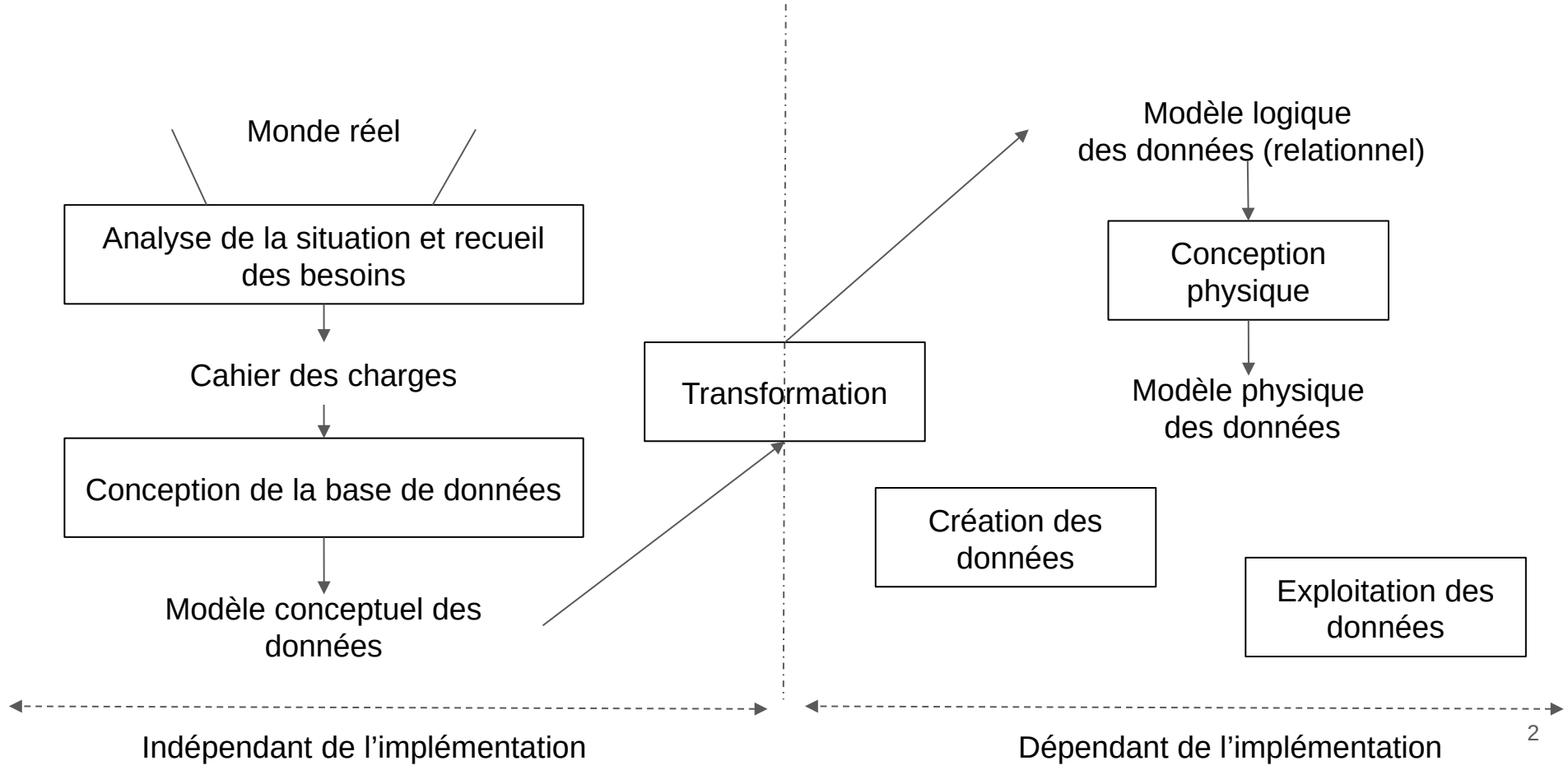
BDD 2

Modélisation et conception de base de données

Ioan Marius BILASCO et Maude PUPIN

certaines parties sont inspirées des supports de cours d’A. Etien et J.C. Marti

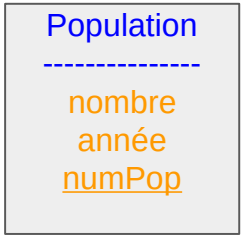
Conception et exploitation d'une base de données

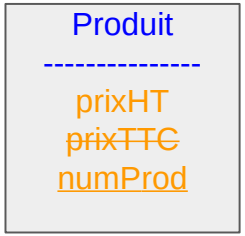


MCD - Entités (ou Classes) - Synthèse

- Correspondent à des concepts indépendants et importants pour le domaine visé
- Portent un nom permettant de les distinguer clairement dans l'écosystème
- Disposent d'une liste de propriétés (atomiques)
- Disposent *souvent* d'une ou plusieurs propriétés permettant de les identifier de manière unique

- Éviter autant que possible les dépendances fonctionnelles entre propriétés d'une ou plusieurs entités
  - Parfois, si le calcul de la propriété dérivée est long et il est fait souvent, on accepte certaines dépendances
    - ex : montant d'une commande incluant de nombreux produits



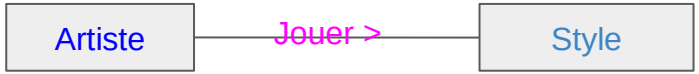


prixTTC = f(prixHT)

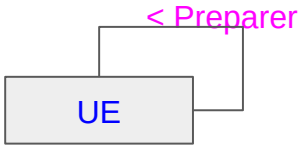
MCD - Relations (ou Associations)

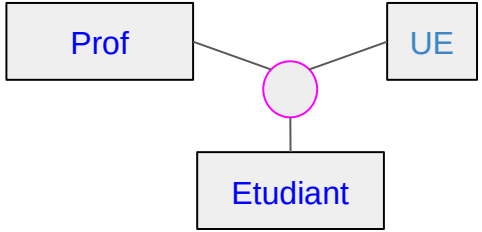
- Décrire des liens entre entités/classes
- Nom d'association - il est souhaitable de la nommer par un verbe à l'infinitif

- Éventuellement des propriétés atomiques caractérisent la mise en relation
- Associer des propriétés complexes (non-atomiques, contraintes) aux entités



- Arité : binaire, ternaire, n-aire
- Réflexivité

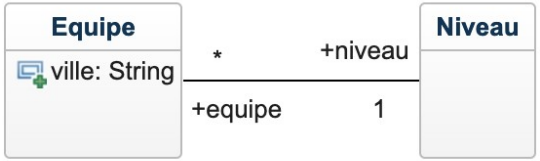




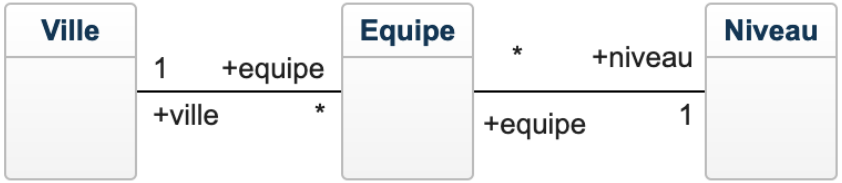
- Cardinalité : nombre d'instances mises en relation avec d'autres instances

MCD - Contraindre un ensemble de valeurs

- Explicite – catégories/types



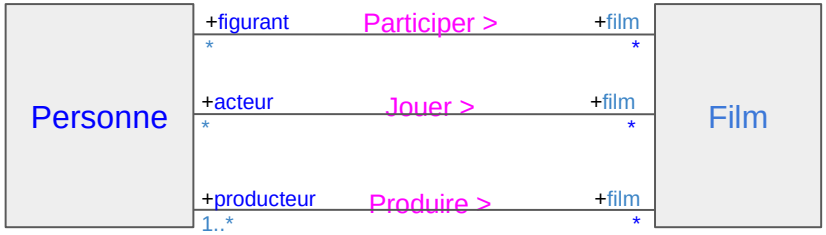
- Implicite
  - Dans cet exemple, comment normer l'écriture des noms de pays/villes ?



  - Parfois, si l'ensemble des valeurs est restreint on peut contraindre au niveau de la création de la table avec CHECK champ IN (val1, val2, ..., val9) (voir TP5)

MCD - Relations (ou Associations) - rôles

- Rôles en UML
  - Préciser le rôle joué par une instance d'une entité dans la relation
  - Utile lorsque l'entité participe à plusieurs relations
  - Utile lorsque la même entité participe plusieurs fois à la même relation



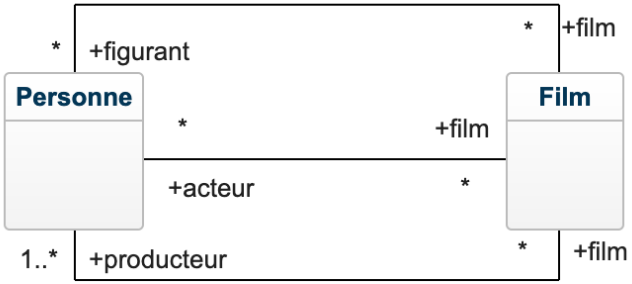
1 personne en tant que figurant participe à (*) 0 ou plusieurs films
Pour 1 film, on fait participer (*) 0 ou plusieurs figurants (personnes)

1 personne en tant qu'acteur joue dans (*) 0 ou plusieurs films
Pour 1 film, on fait jouer (*) 0 ou plusieurs acteurs (personnes)

1 personne en tant que producteur produit (*) 0 ou plusieurs films
1 film est produit par (1..*) 1 ou plusieurs producteurs (personnes)

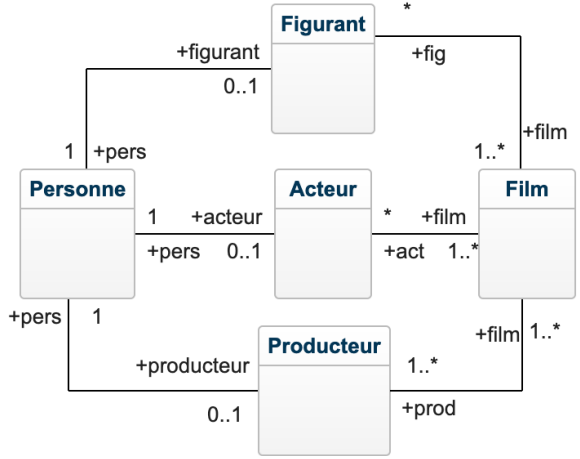
- Mais parfois, cette modélisation n'est pas suffisament précise

MCD - Rôles vs Spécialisation

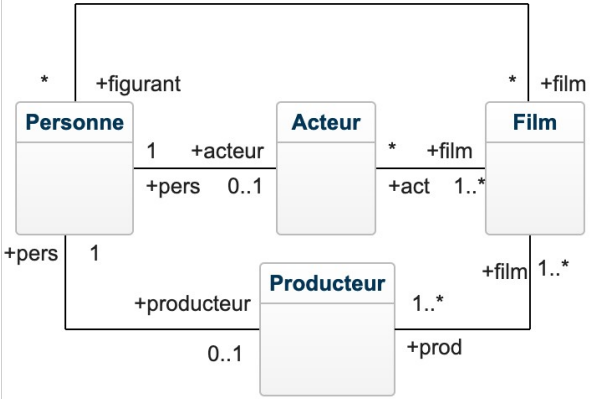


Cette solution suffit pour pouvoir réprésenter toutes les données, mais ne permet pas de distinguer entre propriétés spécifiques par rôle, ni d'éventuelles relations spécifique par rôle.

Un producteur ou un acteur peuvent recevoir une récompense, mais pas les figurants.

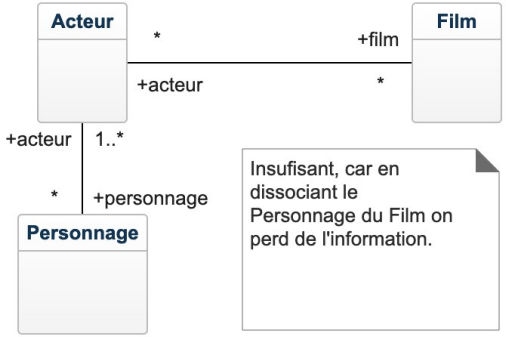


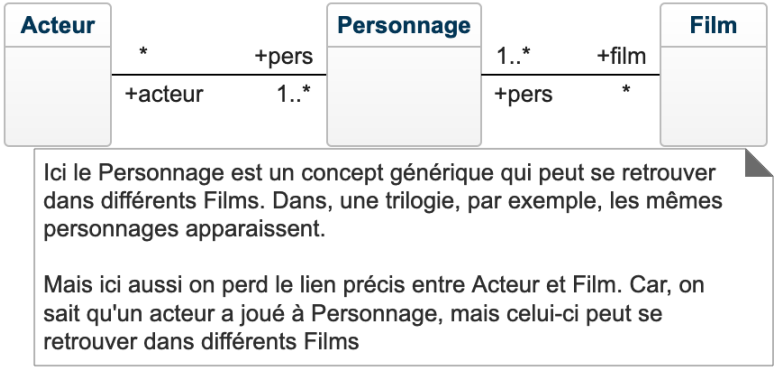
Lorsque Figurant, Acteur et Producteur ont des propriétés spécifiques ou des relations spécifiques avec d'autres entités (par exemple Recompense).
Cela empêchera un figurant d'être recompensé.

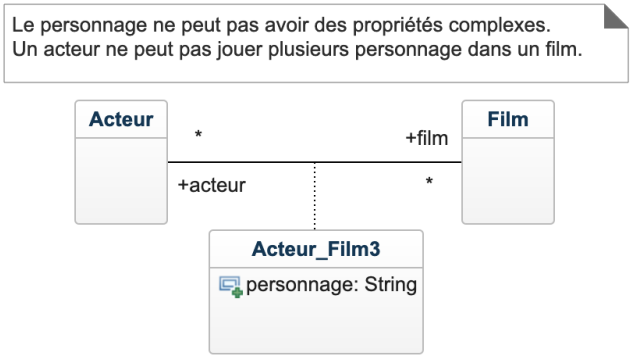


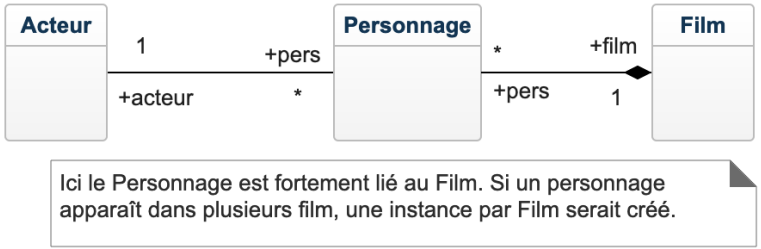
Si le Figurant n'a pas de propriétés spécifiques, ni relations spécifiques, il n'est pas utile d'introduire une entité spécifique.

MCD - Classe Association ou Entité dépendante









MCD - Quelques recommandations

- Pas de solution unique, mais les choix ont un large impact sur l'organisation et l'usage des données
- Se limiter au domaine d'étude (voir recueil des besoins)
- Limiter les dépendances fonctionnelles
- S'assurer que l'intégralité des informations peuvent être instanciées dans votre modèle

  → soit en tant qu'entité, soit en tant que propriété, soit en tant que tuple
- S'assurer que les contraintes (cardinalités, cycle de vie) sont respectées
- Se référer au client (ici vos enseigant(e)s)
- En absence de précisions, laisser en commentaires libres les choix faits !

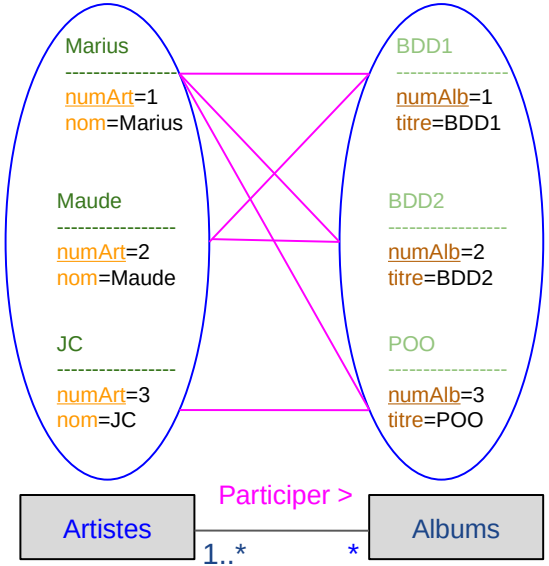
  → *veillez toutefois à ce que cela ne simplifie de trop le problème…*

MLD

- Représentation du modèle conceptuel sous la forme de tables
  - Entités –> Tables
  - Relations -> nouvelles tables ou ajout de clés étrangères et attributs aux tables existantes
- MLD – capacités de représentation tout aussi forte que le MCD
- Les cardinalités maximales guident le processus de transformation Systématiquement, [1..]* ---- [1..]* création de nouvelles tables.
- Les cardinalités minimales viennent préciser si certaines clés étrangères (et attributs migrés depuis une classe-association) peuvent être NULL.
- Les cardinalités minimales peuvent justifier la création de nouvelles tables (0..1---*)
- Ordre de transformation : priorité aux compositions qui modifient les clés primaires des tables

Relations/Associations de type 1..* ← ... → *

L'objectif de la transformation est de permettre de matérialiser efficacement la mise en relation entre instances

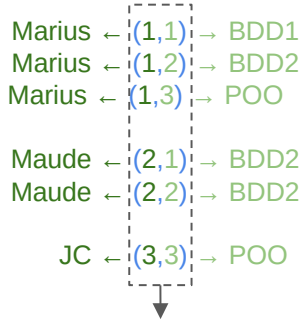


Participer (artiste,album)

(Marius, BDD1)
(Marius, BDD2)
(Marius, POO)
(Maude, BDD1)
(Maude, BDD2)
(JC, POO)

Noms des entités associées.

Représenter les tuples en se servant des clés des entités



Construire une nouvelle table Participer(numArt, numAlb)

Les clés numArt et numAlb identifient de manière unique les artistes et les albums.

Elles migrent vers la nouvelle table Participer en tant que *clés étrangères*. On utilise la notation #. Participer((numArt#, numAlb#))

Comme une seule mise en relation entre un artiste et un album est possible, les *clés étrangères* forment la *clé primaire*.

Participer((numArt#, numAlb#))

Mais comment garantir qu'un album est au moins en relation avec un artiste ?

1..* A --- B * : cela générera la table AB(*(numA#,numB#)*)

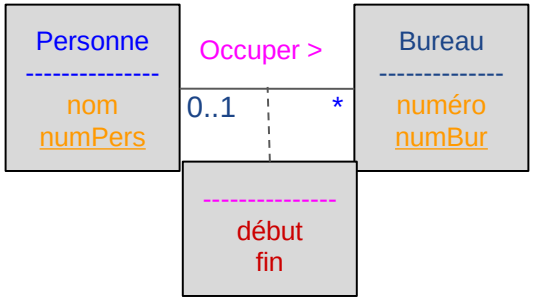
Comment garantir qu'un B est au moins en relation avec un A dans la table AB?

- Cette contrainte ne peut pas être directement renforcée au niveau du MLD
- Par ailleurs, elle ne pourra pas être garantie tout le temps.
- Lors de l'ajout d'un B
  - *Créer un nouvel enregistrement b1 en B* → lors de cette étape la contrainte ne tient plus car il n'y a aucun enregistrement dans AB correspondant à b1
  - *Créer un enregistrement dans AB qui met en relation b1 avec* a1 → à l'issue de cette étape la contrainte est à nouveau respectée
- La vérification peut être réalisée dans le cadre d'une transaction
  - *Transaction = suite d'opérations qui est traitée de manière atomique.*
  - *Si l'une des opérations échoue, les effets de l'ensemble des opérations sont annulés.*
- Ajouter une requête au processus pour valider que la contrainte est respectée
  - *Select numB from B natural left join AB where numA is NULL*

Si la requête renvoie des enregistrements → contrainte invalidée → annuler transaction

Relations/Associations de type 0..1 ← ... → */1..*/0..1

$\mathcal{R}\,01^{*}p$ Dans le cadre d'une relation porteuse d'information de type 0..1 ← ... → 1..*/*/0..1 on migre la *clé de l'entité forte* / de référence *ainsi que les données portées*, vers l'entité faible. Tous ces attributs y compris la clé étrangère peuvent être NULL.



1/ Transformation des entités : Bureau(numBur, …) et Personne (numPers, nom)

2/ Transformation de la relation Personne 0..1 ← Occuper → * Bureau selon *R 01**
→ Migration de la clé numPers de Personne vers Bureau en tant que *clé étrangère*
Bureau(numBur, … , numPers#) avec numPers- NULL
→ Migration des attributs début et fin vers Bureau en tant qu'attributs.
Bureau(numBur, … , numPers#, début, fin) avec numPers#, début, fin - NULL

Remarque: Pour éviter les valeurs NULL, Donc, si vous estimez qu'il y aura bcp. de bureaux sans personne, il vaut mieux appliquer *R***
… (mais est-ce souvent le cas ? ;))
Si c'est le cas

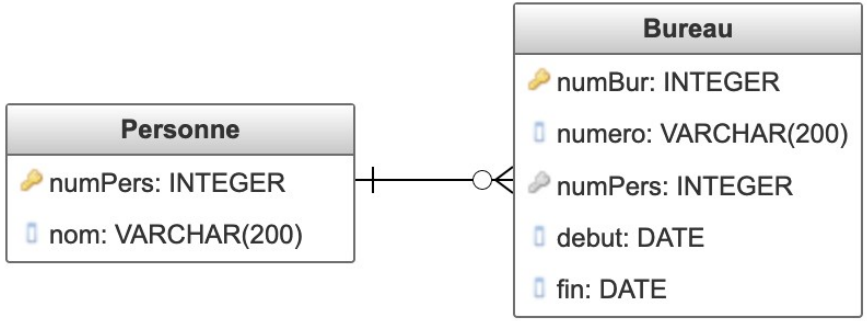
→ on peut envisager de traiter ces associations selon *R***

1/ Transformation des entités : Bureau(numBur, …) et Personne (numPers, nom)

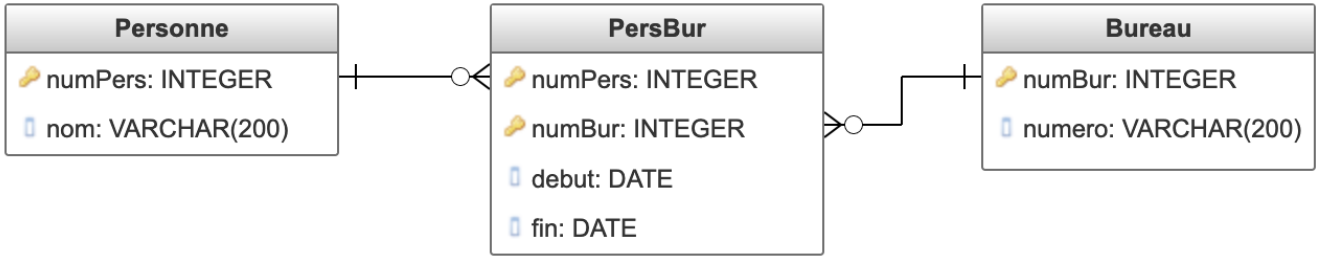
2/ Transformation de la relation Personne 0..1 ← Occuper → * Bureau selon *R ***
→ Nvl. table avec pour clés étrangères/primaires : numPers et numBur
Occupation((numBur#, numPers#))

3/ Migration des attributs début et fin vers Occupation((numBur#, numPers#), début, fin)

Migration



Création table



Migration

```
create table Personne(numPers serial, nom varchar(20));
alter table Personne add constraint PK_Pers PRIMARY KEY(numPers);

create table Bureau (numBur serial, numero varchar(20), numPers integer,
debut date, fin date);
alter table Bureau add constraint PK_Bureau PRIMARY KEY (numBur);
alter table Bureau add constraint FK_Pers FOREIGN KEY (numPers)
REFERENCES Personne;
```

```
insert into Personne(nom) VALUES ('Marius');
insert into Bureau(numero,numPers,debut,fin) VALUES
('D321',1,'2020/09/01','2020/12/01'), ('D322',NULL,NULL,NULL);
insert into Bureau(numero) VALUES ('D323'),('D324'),('D325');
```

numpers	nom
1	Marius

numbur	numero	numpers	debut	fin
1	D321	1	2020-09-01	2020-12-01
2	D322	*NULL*	*NULL*	*NULL*
3	D323	*NULL*	*NULL*	*NULL*
4	D324	*NULL*	*NULL*	*NULL*
5	D325	*NULL*	*NULL*	*NULL*

Création table

```
create table Personne(numPers serial, nom varchar(20));
alter table Personne add constraint PK_Pers PRIMARY KEY(numPers);

create table Bureau (numBur serial, numero varchar(20));
alter table Bureau add constraint PK_Bureau PRIMARY KEY (numBur);

create table PersBur(numBur serial, numPers serial, debut date, fin date);
alter table PersBur add constraint PK_PersBur PRIMARY KEY (numBur,
numPers);
alter table PersBur add constraint FK_Pers FOREIGN KEY (numPers)
REFERENCES Personne;
alter table PersBur add constraint FK_Bur FOREIGN KEY (numBur)
REFERENCES Bureau;
```

```
insert into Personne(nom) VALUES ('Marius');
insert into Bureau(numero) VALUES ('D321'),
                ('D322'),('D323'),('D324'),('D325');
insert into PersBur values (1,1,2020-09-01,'2020-12-01')
```

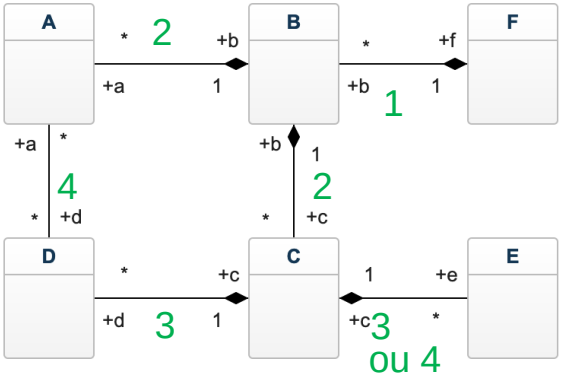
numbur	numero
1	D321
2	D322
3	D323
4	D324
5	D325

numpers	nom
1	Marius

numbur	numpers	debut	fin
1	1	2020-09-01	2020-12-01

MLD – ordre de transformation

- Priorité aux compositions, mais parmi les compositions il faut également définir un ordre
- On commence toujours par les compositions dont les entités fortes ne sont pas entités faibles dans d'autres compositions
- Le sens de lecture ou bien la position relative dans le MCD ne compte pas



Création de tables : Unique vs Primary Key

```
CREATE TABLE test (b integer);
ALTER TABLE ONLY test ADD CONSTRAINT test_b_key UNIQUE (b)
```

- Unique assure l'unicité d'un attribut ou d'un ensemble d'attributs
- Plusieurs contraintes d'unicité peuvent être définies par table
- Unique n'impose pas que la valeur soit NOT NULL
- Les valeurs NULL peuvent apparaître plusieurs fois car

  NULL == NULL produit NULL comme résultat !

```
INSERT INTO test VALUES (NULL);
INSERT INTO test VALUES (NULL);
INSERT INTO test VALUES (2);
INSERT INTO test VALUES (2);
```

→ cela n'engendre pas de violation de contrainte

→ cela n'engendre pas de violation de contrainte

→ cela engendre une violation de contrainte car 2==2

Création de tables : Unique vs Primary Key

```
CREATE TABLE test (b integer);
ALTER TABLE ONLY test ADD CONSTRAINT test_b_pkey PRIMARY KEY (b)
```

- PRIMARY KEY précise que l' attribut ou l'ensemble d'attributs identifie de manière unique un enregistrement – une seule clé par table
- PRIMARY KEY impose NOT NULL + UNIQUE

```
INSERT INTO test VALUES (NULL);
INSERT INTO test VALUES (2);
INSERT INTO test VALUES (2);
```

→ cela engendre une violation de contrainte

→ cela engendre une violation de contrainte car 2==2

- PRIMARY KEY sert également à la construction de clés étrangères
  - Utilisation des valeurs dans le cadre des opérations de vérification de contraintes étrangères, et lors de jointures
  - Optimiser les comparaisons tout en garantissant la cohérence du modèle
    - Privilégier les entiers ou ensemble d'entiers (si composition) en tant que clé primaire

Unique vs Primary Key

- Produit1(numProd serial PRIMARY KEY, nomProd varchar(200) UNIQUE)
  - + : verifications et jointures plus rapides
  - - : utilisation d’un entier en plus
- Produit2(nomProd varchar(200) PRIMARY KEY)
  - + : représentation plus compacte
  - - : vérifications et jointures moins rapides
- Quel ratio entre nombre de produits et nombre de jointures ?
- FOREIGN KEY peut pointer vers PRIMARY KEY ou UNIQUE

FOREGIN KEY et ON DELETE/UPDATE

- Interdiction de supprimer *ou de mettre à jou*r une ligne si elle est déjà liée à au moins une ligne d'une autre table via une clé étrangère
  - Comportement par défaut : ON DELETE RESTRICT / ON UPDATE RESTRICT
- La suppression *ou la mise à jour* d'une ligne qui contient une clé primaire provoque la suppression des lignes des autres tables qui contiennent les clés étrangères qui y sont liées
  - ON DELETE CASCADE / ON UPDATE CASCADE
- La suppression *ou la mise à jour* d'une ligne provoque le changement des clés étrangères qui y sont liées pour une valeur par défaut
  - ON DELETE SET DEFAULT / ON UPDATE SET DEFAULT
- La suppression *ou la mise à jour* d'une ligne provoque la mise à NULL des clés étrangères qui y sont liées → utile lorsqu'on a des cardinalités = 0
  - ON DELETE SET NULL / ON UPDATE SET NULL
- Le comportement est porté par la clé étrangère

FOREGIN KEY et ON DELETE/UPDATE

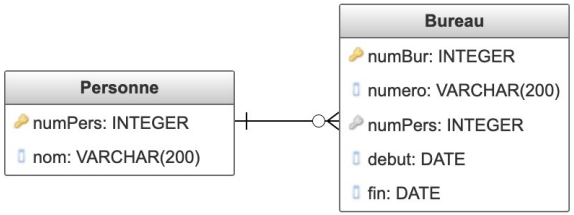
- Difficile de décider du comportement des ON DELETE/ON UPDATE à partir du MCD seul, car cela dépend des règles de gestion qui ne sont pas nécessairement présentes dans le MCD

- En revanche, on peut identifier des incohérences
  - Pas de SET NULL ou SET DEFAULT en présence d’une composition
  - Pas de SET NULL en présence d’une cardinalité 1 côté entité forte
  - Pas de CASCADE en présence d’une association simple (ou agrégation)

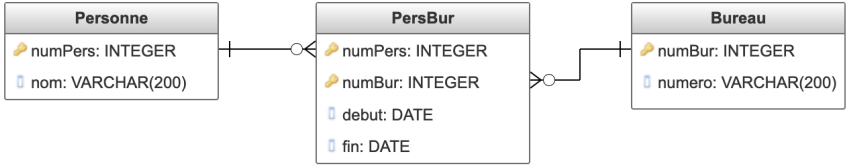
CREATE VIEW

- Simplifier l'écriture de requêtes complexes

- Remettre ensemble des bribes d'informations qui ont été reparties entre différentes tables pour éviter des redondances
  - S'abstraire de la manière dont l'information est stockée dans la base



Create View PersonneBureau as
(Select nom, numero, debut, fin From
Personne Natural Join Bureau)



Create View PersonneBureau as
(Select nom, numero, debut, fin From
Personne Natural Join PersBur Natural Join Bureau)

Select * From PersonneBureau