Bases de Données - M1 Info

C5 - UML et SQL 2/3

Lina Soualmia

Université de Rouen
LITIS - Équipe TIBS-CISMeF
lina.soualmia@chu-rouen.fr

30 septembre 2015

Plan

- Introduction
- Partie 1 : de UML à SQL2
  (du conceptuel au relationnel étendu – objet-relationnel)
- Partie 2 : de UML à SQL3
  (du conceptuel à l'orienté objet)
- Conclusion

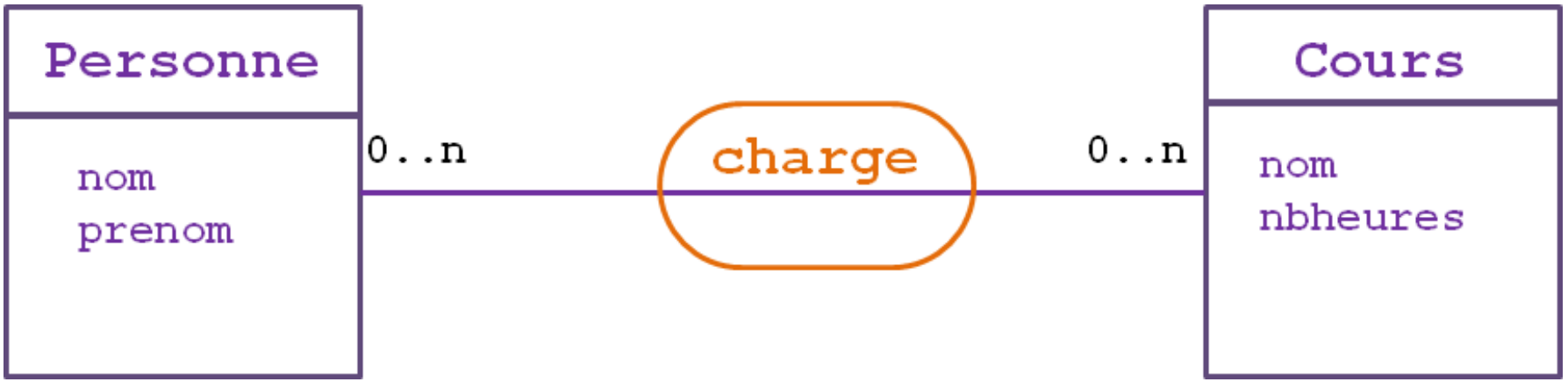
Introduction

1. Étape conceptuelle : Conception et Modélisation de bases de données
2. Étape logique : Implantation d'une base de données
3. Étape physique
4. Logiciels (SGBD, Interfaces, ...) & Matériels

Modélisation de Bases de Données

- Phase d'analyse : définition d'un schéma conceptuel
- Schéma Conceptuel de Données (SCD) : selon le formalisme utilisé
  - ensemble d'Entités et d'Associations
  - ou ensemble de Classes

- Formalisme EA, ER :



- Formalisme UML :

Formalismes

Différents formalismes de modélisation de schémas conceptuels de BD :

- Formalisme EA, ER, EER
  - Modèle Entité-Association (*Entity-Relationship Model*)
  - Modèle Entité-Association Etendu *(Extended Entity-Relationship Model)*
- Formalisme UML (*Unified Modelling Language*)

Modèle Entité-Association

- Entité :
  - tout concept concret ou abstrait individualisable
- Classe ou Type d'Entités :
  - regroupement d'entités de même nature (niveau générique)
- Association :
  - relation liant plusieurs entités
- Classe ou Type d'Associations :
  - regroupement d'associations présentant les mêmes caractéristiques

Modèle Entité-Association Étendu

- Modèle Entité-Association :
  - jeu de concept réduit mais suffisant pour la modélisation de problèmes simples (ou peu complexes)
- Modèle Entité-Association Étendu :
  - Modélisation plus précise et plus expressive de problèmes complexes et de grande taille
  - Introduction de mécanismes d'abstraction
    - de classification
    - d'héritage
    - d'agrégation

Mécanismes d'abstraction

Types Faibles

Type d'entités ou d'associations faibles : existence d'une instance subordonnée à l'existence d'un autre type d'entité ou d'association

Mécanismes d'abstraction

Classification

- Regroupement d'entités dans des classes en fonction de propriétés communes
- Possibilité de classer un objet dans plusieurs classes

Exemples :

- Livre électronique : fichier électronique, et livre
- Autocar : véhicule de transport en commun, véhicule à moteur

Mécanismes d'abstraction

Héritage Spécialisation - Généralisation

Un type d'entité A est une spécialisation d'un autre type d'entité B si :

- chaque entité de A est une entité de B
- une seule entité (au plus) de B est associée à une entité de A

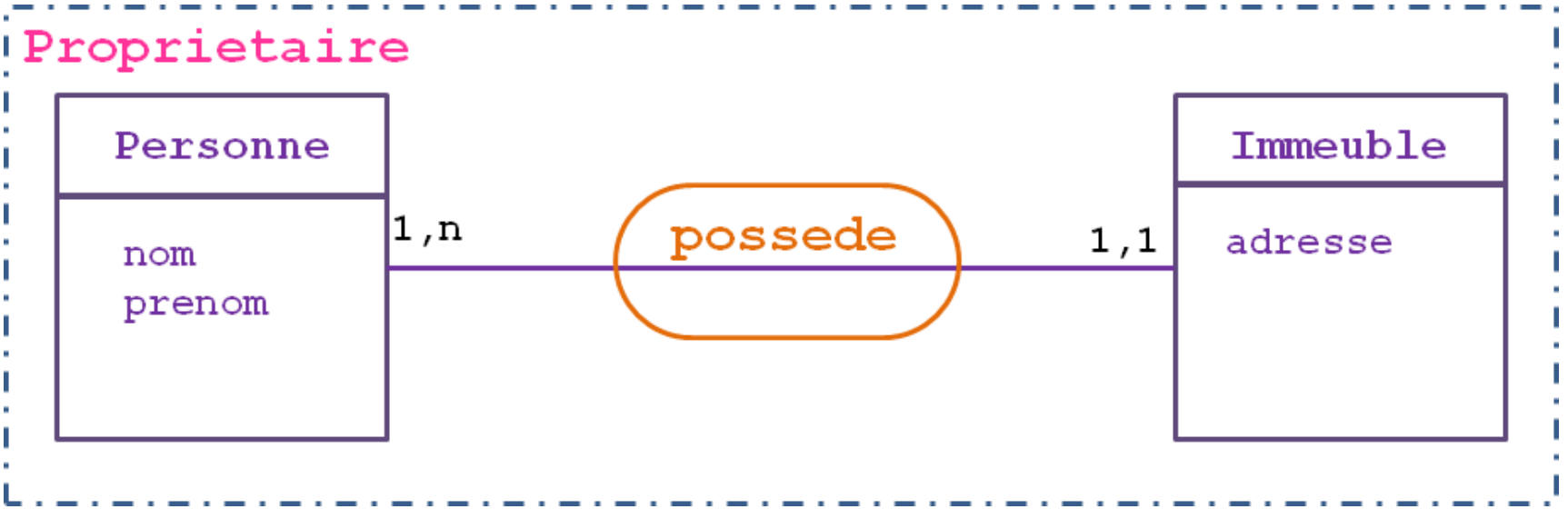
Exemple Médecin et Personnel de santé

Mécanismes d'abstraction

Agrégation :

Description de types d'entités complexes :

- Un type d'associations entre types d'entités est considéré comme un nouveau type d'entités


Proprietaire
Personne
nom
prenom
1,n
possede
1,1
Immeuble
adresse

Les deux formalismes E/R et UML sont très proches

Entité Association	UML
Entité	Objet
Type d'entité	Classe
Relation	Objet
Type d'association	Classe
Attribut/Propriété	Propriété
Rôle / Label	Rôle
	Méthode
Domaine	Contrainte de domaine
Clé	Contrainte de clé
Contrainte	Contrainte
Cardinalité	Multiplicité/Cardinalité
(0,1) (1,1) (0,n) (1,n) (a,b) (a,a)	(0..1) (1) (0..*) (1..*) (a..b) (a)
Diagramme E/A	Diagramme de Classe UML

Traduction EA vers SQL

Objectifs

- Implantation d'un schéma conceptuel (SCD) dans une BD relationnelle
- Exploitation du SCD par le SGBD et les modules de programmation
- Transformation dans un schéma relationnel : Schéma Logique de Données (SLD)

Exemple de schéma conceptuel EA et schéma logique correspondant

Règles de passage

- Tout type d'entité E est traduit en une relation R
  - La clé primaire de R est l'identifiant de E
  - Les attributs de R sont ceux de E
- Tout type d'association est traduit :
  - en *une clé étrangère* dans une relation existante si la cardinalité est du type *1,1* ou *0,1*
  - en une *nouvelle relation* si aucune cardinalité n'est du type 1,1 ou 0,1 (elles sont toutes du type *0,n* ou *1,n*)

Plusieurs algorithmes sont possibles selon l'interprétation de la cardinalité minimale égale à 0.

Traduction de UML vers SQL

- Traduction des associations binaires
- Traduction des associations binaires récursives
- Traduction des associations n-aires (n $>$ 2)
- Traduction des associations d'héritage
- Traduction des contraintes d'héritage
- Traduction des associations d'agrégation
- Traduction des contraintes d'intégrité fonctionnelles (contraintes : Partition, Exclusion, Totalité, Simultanéité, Inclusion)

Intégrité des Données

Les SGBD prennent en compte l'intégrité des données définies via :

- la déclaration de contraintes (**`constraint`**)
- la programmation de
  - fonctions (**`functions`**)
  - de procédures (**`procedures`**) cataloguées
  - de paquetages (**`packages`**)
  - de déclencheurs (**`triggers`**)

Le principe étant d'assurer la cohérence de la base après chaque mise à jour par les commandes ***`insert, update, delete`***

Notations pour les noms des contraintes

- la contrainte *clé primaire* d'une table se nomme `PKTable`
- la contrainte *clé étrangère* d'une table se nomme `FKTable1ColonneTable2`
- la contrainte de *validité d'une colonne* se nomme `CKTableColonne`
- la contrainte de type *non nulle* sur une colonne se nomme `NNTableColonne`
- la contrainte de type *unique* sur une colonne se nomme `UnTableColonne`

Association 1-1 (1)

Un stage est effectué par au plus 1 étudiant

```
create table STAGE
(NumStage NUMBER(2),
NomEntreprise VARCHAR(40),
TelEntreprise VARCHAR(15),
AdrEntreprise VARCHAR(50),
constraint PKStage  primary key (NumStage));
```

Association 1-1 (2)

Un étudiant effectue obligatoirement un stage unique

```
create table ETUDIANT
(NumEtudiant NUMBER(7),
NomEtudiant VARCHAR(10),
DateNaissance DATE,
Genre CHAR (1),
NumStage NUMBER(7),
constraint PKEtudiant primary key (NumEtudiant),
constraint FKEtudiantNumStageStage foreign key
(NumStage)references STAGE(NumStage),
constraint CKEtudiantGenre check (Genre in('M','F')),
constraint NNEtudiantNumStage check (NumStage  IS NOT
NULL),
constraint UnEtudiantNumStage UNIQUE (NumStage));
```

Association du type 1-N (1)

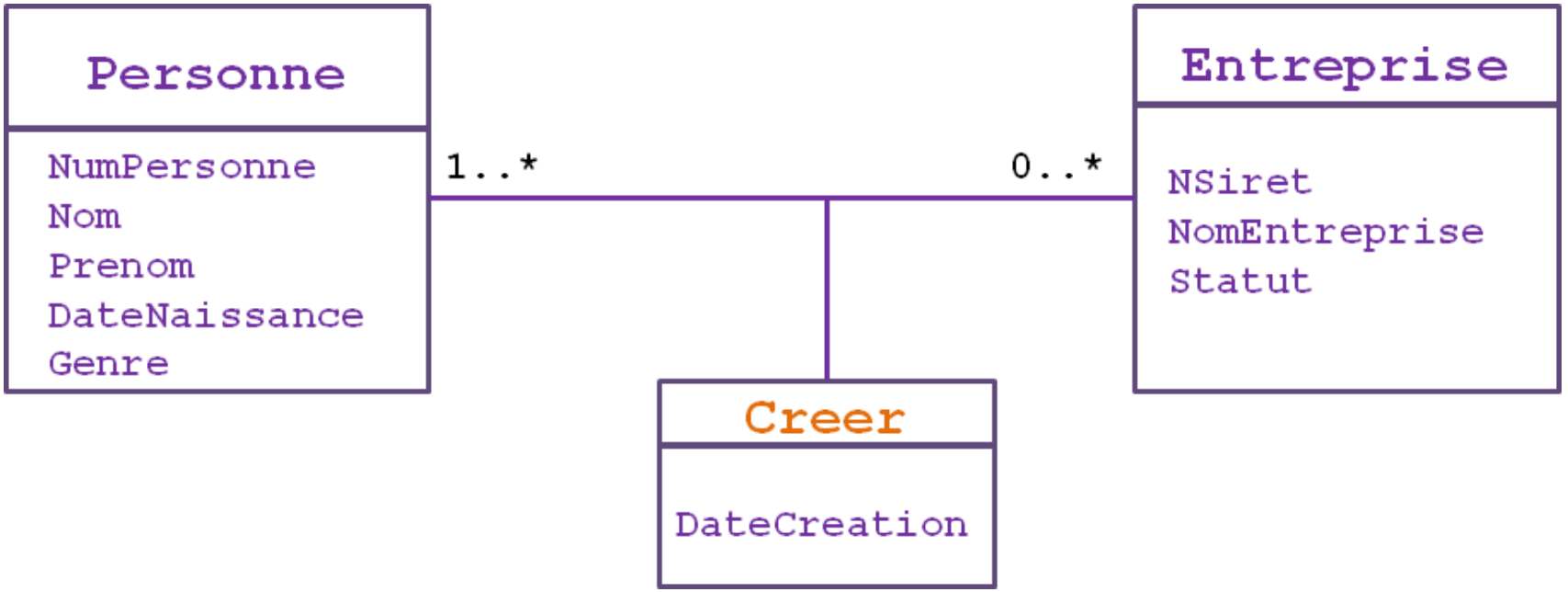
Une personne peut posséder plusieurs voitures

```
create table PERSONNE
(NumPersonne NUMBER(7),
Nom VARCHAR(10),
Prenom VARCHAR(10),
DateNaissance DATE,
Genre CHAR(1),
constraint  PKPersonne  primary key (NumPersonne));
```

Association de type 1-N (2)

Une voiture appartient obligatoirement à une personne

```
create table VOITURE
(NumImmat VARCHAR(15),
Marque VARCHAR(20),
Type VARCHAR(30),
NumPersonne NUMBER(7),
constraint PKVoiture primary key (NumImmat),
constraint FKVoitureNumPersonnePersonne foreign key
(NumPersonne) references PERSONNE(NumPersonne),
constraint NNVoitureNumePersonne check (NumPersonne
is NOT NULL);
```


Personne
NumPersonne
Nom
Prenom
DateNaissance
Genre
1..*
0..*
Entreprise
NSiret
NomEntreprise
Statut
Creer
DateCreation

Association de type N-N (1)

Une personne peut créer plusieurs entreprises

```
create table PERSONNE
(NumPersonne NUMBER(7),
Nom VARCHAR(10),
Prenom VARCHAR(10),
DateNaissance DATE,
Genre CHAR(1),
constraint  PKPersonne  primary key (NumPersonne));
```

Association de type N-N (2)

Une entreprise doit être créée par une ou plusieurs personnes

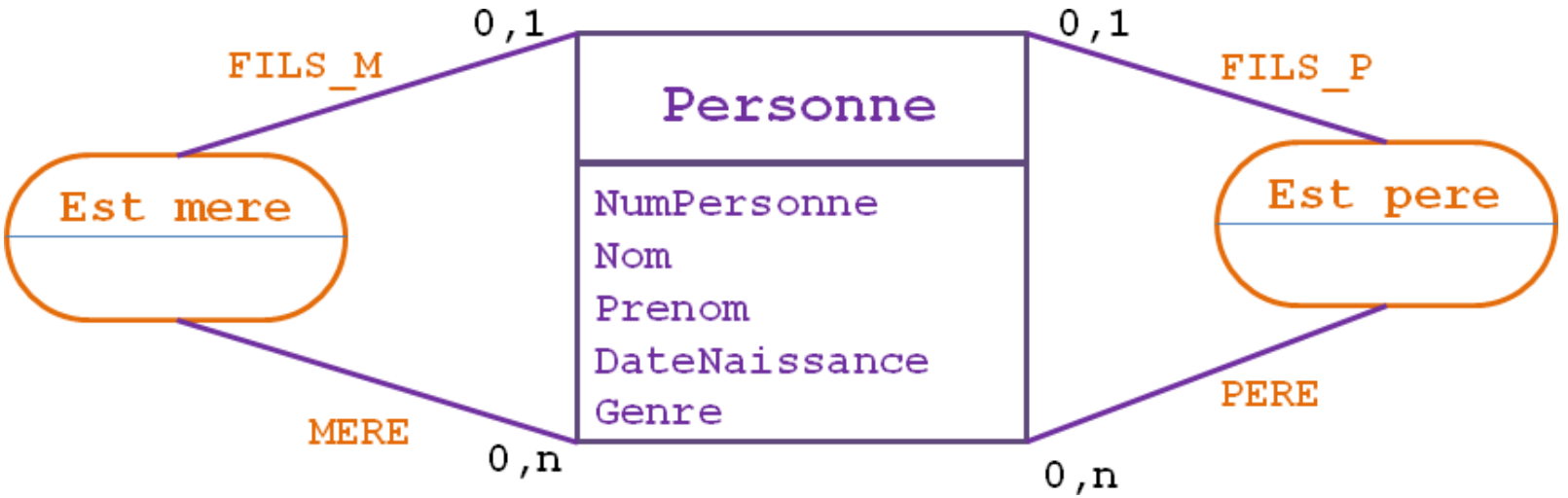
```
create table ENTREPRISE
(NumSiret VARCHAR(20),
NomEntreprise VARCHAR(20),
Statut VARCHAR(10),
constraint  PKEntreprise  primary key (NumSiret));
```

```
create table CREER
(NumPersonne NUMBER(7),
NumSiret VARCHAR(10),
DateCreation DATE,
constraint PKPersonne primary key (NumPersonne,
NumSiret),
constraint FKCreerNumPersonnePersonne foreign key
(NumPersonne) references PERSONNE(NumPersonne),
constraint FKCreerNumiretSiret foreign key
(NumSiret) references ENTREPRISE(NumSiret));
```

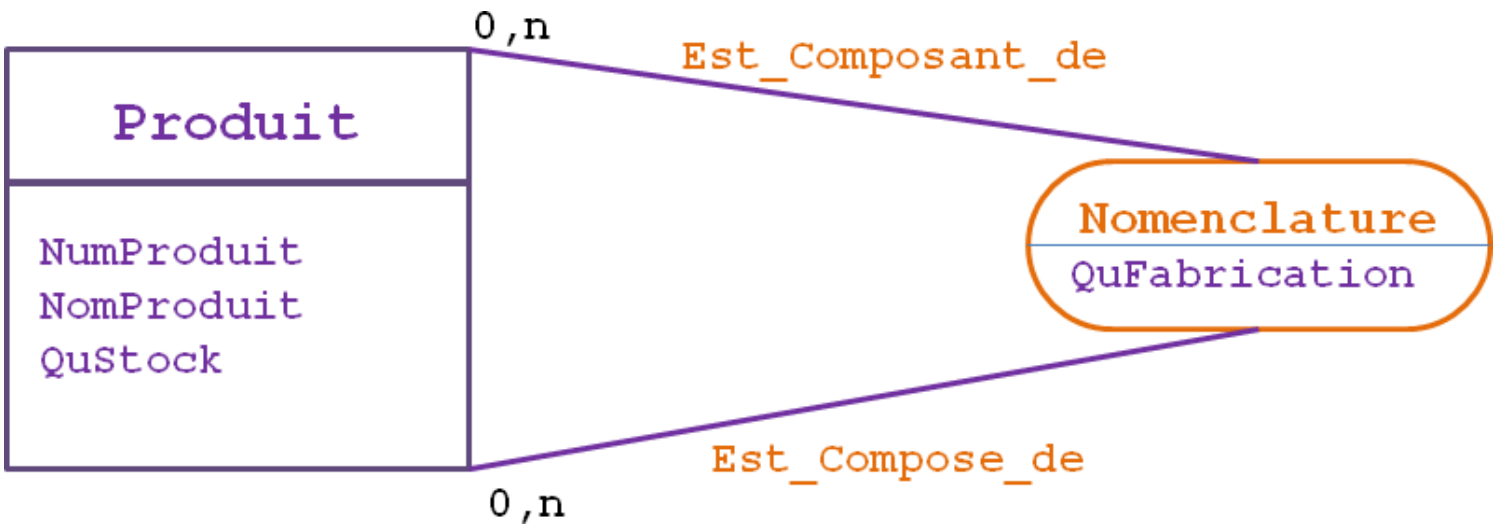
Associaton de type N-N (3)

La cardinalité minimale de l'association créée pourra être testée par une procédure PL/SQL

Association du type Réflexif/Récursif



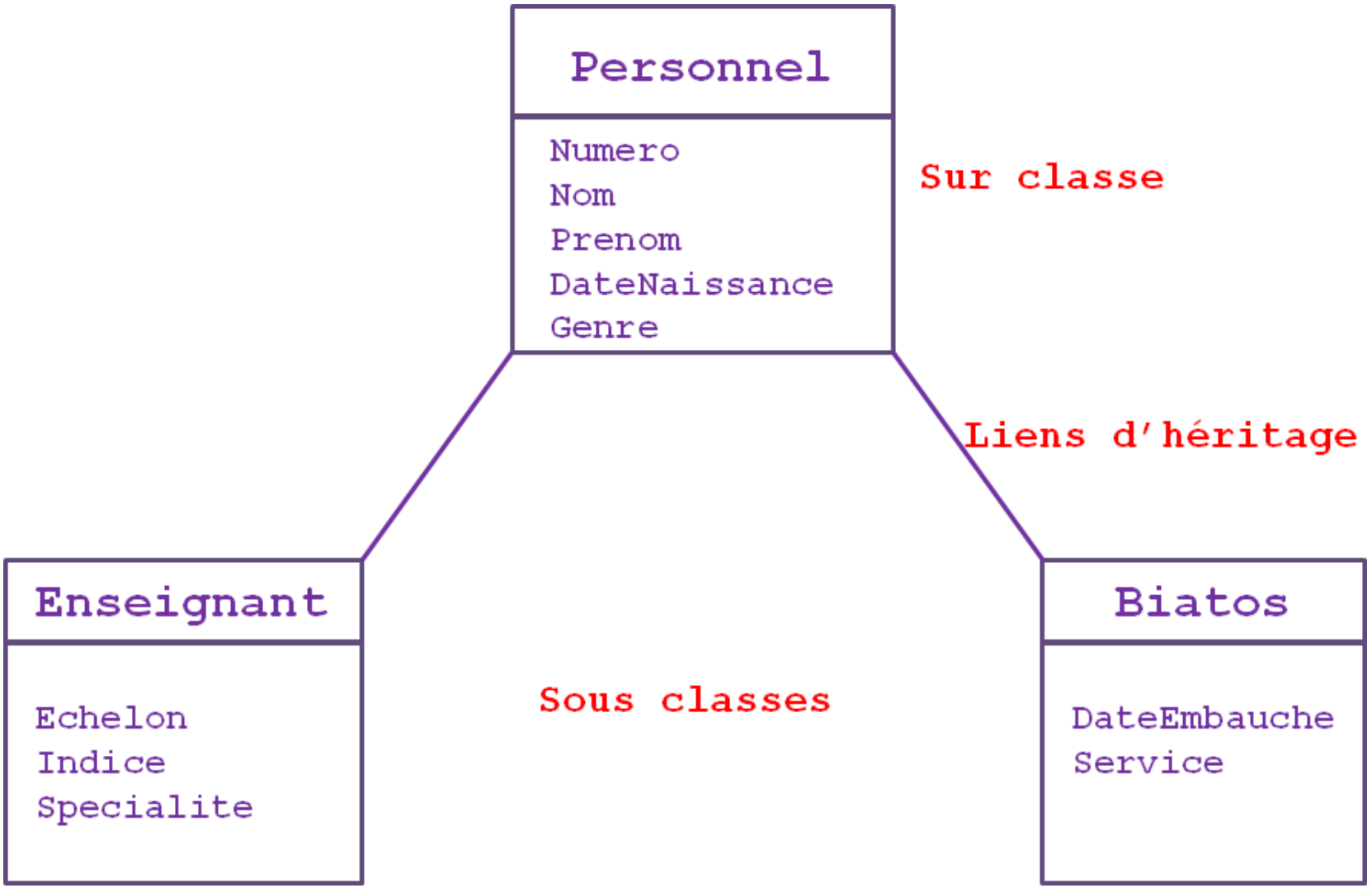
```
create table PERSONNES
(NumPersonne NUMBER(7),
Nom VARCHAR(15),
Prenom VARCHAR(15),
DateNaissance DATE,
Genre CHAR(1),
Pere NUMBER(7),
Mere NUMBER(7),
constraint PKPersonnes primary key NumPersonne,
constraint FKPersonnesPerePersonnes foreign key
(Pere) references PERSONNES,
constraint FKPersonnesMerePersonnes foreign key
(Mere) references PERSONNES,
constraint CKPersonnesGenre check (Genre
in('M','F')));
```


0,n
Est_Composant_de
Produit
NumProduit
NomProduit
QuStock
Nomenclature
QuFabrication
Est_Compose_de
0,n

```
create table PRODUITS
(NumProduit NUMBER(3),
NomProduit VARCHAR(15),
QuStock NUMBER(5),
constraint  PKProduits  primary key NumProduit,
constraint CKProduitsQuStock check (QuStock>=0));
```

```
create table NOMENCLATURE
(Composer NUMBER(3),
Composant NUMBER(3),
QuFabrication NUMBER(5),
constraint  PKNomenclature  primary key
(Composer,Composant),
constraint  FKNomenclatureComposerProduits  foreign
key (Composer) references  PRODUITS,
constraint CKNomenclatureQuFabrication check
(QuFabrication>=0));
```

Gestion du personnel dans une université



Associations d'héritage dans UML (1)

- Recensement des différents cas d'héritage en fonction des instances
- Modélisation des différents héritages, dans le formalisme UML, à l'aide de contraintes

Associations d'héritage dans UML (1)

- Expression des cas d'héritage à l'aide de
  - couverture
  - disjonction d'instances dans une population donnée
- Modélisation des différents héritages, dans le formalisme UML, à l'aide de contraintes
  - partition
  - totalité
  - exclusion
  - absence de contraintes

Contraintes d'héritage : Partition et Totalité

- Disjonction ET Couverture $\Rightarrow$ Partition
- Non-Disjonction ET Couverture $\Rightarrow$ Totalité

Contraintes d'héritage : Exclusion et Absence de contrainte

- Disjonction ET Non-Couverture ⇒ Exclusion
- Non-Disjonction ET Non-Couverture ⇒ Absence de contrainte

Contraintes d'héritage - PARTITION

- Disjonction & Couverture $\longrightarrow$ Partition
- Exemple :
  - Personnel (P) est égal à l'Union de Enseignant (E) et de BIATOS (B) et l'intersection de E et de B est vide

Contraintes d'héritage - TOTALITÉ

- Couverture & Non-Disjonction $\longrightarrow$Totalité
- Exemple :
  - Personnel (P) est égal à l'Union de Enseignant (E) et de Biatoss (B) et l'Intersection de E et de B n'est pas Vide

Contraintes d'héritage - EXCLUSION

- Disjonction & Non-Couverture $\longrightarrow$ Exclusion
- Exemple :
  - L'Union de Enseignant (E) et de Biatoss (B) est incluse dans Personnel (P) et l'Intersection de E et de B est Vide

Contraintes d'héritage ABSENCE DE CONTRAINTES

- Non-Couverture & Non-Disjonction $\longrightarrow$ Absence de contraintes
- Exemple :
  - L'Union de Enseignant (E) et de BIATOS (B) est incluse dans Personnel (P) et l'Intersection de E et de B n'est pas Vide

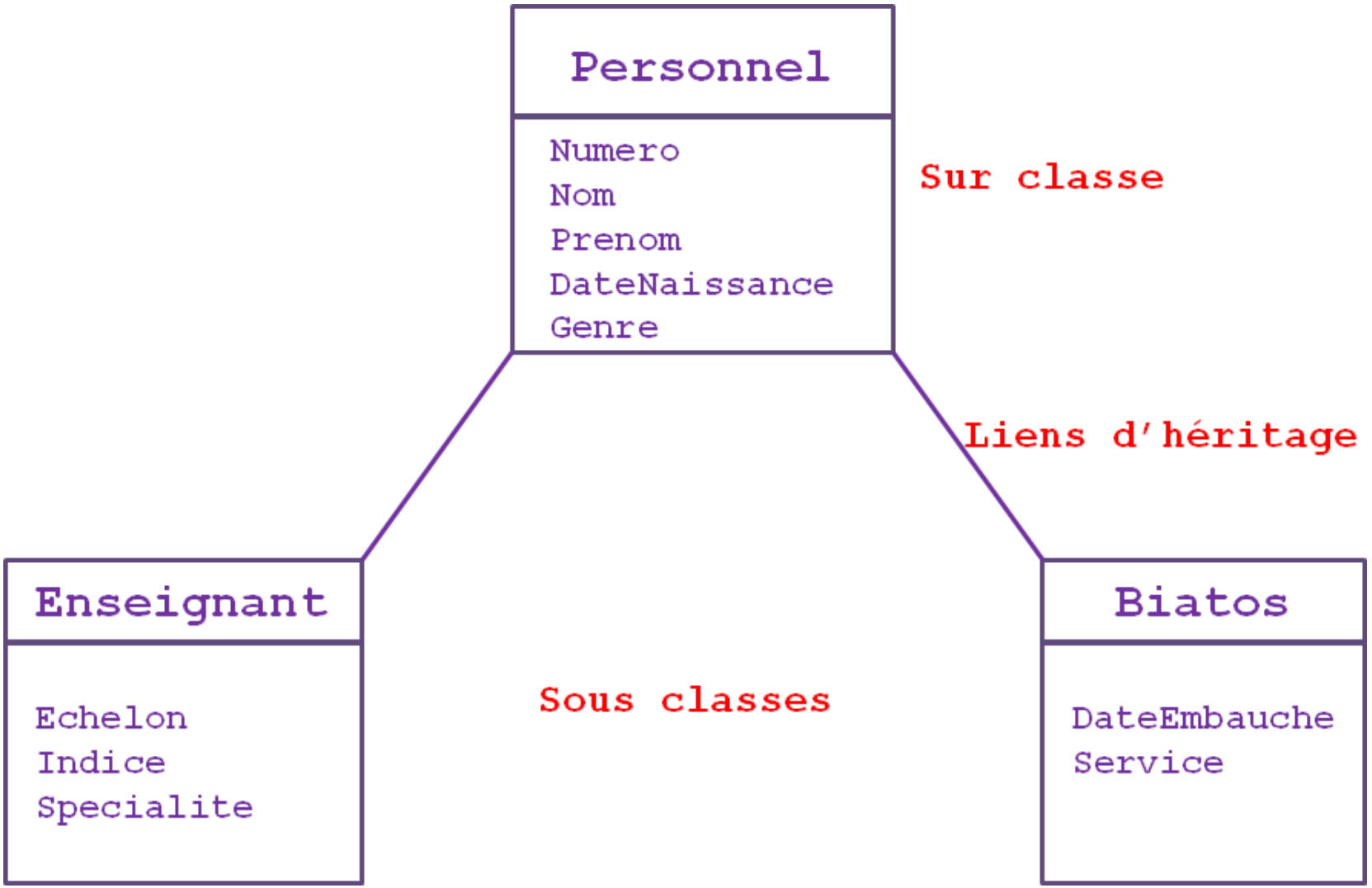
Transformation des associations d'héritage

La traduction d'une association d'héritage se fait en fonction des contraintes de l'association d'héritage
Il y a 3 familles de décomposition :

- Décomposition par *distinction*
- Décomposition *descendante* (push-down)
- Décomposition *ascendante* (push-up)

Décomposition par distinction

- Transformation de chaque sous-classe en une relation
- Migration de la clé primaire de la sur-classe dans la ou les relations issues des sous-classes
- La clé primaire de la sur-classe devient à la fois clé primaire et clé étrangère


Personnel
Numero
Nom
Prenom
DateNaissance
Genre
Sur classe
Liens d'héritage
Enseignant
Echelon
Indice
Specialite
Sous classes
Biatos
DateEmbauche
Service

Exemple - Distinction

PERSONNEL(Numero, Nom, Prénom, DateNaissance, Genre)
ENSEIGNANT(Numero, Echelon, Indice, Specialite)
BIATOS(Numero, DateEmbauche, Service)

Décomposition descendante

Deux cas possibles selon la contrainte d'héritage :

- contrainte de totalité ou de partition sur l'association :
  - possibilité de ne pas traduire la relation issue de la sur-classe
  - migration de tous les attributs dans la ou les relations issues de la ou des sous-classes
- Sinon :
  - migration de tous les attributs dans la ou les reations issues de la ou des sous-classes
  - duplication des données

Décomposition descendante - Exemple

Contrainte de partition :

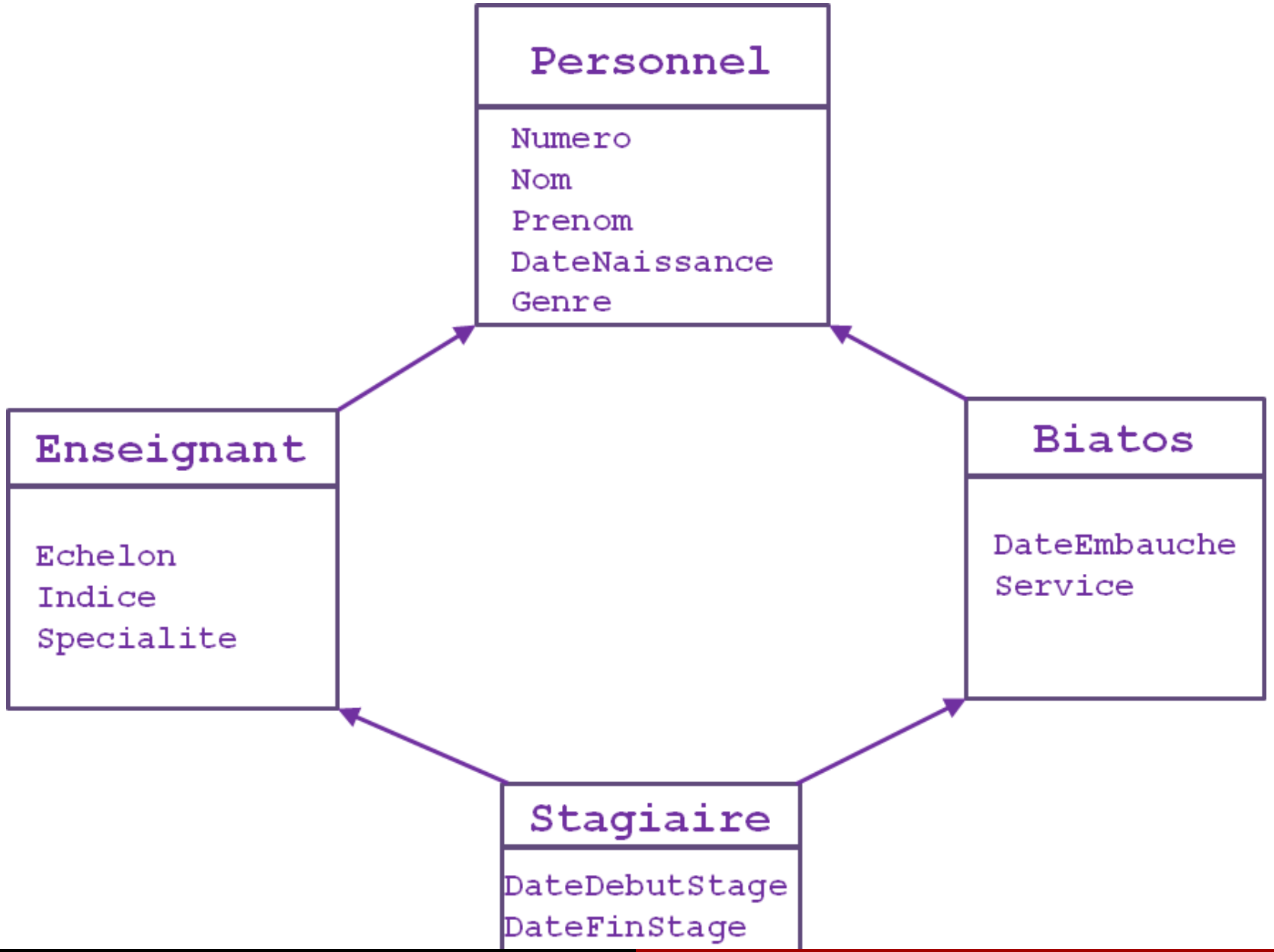
- Aucun Personnel ne peut être à la fois Enseignant et BIATOS
- Il n'existe pas non plus un Personnel n'étant ni Enseignant ni Biatos

`ENSEIGNANT(Numéro, Nom, Prenom, DateNaissance, Genre, Echelon, Indice, Specialite)`
`BIATOS(Numero, Nom, Prénom, DateNaissance, Genre, DateEmbauche, Service)`

Décomposition ascendante

- Supression de la ou des relations issues de la ou des sous-classes
- Migration des attributs dans la relation issue de la sous-classe
- Exemple : (absence de contrainte)

`PERSONNEL(Numero, Nom, Prenom, DateNaissance, Genre, Echelon, Indice, Specialite, DateEmbauche, Service)`


Personnel
Numero
Nom
Prenom
DateNaissance
Genre
Enseignant
Echelon
Indice
Specialite
Biatos
DateEmbauche
Service
Stagiaire
DateDebutStage
DateFinStage

Héritage multiple

- Les mêmes règles sont appliquées avec plusieurs possibilités
- Exemple : décomposition ascendante avec une contrainte d'exclusion sur Enseignant et Biatoss

```
PERSONNEL(Numero, Nom, Prenom, DateNaissance, Genre
ENSEIGNANT(Numero, Echelon, Indice, Specialite,
DateDebutStage, DateFinStage)
BIATOS(Numero, DateEmbauche, Service, DateDebutStage,
DateFinStage)
```

Passage au SQL2 - Décomposition par distinction

PERSONNEL(Numero, Nom, Prenom, DateNaissance, Genre)
ENSEIGNANT(Numero, Echelon, Indice, Specialite)
BIATOS(Numero, DateEmbauche, Service)

un personnel de l'université

```
create table PERSONNEL
(Numero NUMBER(7)
Nom VARCHAR(10)
Prenom VARCHAR(10)
DateNaissanceDATE
Genre CHAR(1),
constraint PKPersonnel primary key (Numero)
constraint CKGenrePersonnel check (Genre in
('M','F')));
```

le personnel Enseignant

```
create table ENSEIGNANT
(Numero NUMBER(7),
Echelon NUMBER(2),
Indice NUMBER(5),
Specialite VARCHAR(20),
constraint  PKEnseignant  primary key (Numero)
constraint  FKEnseigantPersonnel  foreign key (Numero)
references  PERSONNEL);
```

le personnel Biatos

```
create table BIATOS
(Numero NUMBER(7),
DateEmbauche DATE
Service VARCHAR(20),
constraint  PKBiatos  primary key (Numero)
constraint  FKBiatosPersonnel  foreign key (Numero)
references  PERSONNEL);
```

Passage au SQL2 - Décomposition descendante

ENSEIGNANT(Numero, Nom, Prenom, DateNaissance, Genre, Echelon, Indice, Specialite)
BIATOS(Numero, Nom, Prenom, DateNaissance, Genre, DateEmbauche, Service)

Personnel enseignant

```
create table ENSEIGNANT
(Numero NUMBER(7),
Nom VARCHAR(10),
Prenom VARCHAR(10),
DateNaissance DATE
Genre CHAR(1),
Echelon NUMBER(2),
Indice NUMBER(5),
Specialite VARCHAR(20),
constraint CKGenreEnseignant check (Genre in
('M','F')));
constraint PKEnseigant primary key (Numero));
```

Personnel Biatos

```
create table BIATOS
(Numero NUMBER(7),
Nom VARCHAR(10),
Prenom VARCHAR(10),
DateNaissance DATE
Genre CHAR(1),
DateEmbauche DATE
Service VARCHAR(20),
constraint CKGenreBiatos check (Genre in
('M','F')),
constraint PKBiatos primary key (Numero);
```

Passage au SQL2 - Décomposition ascendante

PERSONNEL(Numero, Nom, Prenom, DateNaissance, Genre, Echelon, Indice, Specialite, DateEmbauche, Service)

```
create table PERSONNEL
(Numero NUMBER(7),
Nom VARCHAR(10),
Prenom VARCHAR(10),
DateNaissance DATE
Genre CHAR(1),
Echelon NUMBER(2),
Indice NUMBER(5),
Specialite VARCHAR(20),
DateEmbauche DATE
Service VARCHAR(20),
constraint CKGenrePersonnel check (Genre in ('M','F')),
constraint PKPersonnel primary key (Numero);
```

Décomposition par distinction

- Contraintes :
  - $\longrightarrow$ Contrainte de partition
  - Contrainte de totalité
  - Contrainte d'exclusion
  - Sans contrainte
- Contraintes d'héritage :
  - Contrainte A : Il n'existe pas de personnel à la fois enseignant et BIATOS
  - Contrainte B : Il n'existe pas de personnel ni enseigant ni BIATOS

Décomposition par distinction - Contrainte de partition

- Contrainte A : Il n'existe pas de personnel à la fois enseignant et BIATOS $\longrightarrow$ 2 déclencheurs
  - Déclencheur sur Enseignant
  - Déclencheur sur BIATOS

```
create or replace trigger TrigEnseignant
before insert or update of Numero on ENSEIGNANT
for each row
declare
    num NUMBER;
begin
    select Numero into Num
    from BIATOS where Numero=:new.Numero;
   raise_application_error(-20001,'Le
personnel'||to_char(num)||'est déjà Biatos et ne peut
pas être Enseignant');
exception
    when no_data_found then NULL;
end;
```

```
create or replace trigger TrigBiatos
before insert or update of Numero on BIATOS
for each row
declare
    num NUMBER;
begin
    select Numero into Num
    from ENSEIGNANT where Numero=:new.Numero;
    raise_application_error(-20001,'Le
personnel'||to_char(num)||'est déjà Enseigant et ne
peut pas être Biatos');
exception
    when no_data_found then NULL;
end;
```

Décomposition par distinction - Contrainte de partition

- Contrainte B : Il n'existe pas de personnel ni enseignant ni Biatos
  - procédures cataloguées pour l'Insertion et la Suppression
  - déclencheurs pour la Modification

Ajout d'un enseignant

```
create or replace procedure AjoutEnseignant
(Num NUMBER,Nom VACHAR,Prenom VACHAR,DateNaiss
DATE,Genre CHAREchelon NUMBER,Indice
NUMBER,Specialite VACHAR) is
begin
    insert into PERSONNEL values
(Num,Nom,Prenom,DateNaiss,Genre);
    insert into ENSEIGNANT values
(Echelon,Indice,Specialite);
end;
```

Ajout d'un biatos

```
create or replace procedure AjoutBiatos
(Num NUMBER,Nom VACHAR,Prenom VACHAR,DateNaiss
DATE,Genre CHAR,DateEmb DATE,Service VACHAR) is
begin
    insert into PERSONNEL values
(Num,Nom,Prenom,DateNaiss,Genre);
    insert into BIATOS values (Num,DateEmb,Service);
end;
```

Suppression d'un enseignant

```
create or replace procedure SupprEnseignant
(Num NUMBER) is
begin
    delete from ENSEIGNANT where Numero=Num;
    delete from PERSONNEL where Numero=Num;
end;
```

Suppression d'un biatos

```
create or replace procedure SupprBiatos
(Num NUMBER) is
begin
    delete from BIATOS where Numero=Num;
    delete from PERSONNEL where Numero=Num;
end;
```

Déclencheur pour la répercussion de la modification du numéro PERSONNEL vers ENSEIGNANT et BIATOS

```
create or replace trigger TrigEnseigantBiatos

before update of Numero on PERSONNEL

for each row

begin

    begin

        update ENSEIGNANT set Numero=:new.Numero where Numero=:old.Numero;

    exception

        when no_data_found then NULL;

    end;

    begin

        update BIATOS set Numero=:new.Numero where Numero=:old.Numero;

    exception

        when no_data_found then NULL;

    end;
end;
```

Utilisation

Insertion des données sous SQLPLUS et lancement des procédures

```
execute
AjoutEnseigant(1,'nom1','prenom1','12-09-1968','M',6,780,'Réseau
execute
AjoutEnseigant(2,'nom2','prenom2','14-03-1967','F',6,780,'Compil
execute
AjoutBiatos(3,'nom3','prenom3','05-04-1968','M','01-09-2005','Co
```

Décomposition par distinction - Contrainte de totalité

- Contraintes d'héritage :
  - Contrainte B : Il n'existe pas de personnel ni Enseignant ni Biatos
  - Contrainte C : Il peut exister un personnel à la fois Enseignant et Biatos
- Implémentation :
  - Contrainte B (voir ci dessus) :
    - procédures cataloguées pour l'Insertion et la Suppression
    - déclencheurs pour la Modification
  - Contrainte C :
    - équivaut à ne pas programmer la contrainte A précédente
    - pas de mise en oeuvre des déclencheurs des tables Enseignant et Biatos `TrigEnseignant` et `TrigBiatos`

Décomposition par distinction - Contrainte d'exclusion

- Contraintes d'héritage :
  - Contrainte A : Il n'existe pas de personnel à la fois Enseignant et Biatos
  - Contrainte D : Il peut exister un personnel ni Enseignant ni Biatos
- Implémentation :
  - Contrainte A (voir ci dessus) :
    - déclencheurs sur Enseignant et sur Biatos
  - Contrainte D :
    - équivaut à ne pas programmer la contrainte B précédente
    - pas de mise en oeuvre des 4 procédures (ajout et supression) et du déclencheur `TrigEnseignantBiatos`

Décomposition par distinction - Sans contrainte

Aucune contrainte n'est à programmer

Décomposition descendante

- Contrainte de partition :
  - aucun personnel ne peut être à la fois enseignant et biatos
  - il n'existe pas non plus un personnel n'étant ni personnel ni biatos
- Contrainte de totalité
- Contrainte d'exclusion : il faudra garder la table personnel pour les personnels non enseignant et non Biatos

Décomposition ascendante - Contrainte de partition

- Contraintes d'héritage :
  - Contrainte A : il n'existe pas de personnel à la fois enseignant et Biatos
  - Contrainte B : il n'existe pas de personnel ni enseignant ni Biatos
- Implémentation des contraintes A et B :
  - au niveau de la table `PERSONNEL`
  - à l'aide des contraintes de type **`check`**

Décomposition ascendante - contrainte de partition

- Contrainte A :
  - Vérifier que les colonnes `Echelon, Indice, Specialite, DateEmbauche` et `Service` ne soient pas toutes initialisées
- Contrainte B
  - Vérifier que les colonnes `Echelon, Indice, Specialite, DateEmbauche` et `Service` ne soient pas toutes nulles

Contrainte A :

```
alter table PERSONNEL add constraint CKContrainteA
check ((Echelon is NULL andIndice is NULL and
Specialite is NULL)
or (DateEmbauche is NULL and Service is NULL));
```

Contrainte B

```
alter table PERSONNEL add constraint CKContrainteB
check ((Echelon is NOT NULL or Indice is NOT NULL or
Specialite is NOT NULL)
or (DateEmbauche is NOT NULL or Service is NOT
NULL));
```

Décomposition ascendante - contrainte de totalité

Contraintes d'héritage :

- Contrainte B :
  - il n'existe pas de personnel ni Enseignant ni Biatos
  - Vérifier que les colonnes `Echelon, Indice, Specialite, DateEmbauche` et `Service` ne soient pas toutes initialisées
- Contrainte C :
  - Il peut exister un personnel à la fois enseignant et biatos
  - Supprimer **`(drop constraint)`** ou désactiver **`(disable constraint)`** la contrainte A précédente :
  - si suppression : en cas de récativation il est nécessaire de la recréer **`(add constraint)`**
  - si désactivation : en cas de réactivation utiliser la commande **`enable constraint`**

```
alter table PERSONNEL disable constraint CKContrainteA;
```

Décomposition ascendante - contrainte d'exclusion

Contraintes d'héritage :

- Contrainte A :
  - Il n'existe pas de personnel à la fois Enseignant et Biatos
  - Réactivation de la contrainte A en supprimant au préalable les tuples ne répondant pas à cette contrainte
- Non contrainte B :
  - Il peut exister un personnel ni enseignant ni biatos
  - Désactivation de la contrainte B : `(disable constraint)`

Réactivation de la contrainte A :

```
alter table PERSONNEL enable constraint CKContrainteA;
```

Désactivation de la contrainte B :

```
alter table PERSONNEL disable constraint CKContrainteB;
```

Décomposition ascendante - sans contrainte

Aucune contrainte de type `check` n'est à programmer

Conclusion

Aucune des solutions ne constitue la panacée

Il faut mesurer les performances des requêtes

Voir aussi le type des requêtes