERENCES Classe1 (Ident1),
```

Figure 8 : Traduction d'un rôle

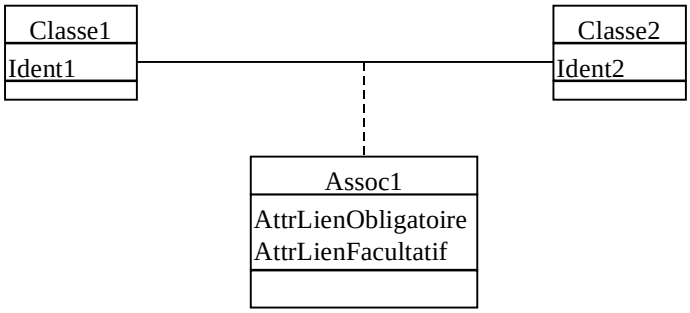
3.4 Association

L'image d'une association contient l'image de chacun de ses rôles.

Si l'association possède une classe de lien (*Link Class*), son image contient les images de tous les attributs de lien (*Link Attributes*), obligatoires ou facultatifs.

La `PRIMARY KEY` est l'image de l'identifiant principal de l'association, qui lui-même dépend des multiplicités de l'association.

Les images des éventuels rôles identifiants secondaires (non représentés ici) portent les clauses `NOT NULL` et `UNIQUE`.




```
CREATE TABLE Assoc1 (
Ident1 <datatype> NOT NULL,
Ident2 <datatype> NOT NULL,
AttrLienObligatoire <datatype> NOT NULL,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident1) REFERENCES Classe1 (Ident1),
FOREIGN KEY (Ident2) REFERENCES Classe2 (Ident2),
PRIMARY KEY (<image de l'identifiant>)) ;
```

Figure 9 : Traduction littérale d'une association

La traduction optimisée n'est possible que si l'association possède (au moins) une multiplicité dont le maximum est 1 (0..1 ou 1..1). Rappelons que l'image de l'association est collée dans l'image de la classe qui assume *l'autre* rôle.

En traduction optimisée, dans l'image de l'association collée :

- La clause `PRIMARY KEY` de l'image de l'identifiant de l'association est confondue avec l'image de celui de la classe,
- La clause `FOREIGN KEY` de l'image de l'identifiant de l'association est supprimée,
- Les clauses `FOREIGN KEY` et les éventuelles clauses `UNIQUE` (pour les identifiants secondaires) des images des autres rôles sont maintenues.

Si la multiplicité est 1..1, le lien est *obligatoire* : chaque objet possède nécessairement un lien. Alors :

- Les clauses `NOT NULL` des images des autres rôles (autres que le rôle identifiant principal) et des attributs obligatoires de l'association sont *maintenues.*

Le maintien des clauses `NOT NULL` impose l'existence obligatoire du lien, exprimée par la valeur 1 du minimum de la multiplicité 1..1.

Si la multiplicité est 0..1, le lien est *facultatif* : il peut exister des objets sans lien. Alors :

- Les clauses `NOT NULL` des rôles, autres que le rôle identifiant principal de l'association, et des attributs obligatoires de l'association sont *supprimées.*

L'absence des clauses `NOT NULL` autorise l'absence du lien, exprimée par la valeur 0 du minimum de la multiplicité 0..1. Toutefois, remarquons que la cohérence de l'image de l'association n'est pas entièrement garantie pour autant. Les colonnes doivent être toutes `NULL` (si le lien n'existe pas) ou posséder toutes une valeur (sauf éventuellement les attributs de lien facultatifs). Cette contrainte doit être imposée par programme.

4. Résumé des règles de traduction

Objet de modélisation UML	*Clauses DDL-SQL*
Attribut obligatoire	`NOT NULL`
Identifiant principal	`PRIMARY KEY`
Identifiant secondaire	`NOT NULL, UNIQUE`
Rôle	`NOT NULL, FOREIGN KEY`
Collage d'une association facultative	suppression des `NOT NULL`

5. Cas particuliers d'associations

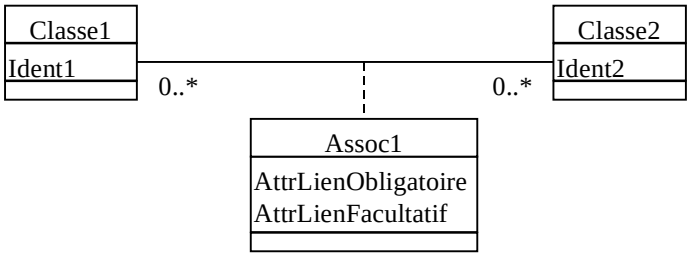
Nous allons, maintenant, passer en revue la traduction des associations binaires les plus courantes.

Dans chacun des cas, une classe de lien est mentionnée pour la complétude de l'exemple. Pour obtenir le résultat de la traduction d'une association sans classe de lien, il suffit, bien sûr, de retirer les images des attributs de la classe de lien.

5.1 Association 0..* / 0..*

L'association est représentée par une table indépendante, la traduction ne peut pas être optimisée.

L'identifiant de l'association est formé de l'ensemble des rôles, la `PRIMARY KEY` est constituée de la concaténation des images des rôles. Les clauses `NOT NULL` sur les images des rôles ne sont plus nécessaires. Les clauses `FOREIGN KEY` subsistent.




```
CREATE TABLE Assoc1 (
Ident1 <datatype>,
Ident2 <datatype>,
AttrLienObligatoire <datatype> NOT NULL,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident1) REFERENCES Classe1 (Ident1),
FOREIGN KEY (Ident2) REFERENCES Classe2 (Ident2),
PRIMARY KEY (Ident1, Ident2)) ;
```

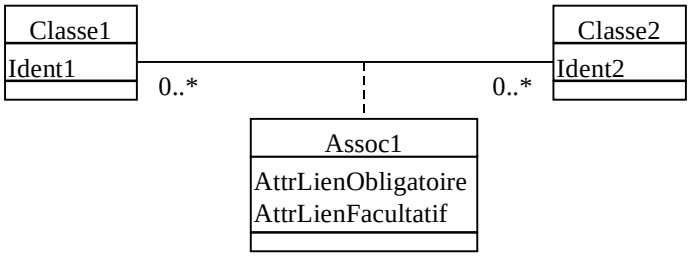
Figure 10 : Association 0..* / 0..*

L'ordre des colonnes dans la `PRIMARY KEY` doit être choisi, il est important. L'index de la `PRIMARY KEY` permet de parcourir efficacement les liens dans un sens, mais pas dans l'autre.

Dans l'exemple, on a choisi l'ordre (Ident1, Ident2). Connaissant un objet de Classe1, on peut retrouver tous les objets de Classe2 qui lui sont liés. Mais il n'est pas possible de retrouver efficacement les objets de Classe1 qui sont liés à un objet donné de Classe2. On peut parcourir Assoc1 de Classe1 vers Classe2.

En inversant l'ordre des colonnes dans la `PRIMARY KEY`, on inverse la situation. On peut parcourir Assoc1 de Classe2 vers Classe2, mais pas l'inverse.

S'il est nécessaire de parcourir l'association dans les deux sens (ce qui est souvent le cas), il faut créer un index supplémentaire.



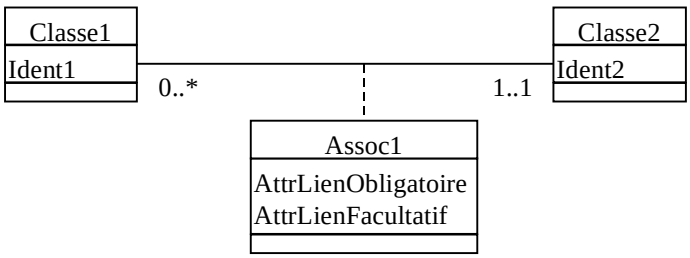

```
CREATE TABLE Assoc1 (
Ident1 <datatype>,
Ident2 <datatype>,
AttrLienObligatoire <datatype> NOT NULL,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident1) REFERENCES Classe1 (Ident1),
FOREIGN KEY (Ident2) REFERENCES Classe2 (Ident2),
PRIMARY KEY (Ident1, Ident2)) ;

CREATE INDEX C1ParC2 ON Assoc1 (Ident2, Ident1) ;
```

Figure 11 : Association 0..* / 0..* bidirectionnelle

5.2 Association 0..* / 1..1

Le rôle 0..* identifie l'association, son image est la `PRIMARY KEY`. Sur l'image de ce rôle, la clause `NOT NULL` est inutile. Par contre, elle subsiste sur l'image de l'autre rôle (1..1).



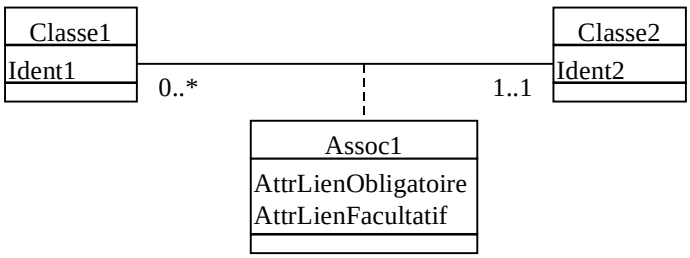

```
CREATE TABLE Assoc1 (
Ident1 <datatype>,
Ident2 <datatype> NOT NULL,
AttrLienObligatoire <datatype> NOT NULL,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident1) REFERENCES Classe1 (Ident1),
FOREIGN KEY (Ident2) REFERENCES Classe2 (Ident2),
PRIMARY KEY (Ident1)) ;
```

Figure 12 : Association 0..* / 1..1 (littérale)

L'index de la `PRIMARY KEY` permet de parcourir efficacement les liens dans un sens, mais pas dans l'autre.

Dans l'exemple, connaissant un objet de Classe1, on peut retrouver l'objet de Classe2 qui lui est lié. Mais il n'est pas possible de retrouver efficacement les objets de Classe1 qui sont liés à un objet donné de Classe2. On peut parcourir Assoc1 de Classe1 vers Classe2, mais pas l'inverse.

Le sens du parcours est imposé par les multiplicités. S'il est nécessaire de parcourir l'association dans l'autre sens (ce qui est souvent le cas), il faut créer un index supplémentaire.



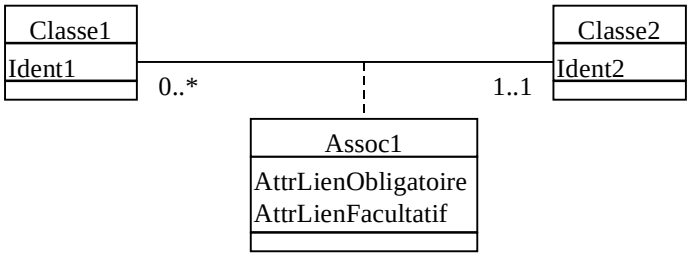

```
CREATE TABLE Assoc1 (
Ident1 <datatype>,
Ident2 <datatype> NOT NULL,
AttrLienObligatoire <datatype> NOT NULL,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident1) REFERENCES Classe1 (Ident1),
FOREIGN KEY (Ident2) REFERENCES Classe2 (Ident2),
PRIMARY KEY (Ident1)) ;

CREATE INDEX C1ParC2 ON Assoc1 (Ident2, Ident1) ;
```

Figure 13 : Association 0..* / 1..1 bidirectionnelle (littérale)

A cause de la multiplicité 1..1, l'image de l'association peut être collée (traduction optimisée). Le collage permet d'imposer la contrainte d'existence du lien (minimum de la multiplicité 1..1) en maintenant les clauses NOT NULL sur l'image de l'association collée dans l'image de la classe.

Les conditions de parcours bidirectionnel de l'association sont les mêmes que pour la traduction littérale.




```
CREATE TABLE Classe1 (
Ident1 <datatype>,
Ident2 <datatype> NOT NULL,
AttrLienObligatoire <datatype> NOT NULL,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident2) REFERENCES Classe2 (Ident2),
PRIMARY KEY (Ident1)) ;

CREATE INDEX C2ParC1 ON Classe1 (Ident2, Ident1) ;
```

Figure 14 : Association 0..* / 1..1 bidirectionnelle (optimisée)

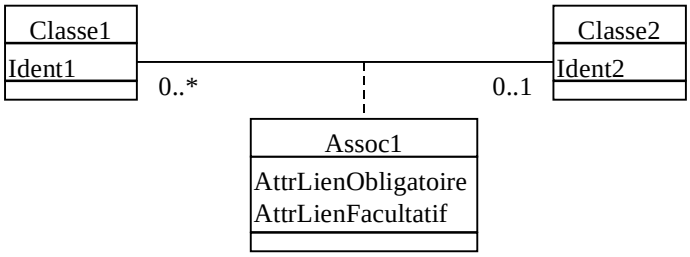
5.3 Association 0..* / 0..1

Le rôle 0..* identifie l'association, son image est la `PRIMARY KEY`. Sur l'image de ce rôle, la clause `NOT NULL` est inutile.

L'index de la `PRIMARY KEY` permet de parcourir efficacement les liens dans un sens, mais pas dans l'autre.

La traduction littérale est semblable à celle de l'association 0..* / 1..1, ainsi que les conditions de parcours bidirectionnel. A cause de la multiplicité 0..1, l'image de l'association peut être collée (traduction optimisée). Nous ne donnerons qu'un exemple de traduction optimisée avec parcours bidirectionnel.

Le lien est facultatif (minimum de la multiplicité 0..1). Dans la traduction optimisée, les clauses `NOT NULL` sont éliminées sur l'image de l'association collée dans l'image de la classe. La cohérence de l'image de l'association doit être imposée par programme.



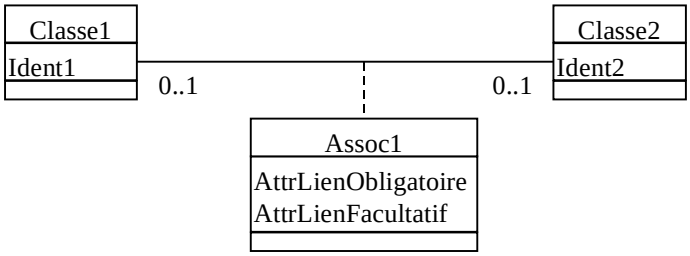

```
CREATE TABLE Classe1 (
Ident1 <datatype>,
Ident2 <datatype>,
AttrLienObligatoire <datatype>,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident2) REFERENCES Classe2 (Ident2),
PRIMARY KEY (Ident1)) ;

CREATE INDEX C2ParC1 ON Classe1 (Ident2, Ident1) ;
```

Figure 15 : Association 0..* / 0..1 (optimisée et bidirectionnelle)

5.4 Association 0..1 / 0..1

Les multiplicités des deux rôles sont identiques et possèdent un maximum de 1, donc chaque rôle identifie l'association. Elle possède deux identifiants, il faut choisir l'identifiant principal parmi eux, l'autre sera un identifiant secondaire.




```
CREATE TABLE Assoc1 (
Ident1 <datatype>,
Ident2 <datatype> NOT NULL,
AttrLienObligatoire <datatype> NOT NULL,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident1) REFERENCES Classe1 (Ident1),
FOREIGN KEY (Ident2) REFERENCES Classe2 (Ident2),
PRIMARY KEY (Ident1)
UNIQUE (Ident2)) ;
```

ou

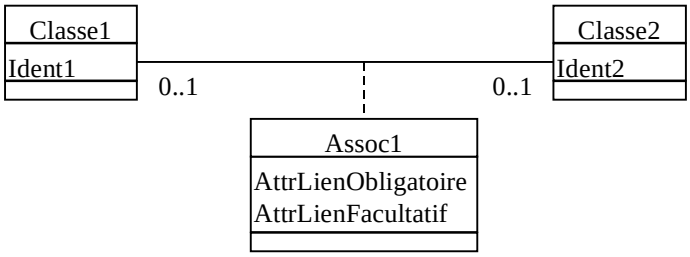
```
CREATE TABLE Assoc1 (
Ident2 <datatype>,
Ident1 <datatype> NOT NULL,
AttrLienObligatoire <datatype> NOT NULL,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident2) REFERENCES Classe2 (Ident2),
FOREIGN KEY (Ident1) REFERENCES Classe1 (Ident1),
PRIMARY KEY (Ident2)
UNIQUE (Ident1)) ;
```

Figure 16 : Association 0..1 / 0..1 (littérale)

Comme l'identifiant secondaire porte une clause `UNIQUE`, il possède aussi un index, qui autorise le parcours bidirectionnel de l'association.

Dans la traduction optimisée, la clause `FOREIGN KEY` de l'image du rôle identifiant principal est supprimée et les clauses `NOT NULL` de l'image doivent être retirées, puisque le lien est facultatif.

La cohérence de l'image de l'association doit être imposée par programme.




```
CREATE TABLE Classe1 (
Ident1 <datatype>,
Ident2 <datatype>,
AttrLienObligatoire <datatype>,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident2) REFERENCES Classe2 (Ident2),
PRIMARY KEY (Ident1)
UNIQUE (Ident2)) ;
```

ou

```
CREATE TABLE Classe2 (
Ident2 <datatype>,
Ident1 <datatype>,
AttrLienObligatoire <datatype>,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident1) REFERENCES Classe1 (Ident1),
PRIMARY KEY (Ident2)
UNIQUE (Ident1)) ;
```

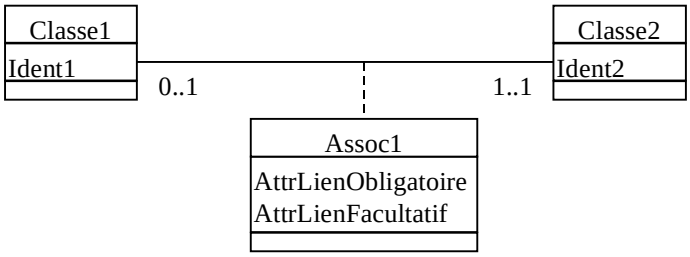
Figure 17 : Association 0..1 / 0..1 (optimisée)

5.5 Association 0..1 / 1..1

Le maximum de la multiplicité de chaque rôle est 1, chacun identifie l'association. Comme elle possède deux identifiants, il faut choisir l'identifiant principal.

Avec une multiplicité 1..1, il est rare qu'on renonce à coller l'image de l'association pour imposer l'existence du lien, exprimée par le minimum de la multiplicité 1..1. On privilégiera le rôle 0..1 comme identifiant principal. L'autre rôle reste identifiant secondaire, on remarque qu'il est également identifiant secondaire de Classe1.

Comme l'identifiant secondaire porte une clause `UNIQUE`, il possède aussi un index, qui autorise le parcours bidirectionnel de l'association.



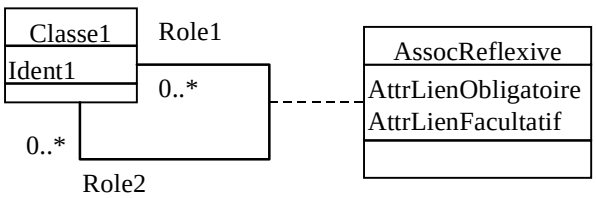

```
CREATE TABLE Classe1 (
Ident1 <datatype>,
Ident2 <datatype> NOT NULL,
AttrLienObligatoire <datatype> NOT NULL,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident2) REFERENCES Classe2 (Ident2),
PRIMARY KEY (Ident1)
UNIQUE (Ident2)) ;
```

Figure 18 : Association 0..1 / 1..1

5.6 Associations réflexives

Dans ce qui suit, nous ne donnerons que la traduction optimisée (lorsqu'elle est possible) en autorisant un parcours bidirectionnel.

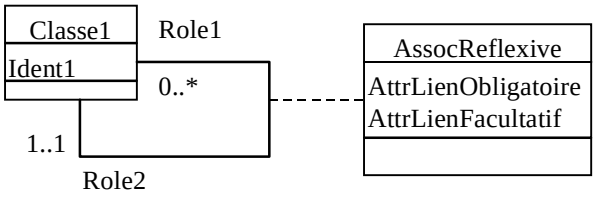
La cohérence de l'image de l'association doit être imposée par programme pour les associations réflexives 0..* / 0..1 et 0..1 / 0..1.




```
CREATE TABLE AssocReflexive (
Ident1Role1 <datatype>,
Ident1Role2 <datatype>,
AttrLienObligatoire <datatype> NOT NULL,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident1Role1) REFERENCES Classe1 (Ident1),
FOREIGN KEY (Ident1Role2) REFERENCES Classe1 (Ident1),
PRIMARY KEY (Ident1Role1, Ident1Role2)) ;

CREATE INDEX C1ParRole2 ON Assoc1 (Ident1Role2, Ident1Role1) ;
```

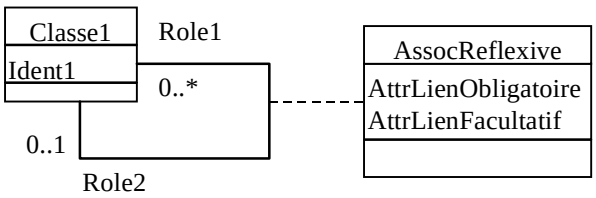
Figure 19 : Association réflexive 0..* / 0..*




```
CREATE TABLE Classe1 (
Ident1 <datatype>,
Ident1Role2 <datatype> NOT NULL,
AttrLienObligatoire <datatype> NOT NULL,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident1Role2) REFERENCES Classe1 (Ident1),
PRIMARY KEY (Ident1)) ;

CREATE INDEX C1ParRole2 ON Classe1 (Ident1Role2, Ident1) ;
```

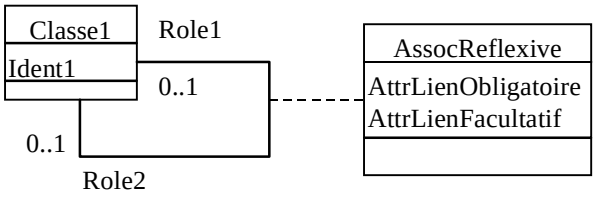
Figure 20 : Association réflexive 0..* / 1..1 (optimisée et bidirectionnelle)




```
CREATE TABLE Classe1 (
Ident1 <datatype>,
Ident1Role2 <datatype>,
AttrLienObligatoire <datatype>,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident1Role2) REFERENCES Classe1 (Ident1),
PRIMARY KEY (Ident1)) ;

CREATE INDEX ParRole2 ON Classe1 (Ident1Role2, Ident1) ;
```

Figure 21 : Association réflexive 0..* / 0..1 (optimisée et bidirectionnelle)

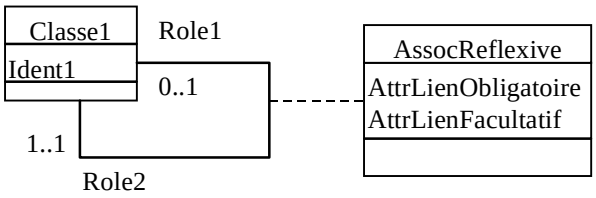



```
CREATE TABLE Classe1 (
Ident1 <datatype>,
Ident1Role2 <datatype>,
AttrLienObligatoire <datatype>,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident1Role2) REFERENCES Classe1 (Ident1),
PRIMARY KEY (Ident1)
UNIQUE (Ident1Role2)) ;
```

ou

```
CREATE TABLE Classe1 (
Ident1 <datatype>,
Ident1Role1 <datatype>,
AttrLienObligatoire <datatype>,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident1Role1) REFERENCES Classe1 (Ident1),
PRIMARY KEY (Ident1)
UNIQUE (Ident1Role1)) ;
```

Figure 22 : Association réflexive 0..1 / 0..1 (optimisée)




```
CREATE TABLE Classe1 (
Ident1 <datatype>,
Ident1Role2 <datatype> NOT NULL,
AttrLienObligatoire <datatype> NOT NULL,
AttrLienFacultatif  <datatype>,
FOREIGN KEY (Ident1Role2) REFERENCES Classe1 (Ident1),
PRIMARY KEY (Ident1)
UNIQUE (Ident1Role2)) ;
```

Figure 23 : Association réflexive 0..1 / 1..1