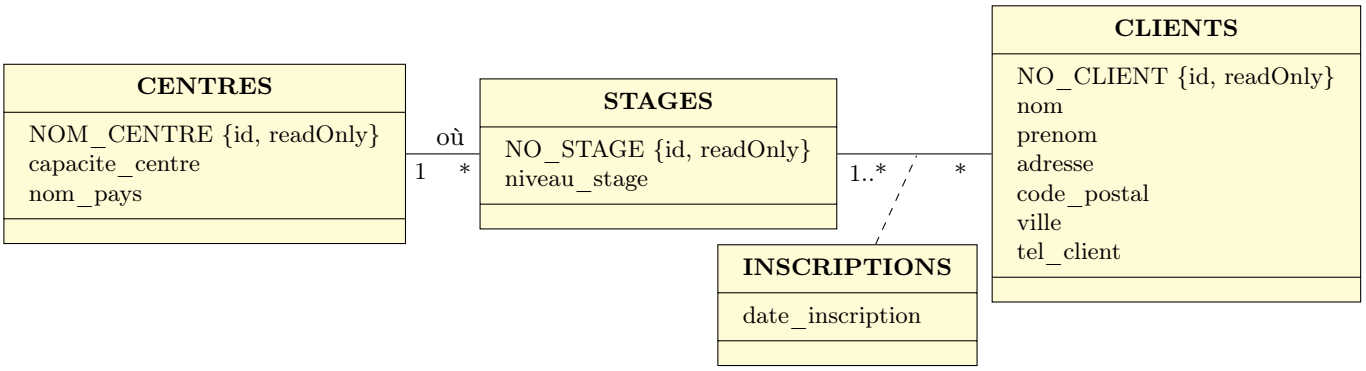
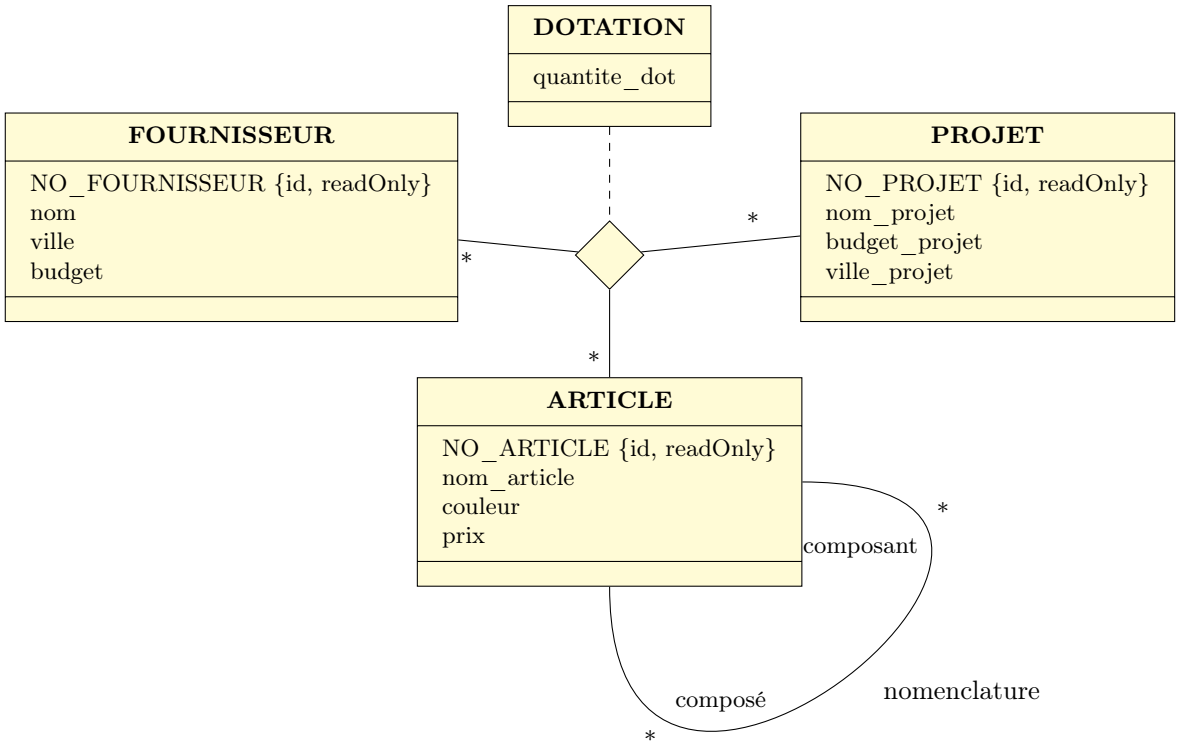
Passage d'UML au Modèle Relationnel

Exercice 1 Passage du modèle UML au modèle relationnel
Pour chaque schéma conceptuel apparaissant ci dessous, on demande de construire un schéma relationnel équivalent. Chaque schéma sera exprimé en *Tutorial D*. Vous penserez à préciser évidemment la ou les clés candidates, ainsi que les éventuelles clés étrangères. Vous proposerez à chaque fois un graphe des contraintes d'intégrité référentielles. Un script réalisant l'implantation du schéma sous PostgreSQL sera élaboré pour les cas où la mention (+SQL) est indiquée.

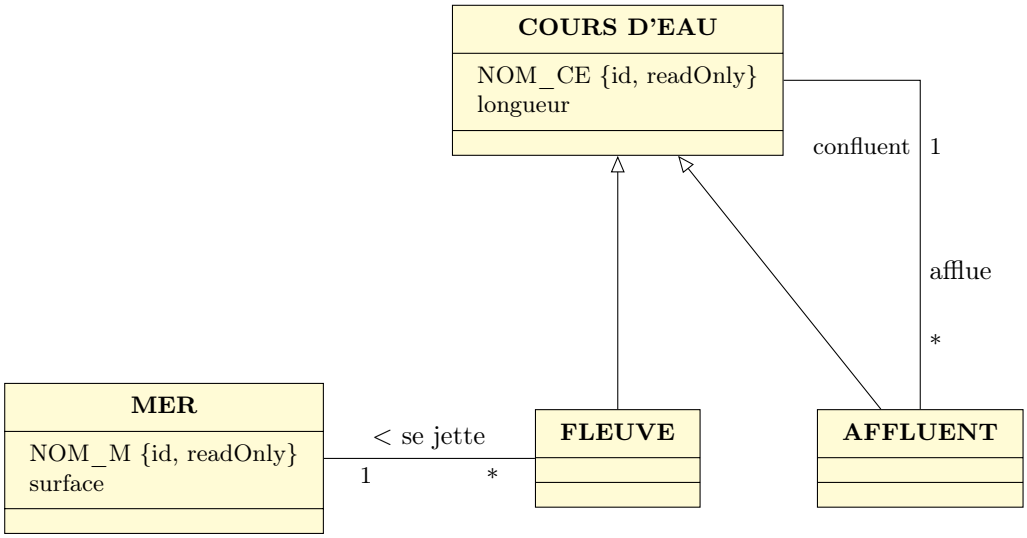
1. Organisation des stages



2. Gestion de projets



3. Fleuves et affluents : (+SQL)



ANNEXE :

Nom	Alias	type	I	O	C	descriptif	rattachement
nb_habit		Décimal		X		Exprimé en millions d'habitants avec une précision de 1 millier d'habitants.	Ville
nom_P		Chaine(20)	X	X			Pays
nom_V		Chaine(20)	X	X			Ville
popul		Entier		X		Exprimée en millions d'habitants	Pays
superficie		Entier		X		Exprimée en Km2	Pays
longueur		Entier(3)		X		Exprimé en kilomètres	Cours_d_Eau
nom_CE		Chaine(20)	X	X			Cours_d_Eau
nom_M		Chaine(20)	X	X			Mer
surface		Entier(7)		X		Exprimée en Km2	Mer

TABLE 1 – Liste des Propriétés - *I = Identifiant, O = Obligatoire, C = Calculée*

Passage d'UML au Modèle Relationnel

Correction 1

1. **Organisation des stages** Principe : Traduire chaque classe par une relation, chaque association par une relation et réfléchir ensuite quand on a des muliplicités 1..1.

```
VAR clients BASE RELATION{
   no_client    DNUMCLI,
   nom          DNOM,
   prenom       DPRENOM,
   adresse      DADRESSE,
   code_postal  DCP,
   ville        DVILLE,
   tel_client   DTELEPHONE
} KEY {no_client}
```

```
VAR centres BASE RELATION{
   nom_centre      DNOMCTR,
   capacite_centre DCAPA,
   nom_pays        DPAYS
} KEY {nom_centre}
```

Remarque : Seuls la classe-association `inscriptions` est traduite par une relation car l'association *où* a une multiplicité 1..1.

```
VAR stages BASE RELATION{
   no_stage      DNOSTAGE,
   niveau_stage  DNIVSTAG,
   code_act      DCODEACT,
   no_semaine    DSEMAINE,
   nom_centre    DNOMCTR
} KEY {no_stage};
CONSTRAINT stages_fk2
   stages{nom_centre} <= centres{nom_centre}
```

```
VAR inscriptions BASE RELATION{
   no_client        DNUMCLI,
   no_stage         DNOSTAGE,
   date_inscription DDATE
}
KEY {no_client,no_stage}
CONSTRAINT inscriptions_fk1
   inscriptions{no_client} = clients{no_client}
CONSTRAINT inscriptions_fk2
   inscriptions{no_stage} <= stages{no_stage}
```

Remarque : il y a égalité entre l'ensemble de `no_client` dans `CLIENTS` et dans `INSCRIPTIONS` du fait de la multiplicité minimale de 1 (un client est toujours inscrit au moins une fois).

2. Gestion de projets

```
VAR fournisseur BASE RELATION{
   no_fournisseur   DNOFOUR,
   nom              DNOMFOUR,
   ville            DVILLE,
   budget           DMONTANT
}
KEY {no_fournisseur}
```

```
VAR projet BASE RELATION{
   no_projet       DNUMPROJ,
   nom_projet      DNOMPROJ,
   budget_projet   DMONTANT,
   ville_projet    DVILLE
}
KEY {no_projet}
```

```
VAR article BASE RELATION{
   no_article      DNOART,
   nom_article     DNOMART,
   couleur         DCOULEUR,
   prix            DMONTANT
}
KEY {no_article}
```

```
VAR dotation BASE RELATION{
   no_fournisseur  DNOFOUR,
   no_projet       DNUMPROJ,
   no_article      DNOART,
   quantite_dot    DQUANTDOT
}
KEY {no_fournisseur,no_projet,no_article}
```

Seule petite difficulté ici : une association réflexive. Il faut identifier les rôles de chaque instance dans l'association (voir les étiquettes décrivant le rôle) et les retranscrire dans le nom des attributs utilisés dans la relation `compose`.

```
VAR nomenclature BASE RELATION{
   num_composant   DNOART,
   num_compose     DNOART
}
KEY {num_composant,num_compose}
CONSTRAINT nomenclature_fk1
  nomenclature{num_composant} <=
        article{num_composant}
CONSTRAINT nomenclature_fk2
  nomenclature{num_compose} <=
        article{num_compose}
```

3. Fleuves et affluents : (+SQL)
Ici, il peut être tentant de traduire les associations avec multiplicité 0..1 par des références (clés étrangères). Ce n'est pas un bon choix pour respecter le modèle relationnel car cela donnerait des clés étrangères avec des valeurs possiblement à NULL. On traduit donc chaque classe par une relation, et chaque association par une relation. Par contre, les multiplicités 1 traduisent bien

une DF (dépendance fonctionnelle) forte et sont traduites par une clé étrangère (ici pour affluе).

```
VAR cours_d_eau BASE RELATION{
   nom_ce        DNOMCE,
   longueur      DLONGUEUR
}
KEY {nom_ce}

VAR mer BASE RELATION{
   nom_M         DNOMM,
   surface       DSURFACE
}
KEY {nom_M}

VAR fleuve BASE RELATION{
   nom_ce        DNOMCE
} KEY {nom_ce}
CONSTRAINT fleuve_fk1
   fleuve{nom_ce} <=  cours_d_eau{nom_ce}

VAR se_jette BASE RELATION{
   nom_ce        DNOMCE,
   nom_M         DNOMM
}
KEY {nom_ce}
CONSTRAINT se_jette_fk1
   se_jette{nom_ce} <=  fleuve{nom_ce}
CONSTRAINT se_jette_fk1
   se_jette{nom_m} <= mer{nom_m}

VAR affluent BASE RELATION{
   nom_ce        DNOMCE,
   confluent      DNOMCE
} KEY {nom_ce}
CONSTRAINT affluent_fk1
   affluent{nom_ce} <=  cours_d_eau{nom_ce}
CONSTRAINT affluent_fk2
   affluent{confluent} <=  cours_d_eau{nom_ce}
        RENAME { nom_ce AS confluent}
```

En SQL, cela donnerait :

```
CREATE TABLE cours_d_eau (
   nom_ce        VARCHAR(20),
   longueur      NUMERIC(3),
   CONSTRAINT cours_d_eau_pk
       PRIMARY KEY(nom_ce)
)

CREATE TABLE mer (
   nom_M         VARCHAR(20),
   surface       NUMERIC(7),
   CONSTRAINT mer_pk
       PRIMARY KEY(nom_M)
)

CREATE TABLE fleuve(
   nom_ce        VARCHAR(20),
   CONSTRAINT fleuve_pk
       PRIMARY KEY(nom_ce),
   CONSTRAINT fleuve_fk1
      FOREIGN KEY (nom_ce)
      REFERENCES cours_d_eau(nom_ce)
)

CREATE TABLE se_jette(
   nom_ce        VARCHAR(20),
   nom_M         VARCHAR(20),
   CONSTRAINT mer_pk
       PRIMARY KEY(nom_ce),
   CONSTRAINT se_jette_fk1
      FOREIGN KEY (nom_ce)
      REFERENCES fleuve(nom_ce),
   CONSTRAINT se_jette_fk2
      FOREIGN KEY(nom_M)
      REFERENCES mer(nom_M)
)

CREATE TABLE affluent(
   nom_ce        VARCHAR(20),
   confluent      VARCHAR(20),
   CONSTRAINT affluent_pk
       PRIMARY KEY(nom_ce),
   CONSTRAINT affluent_fk1
      FOREIGN KEY (nom_ce)
      REFERENCES cours_d_eau(nom_ce),
   CONSTRAINT affluent_fk2
      FOREIGN KEY (confluent)
      REFERENCES cours_d_eau(nom_ce),
)
```