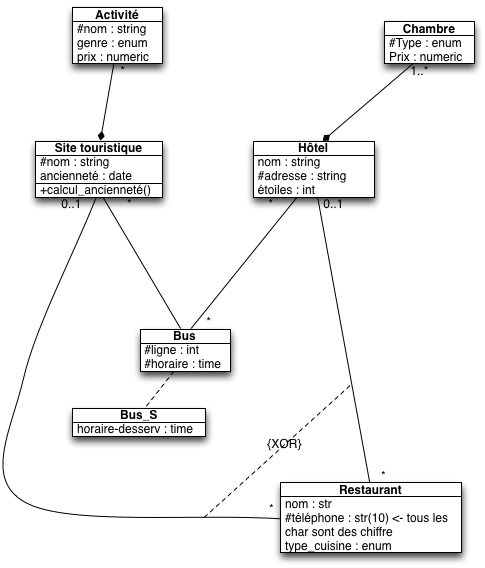
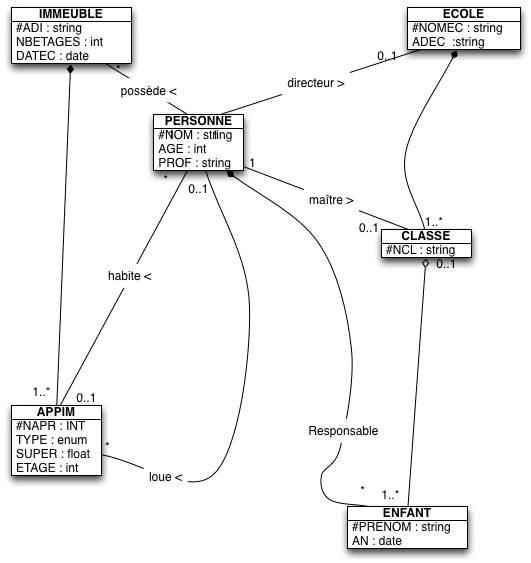
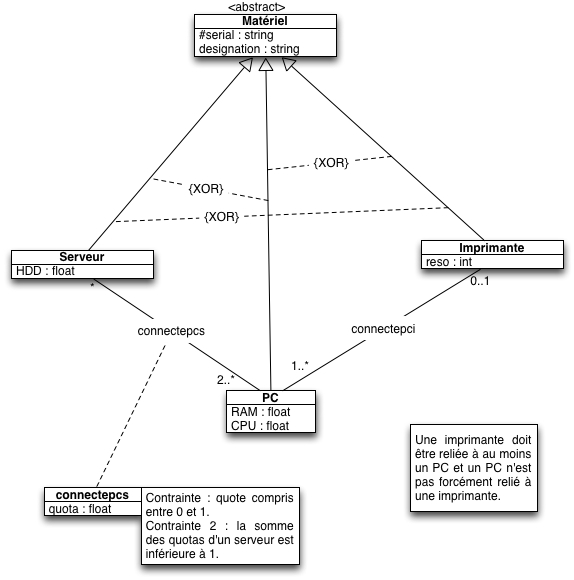
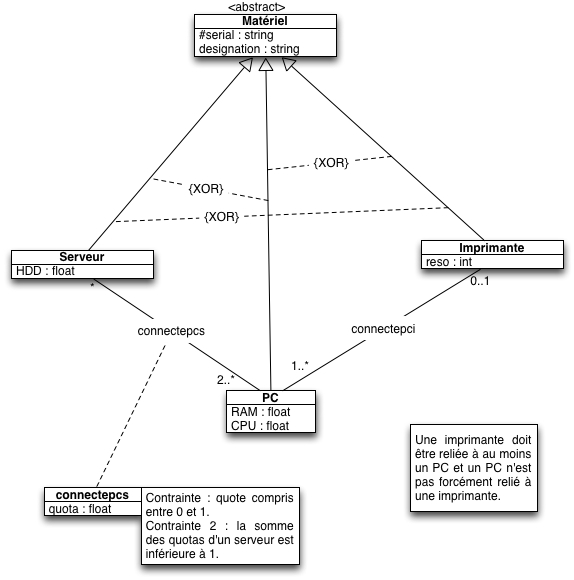
**Exercice 2**

* Une commune veut mieux tenir à jour et représenter les offres disponibles en terme de tourisme, et cherche ainsi à en établir une base de données.
* Elle a répertorié :
* la liste de ses sites touristiques : leur nom, et leur ancienneté. Chaque site propose différentes activités (visite , visite guidée, concerts, ateliers, etc.) avec un nom, un genre, et un prix.
* la liste de ses hôtels : leur nom, leur adresse, leur nombre d'étoiles et les différentes chambres qu'ils proposent (type de chambre et prix de chaque type de chambre),
* la liste de ses transports en commun, permettant de se déplacer des hôtels aux sites touristiques (on ne considéra ici que les bus). Un bus peut desservir plusieurs hôtels et sites touristiques, et il est identifié par son numéro de ligne et son horaire.
* la liste de ses restaurants : les restaurants répertoriés se trouvent dans les hôtels ou dans les sites touristiques. Ils possèdent un nom, un numéro de téléphone, et un type de cuisine (traditionnel, *fast food*, pizzeria, etc.).
* 
* Site touristique(#nom : string, ancienneté : date)
* Activité(#nom : string, genre : enum, prix : numeric, #site : string => Site touristique(nom))
* Hôtel(nom : string, #adresse : string, étoiles : int)
* Chambre(#type : enum, prix : numeric, #hôtel : string => Hôtel(adresse))
* +PROJ
* Bus(#ligne : int, #horaire : time)
* Bus\_Hotel(#ligne : int =>Bus(ligne), #horaire : time =>Bus(horaire), #hôtel : string => Hôtel(adresse), horaire-desserv : time)
* Bus\_Site(#ligne : int =>Bus(ligne), #horaire : time =>Bus(horaire), #site : string => Site touristique(nom), horaire-desserv : time)
* Rstaurants (nom : string, #téléphone : str(10), type\_cuisine : enum, hôtel : string => Hôtel(adresse), site : string => Site touristique(nom))
* Si hôtel NOT NULL, site doit être NULL et vice versa.
* **Exercice : Le chemin des écoliers**
* IMMEUBLE (#ADI, NBETAGES, DATEC, PROP)
* APPIM (#ADI, #NAPR, OCCUP, TYPE, SUPER, ETAGE)
* PERSONNE (#NOM, AGE, PROF, ADR, NAPR)
* ÉCOLE (#NOMEC, ADEC, DIR)
* CLASSE (#NOMEC, #NCL, MAITRE)
* ENFANT (#NOMP, #PRENOM, AN, NOMEC, NCL)
* Clés étrangères :
* IMMEUBLE
  + PROP : PERSONNE(NOM)
* APPIM
  + ADI : IMMEUBLE(ADI)
  + OCCUP : PERSONNE(NOM)
* PERSONNE
  + ADR : APIM(ADI)
  + NAPR : APPIM(NAPR)
* ECOLE
  + DIR : PERSONNE(NOM)
* CLASSE
  + NOMEC : ECOLE(NOMEC)
  + MAITRE : PERSONNE(NOM)
* ENFANT
  + NOMP : PERSONNE(NOM)
  + NOMEC : ECOLE(NOMEC)
  + NCL : CLASSE(NCL)
* 
* **Exercice : Gestion d'une agence immobilière**
* Une agence immobilière souhaite créer une base de données pour la gestion des biens immobiliers mis à sa disposition et pour l'exploitation statistique et/ou fiscale des informations accumulées. Pour chaque logement on possède plusieurs informations : l'adresse, le nom du propriétaire, le type (maison/appartement), le nombre de pièces constituant, superficie habitable, l'état d'habitation (neuf, bon état, très bon état, mauvais état), prix de mise en vente, la date de disponibilité, la ville. Chaque propriété peut avoir un ou plusieurs garages. Ces derniers sont caractérisés par le type (box, emplacement numérotés, etc.) et dans certain cas peuvent avoir des adresses différentes de celle de la propriété. Un logement peut être vendu ou acheté par une personne caractérisée par son nom et son adresse de contact. Pour chaque transaction de vente, l'agence touche une commission qui correspond à un pourcentage du prix de vente (qui est composé d'une valeur fixe de 1000 euros à laquelle on additionne entre 3 et 5% en fonction de la négociation). L'agence organise et gère également les visites des propriétés que les acheteurs potentiels (clients) pourraient réclamer. Pour des besoins d'exploitation fiscale, l'agence souhaite garder la trace du montant total de chaque transaction de vente effectuée (prix de vente) ainsi que de la somme des commissions.
* 
* **Exercice : Gestion de parc informatique**
* Vous avez en charge la réalisation d'un modèle de base de données pour la gestion d'un parc informatique.
* L'analyse des besoins révèlent les informations suivantes : tout matériel informatique est identifié de façon unique par un numéro de série et est décrit par une désignation. Il existe trois types de matériel informatique : les PC, les serveurs et les imprimantes. Pour les PC les informations que l'on veut gérer sont la taille de la mémoire vive et la cadence du micro-processeur, pour les serveurs on veut gérer leur volume de disque dur et pour les imprimante leur résolution maximale d'impression. On veut également gérer les connections réseau sachant que tout PC peut être relié à un ou plusieurs serveur et que chaque serveur sert bien entendu plusieurs PC ; et qu'un PC peut être relié à une imprimante, qui est également utilisée par plusieurs PC. Quand un PC est relié à un serveur, on veut gérer le quota de disque dont il dispose sur ce serveur.
* 
* Serveur (#serial : string, designation : string, HDD : float)
* PC (#serial : string, RAM : float, CPU : float, Imprimante=>Imprimante(serial) : string)
* Imprimante(#serial : string, designation ; string, reso : int)
* Connexsionpcs(#PC=>PC(serial) : string, #serveur=>serveur(serial) : string, quota : float)
* Contraintes :
* un serveur est relié au moins à 2 pc ;
* Proj(Imprimante, serial) ⊂ Proj(PC, Imprimante) ;
* quota compris entre 0 et 1 ;
* la somme des quotas d'un serveur est inférieure à 1 ;
* …
* **Algèbre relationnelle : tous les opérateurs algébriques utilisables :**
* Union
* Intersection
* Différence
* Projection
* Restriction
* Jointure
* Produit Cartésien
* Division
* Attention : Union, Intersection, Différence ne marchent que si les ensembles ont le même type
* **Exercice : Gestion du personnel**
* Lister les employés ayant des revenus supérieurs à 10.000 euros.
* R = Rest(EMP, SAL+COM≥10000)
* Trouver le nom et la profession de l'employé numéro 10.
* R = Proj(Rest(EMP, ENO=10), ENOM, PROF)
* Lister les noms des employés qui travaillent à Paris.
* R1 = Join(EMP, DEPT, EMP.DNO = DEPT.DNO) *Notes : R1 a 2 DNO en attributs, typiquement les SGBD les renommeraient en leur ajoutant un chiffre, type DNO1, DNO2 ; impossible d'imposer VILLE = 'Paris' comme condition pour une jointure.*
* R2 = Rest(R1, VILLE='Paris')
* R3 = Proj(R2, ENOM)
* Trouver le nom du directeur du département ''Commercial''.
* R1 = Rest(DEPT, DNOM='Commercial')
* R2 = Join(R1, EMP, R1.DIR=EMP.ENO)
* R3 = Proj(R2, ENOM)
* Trouver les professions des directeurs des départements.
* R1 = Join(DEPT, EMP, DEPT.DIR=EMP.ENO)
* R2 = Proj(R1, PROF)
* Trouver le nom des directeurs de département ayant comme profession "Ingénieur".
* R1 = Rest(EMP, PROF='Ingénieur')
* R2 = Join(R1, DEPT, R1.ENO=DEPT.DIR)
* R3 = Proj(R2, ENOM)
* **Exercice : Inviter ses amis**
* Quels sont les invités du repas du 01/05/97 ?
* R1 = Rest(REPAS, date='01/05/97')
* R2 = Proj(R1, invité)
* Quels sont les plats qui ont été servis à Alice ?
* R1 = Rest(REPAS, invité='Alice')
* R2 = Join(MENU, R1, MENU.date=R1.date)
* R3 = Proj(R2, plat)
* Quels sont les invités qui lors d'un repas ont eu au moins un de leur plat préféré ?
* R1 = Join(REPAS, MENU, date)
* R2 = Join(R1, PREFERENCE, plat)
* R3 = Rest(R2, invité=personne)
* R4 = Proj(R3, invité)
* Quelles sont les personnes qui n'ont jamais été invitées ?
* R1 = Proj(PREFERENCE, personne)
* R2 = Proj(REPAS, invité)
* R3 = R1 - R2
* Quels sont les invités qui sont venus à tous les repas ?
* R1 = REPAS / Proj(MENU, date) *"Tous les machins qui tous les bidules"*
* **Exercice : Faire du cinéma**
* Lister les films français (titre, année, réalisateur).
* R1 = Rest (LESFILMS, PAYS='France')
* R2 = Proj(R2, TITRE, ANNEE, REALISATEUR)
* Donnez les années de sortie des fims où a joué GABIN.
* Proj (Join (Rest (LESACTEURS, ACTEUR='GABIN')), LESFILMS, TITRE), ANNEE)
* Trouver les acteurs qui ont tourné avec TRUFFAUT comme réalisateur.
* R1 = Rest(LESFILMS, REALISATEUR='TRUFFAUT')
* R2 = Join(R2, LESACTEURS, TITRE)
* R3 = Proj(R2, ACTEUR)
* Trouver tous les acteurs qui ont été partenaires de DENEUVE.
* R1 = Rest(LESACTEURS, ACTEUR='DENEUVE')
* R2 = Join(LESACTEURS, R1, TITRE)
* R3 = Proj(R2, ACTEUR)
* R4 = R3 - {Deneuve}
* Lister les films dans lesquels le réalisateur est aussi acteur.
* R1 = Join(LESFILMS, LESACTEURS, TITRE)
* R2 = Rest(R1, ACTEUR=REALISATEUR)
* R3 = Proj(R2, LESACTEURS.ACTEUR)
* Lister les réalisateurs n'ayant joué comme acteurs que dans des films qu'ils ne réalisaient pas eux-mêmes.
* R1 = Join(LESFILMS, LESACTEURS, REALISATEUR=ACTEUR)
* R2 = Rest(R1, ACTEUR=REALISATEUR)
* R3 = R1 - R2
* R4 = Proj(R3, ACTEUR)
* Lister les réalisateurs ayant joué comme acteurs dans des fims qu'ils ne réalisaient pas eux-mêmes.
* R1 = Join(LESFILMS, LESACTEURS, REALISATEUR=ACTEUR)
* R2 = Rest(R1, LESACTEURS.TITRE≠LESFILMS.TITRE)
* R3 = Proj(R2, REALISATEUR)
* Donnez les acteurs qui jouent dans tous les fims de TRUFFAUT.
* LESACTEURS / Proj(Rest(LESFILMS, REALISATEUR='Truffaut'), Titre)
* **Exercice : Représentation des représentants**
* Soit le schéma relationnel suivant :
* REPRESENTANTS (#NR, NOMR, VILLE)
* PRODUITS (#NP, NOMP, COUL, PDS)
* CLIENTS (#NC, NOMC, VILLE)
* VENTES (#NR=>REPRESENTANTS, #NP=>PRODUITS, #NC=>CLIENTS, QT)
* Ecrire en SQL les requêtes permettant d'obtenir les informations suivantes.
* Tous les détails de tous les clients.
* SELECT \* FROM CLIENTS;
* Les numéros et les noms des produits de couleur rouge et de poids supérieur à 100.
* SELECT NP, NOMP FROM PRODUITS WHERE UPPER(COUL)='ROUGE' AND PDS>100;
* **UPPER(str)** : *met la chaîne str en majuscules. Très pratique dans la mesure où dans une chaîne de caractères, la casse est importante.*
* **LOWER(str)** : *met la chaîne str en minuscules.*
* Les différents représentants ayant vendu au moins un produit.
* SELECT DISTINCT NR FROM VENTES;
* Les noms des clients de Lyon ayant acheté un produit pour une quantité supérieure à 180.
* SELECT DISTINCT C.NOMC FROM CLIENTS C JOIN VENTES V ON C.NC=V.NC WHERE UPPER(C.VILLE)='LYON' AND V.QT>180;
* ***Autre moyen*** *: avec un produit cartésien :*
* SELECT DISTINCT C.NOMC FROM CLIENTS C, VENTES V WHERE C.NC=V.NC AND UPPER(C.VILLE)='LYON' AND V.QT>180;
* Les noms des représentants et des clients à qui ces représentants ont vendu un produit de couleur rouge pour une quantité supérieure à 100.
* SELECT C.NOMC, R.NOMR FROM CLIENTS C, REPRESENTANTS R, VENTES V, PRODUITS P WHERE C.NC=V.NC AND R.NR=V.NR, V.NP=P.NP AND UPPER(P.COUL)='ROUGE' AND V.QT>100;
* ***Autre moyen*** *: avec les jointures* :
* SELECT C.NOMC, R.NOMR FROM ((CLIENTS C JOIN VENTES V ON C.NC=V.NC) JOIN PRODUITS P ON P.NP=V.NP) JOIN REPRESENTANTS R ON V.NR=R.NR WHERE UPPER(P.COUL)='ROUGE' AND V.QT>100;
* ***Note*** *: on peut renommer les jointures comme on nomme les tables.*
* Le nombre de clients.
* SELECT COUNT(NC) FROM CLIENTS;
* *Permet de compter le nombre de NC qu'il y a dans clients.*
* *Aussi :*
* SELECT COUNT(\*) FROM CLIENTS;
* Le nombre de produits de couleur rouge.
* SELECT COUNT(NP) FROM PRODUITS WHERE UPPER(COUL)='ROUGE'
* Le nombre de clients par ville.
* SELECT VILLE, COUNT(NC) FROM CLIENTS GROUP BY VILLE;
* *Opérateurs d'agrégats classique* : **COUNT, SUM, AVG**
* La quantité totale des produits rouge vendus par chaque représentant.
* SELECT V.NR, SUM(V.QT) AS QT\_ROUGE\_VENDUE FROM VENTES V, PRODUITS P WHERE UPPER(P.COUL)='ROUGE' AND V.NP=P.NP GROUP BY V.NR;
* ***Notes :***
* **WHERE** *avant* **GROUP BY**
* **AS** *permet de renommer ce qu'on projette*
* La quantité totale de produits rouges vendus pour chaque représentant ayant vendu plus de 10 fois des produits rouges (ayant réalisé plus de 10 ventes différentes de produits rouges).
* SELECT V.NR, SUM(V.QT) AS QT\_ROUGE\_VENDUE FROM VENTES V, PRODUITS P WHERE UPPER(P.COUL)='ROUGE' AND V.NP=P.NP GROUP BY V.NR HAVING COUNT(\*)>10;
* **HAVING** : *permet d'ajouter une condition supplémentaire sur un agrégat.*
* **Exercice : Gestion documentaire**
* En algèbre relationnel et en SQL, afficher toutes les informations présentes dans la base de données.
* SELECT \* FROM user, document, version
* ***Note*** *: si on veut faire appel dans la restriction à plusieurs attributs d'une table, il faut la mettre plusieurs fois dans le FROM et la renommer avec des noms différents !!!*
* Exemple : SELECT \* FROM user u1, document, version, user u2 WHERE document.id=version.doc AND d.creator=u1.login AND u2.login=v.contrib;
* En **algèbre relationnel** et en **SQL**, afficher le nom de l'auteur du document dont le titre est "Le Capital".
* SELECT u.name FROM document d, user u WHERE d.creator=u.login AND UPPER(d.title)='Le Capital';
* En **algèbre relationnel** et en **SQL**, afficher les logins des users qui sont à la fois creator et contrib et qui commencent par "k".
* SELECT DISTINCT d.creator FROM document d, version v WHERE d.creator=v.contrib AND d.creator LIKE 'k%';
* En **algèbre relationnel** et en **SQL**, Afficher les logins des users qui n'ont ni créé, ni contribué.
* SELECT login FROM user WHERE login NOT IN (SELECT creator FROM document) AND login NOT IN (SELECT contrib FROM version);
* ***Autre moyen :***
* SELECT u.login FROM user u WHERE NOT EXISTS (SELECT d.creator FROM document d WHERE u.login=d.creator) AND NOT EXISTS (SELECT v.contrib FROM version v WHERE u.login=v.creator)
* ***Notes :***
* portée de user u : toute la requête
* portée de document d et version v : seulement leur sous-requête
* ***Autre moyen :***
* SELECT login FROM user EXCEPT ((SELECT creator FROM document) UNION (SELECT contrib FROM version)
* ***Autre moyen :***
* SELECT u.login FROM (user u LEFT OUTER JOIN document d ON u.login=d.creator) LEFT OUTER JOIN version v ON u.login=v.contrib WHERE d.creator IS NULL AND v.contrib IS NULL;
* En **SQL**, afficher la dernière version (celle qui a le numéro le plus élevé) de chaque document, la liste sera triée par id des documents croissants.
* SELECT doc, MAX(num) FROM version GROUP BY doc ORDER BY doc ASC;
* En **SQL**, afficher le nombre de contributions pour chaque utilisateur, en face de leur nom, à condition que l'utilisateur ait contribué au moins 2 fois, la liste sera triée par nombre de contributions décroissant.
* SELECT u.name, COUNT(v.contrib) AS nb\_contrib FROM user u, version v WHERE v.contrib=u.login GROUP BY u.name HAVING nb\_contrib>2 ORDER BY nb\_contrib DESC;
* **Exercice : Représentation de représentants, deuxième partie**
* Les noms des vendeurs qui n'ont jamais rien vendu.
* SELECT R.NOMR FROM REPRESENTANTS R WHERE R.NR NOT IN (SELECT NR FROM VENTES, REPRESENTANTS);
* Les numéros de tous les produits tels qu'il n'en existe aucun autre ayant un poids plus faible.
* SELECT NP FROM PRODUITS WHERE PDS = (SELECT MIN(PDS) FROM PRODUITS);
* Les noms des représentants ayant vendu quelque chose à tous les clients.
* SELECT R.NR, R.NOMR FROM REPRESENTANTS R WHERE (SELECT COUNT(\*) FROM CLIENTS)=(SELECT COUNT DISTINCT(V.NC) FROM VENTES V WHERE V.NR=R.NR)
* ***Autre moyen*** *: R /* ∄ *c / (*∄ *v de R à C)*
* SELECT R.NR, R.NOMR FROM REPRESENTANTS R WHERE NOT EXISTS (SELECT NC FROM CLIENTS WHERE NOT EXISTS (SELECT \* FROM VENTES V WHERE V.NR=R.NR AND V.NC=C.NC));
* Les noms des représentants ayant vendu quelque chose aux clients NC1 et NC2.
* SELECT DISTINCT R.NOMR FROM REPRESENTANTS, VENTES V1, VENTES V2 WHERE R.NR=V1.NR AND R.NR=V2.NR AND V1.NC=NC1 AND V2.NC=NC2;
* Les numéros des produits vendus à un client par un représentant issu de la même ville.
* SELECT DISTINCT V.NP FROM CLIENTS C, REPRESENTANTS R, VENTES V WHERE C.VILLE=R.VILLE AND R.NR=V.NR AND C.NC=V.NC;
* Les numéros des produits vendus à un client de Lyon par un représentant de Lyon.
* SELECT DISTINCT V.NP FROM CLIENTS C, REPRESENTANTS R, VENTES V WHERE C.VILLE=R.VILLE AND R.NR=V.NR AND C.NC=V.NC AND UPPER(C.VILLE)='LYON';
* Les numéros des clients ayant acheté quelque chose à au moins un représentant issu de la même ville.
* SELECT DISTINCT C.NC FROM CLIENTS C, REP R, VENTES V WHERE C.VILLE=R.VILLE AND R.NR=V;NR AND V.NC=C.NC;
* Les numéros des clients qui n'ont rien acheté à un représentant de Lyon.
* SELECT C.NC FROM CLIENTS C WHERE NOT EXISTS (SELECT \* FROM VENTES V, REPRESENTANTS R WHERE V.NC=C.NC AND V.NR=R.NR AND UPPER(R.VILLE)='LYON');
* Les couples des représentants et le nom de la ville dans laquelle le premier a vendu plus que le second.
* SELECT R1.NR, R2.NR, C.VILLE FROM REPRESENTANTS R1, REPRESENTANTS R2, CLIENTS C WHERE (SELECT COUNT(\*) FROM VENTES V, CLIENTS C2 WHERE V.NR=R1.NR AND C2.NC=V.NC AND C2.VILLE=C.VILLE ) > (SELECT COUNT(\*) FROM VENTES V, CLIENTS C2 WHERE V.NR=N2.NR AND C2.NC=V.NC AND C2.VILLE=C.VILLE);
* **Exercice : Carnets de voyages**
* Une clé est un groupe minimum d'attributs qui permet d'identifier de façon unique un tuple.
* (numAuteur, numVille) est une clé.
* ⟶numAuteur, numVille, nomAuteur, prenomAuteur, nomVille, nomPays, description
* *Dépendance fonctionnelle élémentaire : dépendance fonctionnelle irréductible ni à gauche à ni à droite.*
* 1NF : Oui, les attributs de cette relation sont atomiques
* 2NF : Non, car, par exemple, numAuteur⟶nomAuteur
* Transformation en 3NF :
* Auteur(#numAuteur, nomAuteur, prenomAuteur)
* Ville(#numVille, nomVille, nomPays)
* CarnetVoyage(#numA=>Auteur(numAuteur), #numV=>Ville(numVille), desc)
* Code SQL :
* CREATE TABLE Auteur(
* numA int,
* nomA, varchar(50)
* prenomA, varchar(50),
* PRIMARY KEY(numA)
* CHECK (nomA NOT NULL));
* ou
* CREATE TABLE Auteur(
* numA int PRIMARY KEY,
* nomA varchar(50) NOT NULL,
* prenomA varchar(50));
* CREATE TABLE Ville(
* numV int,
* nomV varchar(50),
* nomPays varchar(50),
* PRIMARY KEY(numV));
* CREATE TABLE CarnetVoyage(
* numV int REFERENCES Villes(numV),
* numA int REFERENCES Auteurs(numA),
* desc VARCHAR(100),
* PRIMARY KEY (numV, numA),
* FOREIGN KEY (numV) REFERENCES Villes(numV),
* FOREIGN KEY (numA) REFERENCES Auteurs(numA));
* Trajet(#depart=>Ville(numV), #arrivee=>Ville(numV), #nom, distance)
* GRANT ALL PRIVILEGES ON Auteur TO Admin;
* Pareil ON Villes, CarnetVoyage, Trajet
* GRANT SELECT ONT Auteur TO User;
* **Exercice : Jeanne et Serge**
* Equipe(#nom : string)
* Joueur(#prenom : string, #equipe : string => Equipe(nom))
* Match(gagnant : string => Equipe(nom), perdant : string => Equipe(nom), date : date, score : int)
* Une équipe ne peut pas jouer contre elle-même
* Le score est 0 ou 1
* 2 à 6 joueurs par équipe
* CREATE TABLE Equipe(
* nom VARCHAR(50),
* PRIMARY KEY(nom));
* CREATE TABLE Joueur(
* prenom VARCHAR(30),
* equipe VARCHAR(50),
* FOREIGN KEY (equipe) REFERENCES Equipe(nom)
* PRIMARY KEY (prénom, equipe));
* CREATE TABLE Match(
* gagnant VARCHAR(50),
* perdant VARCHAR(50),
* date date NOT NULL,
* score int NOT NULL,
* PRIMARY KEY (gagnant, perdant),
* FOREIGN KEY (gagnant) REFERENCES Equipe(nom),
* FOREIGN KEY (perdant) REFERENCES Equipe(nom),
* CHECK ((score=0 OR score = 1) AND (gagnant <> perdant)));
* *Pour la contrainte 2 à 6 joueurs par équipe : utiliser un trigger. Il s'agit d'une opération SQL qui va être exécutée automatiquement à chaque fois qu'il y a un déclenchement. Ici, ce serait un "INSERT Joueur". Mais ça ne contraint pas le minimum de joueur, seulement le maximum.*
* R1 = PROJ(Match, gagnant)
* R2 = PROJ(Match, perdant)
* R3 = INETR(R1, R2)
* ou
* R1=JOIN(Match1, Match2, Match1.perdant=Match2.gagnant)
* R2=proj(R1, gagnant)
* SELECT M.gagnant, COUNT(\*)\*2 as nbPts
* FROM Match M
* GROUP BY M.gagnant
* ORDER BY nbPts DESC;
* **Exercice : Le chemin à l'envers**
* Clé primaire de t\_personnes : pk\_n, il s'agit d'une clé artificielle.
* numms est une clé candidate.
* conjoint est un attribut unique pour une donnée mais pas obligatoire, ce n'est dont pas une clé candidate.
* Leur domaine n'est pas le même.
* ALTER TABLE t\_personnes
* MODIFY conjoint number(4) UNIQUE;
* ALTER TABLE t\_personnes
* ADD FOREIGN KEY(conjoint) REFERENCES t\_personnes(pk\_n);
* Les DF :
* pk\_n détermine tout.
* numss aussi.
* chanteur\_préféré⟶nationalité\_chanteur\_préféré
* Csqce : 2NF oui, puisque les clés (primaire et candidate) sont atomiques.
* 3NF non, car chanteur\_préféré⟶nationalité\_chanteur\_préféré, et chanteur\_préféré n'est pas une clé, nationalité\_chanteur\_préféré non plus.
* Transformation en 3NF :
* Chanteur(#nom, nationalité)
* t\_personnes => on remet tout sauf chanteur\_préféré et nationalité\_chanteur\_préféré, à la place on met Chanteur=>Chanteur(nom).
* CREATE TABLE Chanteur(
* nom char(50),
* nationalité char(50),
* PRIMARY KEY(nom));
* INSERT INTO Chanteur (nom, nationalité)
* SELECT DISTINCT chanteur\_préféré, nationalité\_chanteur\_préféré
* FROM t\_personnes;
* ALTER TABLE t\_personnes
* DROP nationalité\_chanteur\_préféré;
* ALTER TABLE t\_personnes
* ADD FOREIGN KEY chanteur\_préféré REFRENCES Chanteur(nom);
* SELECT COUNT(\*)
* FROM t\_personnes p, t\_personnes c
* WHERE p.chanteur\_préféré = c.chanteur.préféré AND p.conjoint = cq.pk\_n