

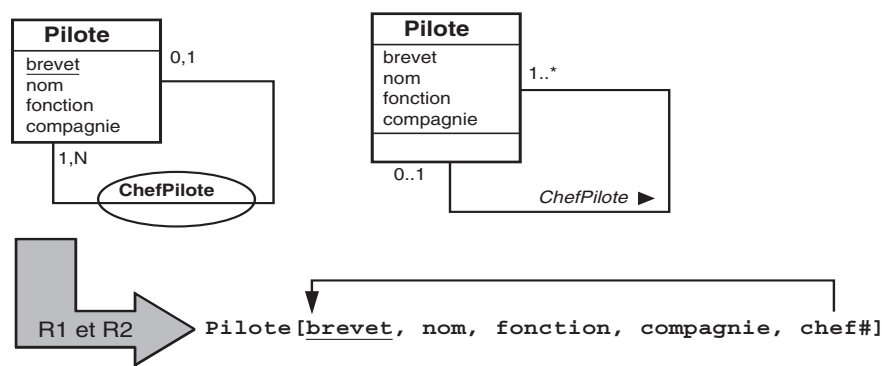
Associations réflexives

Les associations réflexives sont des associations binaires (*un-à-un*, *un-à-plusieurs*, *plusieurs-à-plusieurs*) ou *n*-aires. Les transformations sont analogues aux associations non réflexives.

Un-à-plusieurs

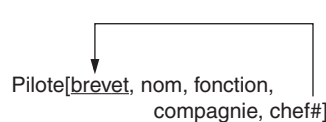
Les règles R1 et R2 sont appliquées à l'association réflexive *un-à-plusieurs* de l'exemple 3-12. La clé étrangère contiendra le code du chef pilote pour chaque pilote. Pour tout chef, cette clé étrangère ne contiendra pas de valeur (NULL).

Figure 3-12 Exemple d'association réflexive un-à-plusieurs



La partie SQL indiquée en gras souligne la particularité de l'association réflexive.

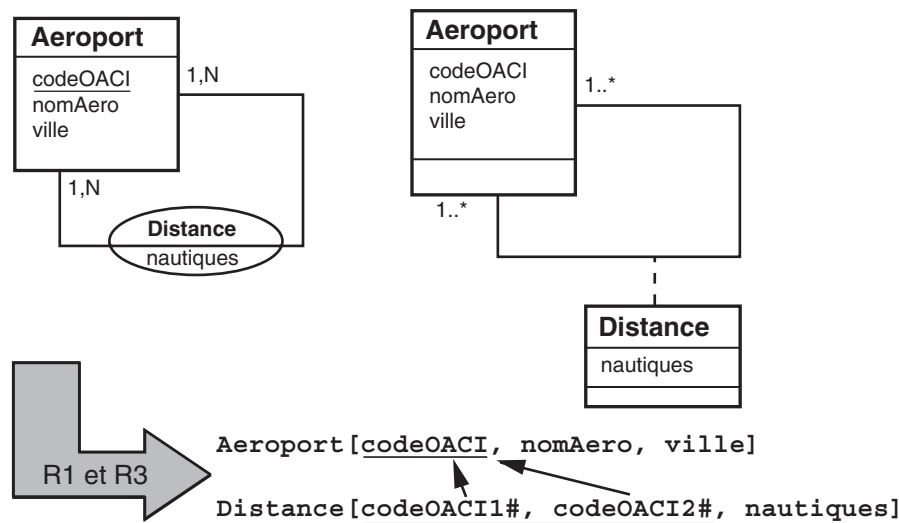
Tableau 3.9 Association réflexive un-à-plusieurs

Schéma logique	Script SQL2
 <p>Pilote[brevet, nom, fonction, compagnie, chef#]</p>	<pre>CREATE TABLE pilote (brevet VARCHAR(8), nom VARCHAR(30), fonction VARCHAR(4), compagnie VARCHAR(4), chef VARCHAR(8), CONSTRAINT pk_pilote PRIMARY KEY(brevet), CONSTRAINT fk_pilote_chef_pilote FOREIGN KEY(chef) REFERENCES pilote(brevet))</pre>

Plusieurs-à-plusieurs

Les règles R1 et R3 sont appliquées à l'association réflexive *plusieurs-à-plusieurs* de l'exemple 3-13 (modélisation de la distance entre deux aéroports).

Figure 3-13 Exemple d'association réflexive plusieurs-à-plusieurs



Le script SQL est le suivant.

Tableau 3.10 Association réflexive plusieurs-à-plusieurs

Schéma logique	Script SQL2
<p>Aeroport[<u>codeOACI</u>, nomAero, ville]</p> <p>Distance[<u>OACI1#</u>, <u>OACI2#</u>, nautiques]</p>	<pre>CREATE TABLE Aeroport (codeOACI VARCHAR(8), nomAero VARCHAR(30), ville VARCHAR(20), CONSTRAINT pk_Aeroport PRIMARY KEY(codeOACI)) CREATE TABLE Distance (OACI1 VARCHAR(8), OACI2 VARCHAR(8), nautiques NUMBER(4), CONSTRAINT pk_Distance PRIMARY KEY(OACI1, OACI2), CONSTRAINT fk_Distance_Aeroport1 FOREIGN KEY(OACI1) REFERENCES Aeroport(codeOACI), CONSTRAINT fk_Distance_Aeroport2 FOREIGN KEY(OACI2) REFERENCES Aeroport(codeOACI))</pre>

Solution universelle

Il est possible de modéliser toute association par une table supplémentaire, qui contiendra autant de clés étrangères qu'il y a de tables à relier, et sur lesquelles existera ou non une contrainte de type UNIQUE. Ce principe s'apparente à la règle R3.



Cette solution présente l'avantage de pouvoir faire évoluer le schéma plus facilement si les cardinalités viennent à changer dans le temps. En effet, il n'y a besoin de modifier la structure d'aucune table, seules des contraintes UNIQUE devront être désactivées.

Considérons à nouveau l'exemple des stages des étudiants. La table supplémentaire (Effectuer) contient deux colonnes réalisant l'association. Sur chaque colonne, il sera impératif de définir une contrainte UNIQUE qui garantira les cardinalités maximales de l'association (ici *un-à-un*).

Tableau 3.11 Solution universelle pour une association un-à-un

Schéma logique	Script SQL2
<pre> graph TD Stage[Stage[nstage, entreprise]] <--> Effectuer[Effectuer[netu#, nstage#]] Effectuer --> Etudiant[Etudiant[netu, nometu]] </pre>	<pre> CREATE TABLE Stage (nstage VARCHAR(4), entreprise VARCHAR(30), CONSTRAINT pk_stage PRIMARY KEY(nstage)) CREATE TABLE Etudiant (netu VARCHAR(2), nometu VARCHAR(30), CONSTRAINT pk_etudiant PRIMARY KEY(netu)) CREATE TABLE Effectuer (netu VARCHAR(2), nstage VARCHAR(4), CONSTRAINT pk_Effectuer PRIMARY KEY(netu,nstage), CONSTRAINT fk_Effectuer_nstage_Stage FOREIGN KEY(nstage) REFERENCES Stage(nstage), CONSTRAINT fk_Effectuer_netu_Etudiant FOREIGN KEY(netu) REFERENCES Etudiant(netu), CONSTRAINT unique_Effectuer_netu UNIQUE (netu), CONSTRAINT unique_Effectuer_nstage UNIQUE (nstage)) </pre>

Ce schéma est évolutif. En effet, si un stage peut être effectué par plusieurs étudiants, il faudra simplement désactiver la contrainte (`ALTER TABLE Effectuer DISABLE CONSTRAINT unique_Effectuer_nstage`). On retrouvera une association *un-à-plusieurs*. Si un étudiant peut effectuer plusieurs stages, il faudra désactiver l'autre contrainte. On se retrouvera alors avec une association *plusieurs-à-plusieurs*.

Traduction des associations d'héritage

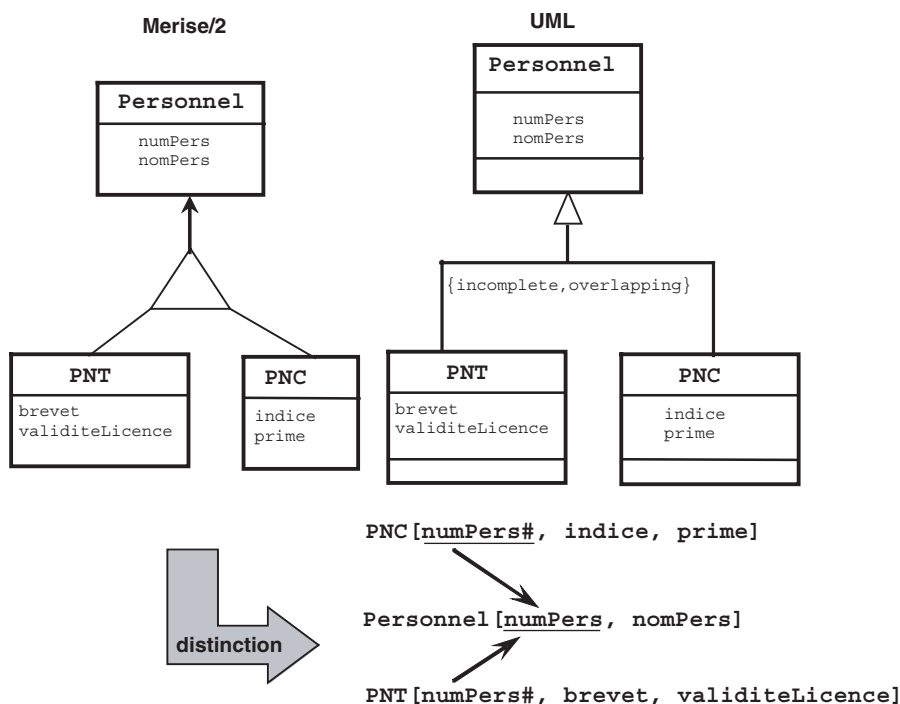
Nous expliquerons dans un premier temps la traduction SQL2 des associations d'héritage en fonction de la décomposition choisie (chapitre 2, section *Héritage*). Nous détaillerons ensuite la traduction d'éventuelles contraintes (partition, totalité et exclusivité).

Nous avons recensé au chapitre précédent trois familles de décomposition au niveau logique pour traduire une association d'héritage : décomposition par distinction, descendante (*push-down*) et ascendante (*push-up*).

Décomposition par distinction

L'héritage 3-14 (sans contrainte avec Merise/2 se modélisant par une contrainte UML) est traduit suivant le principe de décomposition par distinction. La clé primaire issue de la sur-entité (sur-classe) est dupliquée dans les deux tables déduites des sous-entités (sous-classes).

Figure 3-14 Décomposition par distinction d'une association d'héritage



Le schéma physique est composé de trois tables dotées de la même clé primaire. La table `Personnel` stockera notamment les personnels n'étant ni `PNT` ni `PNC`.

Tableau 3.12 Héritage par distinction

Schéma logique	Script SQL2
<p>PNC[<u>numPers#</u>, indice, prime]</p> <p>Personnel[<u>numPers</u>, nomPers]</p> <p>PNT[<u>numPers#</u>, brevet, validiteLicence]</p>	<pre>CREATE TABLE Personnel (numPers NUMBER, nomPers VARCHAR(20), CONSTRAINT pk_Personnel PRIMARY KEY(numPers)) CREATE TABLE PNC (numPers NUMBER, indice NUMBER, prime NUMBER(8,2), CONSTRAINT pk_PNC PRIMARY KEY(numPers), CONSTRAINT fk_PNC_Personnel FOREIGN KEY(numPers) ➔REFERENCES Personnel(numPers)) CREATE TABLE PNT (numPers NUMBER, brevet VARCHAR(10), validiteLicence DATE, CONSTRAINT pk_PNT PRIMARY KEY(numPers), CONSTRAINT fk_PNT_Personnel FOREIGN KEY(numPers) ➔REFERENCES Personnel(numPers))</pre>

Décomposition descendante (push-down)

L'héritage de partition 3-15 exprime le fait qu'aucun personnel ne peut être à la fois PNT et PNC, et qu'il n'existe pas non plus de personnel n'étant ni PNT ni PNC. Nous verrons plus loin comment traduire cette contrainte. Il est question ici uniquement de traduire les relations déduites au niveau logique en tables.

Figure 3-15 Exemple de décomposition descendante (push-down) d'une association d'héritage

