Exercices de conception de bases de données relationnelles normalisées

stph.scenari-community.org/bdd gen2.pdf

Stéphane Crozat

Paternité - Partage des Conditions Initiales à l'Identique : http://creativecommons.org /licenses/by-sa/4.0/fr/

29 janvier 2018

Table des matières

Exercices

Ce module propose des exercices permettant de réviser les fondamentaux de la conception des bases de données normalisées.

1. Exercice : Carnets de voyages

[45 minutes]

Soit la relation suivante, donnée avec une couverture minimale de ses DFE :

```
1 CarnetDeVoyage (numAuteur:integer, nomAuteur:string, prenomAuteur:string,
  numVille:integer, nomVille:string, nomPays:string, description:string)
```

- numAuteur → nomAuteur
- numAuteur → prenomAuteur
- numVille → nomVille
- numVille → nomPays
- numAuteur, numVille → description

Question 1

[solution n°1 p.17*]*

Rappeler la définition formelle d'une clé.

Énoncez la ou les clés existantes.

Pour chaque clé, énoncer les DF qui prouvent que c'est une clé et explicitez les axiomes d'Armstrong utilisés pour établir ces DF.

Dites en quelle forme normale est la relation (montrez pourquoi).

Question 2

[solution n°2 p.17*]*

Proposez un schéma normalisé en 3NF, sans perte, en faisant apparaître les clés.

Question 3

[solution n°3 p.17*]*

Rétro-concevez le modèle UML qui aurait permis d'arriver directement à ce résultat normalisé.

Question 4

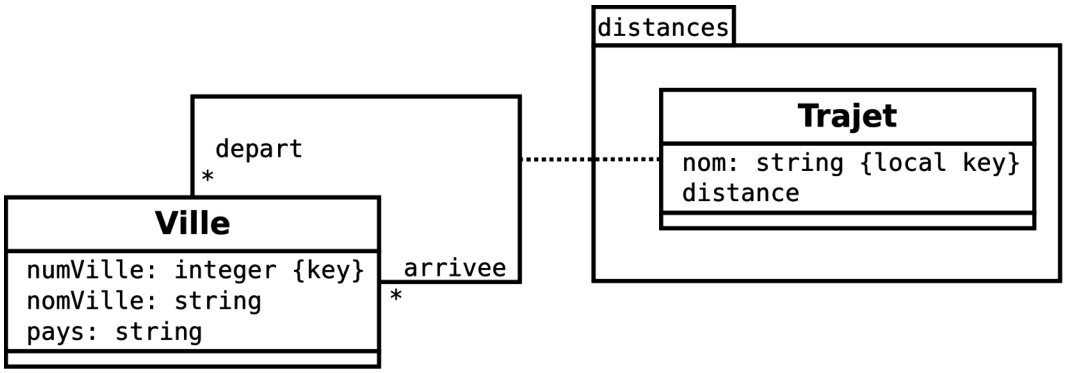
[solution n°4 p.18*]*

Écrivez le code SQL permettant de créer ce schéma normalisé dans une base de données standard.

Question 5

[solution n°5 p.18*]*

De nouvelles informations doivent être ajoutées à la base, afin de connaître les distances entre les villes. Ces nouvelles spécifications sont présentées sous la forme d'un *package* complémentaire. Effectuez le passage au relationnel de ce schéma (n'ajoutez que ce qui est nouveau). Expliquez ce que permet la clé locale `nom` de `trajet`.



Modèle des distances

Question 6

[solution n°6 p.18*]*

Écrivez le code SQL permettant d'implémenter le *package* `distances`.

Puis, donnez tous les droits sur l'ensemble du schéma à un utilisateur `Admin`, et les droits en lecture sur l'ensemble du schéma à tous les utilisateurs de la base.

2. Exercice : Agence de voyages

[45 min]

Une agence de voyage propose des prestations de logement de vacances de type hôtel, location ou gîte. Son catalogue est public et peut être consulté par tous les utilisateurs.

Elle dispose par ailleurs d'un fichier client avec les prestations effectivement vendues, à des prix éventuellement négociés en dessous du prix public. Ces données ne sont accessibles qu'à l'utilisateur "Gerant".

Le schéma UML ci-dessous représente le problème posé. On notera que Code et Numéro sont deux clés naturelles préalablement identifiées (mais il peut y avoir d'autres clés non encore identifiées). le niveau de prestation correspond au nombre d'étoiles de 1 à 5.

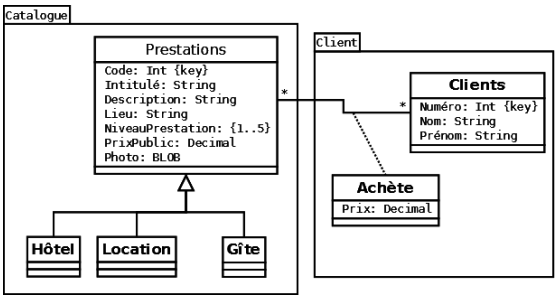


Schéma UML du système de l'agence de voyage

Question 1

[solution n°7 p.18*]*

Analyser la relation d'héritage et proposer la meilleure solution pour le passage au modèle relationnel.

Question 2

[solution n°8 p.19]

Proposer un modèle logique relationnel.

Question 3

[solution n°9 p.19]

Écrire en algèbre relationnelle la ou les vues induites par la transformation de la relation d'héritage.

Question 4

[solution n°10 p.19]

Noter que l'*intitulé* d'une prestation est unique et toujours renseigné d'une part ; et que le *niveau* de la prestation (nombre d'étoiles), le *lieu* ainsi le *type* de logement (hôtel, location ou gîte) permettent de déterminer le *prix public* d'autre part.

Préciser pourquoi le fait que les prix pratiqués peuvent être négociés en dessous du prix public permet d'écarter la DF `Prestation Prix` dans la relation `Achete`.

Énoncer pour chaque relation du schéma la liste des clés et des DF*.

Question 5

[solution n°11 p.19]

En quelle forme normale est le schéma relationnel ? Justifier avec précision. Le schéma est-il redondant ? Si oui donner un exemple, sinon expliquer pourquoi.

Question 6

[solution n°12 p.20]

Écrivez les instructions SQL LDD* permettant de créer la base de données PostgreSQL correspondant au schéma relationnel.

Indice :

Pour la gestion des BLOB*, on s'inspirera du code ci-dessous valide sous PostgreSQL :

```
1 CREATE TABLE Fruit (name CHAR(30), image OID);
2 INSERT INTO Fruit (name, image) VALUES ('peach', lo_import('/usr/images/peach.
  jpg'));
```

Question 7

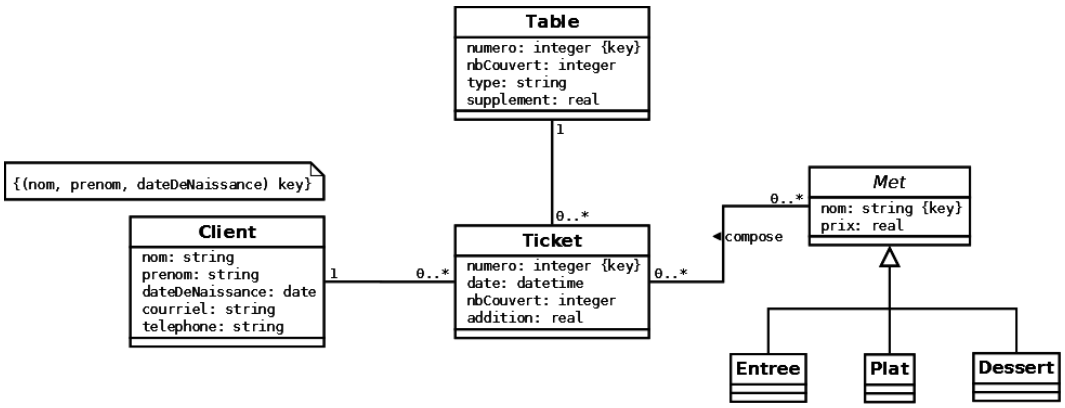
[solution n°13 p.20]

Afin d'assurer la confidentialité du système écrivez les instructions SQL LCD* adaptées.

3. Exercice : Cuisine italienne

[45 min]

Un groupe de projet NF17 a réalisé un MCD pour constituer une base de données des consommations des clients d'un restaurant italien.



UML proposé pour la base de données des clients du restaurant

Informations extraites du document de synthèse fourni par les étudiants : *« Une table est caractérisée par un numéro et possède toujours une capacité maximale théorique et un type. Un client, en réalité celui qui paye l'addition, possède un nom, un prénom et une date de naissance et peut avoir en plus une adresse courriel ou un numéro de téléphone (tous les deux facultatifs). Un ticket concerne un client, et chaque ticket possède un numéro unique et toujours renseigné, un nombre de couvert et le montant de l'addition réglé. Nous n'avons pas utilisé la date du ticket comme clé primaire car il est possible d'avoir la génération de deux tickets exactement au même moment. Un met est une entrée, un plat ou un dessert ; et il possède un prix. »*

« NB : Un attribut supplément dans la classe table permet de faire payer un potentiel supplément en fonction du type de table (petite terrasse, grande terrasse, etc.). »

Question 1

[solution n°14 p.20*]*

Proposez un modèle logique de données en relationnel à partir de l'UML, en ajoutant éventuellement des clés artificielles lorsque c'est utile, et en explicitant la solution adoptée pour transformer l'héritage.

Question 2

[solution n°15 p.21*]*

Écrivez en algèbre relationnelle la ou les vues induites par la transformation de la relation d'héritage.

Question 3

[solution n°16 p.21*]*

Énoncez pour chaque relation du schéma la liste des DF sous la forme d'une fermeture transitive.

Question 4

[solution n°17 p.21*]*

Ce MLD est-il en 3FN ? Si oui, justifiez précisément ; sinon proposez une solution pour respecter la troisième forme normale.

Question 5

[solution n°18 p.21*]*

Écrivez les instructions SQL LDD permettant de créer la base de données PostgreSQL correspondant au modèle en 3NF.

4. Exercice : Le chemin à l'envers

[45 minutes]

Soit une base de données, composée d'une seule table et de deux vues, permettant de gérer les chanteurs préférés des françaises et des français.

Cette base est définie par le code SQL LDD ci-après.

```
1 CREATE TABLE t_personnes (
2   pk_n INTEGER,
3   numss CHAR(13) UNIQUE NOT NULL,
4   nom VARCHAR(50),
5   prenom VARCHAR(50),
6   sexe CHAR(1),
7   conjoint CHAR(13) UNIQUE,
8   chanteur_prefere CHAR(50) NOT NULL,
9   nationalite_chanteur_prefere CHAR(20),
10  PRIMARY KEY (pk_n),
11  CHECK (sexe IN ('H', 'F'))
12 ) ;
```

```
1 CREATE VIEW v_hommes (pk_n, numss, nom, prenom, conjoint, chanteur_prefere,
  nationalite_chanteur_prefere)
2   AS
3     SELECT pk_n, numss, nom, prenom, conjoint, chanteur_prefere,
  nationalite_chanteur_prefere
4     FROM t_personnes
5     WHERE sexe='H' ;
6
7 CREATE VIEW v_femmes (pk_n, numss, nom, prenom, conjoint, chanteur_prefere,
  nationalite_chanteur_prefere)
8   AS
9     SELECT pk_n, numss, nom, prenom, conjoint, chanteur_prefere,
  nationalite_chanteur_prefere
10    FROM t_personnes
11    WHERE sexe='F';
```

On notera que, selon ce modèle, le conjoint d'une femme ou d'un homme est un homme ou une femme ; que X est conjoint de Y n'implique pas que Y est conjoint de X ; et que X peut être conjoint de X.

Question 1

[solution n°19 p.22*]*

Quel attribut est la clé primaire de *t_personnes* ? Comment appelle-t-on ce genre de clé ? Quel est le statut de `numss` ? Quel est le statut de `conjoint` ?

Question 2

[solution n°20 p.22*]*

Expliquez pourquoi si `conjoint` référence la table *t_personnes*, alors son domaine n'est pas correct.

Modifiez le domaine de `conjoint`, puis ajoutez une contrainte à la table *t_personnes* pour que `conjoint` soit une clé étrangère vers *t_personnes*, en s'assurant que les contraintes d'intégrité référentielles seront respectées. Vous utiliserez pour cela deux instructions ALTER.

Question 3

[solution n°21 p.23*]*

Énoncez les DF du modèle relationnel sous-jacent à cette implémentation (en se fondant sur la vraisemblance) sous la forme d'une couverture minimale des DFE.

À partir de la couverture minimale des DFE, prouvez que ce schéma est en 2NF, mais pas en 3NF.

Question 4

[solution n°22 p.23*]*

Proposez un programme SQL permettant de décomposer le schéma de cette BD afin qu'il soit en 3NF, sans perdre d'information, sans perdre de DF et sans perdre les données déjà existantes dans la BD.

Pour ce faire vous utiliserez une instruction `CREATE TABLE` permettant de créer la nouvelle table engendrée par la décomposition, puis une instruction `INSERT` permettant d'initialiser correctement cette nouvelle table avec les valeurs existantes dans *t_personnes*, et enfin deux instructions `ALTER` pour modifier la table *t_personne* de façon à en supprimer la redondance et à établir la référence à la nouvelle table.

On fera l'hypothèse que la BD ne contient pas d'incohérence.

Question 5

[solution n°23 p.23*]*

En UML, rétro-concevez le MCD qui aurait permis d'arriver directement à votre modèle après normalisation. Justifiez.

On notera la présence des vues dans le schéma initial de la BD et l'on ne reportera pas sur ce schéma les clés artificielles.

Question 6

[solution n°24 p.24*]*

Écrivez une requête qui compte le nombre de personnes qui ont le même chanteur préféré que leur conjoint.

On notera que cette requête est équivalente sur le schéma avant ou après normalisation.

5. Exercice : Zéro Faute

[40 min]

L'entreprise ZéroDéfo veut répertorier ses *fautes* de production. Une faute est définie dans le manuel qualité de l'entreprise comme *« un défaut constaté sur une pièce produite en fin de chaîne »*. L'entreprise veut associer les fautes aux produits concernés. Chaque faute est classifiée dans des catégories et sous-catégories. Chaque produit est basé sur un modèle.

- Pour chaque modèle, on veut gérer son code constitué de 8 caractères alphanumériques, son nom et la date de mise sur le marché.
- Pour chaque produit, on veut connaître le modèle associé, le numéro de série (6 chiffres) et le numéro de produit (max. 4 caractères) ainsi que l'année de production. Un produit est identifié par son numéro de série et son numéro de produit.
  Plusieurs produits partagent le même numéro de série (tous les produits de cette série), et deux produits peuvent avoir (par hasard) le même numéro de produit, dans des séries différentes (qui ont adopté le même système de codage des produits).

- Une faute concerne toujours un produit. Elle possède un code unique, un titre et la date de détection. Ell peut éventuellement avoir un commentaire et la date de réparation si le produit a été réparé.
- Les fautes sont classifiées dans des sous-catégories et chaque sous-catégorie fait partie d'une catégorie. Une faute est toujours classifiée dans une sous-catégorie au moins (elle peut être classifiée dans plusieurs).
- Les catégories et les sous-catégories possèdent un nom et, optionnellement, une description.

Question 1

[solution n°25 p.24]

Proposez un diagramme UML répondant aux besoins de cette entreprise.

Question 2

[solution n°26 p.24]

Proposez un modèle relationnel en 3NF basé sur votre modèle UML. Justifiez les choix que vous faites (transformation des héritages, associations 1-1...).

Question 3

[solution n°27 p.25]

Écrivez le code SQL LDD permettant de créer la base de données correspondant au modèle relationnel.

Question 4

[solution n°28 p.26]

Écrivez une requête SQL permettant de lister le nombre de fautes par nom de modèle et par numéro de série.

6. Exercice : Cars

[45 min]

Soit la base de données définie par le code SQL LDD ci-après.

```sql
CREATE TABLE Personne(
pk_id NUMERIC PRIMARY KEY,
nom VARCHAR(30),
prenom VARCHAR(30),
code_postal NUMERIC(5),
ville VARCHAR(30)
);

CREATE TABLE Voiture(
pk_immatriculation CHAR(7) PRIMARY KEY,
modele VARCHAR(30),
marque VARCHAR(30),
couleur VARCHAR(30),
fk_proprietaire NUMERIC REFERENCES Personne(pk_id)
);

INSERT INTO Personne (pk_id, nom, prenom, code_postal, ville) VALUES (1, 'Crozat', 'Stéphane', 60200, 'Compiègne');
INSERT INTO Personne (pk_id, nom, prenom, code_postal, ville) VALUES (2, 'Bernier', 'Emmanuel', 60420, 'Dompierre');
INSERT INTO Personne (pk_id, nom, prenom, code_postal, ville) VALUES (4, 'Vincent', 'Antoine', 60420, 'Mery-la-Bataille');
INSERT INTO Personne (pk_id, nom, prenom, code_postal, ville) VALUES (5, 'Boscolo', 'Corinne', 60680, 'Canly');

```

```
22 INSERT INTO Voiture (pk_immatriculation, modele, marque, couleur,
   fk_proprietaire) VALUES ('AA123AA', 'Clio', 'Renault', 'Noir', 4);
23 INSERT INTO Voiture (pk_immatriculation, modele, marque, couleur,
   fk_proprietaire) VALUES ('AB123NB', '807', 'Peugeot', 'Bleu', 1);
24 INSERT INTO Voiture (pk_immatriculation, modele, marque, couleur,
   fk_proprietaire) VALUES ('DE001TR', 'Clio', 'Renault', 'Rouge', 2);
25 INSERT INTO Voiture (pk_immatriculation, modele, marque, couleur) VALUES (
   'AM007JB', '205', 'Peugeot', 'Rose');
26 INSERT INTO Voiture (pk_immatriculation, modele, marque, couleur,
   fk_proprietaire) VALUES ('BK200OB', 'Cayenne', 'Porsche', 'Noir', 2);
27 INSERT INTO Voiture (pk_immatriculation, modele, marque, couleur,
   fk_proprietaire) VALUES ('ZX987FR', 'Twingo', 'Renault', 'Jaune', 5);
```

Question 1

[solution n°29 p.27]

Dessiner les tableaux correspondant à cette base de données.

Question 2

[solution n°30 p.27]

Rétro-concevoir le MLD correspond à cette base de données.

Question 3

[solution n°31 p.27]

Établissez les DF du modèle sachant que l'on pose que :

- Paris a plusieurs codes postaux ;
- Il n'existe pas deux voitures avec le même nom de modèle.

Question 4

[solution n°32 p.27]

Ce MLD respecte-t-il la 3FN ? Sinon proposer un second MLD respectant la 3FN.

Question 5

[solution n°33 p.28]

Rétro-concevoir le MCD correspondant au MLD en 3NF.

Question 6

[solution n°34 p.28]

Écrire une requête (en algèbre relationnel et en SQL) permettant de trouver les véhicules dont le modèle commence par la lettre C.

Question 7

[solution n°35 p.28]

Écrire une requête (en algèbre relationnel et en SQL) permettant de trouver tous les véhicules n'ayant pas de propriétaire.

Question 8

[solution n°36 p.28]

Écrire une requête (en SQL) permettant de trouver les personnes possédant au moins une voiture noire, trié par ordre alphabétique.

Question 9

[solution n°37 p.28*]*

Écrire une requête (en SQL) permettant de lister les personnes qui possèdent au minimum deux véhicules, par nombre décroissant de voitures.

7. Questions en question

Soit le schéma UML *Q* suivant permettant de modéliser des questionnaires.

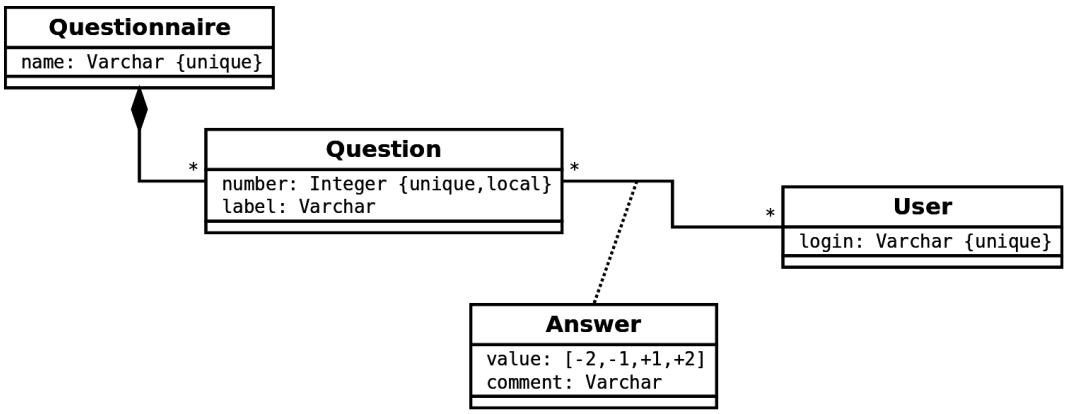


Schéma UML Q

7.1. Exercice : Analyse du modèle UML

[solution n°38 p.29*]*

Sélectionner toutes les réponses correctes.

- ☐ Tout questionnaire contient au moins une question.
- ☐ Une question peut être utilisée dans plusieurs questionnaires.
- ☐ Deux questions peuvent avoir un même numéro `number` au sein d'un même questionnaire.
- ☐ Deux questions peuvent avoir un même numéro `number` au sein de deux questionnaires différents.
- ☐ Un utilisateur peut répondre plusieurs fois à la même question.
- ☐ Un utilisateur est obligé de répondre à au moins une question.

7.2. Exercice : Implémentation relationnelle

Question 1

[solution n°39 p.29*]*

Produire le modèle relationnel R correspondant au schéma UML Q.

Question 2

Énoncer la fermeture transitive des dépendances fonctionnelles. En déduire la forme normale respectée (en justifiant). Normaliser le schéma en 3NF.

Question 3

[solution n°40 p.29*]*

Implémenter le modèle R en SQL.

Indice :

On cherchera à implémenter toutes les contraintes spécifiées au niveau conceptuel.

7.3. Exercice : Collaboration

[solution n°41 p.30*]*

Un second concepteur, identifié comme `adam` dans PosgreSQL, souhaite exécuter des requêtes de type SELECT dans la base de données. Donner les droits d'interroger toutes les tables et vues à l'utilisateur `adam`.

GRANT ;

GRANT ;

GRANT ;

GRANT ;

7.4. Exercice : Questions

Question 1

[solution n°42 p.30*]*

Créer en SQL la vue `v_questionnaire` permettant d'afficher tous les questionnaires avec leurs questions sous la forme suivante.

questionnaire	number	label
Questionnaire BDD	1	Aimez vous l'UML ?
Questionnaire BDD	2	Aimez vous le Relationnel ?
Questionnaire BDD	3	Aimez vous le SQL ?
Questionnaire DW	1	Aimez vous le dimensionnel ?
Questionnaire DW	2	Aimez vous le relationnel-objet ?
...		

Indice :

Les questions d'un même questionnaire doivent se suivre et être dans l'ordre croissant de leur numéro `number`.

Question 2

[solution n°43 p.30*]*

Écrire en algèbre relationnel et en SQL la requête permettant d'afficher toutes les questions avec les logins des utilisateurs qui y ont répondu et leur réponse.

number	label	login	value
1	Aimez vous l'UML ?	nf17p001	-1
1	Aimez vous l'UML ?	nf17p002	2
1	Aimez vous l'UML ?	nf17p003	-1
...			
2	Aimez vous le Relationnel ?	nf17p001	1
2	Aimez vous le Relationnel ?	nf17p002	2
...			

Question 3

[solution n°44 p.30*]*

Écrivez en SQL la requête permettant d'obtenir la moyenne pour chaque question de la base de données. [1pt]

questionnaire	question	label	moyenne
Questionnaire BDD	1	Aimez vous l'UML ?	-1.2
Questionnaire BDD	2	Aimez vous le Relationnel ?	1.9
Questionnaire BDD	3	Aimez vous le SQL ?	0.1
Questionnaire DW	1	Aimez vous le dimensionnel ?	1.1
Questionnaire DW	2	Aimez vous le relationnel-objet ?	-1.2
...			

7.5. Exercice : Évolution du schéma

[solution n°45 p.30*]*

Modifier la base de données, à l'aide d'une seule instruction SQL LDD, pour ajouter la *clé candidate* `description` de type `varchar` à la table `questionnaire`.

Devoir

II

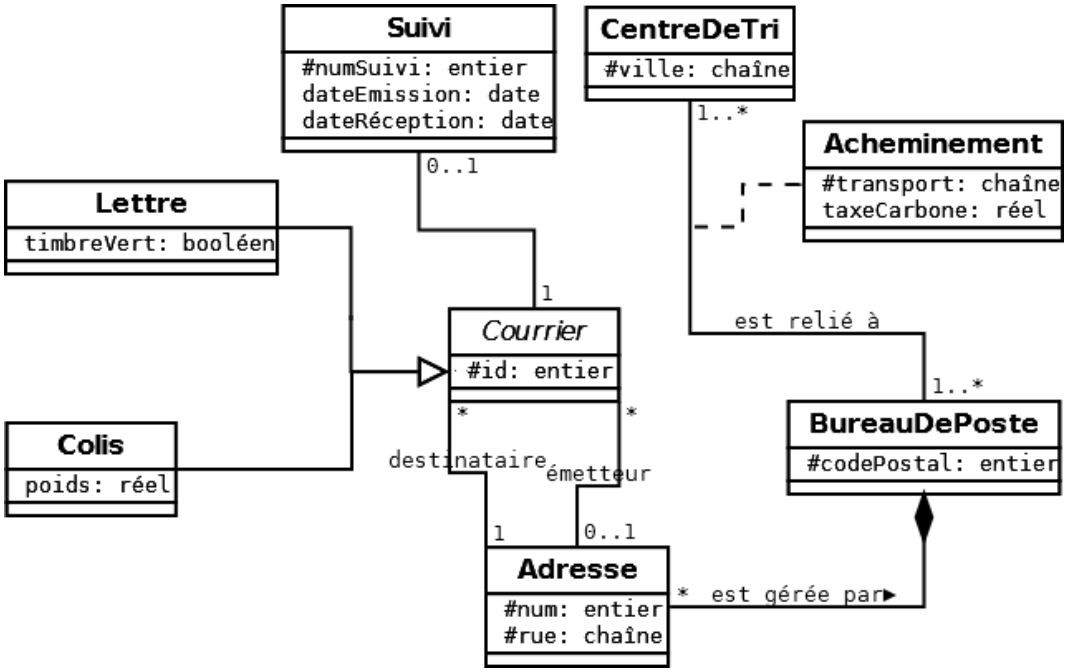
1. Exercice : La Poste

[45 min]

Des ingénieurs de La Poste ont conçu un MCD en UML dans le but de créer une base de données leur permettant de recueillir des statistiques sur les courriers acheminés.

Pour chaque courrier, on enregistre l'adresse (*rue et numéro uniquement*) du destinataire (obligatoire) et de l'émetteur (facultative) ainsi que le type (lettre ou colis). Un suivi peut être effectué pour savoir quand le courrier a été envoyé et reçu.

Chaque adresse est gérée par un bureau de poste identifié par un code postal. Les bureaux de poste sont reliés à des centres de tri qui jouent le rôle d'intermédiaire dans l'acheminement des courriers entre bureaux de poste. Un acheminement entre un centre de tri et un bureau de poste est réalisé, dans les deux sens possibles, par un ou plusieurs types de transport (voiture, train, avion...). À chaque type de transport correspond une valeur de taxe carbone.



MCD pour les statistiques de la Poste

Question 1

Traduisez ce MCD en langage relationnel, en spécifiant les clés (candidates, primaires et étrangères) ainsi que les contraintes (unicité, non-nullité, etc.). Il n'est pas demandé de préciser le type des attributs.

Justifiez les choix que vous faites. Faites des hypothèses si nécessaire.

Question 2

Écrivez les instructions SQL pour créer les tables des relations correspondant uniquement aux classes `BureauDePoste`, `CentreDeTri` et `Acheminement`.

Question 3

Déterminez en quelle forme normale est le schéma que vous avez posé, et faites les éventuelles modifications nécessaires pour atteindre la 3NF. Justifiez.

Solutions des exercices

> **Solution** n°1

Exercice p. 3

Clé

Une clé est un groupe d'attributs minimal qui détermine tous les attributs de la relation.

Il y a une unique clé (numAuteur, numVille).

- numAuteur, numVille → nomAuteur
- numAuteur, numVille → prenomAuteur
- numAuteur, numVille → nomVille
- numAuteur, numVille → nomPays
- numAuteur, numVille → description

La réflexivité et la transitivité sont utilisées à chaque fois de la même façon, par exemple, pour le premier :

numAuteur, numVille → numAuteur ET numAuteur → nomAuteur DONC numAuteur, numVille → nomAuteur

NF

La relation est en 1NF, on a identifié une clé et les attributs sont atomiques. Elle n'est pas en 2NF car des attributs faisant partie de la clé déterminent d'autres attributs, par exemple : numAuteur → nomAuteur.

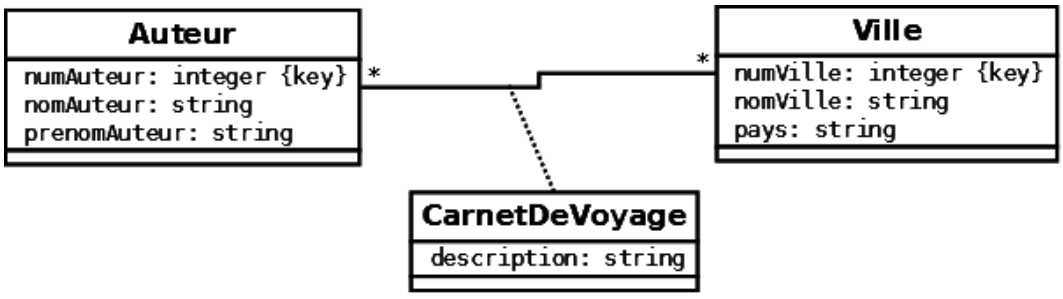
> **Solution** n°2

Exercice p. 3

```
1 Auteur(#numAuteur:integer, nomAuteur:string, prenomAuteur:string)
2 Ville(#numVille:integer, nomVille:string, nomPays:string)
3 CarnetDeVoyage(#numAuteur=>Auteur(numAuteur), #numVille=>Ville(numVille),
  description:string)
```

> **Solution** n°3

Exercice p. 3



Modèle des carnets de voyage

> **Solution** n°4

Exercice p. 3

```
1 CREATE TABLE Auteur (
2   numAuteur INTEGER PRIMARY KEY,
3   nomAuteur VARCHAR,
4   prenomAuteur VARCHAR
5 );
6
7 CREATE TABLE Ville (
8   numVille INTEGER PRIMARY KEY,
9   nomVille VARCHAR,
10  nomPays VARCHAR
11 );
12
13 CREATE TABLE CarnetDeVoyage (
14  numAuteur INTEGER REFERENCES Auteur(numAuteur),
15  numVille INTEGER REFERENCES Ville(numVille),
16  description VARCHAR,
17  PRIMARY KEY (numAuteur, numVille)
18 );
```

> **Solution** n°5

Exercice p. 4

```
1 Trajet (#nom:string, #depart=>Ville(numVille), #arrivee=>Ville(numVille),
  distance:integer)
```

`#nom` permet de définir plusieurs trajets différents entre deux mêmes villes.

> **Solution** n°6

Exercice p. 4

```
1 CREATE TABLE Trajet (
2   nom VARCHAR,
3   depart INTEGER  REFERENCES Ville(numVille),
4   arrivee INTEGER REFERENCES Ville(numVille),
5   distance INTEGER,
6   PRIMARY KEY (nom, depart, arrivee)
7 );
```

```
1 GRANT ALL PRIVILEGES ON Auteur TO Admin ;
2 GRANT ALL PRIVILEGES ON Ville TO Admin ;
3 GRANT ALL PRIVILEGES ON CarnetDeVoyage TO Admin ;
4 GRANT ALL PRIVILEGES ON Trajet TO Admin ;
5
6 GRANT SELECT ON Auteur TO PUBLIC ;
7 GRANT SELECT ON Ville TO PUBLIC ;
8 GRANT SELECT ON CarnetDeVoyage TO PUBLIC ;
9 GRANT SELECT ON Trajet TO PUBLIC ;
```

> **Solution** n°7

Exercice p. 4

L'héritage est complet et exclusif, la classe mère est abstraite. Les deux solutions de transformation par la classe mère et pas les classes filles sont possibles, mais l'héritage par la classe mère est beaucoup plus simple dans ce cas (solution à 3 tables seulement).

On choisit donc une transformation de l'héritage *par la classe mère*.

> **Solution** n°8

Exercice p. 5

```
Prestations (#code:integer, intitule:string, description:string, lieu:string,
niveau:integer, prixPublic:float, photo:BLOB, type:{H,L,G}) WITH type NOT NULL
Clients (#numero:integer, nom:string, prenom:string)
Achete (#client=>Clients(numero), #prestation=>Prestation(code), prix:float)
- Constraints : {niveau ≤ 5}
```

> **Solution** n°9

Exercice p. 5

```
vHotel = Restriction (Prestations, Prestations.type = 'H')
vLocation = Restriction (Prestations, Prestations.type = 'P')
vGite = Restriction (Prestations, Prestations.type = 'G')
```

> **Solution** n°10

Exercice p. 5

Si les prix sont fixes pour tous les clients, alors `Prestation  Prix` dans la relation `Achete`. Mais comme les prix peuvent être négociés pour chaque client, il faut connaître la prestation *et* le client pour connaître le prix. Par exemple si le prix public est de 100 et que le client2 a eu 10% de remise alors on obtient dans la relation `Achete` : `(client1, prestat1, 100) ; (client2, prestat1, 90)`.

```
Prestations : les clés sont (code) et (intitule)
{code → intitule, description, lieu, niveau, prixPublic, photo, type ; intitule →
code, description, lieu, niveau, prixPublic, photo, type ; (niveau, lieu, type) →
prixPublic}
Clients : La clé est (numero)
{numero → nom, prenom}
Achete : La clé est (client, prestation)
{(client, prestation) → prix}
```

> **Solution** n°11

Exercice p. 5

Le schéma est en 2NF.

Le schéma est en 1NF car *toutes les relations ont au moins 1 clé* et *tous les attributs sont atomiques*.

Le schéma est en 2NF car :

- Pour *Prestations* et *Clients*, les clés (`code`), (`intitule`), (`numero`) sont composées d'un seul attribut
- Pour *Achete*, `prix` ne dépend pas de *Clients* ou *Prestations* seulement (car le prix peut être négocié par le client pour chaque prestation donc *Prestations* ne détermine pas `prix`).

Le schéma n'est pas en 3NF car `(niveau, lieu, type)` `prix`, donc il existe une DFE* vers un attribut n'appartenant pas à une clé qui n'est pas issue d'une clé.

La schéma est donc redondant, en effet on aura par exemple :

…	Niveau	Lieu	Type	Prix	…
…	…	…	…	…	…
…	2	Paris	Hôtel	75	…
…	3	Paris	Hôtel	100	…
…	3	Paris	Hôtel	100	…
…	…	…	…	…	…

Exemple de données redondantes dans la relation Prestation

> **Solution** n°12

Exercice p. 5

```sql
CREATE TABLE Prestations (
  pkCode INTEGER PRIMARY KEY,
  intitule VARCHAR UNIQUE NOT NULL,
  description VARCHAR,
  lieu VARCHAR,
  niveau INTEGER CHECK (niveau < 6),
  prix DECIMAL(4,2),
  photo OID,
  type CHAR(1) NOT NULL CHECK (type IN ('H','L','G'))
) ;

CREATE TABLE Clients (
  pkNumero INTEGER PRIMARY KEY,
  nom VARCHAR,
  prenom VARCHAR
) ;

CREATE TABLE Achete (
  fkClient INTEGER,
  fkPrestation INTEGER,
  prix DECIMAL (4,2),
  PRIMARY KEY (fkClient, fkPrestation),
  FOREIGN KEY (fkClient) REFERENCES Clients(pkNumero),
  FOREIGN KEY (fkPrestation) REFERENCES Prestations(pkCode)
) ;
```

> **Solution** n°13

Exercice p. 5

```sql
GRANT SELECT ON Prestation TO PUBLIC;

GRANT ALL PRIVILEGES ON Prestation TO Gerant;
GRANT ALL PRIVILEGES ON Client TO Gerant;
GRANT ALL PRIVILEGES ON Achete TO Gerant;
```

> **Solution** n°14

Exercice p. 6

1. Héritage exclusif
2. Héritage complet

3. Classe mère abstraite

On choisit l'héritage par classe mère pour des raisons de simplicité : 2 tables contre 6 à cause de l'association N:M.

```
1 Table(#numero, nbCouvert, type, supplement) avec nbCouvert, type NOT NULL
2 Client(#id:integer, nom, prenom, dateDeNaissance, courriel, telephone) avec (nom,
  prenom, dateDeNaissance) clé candidate
3 Ticket(#numero, date, nbCouvert, addition, table=>Table(numero), client=>Client
  (id)) avec date, nbCouvert, table, client NOT NULL
4 Met(#nom, prix, type:{'E', 'P', 'D'}) avec prix et type NOT NULL
5 Compose(#ticket=>Ticket(numero), #met=>Met(nom))
```

Remarque

Les types non précisés sont ceux du diagramme UML.

> **Solution** n°15

Exercice p. 6

```
1 vEntree = RESTRICTION (Met, Type = 'E')
2 vPlat = RESTRICTION (Met, Type = 'P')
3 vDessert = RESTRICTION (Met, Type = 'D')
```

> **Solution** n°16

Exercice p. 6

Table

- numero -> nbCouvert, type, supplement
- type -> supplement

Client

- id -> nom, prenom, dateDeNaissance, courriel, telephone
- nom, prenom, dateDeNaissance -> id, courriel, telephone

Ticket

- numero -> date, nbCouvert, addition, table, client

Met

- nom -> prix, type

> **Solution** n°17

Exercice p. 6

Le modèle n'est pas en 3NF, car un attribut non clé (`supplement`) est déterminé par un autre attribut non clé (`type`).

Il faut décomposer `table` en :

- `Type_Table(#type, supplement)`
- `Table (#numero, nbCouvert, type=>type_table)`

> **Solution** n°18

Exercice p. 6

```
1 CREATE TABLE Type_Table(
2 type VARCHAR PRIMARY KEY,
3 supplement REAL
4 );
5
6 CREATE TABLE Table(
7 numero INTEGER PRIMARY KEY,
8 nbCouvert INTEGER NOT NULL,
9 type VARCHAR REFERENCES Type_Table(type)
10 );
11
12 CREATE TABLE Client(
13 id INTEGER PRIMARY KEY,
14 nom VARCHAR NOT NULL,
15 prenom VARCHAR NOT NULL,
16 dateDeNaissance DATETIME NOT NULL,
17 courriel VARCHAR,
18 telephone VARCHAR,
19 UNIQUE(nom, prenom, dateDeNaisance)
20 );
21
22 CREATE TABLE Ticket(
23 numero INTEGER PRIMARY KEY,
24 date DATETIME NOT NULL,
25 nbCouvert INTEGER NOT NULL,
26 addition DECIMAL,
27 table INTEGER REFERENCES Table(numero),
28 client INTEGER REFERENCES Client(id)
29 );
30
31 CREATE TABLE Met(
32 nom VARCHAR PRIMARY KEY,
33 prix REAL NOT NULL,
34 type CHAR(1) CHECK (type='E' OR type='P' OR type='D')
35 );
36
37 CREATE TABLE Compose(
38 ticket INTEGER REFERENCES Ticket(numero),
39 met VARCHAR REFERENCES Met(nom),
40 PRIMARY KEY(ticket, met)
41 );
```

> **Solution** n°19

Exercice p. 7

La clé primaire de *t_personnes* est `pk_n`, c'est une clé artificielle. `numss` est une clé candidate car elle est unique pour chaque enregistrement et non nulle. `conjoint` n'est pas une clé, car l'attribut peut être nul.

> **Solution** n°20

Exercice p. 7

Le domaine de `conjoint` doit être celui de la clé primaire de `t_personne`, donc `INTEGER`.

```
DROP VIEW v_hommes ;
DROP VIEW v_femmes ;

ALTER TABLE t_personnes ALTER conjoint TYPE INTEGER USING conjoint::INTEGER;

ALTER TABLE t_personnes ADD FOREIGN KEY (conjoint) REFERENCES t_personnes(pk_n) ;
```

Complément

Sous PostgreSQL, il n'est pas possible de modifier le type d'une colonne utilisé dans une vue ou une règle, il faut donc commencer par supprimer les vues avant d'effectuer la modification. Il suffit de recréer les vues par la suite.

De même, le passage d'un type `CHAR` ou `VARCHAR` vers `INTEGER` n'est pas possible sans utiliser le transtypage qui décrit comment devra être transformé le contenu de la colonne en cours de transformation. En précisant que la conversion est effectuée vers un entier, la modification devient possible.

> **Solution** n°21

Exercice p. 8

- `pk_n  numss`
- `pk_n  nom`
- `pk_n  prenom`
- `pk_n  sexe`
- `pk_n  conjoint`
- `pk_n  chanteur_prefere`
- `numss  pk_n`
- `chanteur_prefere  nationalite_chanteur_prefere`

Le schéma est en 1NF puisqu'il y a une clé et que les valeurs sont atomiques.

Le schéma est en 2NF puisqu'il n'existe pas d'attribut dépendant d'une partie seulement d'une clé. Tout schéma dont les clés ne sont composés que d'un seul attribut est en 2NF.

Le schéma n'est pas en 3NF car `chanteur_prefere` → `nationalite_chanteur_prefere` et donc un attribut non clé dépend d'un autre attribut non clé.

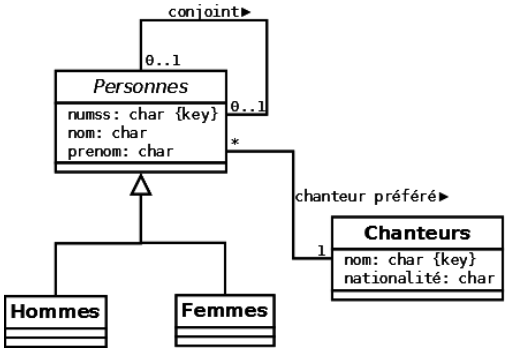
> **Solution** n°22

Exercice p. 8

```
CREATE TABLE t_chanteurs (
  nom VARCHAR(50) PRIMARY KEY,
  nationalite VARCHAR(20)
) ;

INSERT INTO t_chanteurs (nom, nationalite)
  SELECT DISTINCT chanteur_prefere, nationalite_chanteur_prefere
  FROM t_personnes ;

ALTER TABLE t_personnes
  DROP nationalite_chanteur_prefere ;

ALTER TABLE t_personnes
  ADD FOREIGN KEY (chanteur_prefere) REFERENCES t_chanteurs(nom) ;
```

> **Solution** n°23

Exercice p. 8



Modèle UML

On notera que les vues sont le résultat d'une association d'héritage ; et que l'association *conjoint* est une relation 1:1 classique (grâce à la clause d'unicité sur la clé étrangère), et donc (malgré la non conformité avec la réalité) : si A a pour conjoint B, alors B n'a pas forcément pour conjoint A et A peut être conjoint de A.

> **Solution** n°24

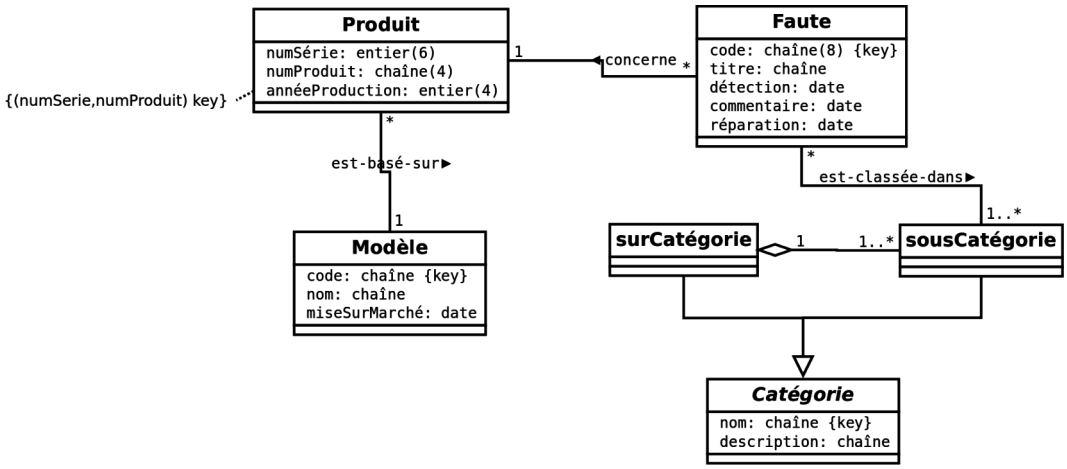
Exercice p. 8

```
1 SELECT COUNT(P1.pk_n) AS result
2 FROM t_personnes P1, t_personnes P2
3 WHERE P1.conjoint = P2.pk_n
4 AND P1.chanteur_prefere = P2.chanteur_prefere ;
```

> **Solution** n°25

Exercice p. 9

MCD



> **Solution** n°26

Exercice p. 9

MLD

```
Modèle(#code : chaîne, nom : chaîne, dateMiseSurMarché : date)
     avec dateMiseSurMarché non-null

Produit(#numSérie : entier, #numProduit : chaîne, annéeProduction : entier,
modèle=>Modèle)
     avec annéeProduction non-nul
     avec modèle non-nul

SurCatégorie(#nom : chaîne, description : chaîne)
SousCatégorie(#nom : chaîne, description : chaîne, surCatégorie=>SurCatégorie)
     avec surCatégorie non-null
     avec PROJ(SurCatégorie, nom) = PROJ(SousCatégorie, surCatégorie)
     avec PROJ(SousCatégorie, nom) INTER PROJ(SurCatégorie, nom) = {}

Faute(#code : chaîne, titre : chaîne, dateDétection : date, dateRéparation :
date, commentaire : chaîne, produitSérie=>Produit(numSérie), produitNuméro=>Produit
(numProduit))
     avec titre non-nul
     avec dateDétection non-nul
     avec produitSérie non-nul
     avec produitNuméro non-nul

ClassementFaute(#faute=>Faute(code), #sousCatégorie=>SousCatégorie(nom))
    PROJ(Faute, code) = PROJ(ClassementFaute, faute)
```

Justification de la 3NF

1. 1NF : toutes les relations ont une clé et chaque attribut est atomique.
2. 2NF :
   - `Modèle`, `SurCatégorie`, `SousCatégorie` et `Faute` ont des clés constituées d'un seul attribut ;
   - `ClassementFaute` ne comporte pas d'attribut en dehors de la clé ;
   - dans `Produit`, il n'existe pas de dépendance fonctionnelle de `numSérie` vers `annéeProduction` ou `modèle`, ou les deux, idem pour `numProduit`.
3. 3NF : dans les relations `Modèle`, `Produit` et `Faute`, il n'existe pas de dépendance fonctionnelle entre attributs ou groupe d'attributs en dehors de la clé.

> **Solution** n°27

Exercice p. 9

```
CREATE TABLE tModele
(
    code CHAR(8) PRIMARY KEY,
    nom VARCHAR NOT NULL,
    dateMiseSurMarche DATE NOT NULL
);
CREATE TABLE tProduit
(
```

```
    numSerie NUMERIC(6) NOT NULL,
    numProduit VARCHAR(4) NOT NULL,
    anneeProduction INTEGER NOT NULL,
    modele CHAR(8) NOT NULL,
    PRIMARY KEY (numSerie, numProduit),
    FOREIGN KEY (modele) REFERENCES tModele(code)
);
CREATE TABLE tSurCategorie
(
    nom VARCHAR PRIMARY KEY,
    description VARCHAR
);
CREATE TABLE tSousCategorie
(
    nom VARCHAR PRIMARY KEY,
    description VARCHAR,
    surCategorie VARCHAR,
    FOREIGN KEY (surCategorie) REFERENCES tSurCategorie(nom)
);
CREATE TABLE tFaute
(
    code VARCHAR PRIMARY KEY,
    titre VARCHAR NOT NULL,
    dateDetection DATE NOT NULL,
    dateReparation DATE,
    commentaire VARCHAR,
    produitSerie NUMERIC(6) NOT NULL,
    produitNumero VARCHAR(4) NOT NULL,
    FOREIGN KEY (produitSerie, produitNumero) REFERENCES tProduit(numSerie,
numProduit)
);
CREATE TABLE tClassementFaute
(
    faute VARCHAR NOT NULL,
    sousCategorie VARCHAR NOT NULL,
    PRIMARY KEY (faute, sousCategorie),
    FOREIGN KEY (faute) REFERENCES tFaute(code),
    FOREIGN KEY (sousCategorie) REFERENCES tSouscategorie(nom)
);
```

> **Solution** n°28

Exercice p. 9

```
SELECT m.nom, p.numSerie, COUNT(*)
FROM tModele m, tProduit p, tFaute f
WHERE m.code = p.modele
    AND p.numSerie = f.produitSerie
    AND p.numProduit = f.produitNumero
GROUP BY m.nom, p.numSerie
;
```

```
SELECT m.nom, p.numeroSerie, COUNT(*)
FROM (tModele m
    JOIN tProduit p
        ON m.code = p.modele)
    JOIN tFaute f
        ON (p.numSerie = f.produitSerie AND p.numProduit = f.produitNumero)
```

```
7 GROUP BY m.nom, p.numSerie
8 ;
```

Attention

Bien que le numéro de produit ne participe pas à la projection (`SELECT`) ni au regroupement (`GROUP BY`), il ne faut pas oublier la jointure entre `tProduit.numProduit` et `tFaute.produitNumero`.

> **Solution** n°29

Exercice p. 10

Personne				
pk_id	**nom**	**prenom**	**code_postal**	**ville**
1	Crozat	Stéphane	60200	Compiègne
2	Bernier	Emmanuel	60420	Dompierre
4	Vincent	Antoine	60420	Mery-la-Bataille
5	Boscolo	Corinne	60680	Canly

Véhicule				
pk_immatriculation	**modele**	**marque**	**couleur**	**fk_proprietaire**
AA123AA	Clio	Renault	Noir	4
AB122NB	807	Peugeot	Bleu	1
DE001TR	Clio	Renault	Rouge	2
AM007JB	205	Peugeot	Rose	
BK200OB	Cayenne	Porche	Noir	2
ZX987FR	Twingo	Renault	Jaune	5

Exemple de données

> **Solution** n°30

Exercice p. 10

```
1 Personne(#pk_id, nom, prenom, code_postal, ville)
2 Voiture(#pk_immatriculation, modele, marque, couleur, fk_proprietaire=>Personne)
```

> **Solution** n°31

Exercice p. 10

pk_id →nom, prenom, code_postal, ville

pk_immatriculation → modele, marque, couleur, fk_proprietaire

modele → marque

> **Solution** n°32

Exercice p. 10

Il n'est pas en 3NF, car `modele` détermine `marque`.

```
1 Personne(#pk_id, nom, prenom, code_postal, ville)
2 Voiture(#pk_immatriculation, fk_modele=>Modele, couleur,
  fk_proprietaire=>Personne)
3 Modele(#pk_modele, marque)
```

pk_id →nom, prenom, code_postal, ville

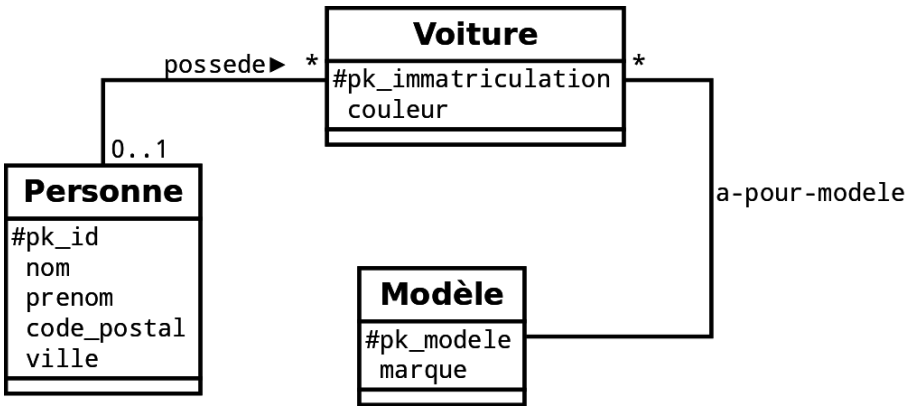
pk_immatriculation → fk_modele, couleur, fk_proprietaire

pk_modele → marque

Toutes les DFE sont du type k → a (avec k clé), on est donc en BCNF.

> **Solution** n°33

Exercice p. 10



Modèle UML

> **Solution** n°34

Exercice p. 10

```
R = Restriction(Voiture, modele COMME 'C%')
```

```
SELECT *
FROM Voiture
WHERE modele LIKE 'C%';
```

> **Solution** n°35

Exercice p. 10

```
R = Restriction(Voiture, fk_proprietaire = NULL)
```

```
SELECT pk_immatriculation
FROM Voiture
WHERE V.fk_proprietaire IS NULL;
```

> **Solution** n°36

Exercice p. 10

```
SELECT DISTINCT nom, prenom
FROM Voiture V, Personne P
WHERE V.fk_proprietaire = P.pk_id
AND couleur = 'Noir'
ORDER BY nom, prenom;
```

> **Solution** n°37

Exercice p. 11

```
SELECT pk_id, nom, prenom, COUNT(pk_immatriculation) AS nombre
FROM Voiture V, Personne P
WHERE V.fk_proprietaire = P.pk_id
GROUP BY pk_id, nom, prenom
```

```
5 HAVING COUNT(pk_immatriculation) > 1
6 ORDER BY nombre DESC;
```

> **Solution** n°38

Exercice p. 11

Sélectionner toutes les réponses correctes.

- ☐ Tout questionnaire contient au moins une question.
- ☐ Une question peut être utilisée dans plusieurs questionnaires.
- ☐ Deux questions peuvent avoir un même numéro `number` au sein d'un même questionnaire.
- ☑ Deux questions peuvent avoir un même numéro `number` au sein de deux questionnaires différents.
- ☐ Un utilisateur peut répondre plusieurs fois à la même question.
- ☐ Un utilisateur est obligé de répondre à au moins une question.

> **Solution** n°39

Exercice p. 12

```
1 Questionnaire(#name)
2 Question(#questionnaire=>Questionnaire, #number, label)
3 User(#login)
4 Answer(#questionnaire=>Questionnaire, #question=>Question, #user=>User, value,
  comment)
```

> **Solution** n°40

Exercice p. 12

```
1 CREATE TABLE user (
2 login VARCHAR PRIMARY KEY
3 );
4
5 CREATE TABLE questionnaire (
6 name VARCHAR PRIMARY KEY
7 );
8
9 CREATE TABLE question (
10 number INTEGER,
11 label VARCHAR,
12 questionnaire VARCHAR REFERENCES questionnaire(name),
13 PRIMARY KEY (number, questionnaire)
14 );
15
16 CREATE TABLE answer (
17 question INTEGER,
18 questionnaire VARCHAR,
19 login VARCHAR,
20 value INTEGER CHECK (value=0 OR value=-2 OR value=-1 OR value=1 OR value=2),
21 comment VARCHAR,
22 PRIMARY KEY (question,questionnaire,login),
23 FOREIGN KEY (login) REFERENCES user(login),
24 FOREIGN KEY (question,questionnaire) REFERENCES question(number,questionnaire)
```

```
25 );
```

> **Solution** n°41

Exercice p. 12

Un second concepteur, identifié comme `adam` dans PosgreSQL, souhaite exécuter des requêtes de type SELECT dans la base de données. Donner les droits d'interroger toutes les tables et vues à l'utilisateur `adam`.

GRANT `SELECT ON user TO adam`;

GRANT `SELECT ON questionnaire TO adam`;

GRANT `SELECT ON question TO adam`;

GRANT `SELECT ON answer TO adam`;

> **Solution** n°42

Exercice p. 13

```
1 CREATE VIEW v_questionnaire AS
2 SELECT questionnaire, number, label
3 FROM question
4 ORDER BY questionnaire, number
5
```

> **Solution** n°43

Exercice p. 13

```
1 Projection (
2 Jointure (answer, question, question=number AND questionnaire=questionnaire),
3 number,label,login,value)
```

```
1 SELECT q.number, q.label, a.login, a.value
2 FROM answer a JOIN question q
3 ON a.question=q.number AND a.questionnaire=q.questionnaire
4
```

> **Solution** n°44

Exercice p. 14

```
1 SELECT a.questionnaire, q.number, q.label, avg(a.value)
2 FROM answer a JOIN question q
3 ON a.question=q.number AND a.questionnaire=q.questionnaire
4 GROUP BY a.questionnaire, q.number, q.label
5 ORDER BY a.questionnaire, q.number
6
```

> **Solution** n°45

Exercice p. 14

Modifier la base de données, à l'aide d'une seule instruction SQL LDD, pour ajouter la *clé candidate* `description` de type `varchar` à la table `questionnaire`.

ALTER TABLE questionnaire ADD description VARCHAR UNIQUE NOT NULL;

Abréviations

BLOB : Binary Large Object

DF : Dépendance Fonctionnelle

DFE : Dépendance Fonctionnelle Elémentaire

LCD : Langage de Contrôle de Données

LDD : Langage de Définition de Données