Module Introduction à la gestion des données
TD 3
Modélisation et Implémentation

Hervé Owsinski
Herve.Owsinski@uphf.fr

Semestre 1

2026-2027

Exercice

Enoncé

Contexte

L'office du tourisme du département de Haute-Savoie souhaite mettre en place un système d'information permettant de stocker les différentes randonnées proposées aux visiteurs.
Ces randonnées sont identifiées par un numéro unique, elles ont également un nom, un niveau de difficulté, un dénivelé et elles sont créées par un guide.
Chaque guide est identifié de façon unique par son nom et son prénom. Il doit avoir un diplôme (qui peut être différent en fonction des guides) et a un niveau.
Pour chaque randonnée, différentes sessions sont proposées durant la saison, ces sessions sont représentées par un numéro unique et possèdent un tarif et une date.
Pour chaque session, un guide encadrant sera défini.
Chaque visiteur, identifié par un numéro, peut réserver un certain nombre de places pour des sessions de randonnée.
Afin d'effectuer des statistiques, l'office de tourisme souhaite conserver pour chaque visiteur son nom, son prénom et son département d'origine.

TD Question 1 : Déterminer un Modèle Entité Association

Enoncé

Contexte

L'office du tourisme du département de Haute-Savoie souhaite mettre en place un système d'information permettant de stocker les différentes **randonnées proposées** aux **visiteurs**.
Ces **randonnées** sont identifiées par un **numéro unique**, elles ont également un **nom**, un **niveau de difficulté**, un **dénivelé** et elles **sont créées** par un **guide**.
Chaque **guide** est identifié de façon unique par **son nom et son prénom**. Il doit avoir un **diplôme** (qui peut être différent en fonction des guides) et a un **niveau**.
Pour chaque **randonnée**, différentes **sessions sont proposées** durant la saison, ces **sessions** sont représentées par un **numéro unique** et possèdent un **tarif** et une **date**.
Pour chaque **session**, un **guide encadrant** sera défini.
Chaque **visiteur**, identifié par un **numéro**, peut **réserver** un certain **nombre de places** pour des **sessions** de randonnée.
Afin d'effectuer des statistiques, l'office de tourisme souhaite conserver pour chaque **visiteur** son **nom**, son **prénom** et son **département** d'origine.

Un Enoncé : plusieurs interprétations possibles

Transformation du texte en schéma E/A : interprétation maximale

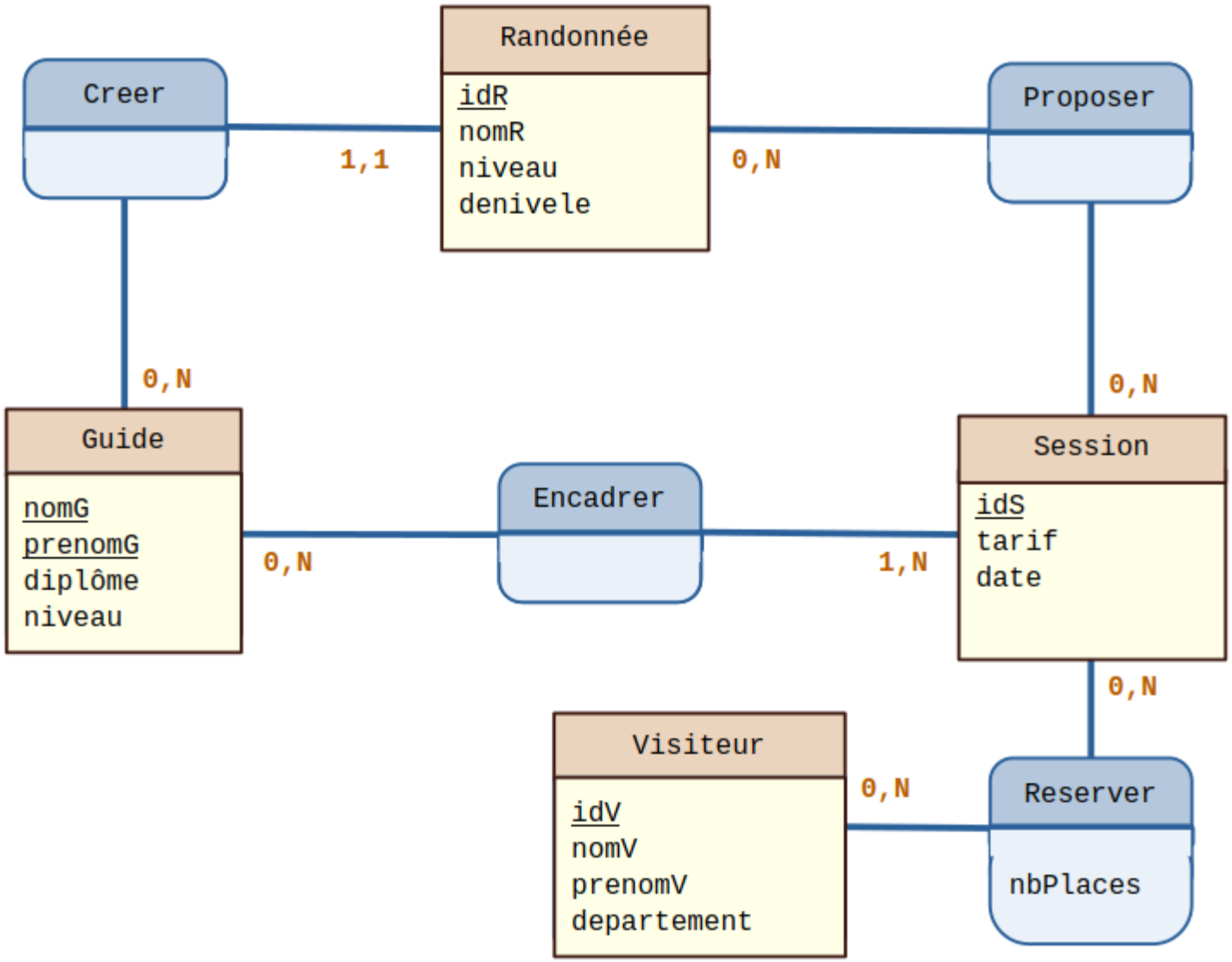
Les choix réalisés ici :

- Possibilité de créer de grands événements en permettant à une session d'être encadrée par plusieurs guides si nécessaire ;
- Possibilité de créer des sessions comportant plusieurs petites randonnées à la suite le même jour.

Pourquoi ? Le texte indique pas précisement "un et un seul" ou "un unique".

Un Enoncé : plusieurs interprétations possibles

Transformation du texte en schéma E/A : interprétation maximale



Un Enoncé : plusieurs interprétations possibles

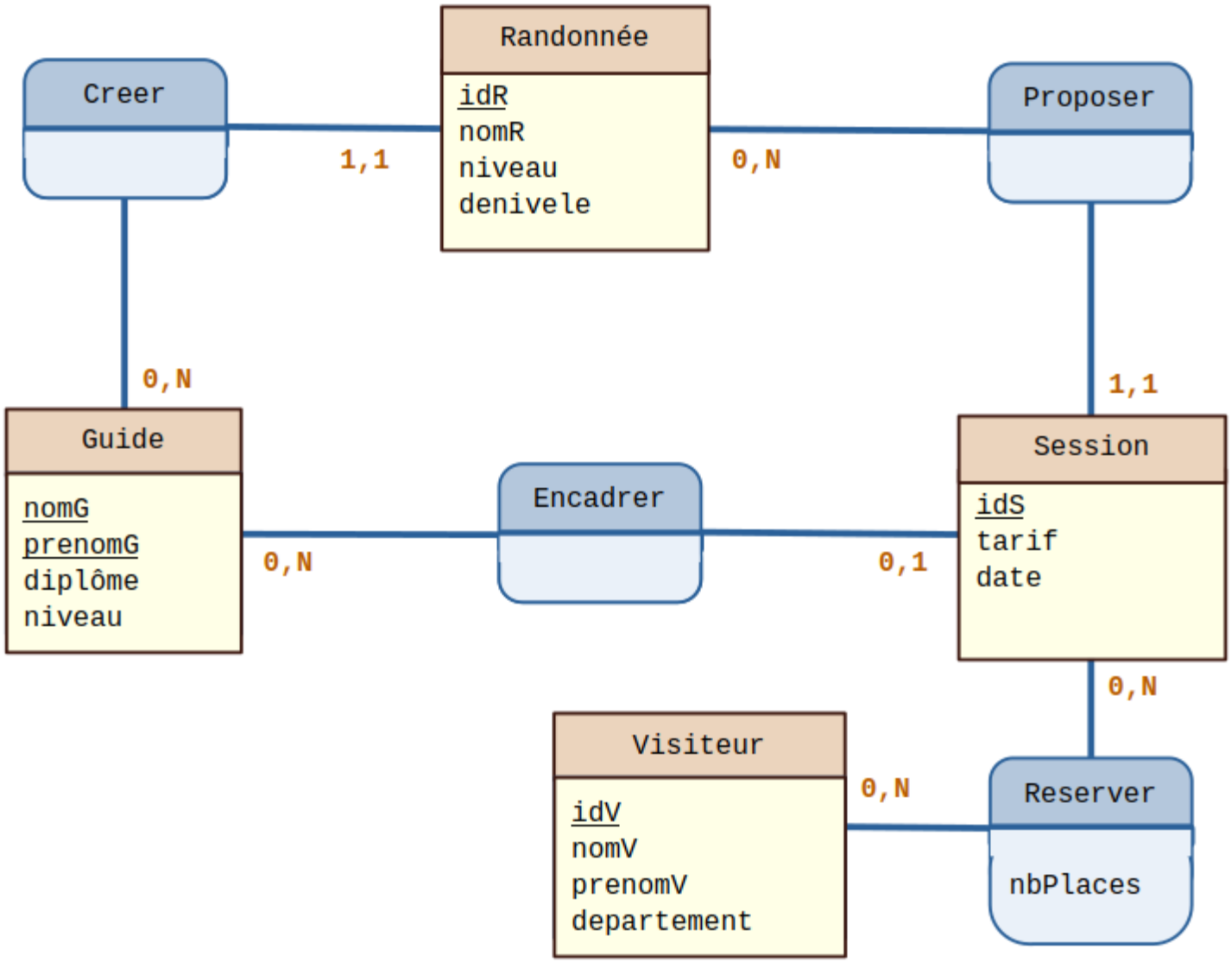
Transformation du texte en schéma E/A : interprétation minimale

Choix réalisés ici :

- Pas de grand événément : une session n'occupe qu'un guide ;
- Une session ne correspond qu'à une unique randonnée.

Un Enoncé : plusieurs interprétations possibles

Transformation du texte en schéma E/A : interprétation minimale

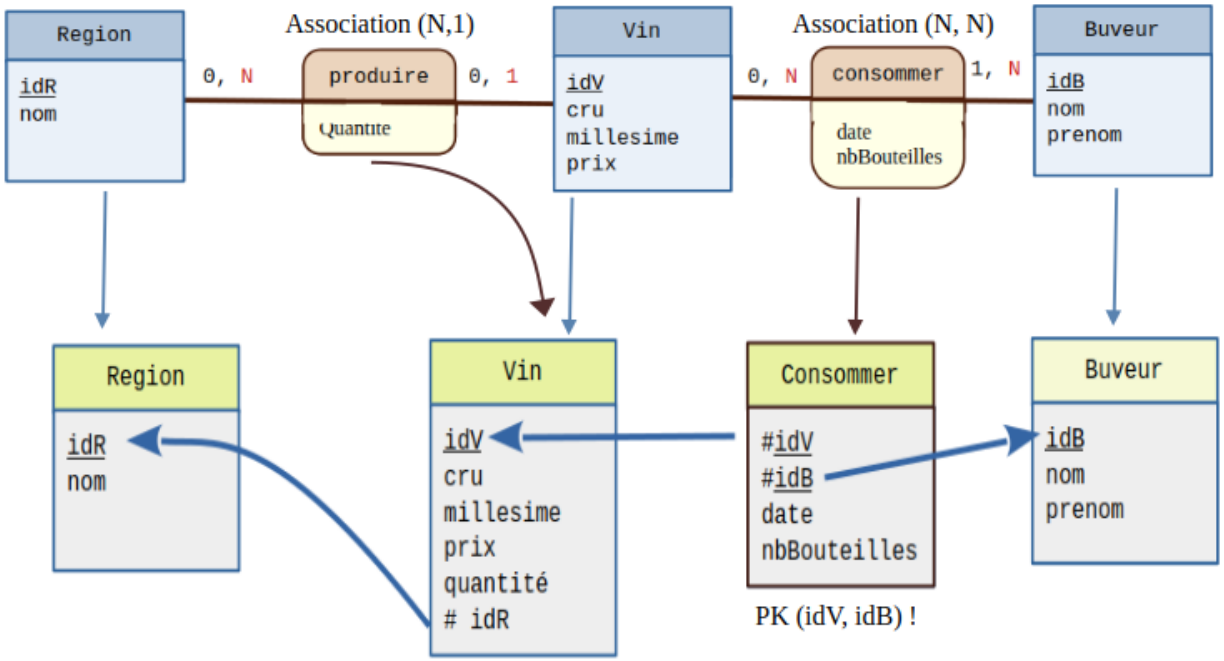


Rappels

Modèle E/A vers Modèle relationnel

Bilan à connaître par coeur

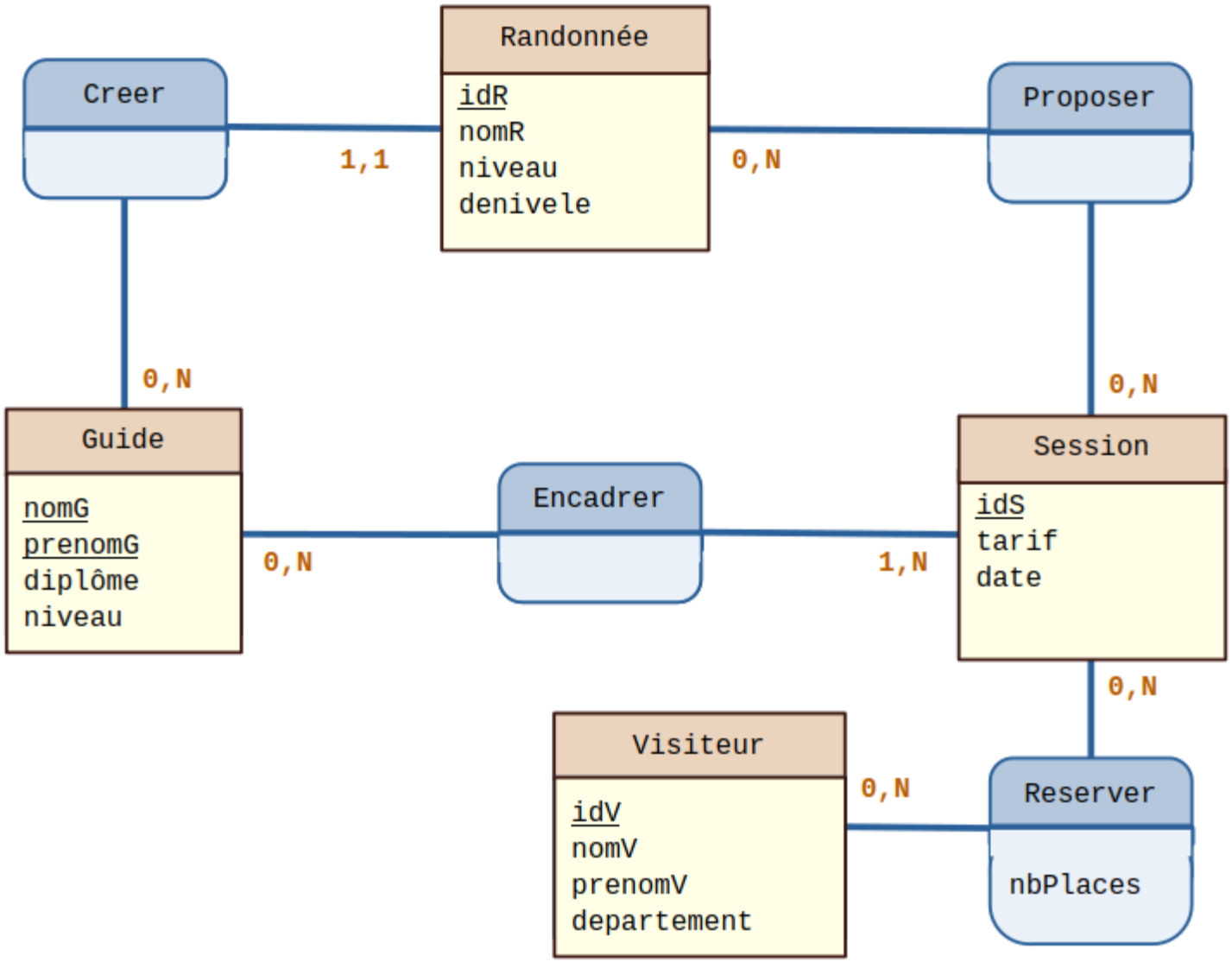
1. Entité → Relation ;
2. Association Many to One → disparait, infos absorbées du côté 1 ;
3. Association Many to Many → Relation,
   clé primaire composée de clés des relations qu'elle relie.



TD Question 2
Passage du MEA au modèle relationnel
Interprétation maximaliste

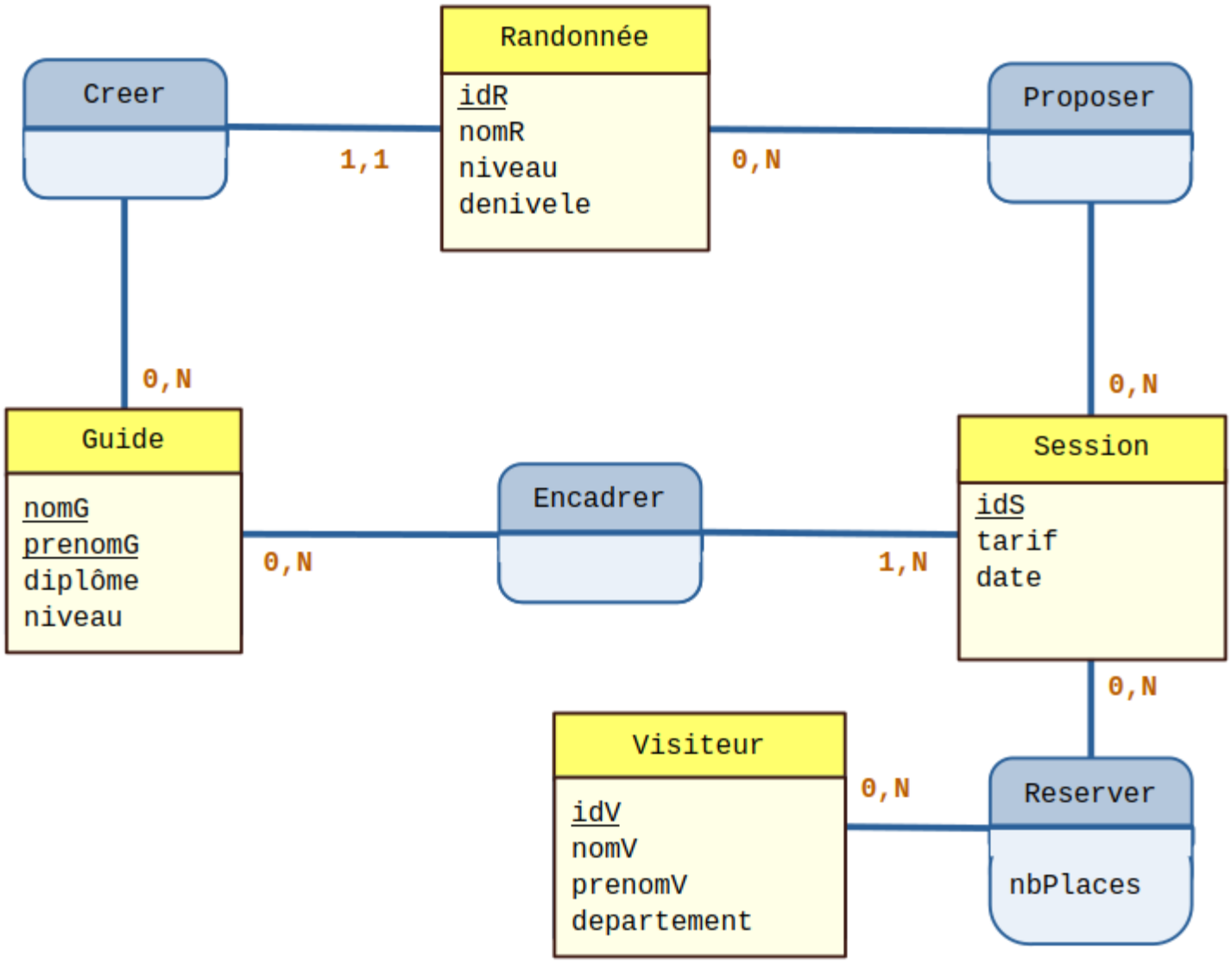
EA vers relationnel

Transformation en modèle relationnel de l'interprétation maximale



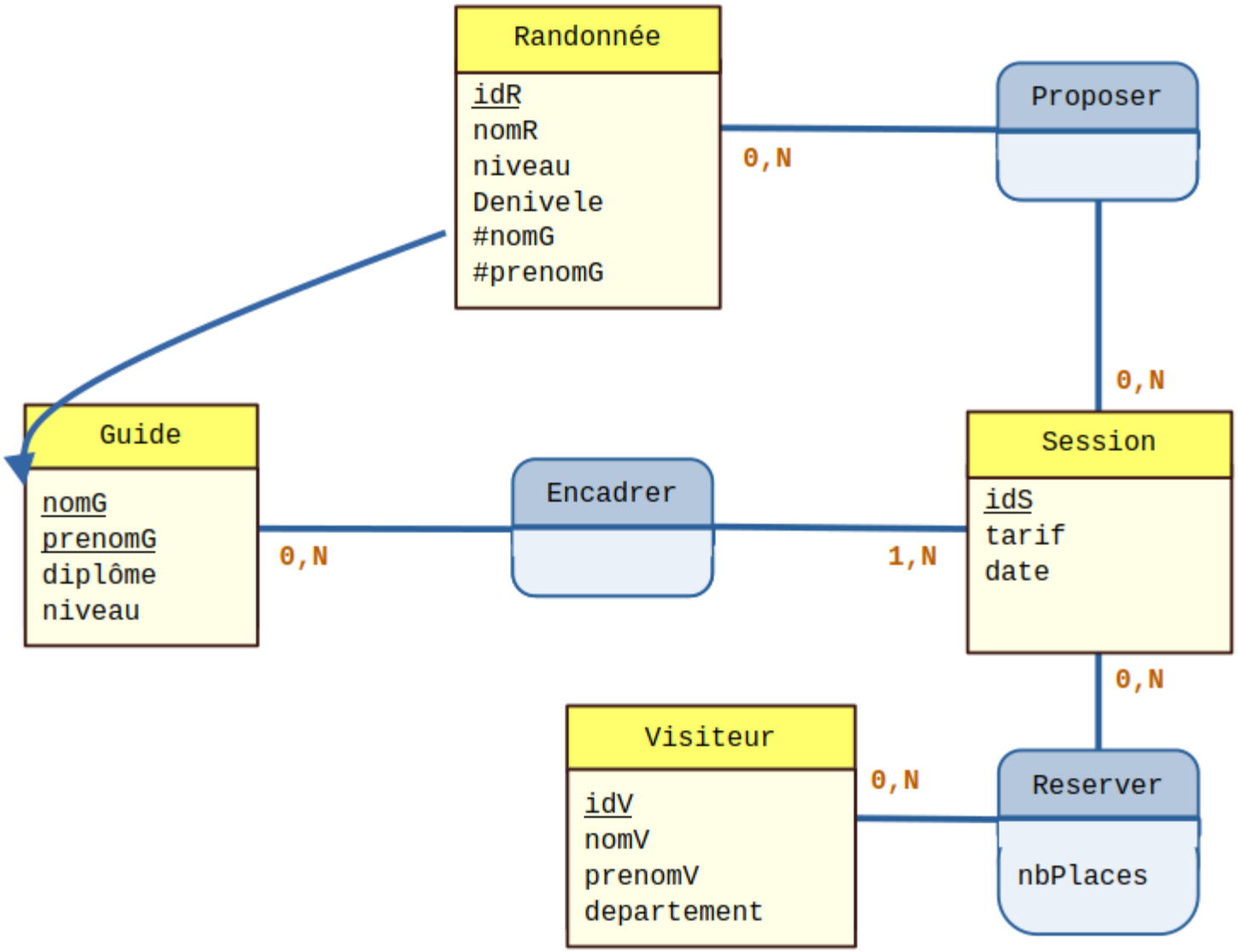
EA vers relationnel

Transformation en modèle relationnel de l'interprétation maximale



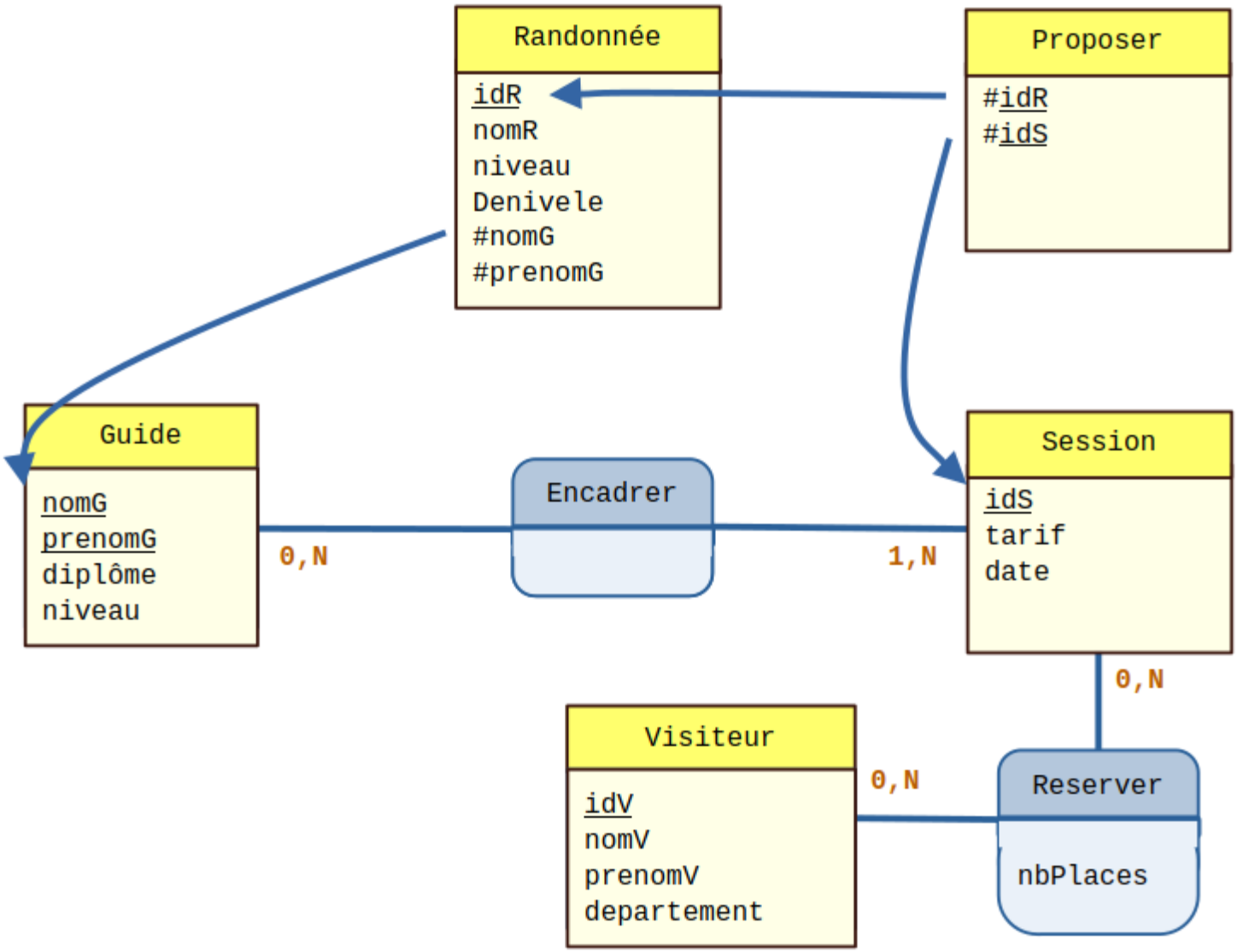
EA vers relationnel

Transformation en modèle relationnel de l'interprétation maximale



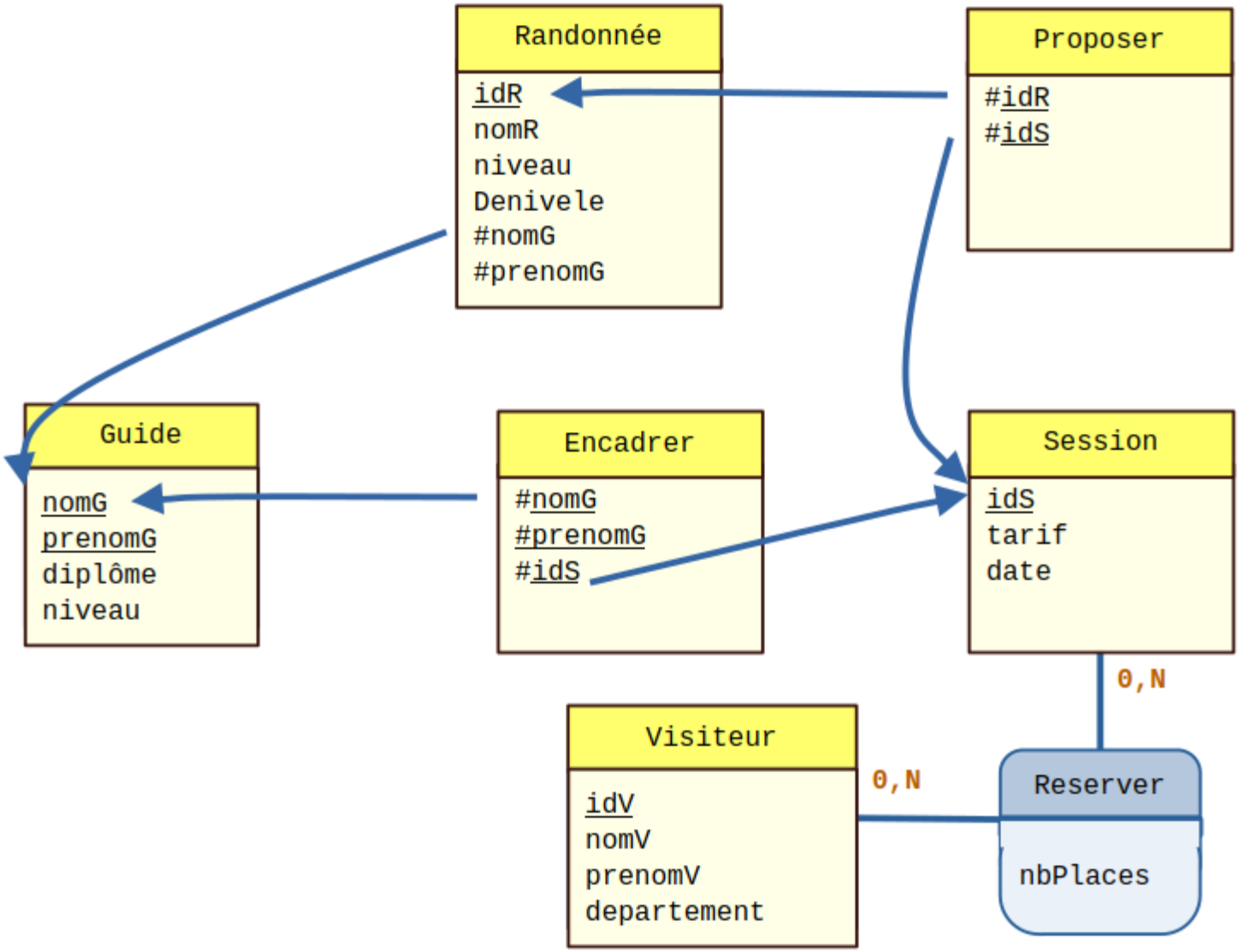
EA vers relationnel

Transformation en modèle relationnel de l'interprétation maximale



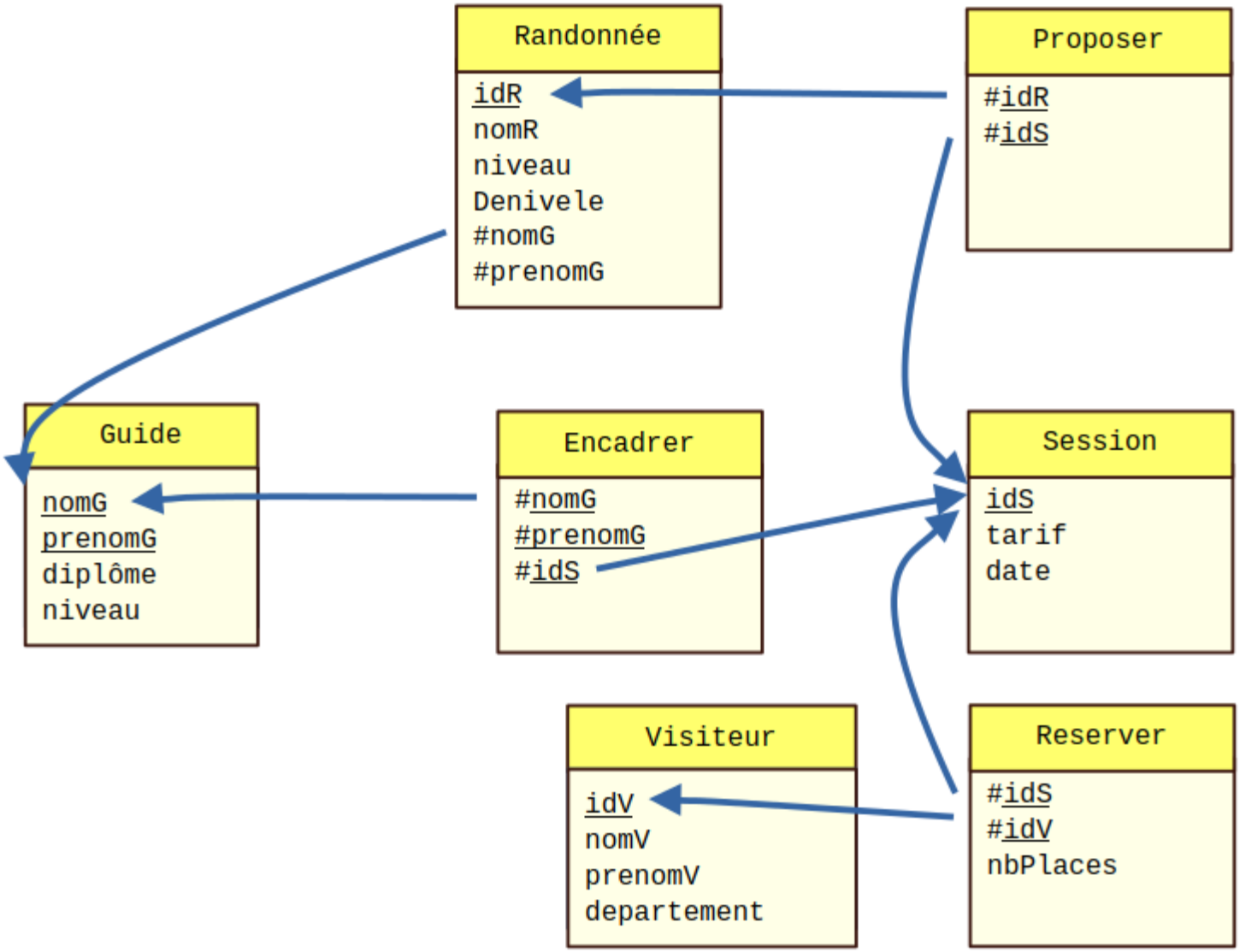
EA vers relationnel

Transformation en modèle relationnel de l'interprétation maximale



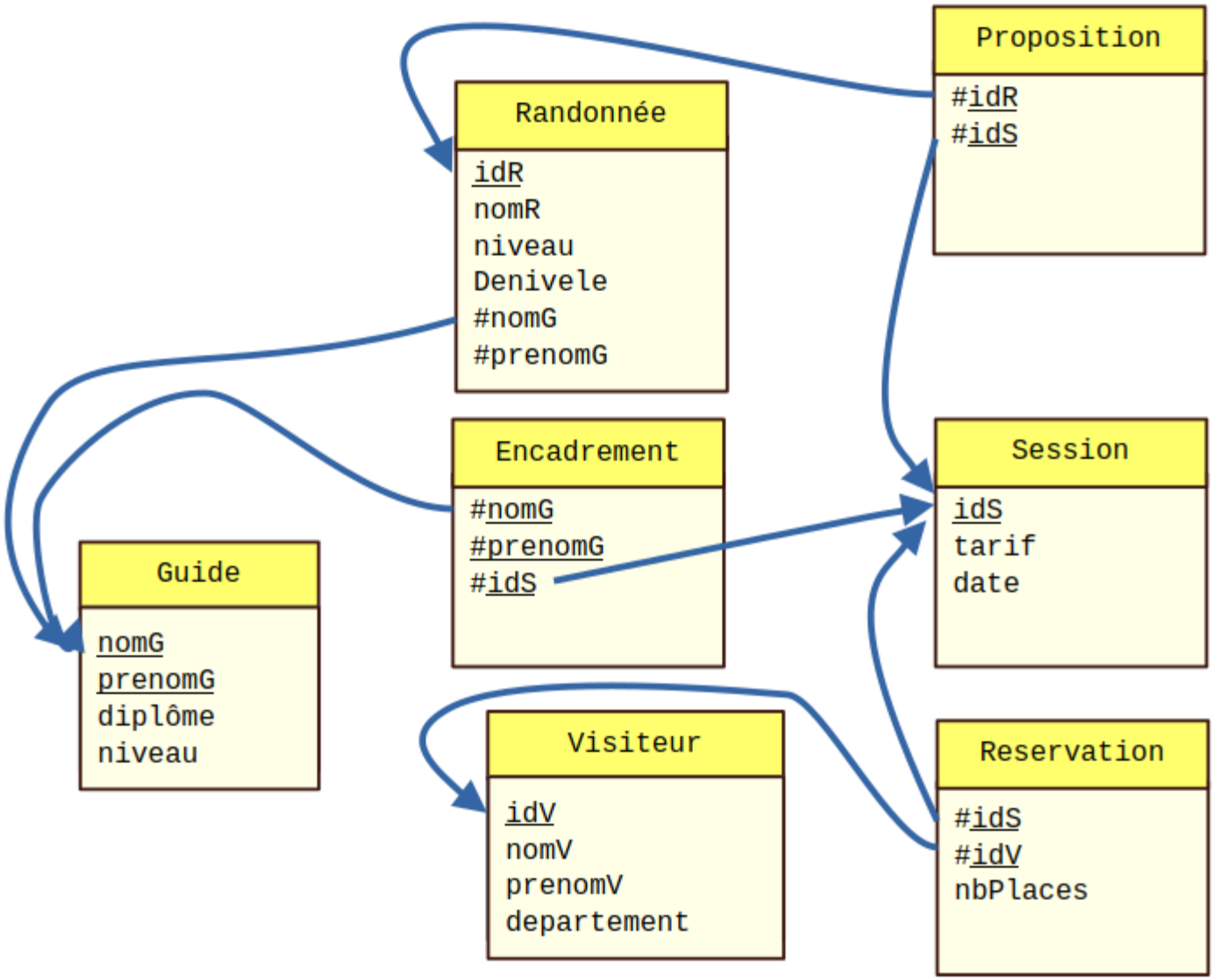
EA vers relationnel

Transformation en modèle relationnel de l'interprétation maximale



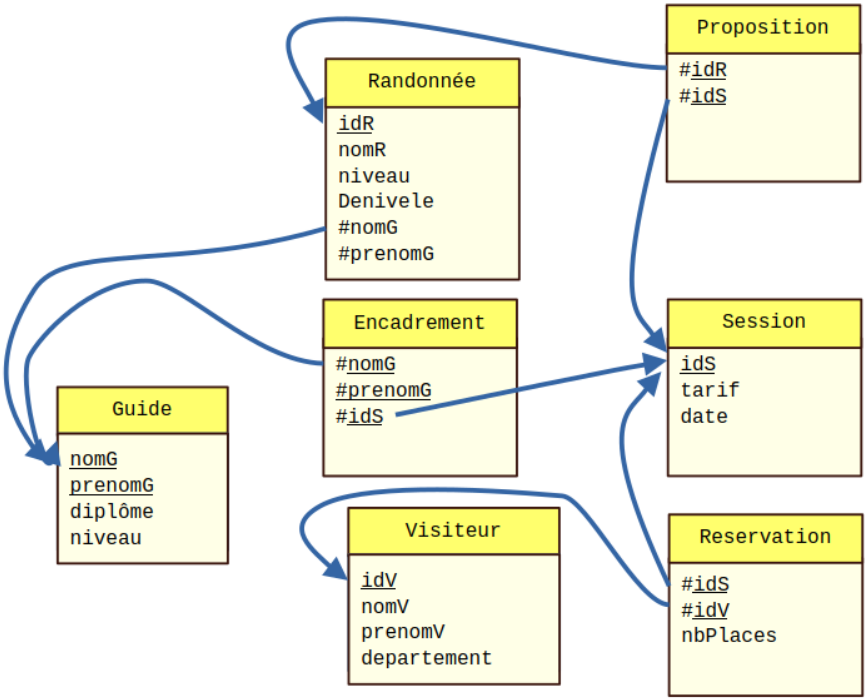
EA vers relationnel

Transformation en modèle relationnel de l'interprétation maximale



EA vers relationnel

Transformation en modèle relationnel de l'interprétation maximale

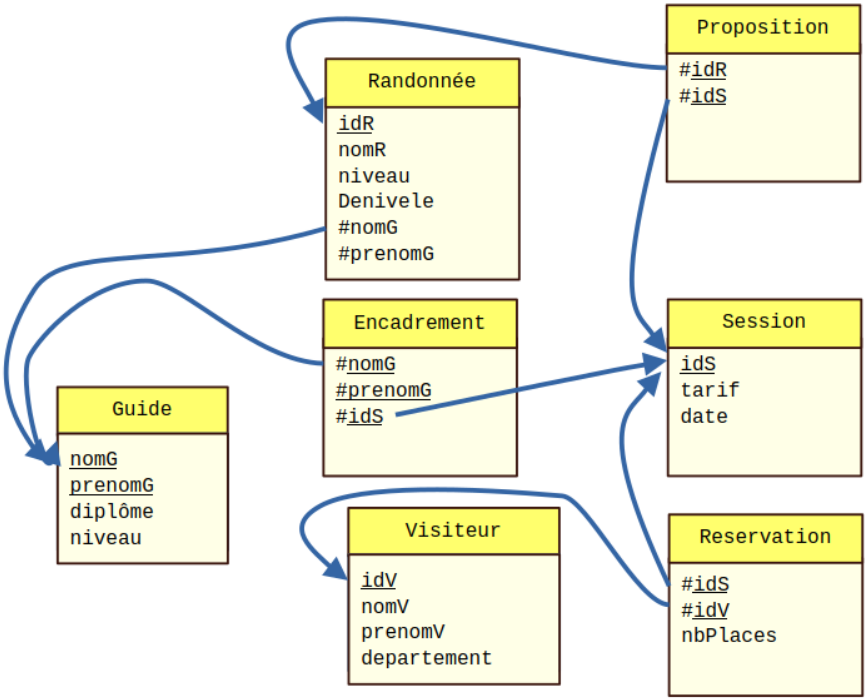


On cherche d'abord les relations qui n'ont pas de clés étrangères.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Session** (idS, tarif, date)
- .
- .
- .
- .

EA vers relationnel

Transformation en modèle relationnel de l'interprétation maximale

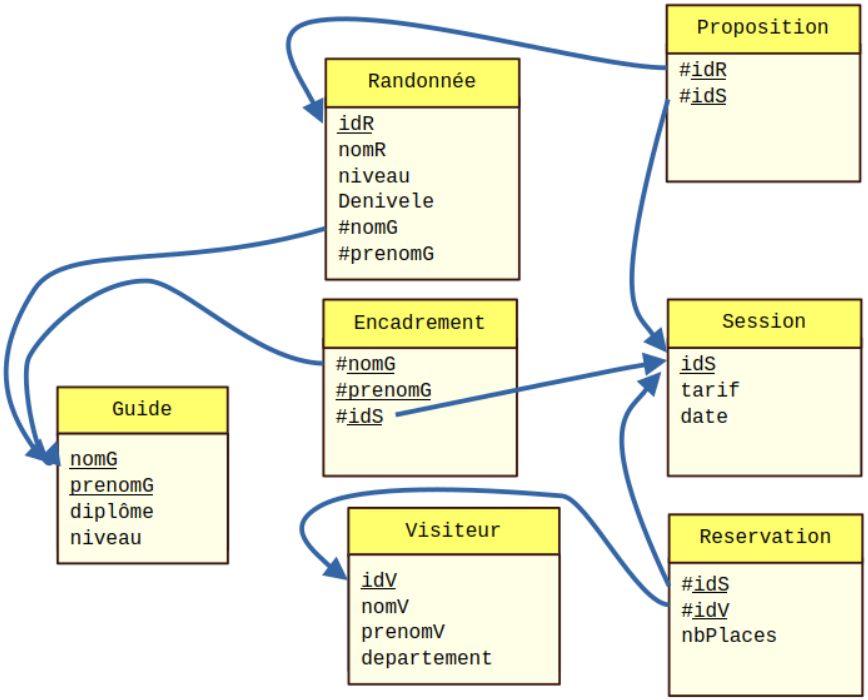


On cherche d'abord les relations qui n'ont pas de clés étrangères.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Session** (idS, tarif, date)
- .
- .
- .
- .

EA vers relationnel

Transformation en modèle relationnel de l'interprétation maximale

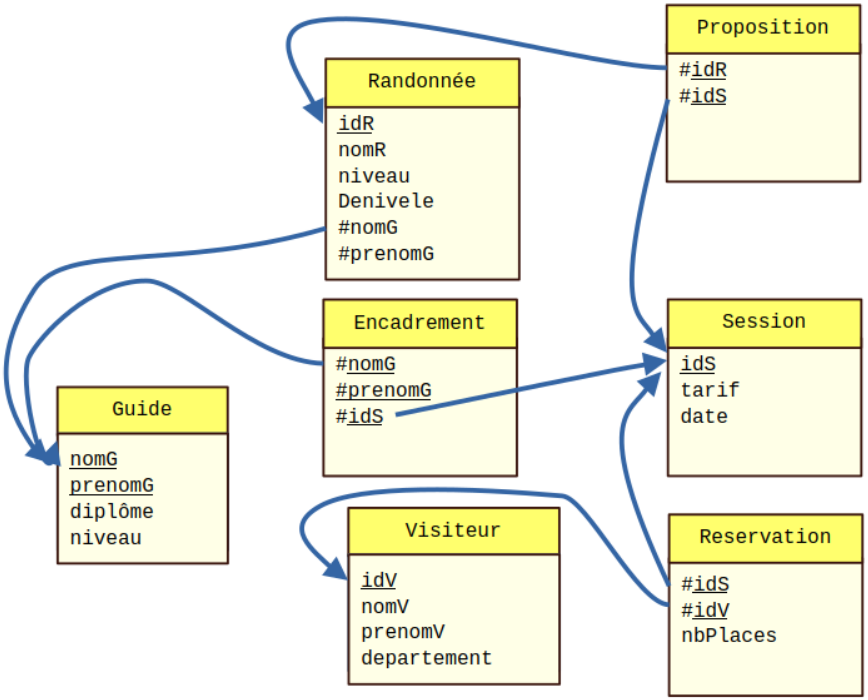


On cherche d'abord les relations qui n'ont pas de clés étrangères.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Session** (idS, tarif, date)
- .
- .
- .
- .

EA vers relationnel

Transformation en modèle relationnel de l'interprétation maximale

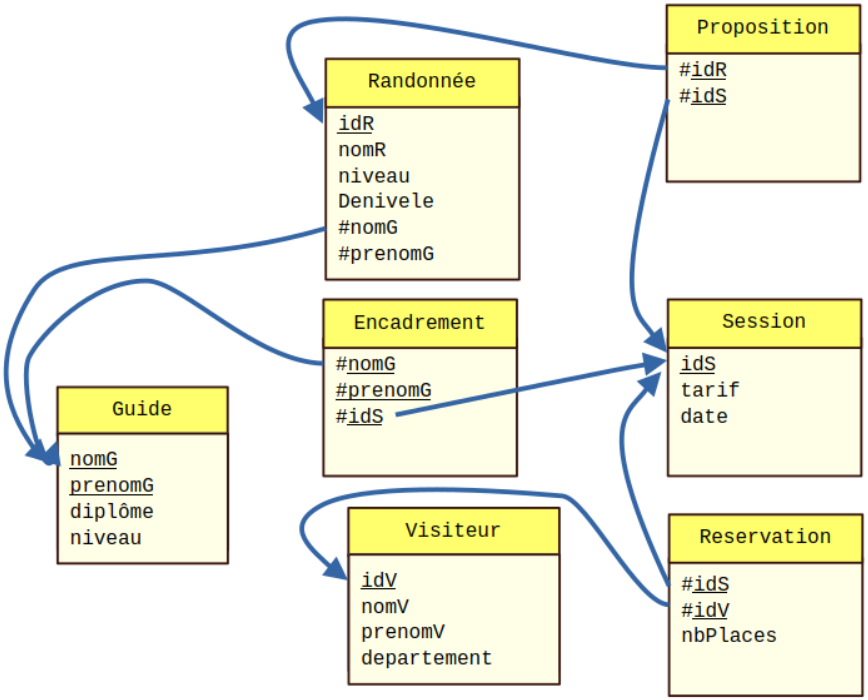


On rajoute peu à peu les dépendantes.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Session** (idS, tarif, date)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Encadrement** (#nomG, #prenomG, #idS)
- **Reservation** (#idS, #idV, nbPlaces)
- **Proposition** (#idR, #idS)

EA vers relationnel

Transformation en modèle relationnel de l'interprétation maximale



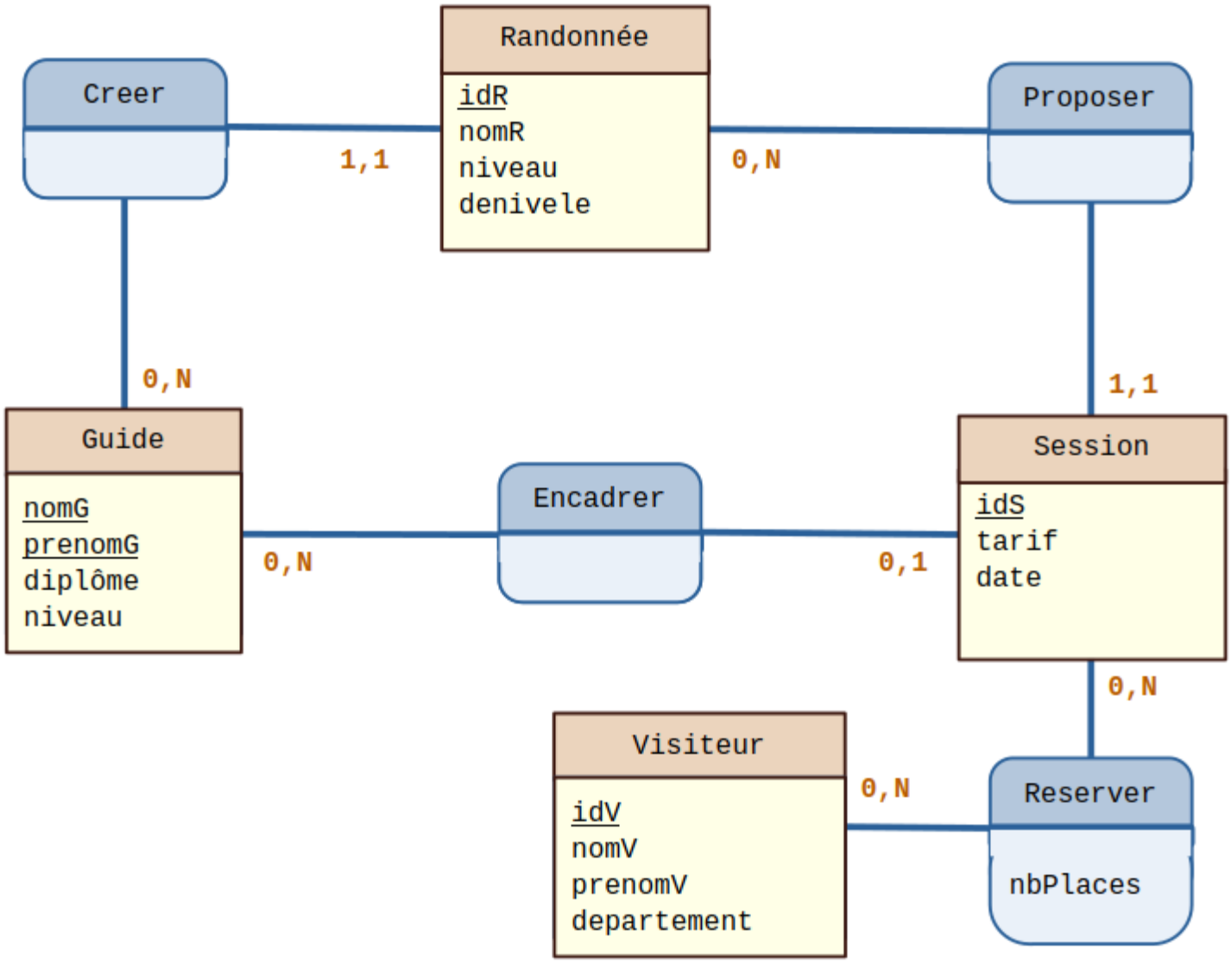
On rajoute peu à peu les dépendantes.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Session** (idS, tarif, date)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Encadrement** (#nomG, #prenomG, #idS)
- **Reservation** (#idS, #idV, nbPlaces)
- **Proposition** (#idR, #idS)

TD Question 2
Passage du MEA au modèle relationnel
Interprétation minimaliste

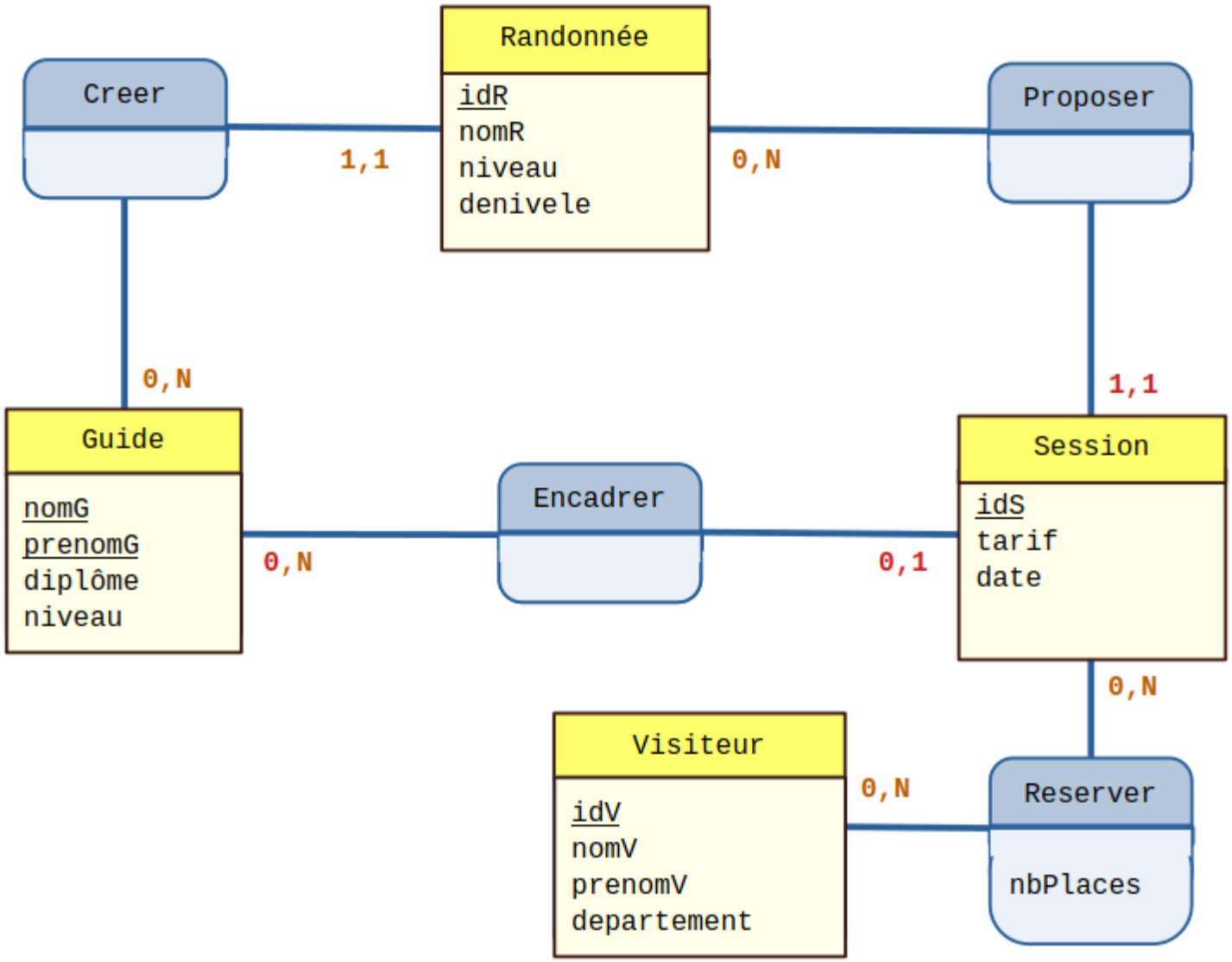
EA vers relationnel

Transformation en modèle relationnel de l'interprétation minimale



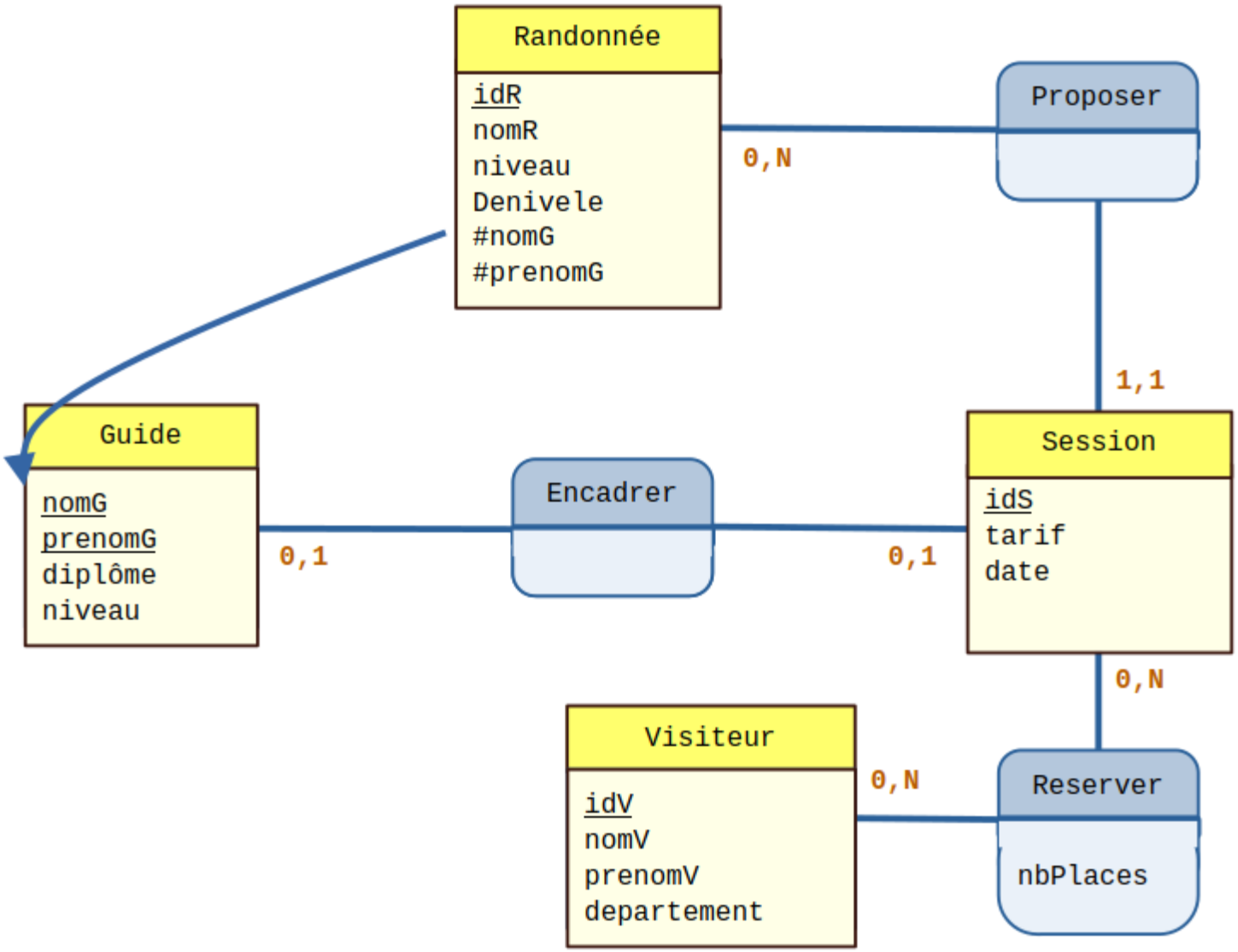
EA vers relationnel

Transformation en modèle relationnel de l'interprétation minimale



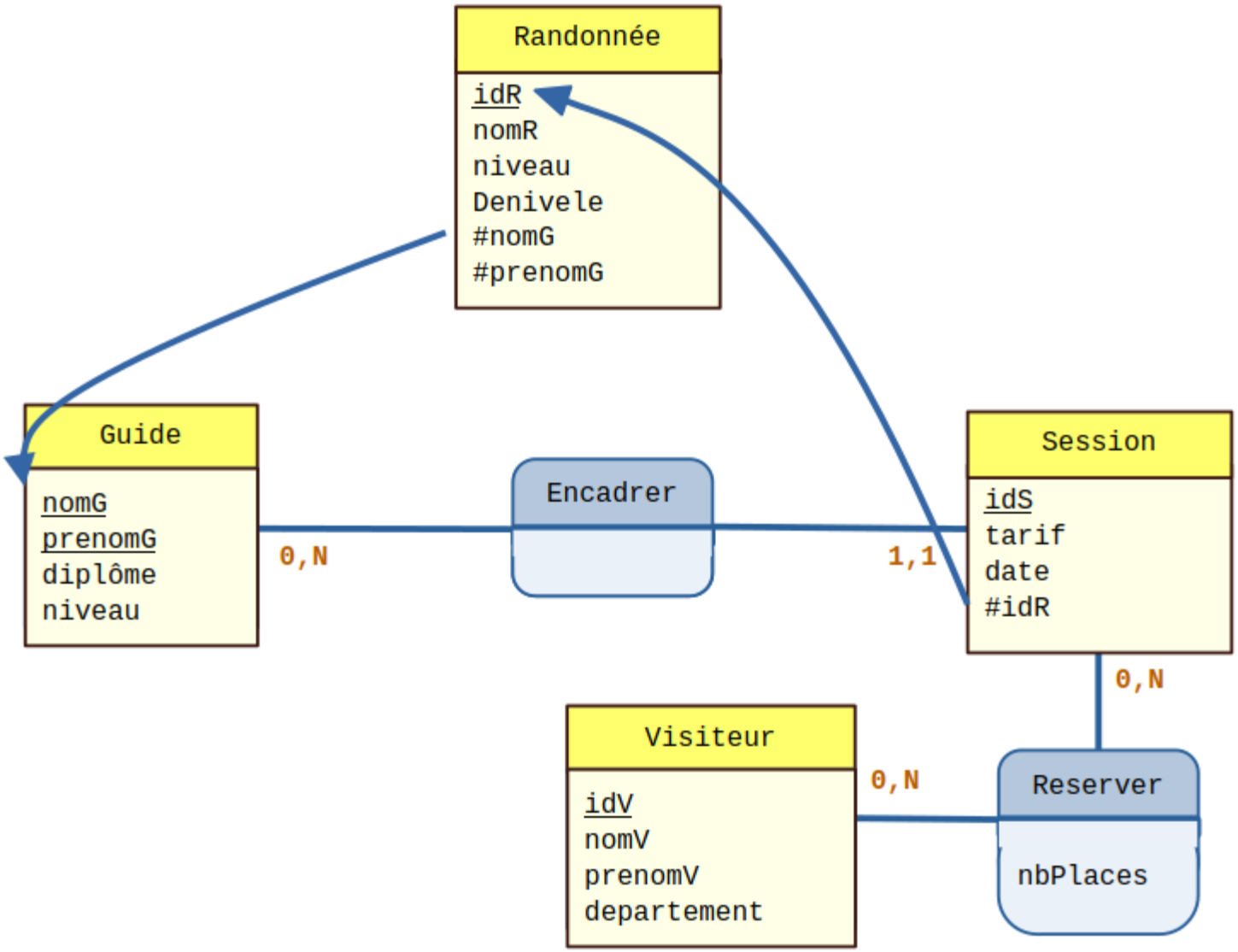
EA vers relationnel

Transformation en modèle relationnel de l'interprétation minimale



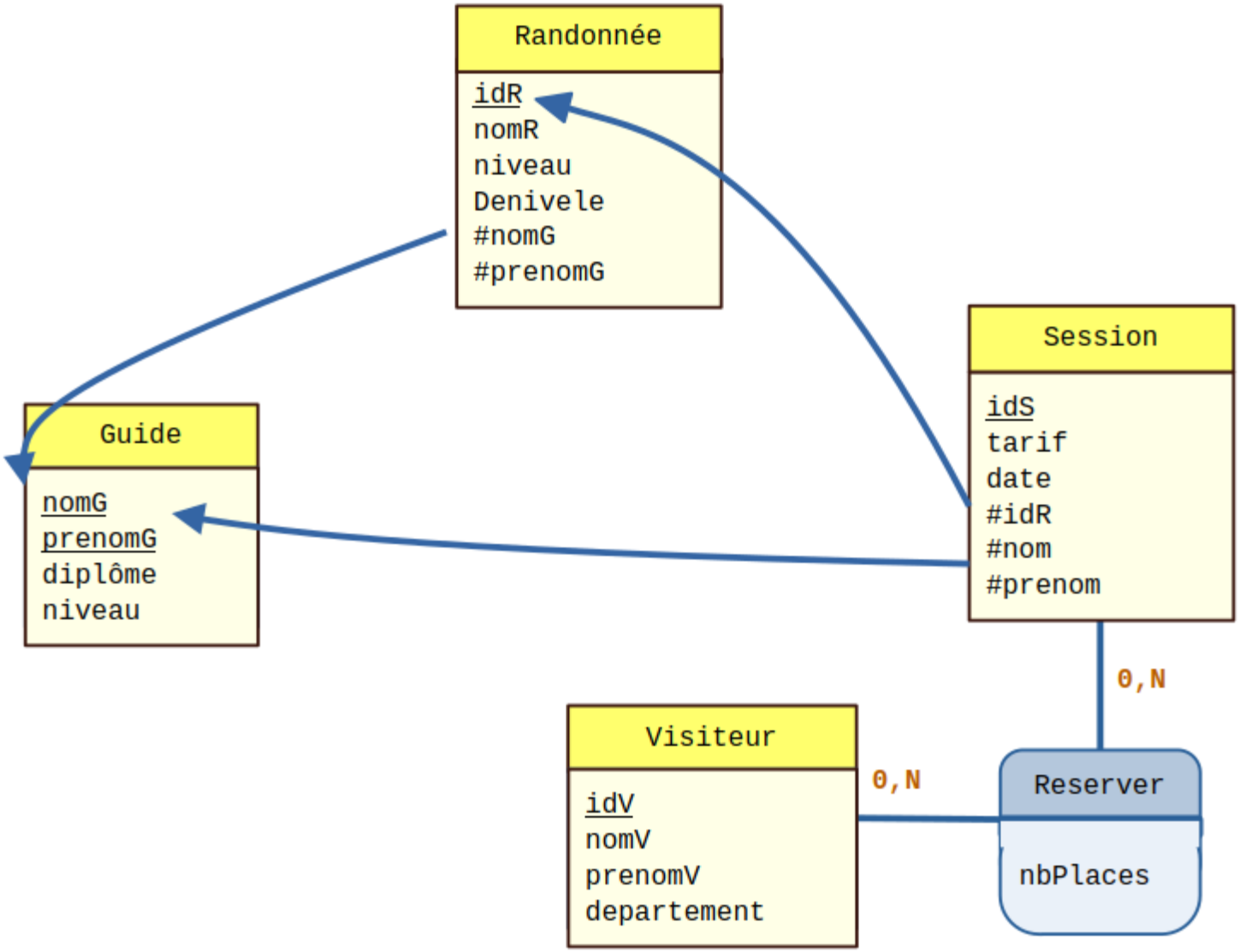
EA vers relationnel

Transformation en modèle relationnel de l'interprétation minimale



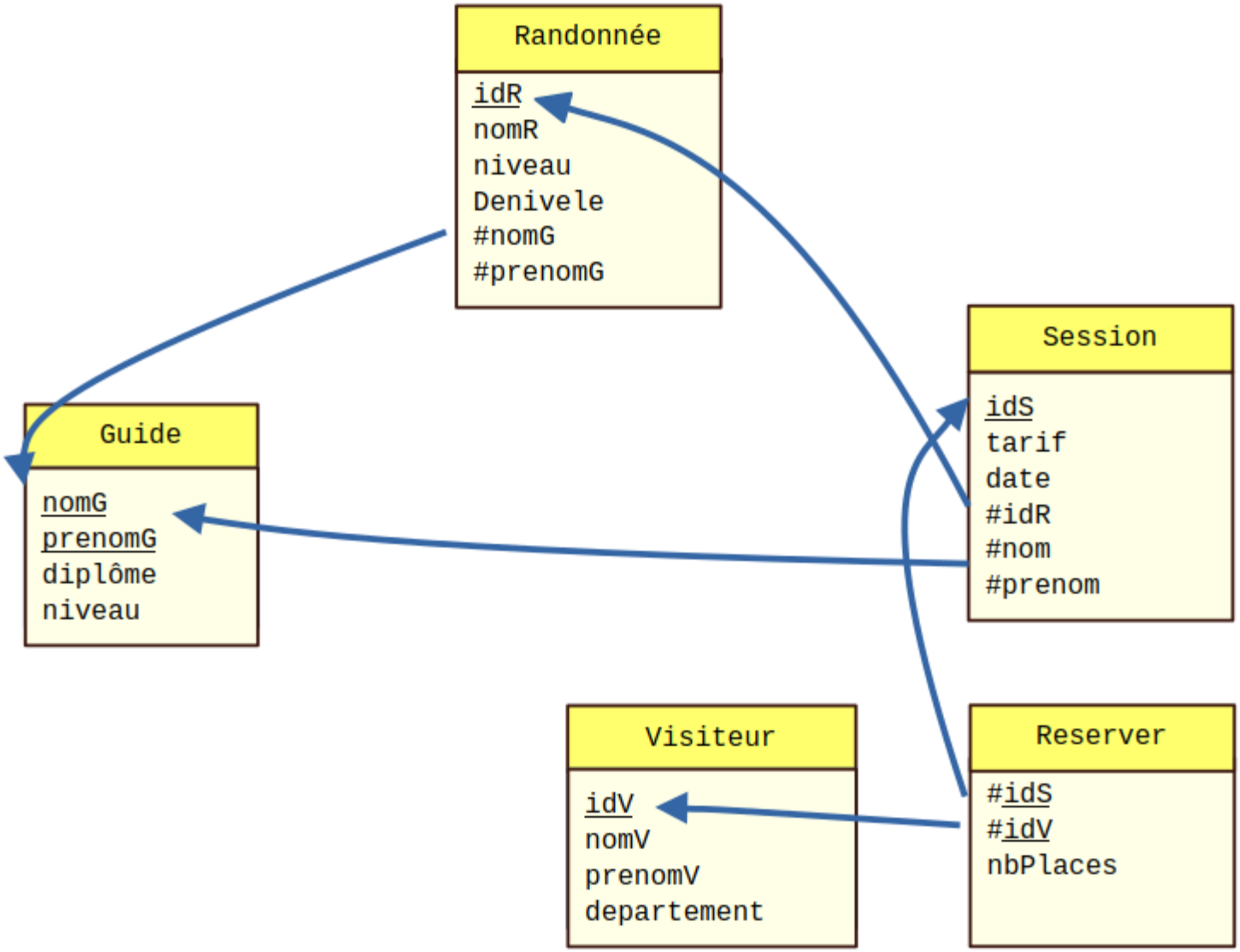
EA vers relationnel

Transformation en modèle relationnel de l'interprétation minimale



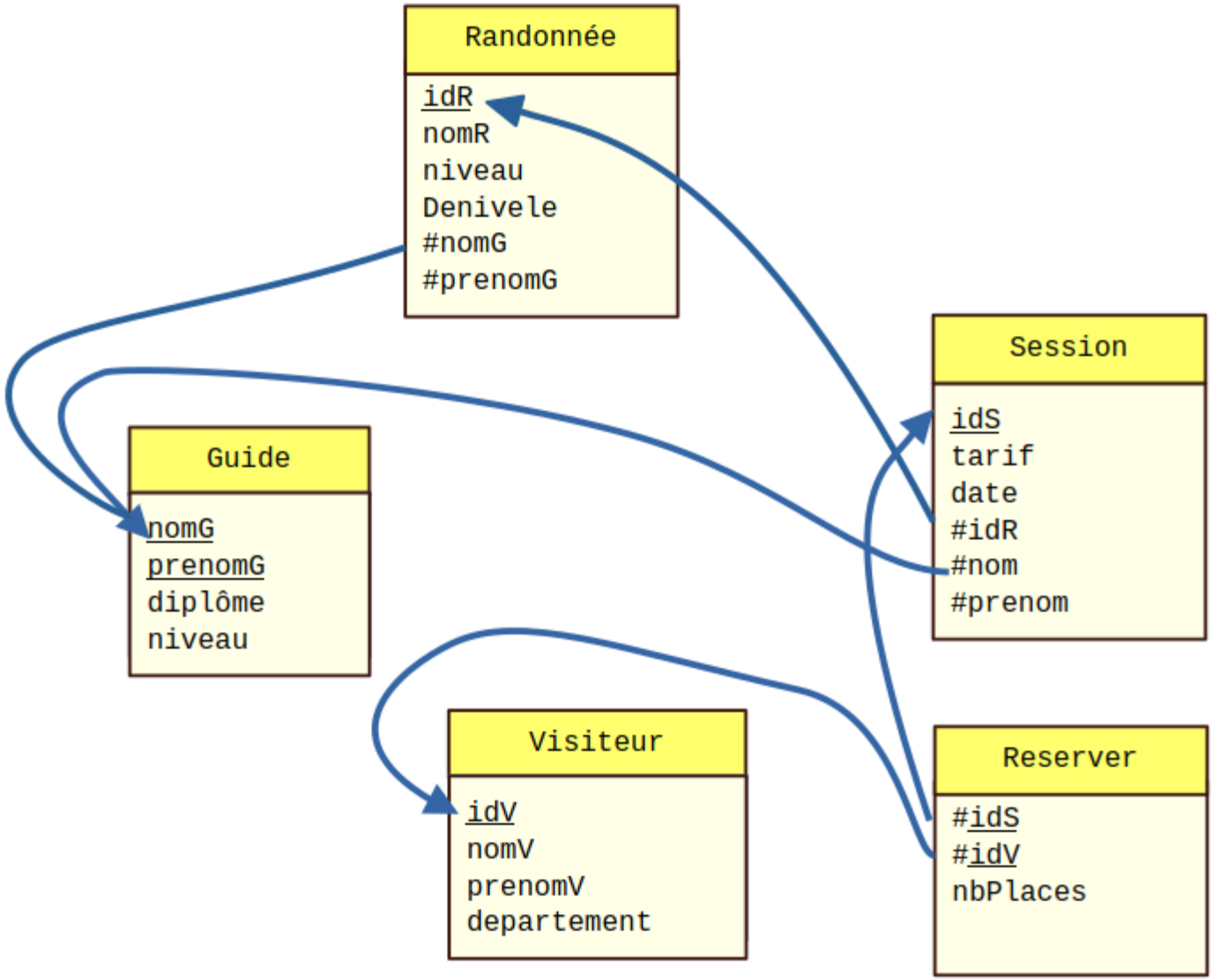
EA vers relationnel

Transformation en modèle relationnel de l'interprétation minimale



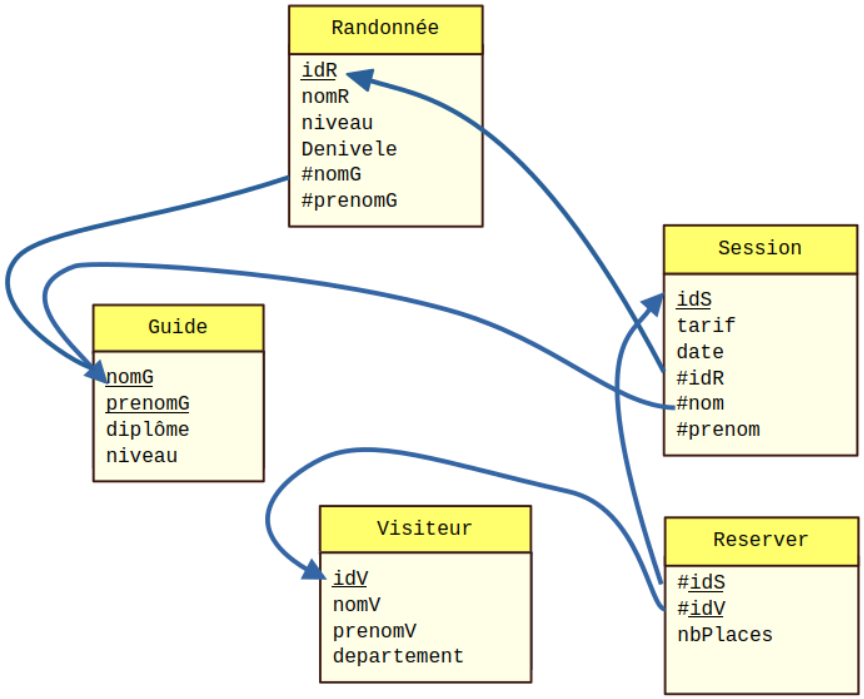
EA vers relationnel

Transformation en modèle relationnel de l'interprétation minimale



EA vers relationnel

Transformation en modèle relationnel de l'interprétation minimale

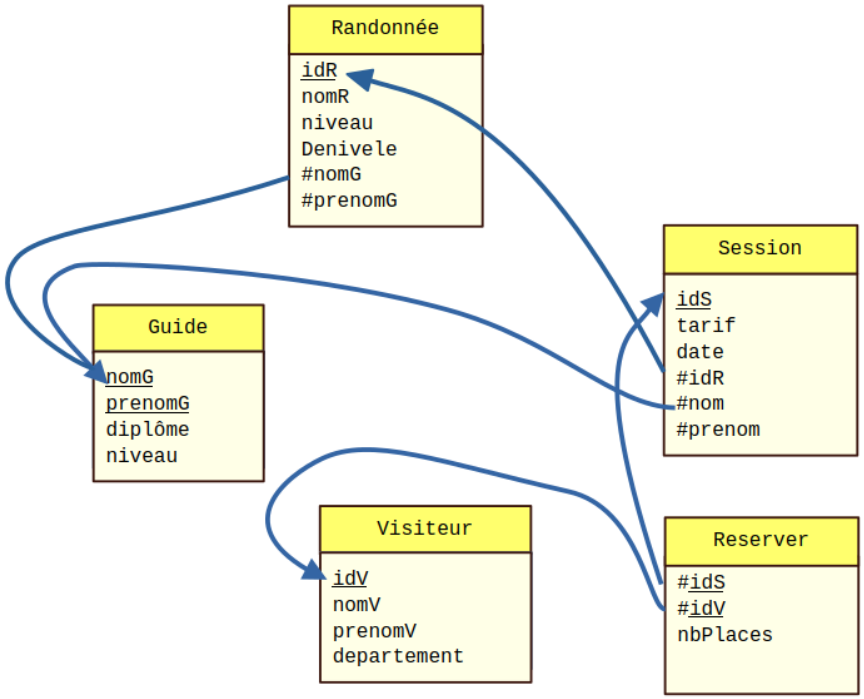


On cherche d'abord les relations qui n'ont pas de clés étrangères.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- .
- .
- .

EA vers relationnel

Transformation en modèle relationnel de l'interprétation minimale

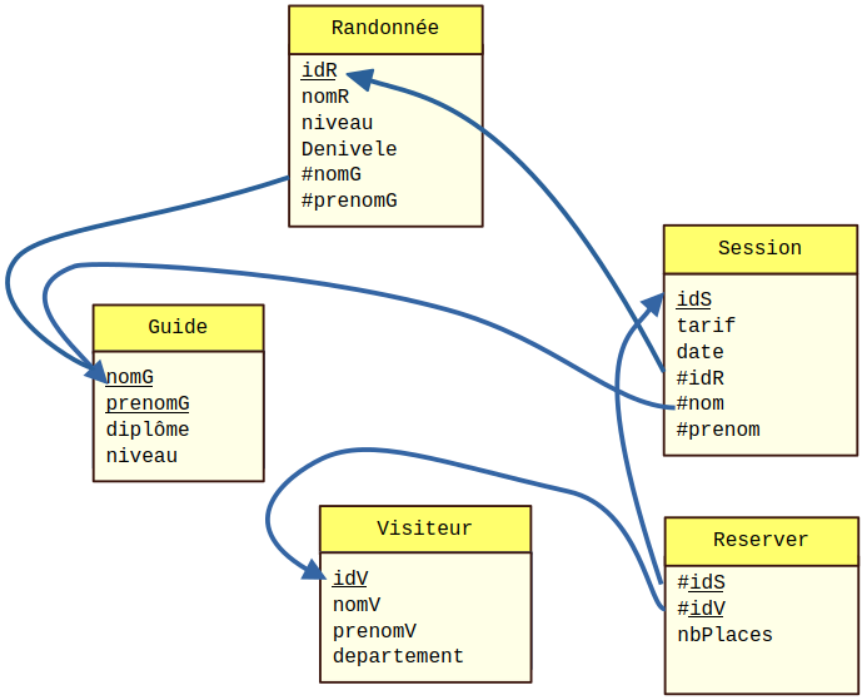


On cherche d'abord les relations qui n'ont pas de clés étrangères.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- .
- .
- .

EA vers relationnel

Transformation en modèle relationnel de l'interprétation minimale

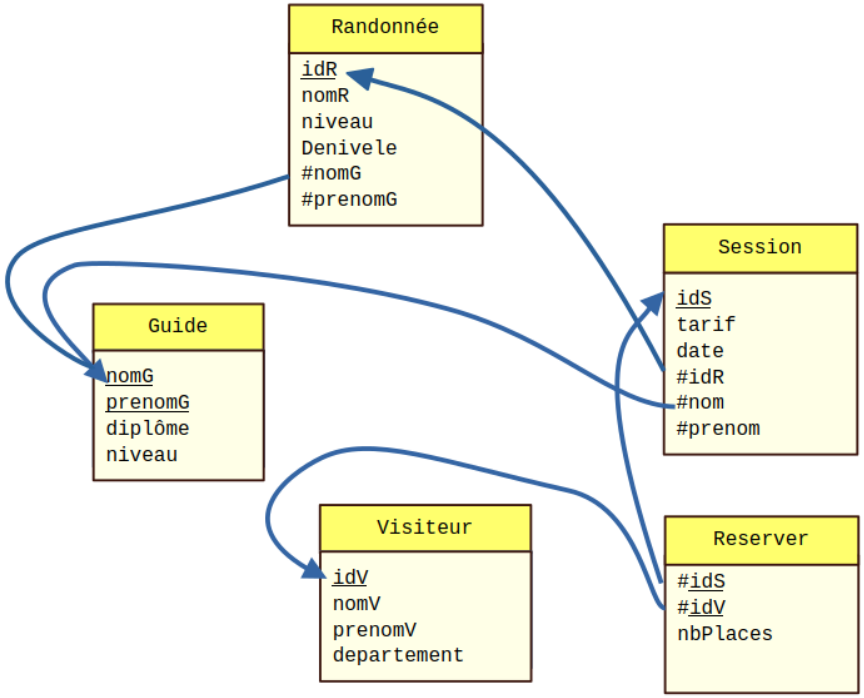


On cherche d'abord les relations qui n'ont pas de clés étrangères.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- .
- .
- .

EA vers relationnel

Transformation en modèle relationnel de l'interprétation minimale

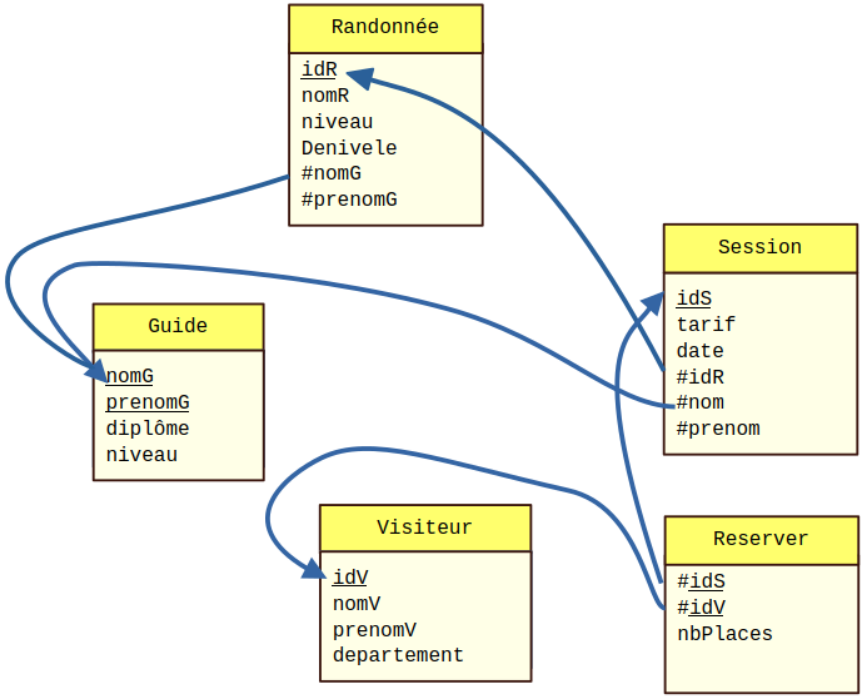


On rajoute peu à peu les dépendantes.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Session** (idS, tarif, date, #idR, #nomG, #prenomG)
- **Reservation** (#idS, #idV, nbPlaces)

EA vers relationnel

Transformation en modèle relationnel de l'interprétation minimale

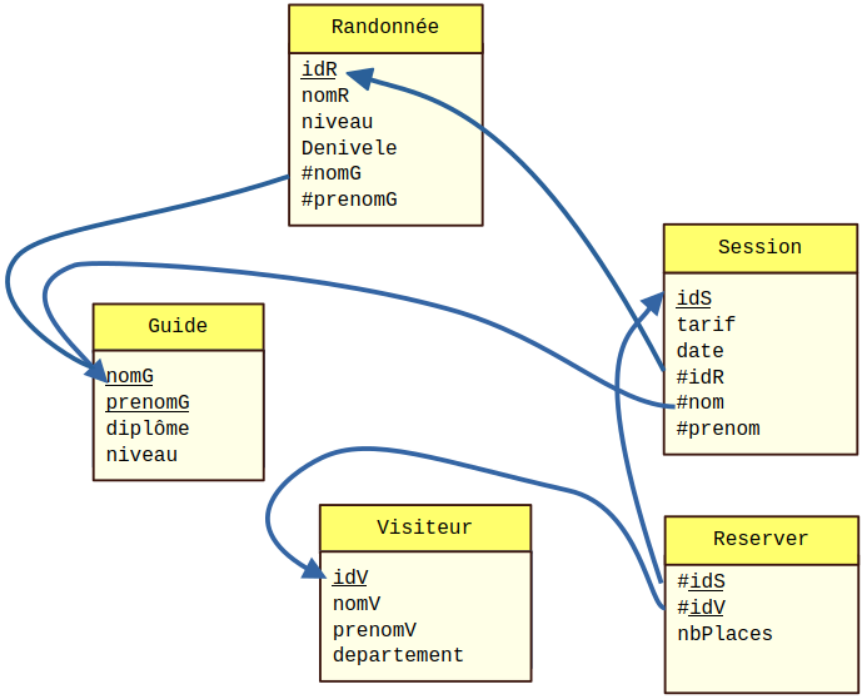


On rajoute peu à peu les dépendantes.

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Session** (idS, tarif, date, #idR, #nomG, #prenomG)
- **Reservation** (#idS, #idV, nbPlaces)

EA vers relationnel

Transformation en modèle relationnel de l'interprétation minimale



On rajoute peu à peu les dépendantes.

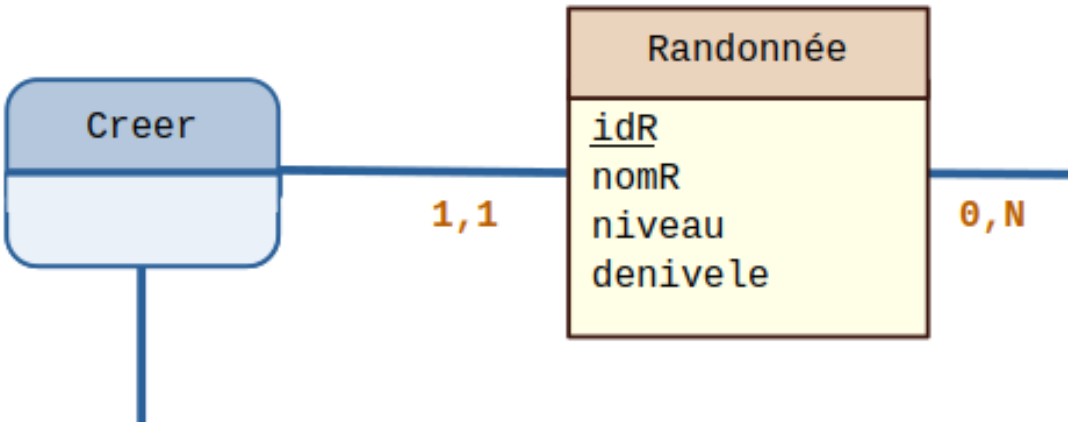
- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Session** (idS, tarif, date, #idR, #nomG, #prenomG)
- **Reservation** (#idS, #idV, nbPlaces)

Vocabulaire des 3 modèles :
Conceptuel : E/A
Logique : relationnel
Physique : SQL

Voc. du Modèle Conceptuel de données (MCD)

Modèle	Entité-Association
Classe d'éléments	Classe d'entité ou d'association
Propriétés	Attributs
Liens	Association
Elément précis	Entité ou association

Modèle conceptuel **Entité-Association** (entity-relationship, dommage pour les français) :

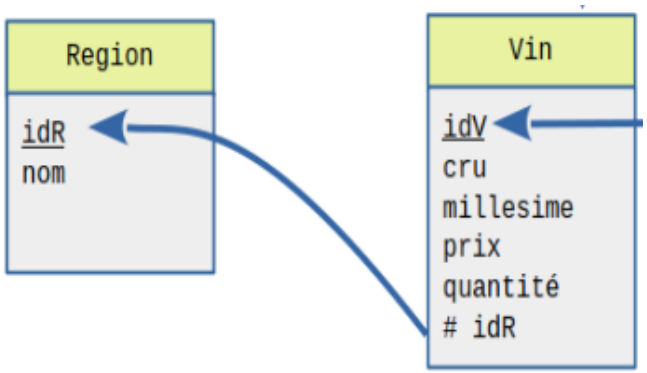


- Mot "Classe" souvent omis : le contexte fixe si on parle de Classe ou d'un élément précis.
- La cardinalité **MIN,MAX** indique combien de liens au minimum et au maximum ;
- Créé à partir d'un texte en français par exemple.

Voc. du Modèle Logique de Données (MLD)

Modèle	Entité-Association	**Relationnel**
Classe d'éléments	Classe d'entité ou d'association	Relation
Propriétés	Attributs	Attributs
Liens	Association	Système clé primaire et clé étrangère
Elément précis	Entité ou association	n-uplet Tuple

Modèle logique **Relationnel** :

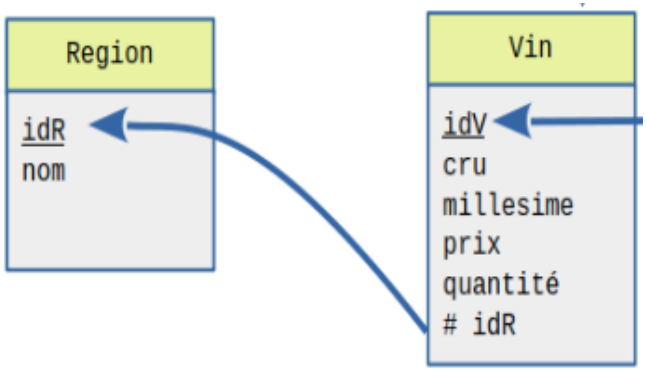


- Chaque élément est un tuple indépendant ;
- Une **relation** est un **ensemble de tuples partageant le même schéma**.
- **Aucun doublon** car les relations sont des ensembles.

Vocabulaire : conceptuel - logique

Modèle	Entité-Association	**Relationnel**
Classe d'éléments	Classe d'entité ou d'association	Relation
Propriétés	Attributs	Attributs
Liens	Association	Système clé primaire et clé étrangère
Elément précis	Entité ou association	N-uplet Tuple

Modèle logique **Relationnel** :

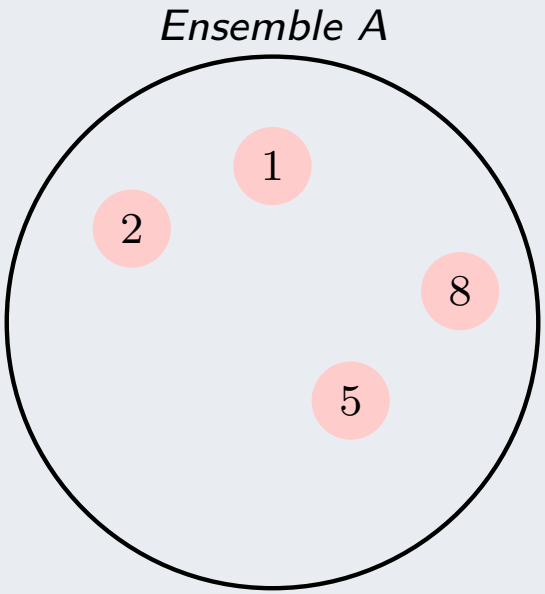


- **Vin** (idV:ENTIER, cru:TEXTE, millesime:ANNEE, prix:REEL, quantite:ENTIER, #idR:ENTIER)
- Les attributs sont typés donc VIDE / NULL n'est pas une valeur autorisée.

Ensemble

Un **ensemble** est une collection **non ordonnée d'éléments distincts**.

Exemple : $A = \{1, 2, 5, 8\} = \{2, 1, 8, 5\} = \{1, 1, 2, 5, 8\}$



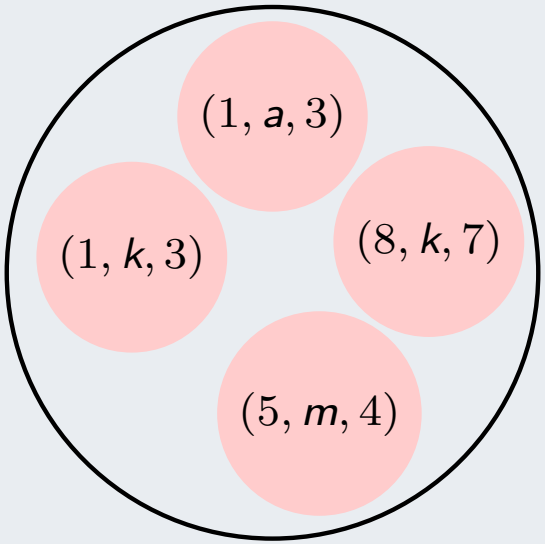
- On note $x \in A$ si l'élément x appartient à l'ensemble A (ex $5 \in A$).
- On note $X \subseteq A$ si tout élément de X est élément de A (ex $\{1, 5\} \subseteq A$).
- $\varnothing = \{\}$ est l'ensemble vide (aucun élément).
- $A \cup B$, $A \cap B$, $A \setminus B$ sont les opérations usuelles.

Relation : ensemble de tuples

Une **relation** est une collection **non ordonnée de tuples distincts** mais de **même schéma**. Ici (x, y, z), avec `x : Entier`, `y : Lettre`, `z : Entier`.

Exemple : $R(x, y, z) = \{(1, a, 3), (1, k, 3), (5, m, 4), (8, k, 7)\}$

Relation ternaire $R(x, y, z)$, sans clé primaire



- On note $t \in R$ si le tuple t appartient à la relation R.
- On a pu ajouter le tuple $(1, k, 3)$ car $(1, k, 3) \neq (1, a, 3)$;
- Impossible de rajouter un second tuple $(1, k, 3)$.

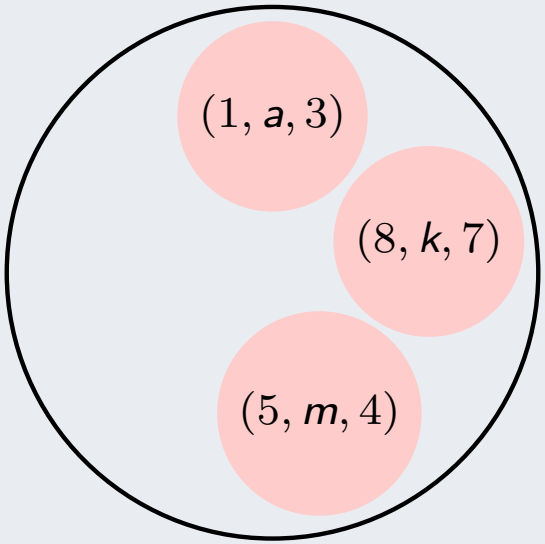
[Complément : Relation avec clé primaire optionnelle]

Relation : ensemble de tuples (chaque tuple n'apparait qu'une fois)

Une **relation** est une collection **non ordonnée de tuples distincts** mais de **même schéma**. Ici (x, y, z), avec `x : Entier`, `y : Lettre`, `z : Entier`.

Exemple : $R(x, y, z) = \{(1, a, 3), \cancel{(1, k, 3)}, (5, m, 4), (8, k, 7)\}$

Relation ternaire $R(x, y, z)$, avec clé primaire (x)



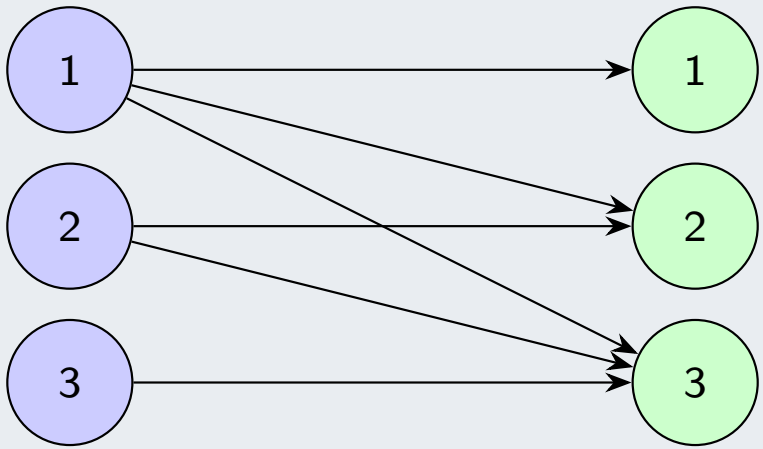
- Si **(x)** est clé primaire : impossible d'ajouter $(\mathbf{1}, k, 3)$;
- Si **(x,y)** est clé primaire : possible d'ajouter $(\mathbf{1, k}, 3)$ car $(1, a) \neq (1, k)$.
- Sans clé définie, c'est tout le tuple (x, y, z) qui doit être unique.

Cas particulier des relations binaires

Relation : peut modéliser une association Many to Many : plusieurs valeurs d'un ensemble A à plusieurs valeurs d'un ensemble B.

On écrit $x\ R\ y$ si $(x, y) \in R$ c'est à dire "x est en relation avec y".

Exemple : `R = {(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3) }`



Ici R pourrait être la relation $\leq$ sur $A = B = \{1, 2, 3\}$.
Ce n'est pas une fonction : dans une fonction car dans une fonction chaque élément de A n'a au plus qu'une image.

Vocabulaire du modèle physique

Définition : Implémentation

Implémenter désigne le fait de passer

- de **construction mentale abstraite** à **réalisation concrète en machine**.

Ainsi :

- Un **algorithme** peut être implémenté sous forme d'un **programme** ;
- Une **relation** est nommée **table**.
- Les **tuples d'une relation** deviennent des **lignes de la table** ;

Vocabulaire du modèle physique

Abstraction mentale est **réalisation concrète** s'éloignent.

- Une **relation** : ensemble de **tuples non ordonnés** et **uniques** ;

Dans une **table** :

- les **lignes sont ordonnées** car placées dans des cases mémoires ordonnées ;
  mais l'ordre n'est pas fiable car changer au cours du temps !
- les **doublons sont possibles** par choix : vérifier serait long !
  mais les clés primaires permettent de limiter le problème ;
- la valeur **NULL/VIDE** devient possible en SQL.
  pratique mais peut provoquer des problèmes !

Vocabulaire du modèle physique

Abstraction mentale est **réalisation concrète** s'éloignent.

- Une **relation** : ensemble de **tuples non ordonnés** et **uniques** ;

Dans une **table** :

- les **lignes sont ordonnées** car placées dans des cases mémoires ordonnées ;

  mais l'ordre n'est pas fiable car changer au cours du temps !
- les **doublons sont possibles** par choix : vérifier serait long !

  mais les clés primaires permettent de limiter le problème ;
- la valeur **NULL/VIDE** devient possible en SQL.

  pratique mais peut provoquer des problèmes !

Vocabulaire du modèle physique

Abstraction mentale est **réalisation concrète** s'éloignent.

- Une **relation** : ensemble de **tuples non ordonnés** et **uniques** ;

Dans une **table** :

- les **lignes sont ordonnées** car placées dans des cases mémoires ordonnées ;
  mais l'ordre n'est pas fiable car changer au cours du temps !
- les **doublons sont possibles** par choix : vérifier serait long !
  mais les clés primaires permettent de limiter le problème ;
- la valeur **NULL/VIDE** devient possible en SQL.
  pratique mais peut provoquer des problèmes !

Vocabulaire du modèle physique

Abstraction mentale est **réalisation concrète** s'éloignent.

- Une **relation** : ensemble de **tuples non ordonnés** et **uniques** ;

Dans une **table** :

- les **lignes sont ordonnées** car placées dans des cases mémoires ordonnées ;
  mais l'ordre n'est pas fiable car changer au cours du temps !
- les **doublons sont possibles** par choix : vérifier serait long !
  mais les clés primaires permettent de limiter le problème ;
- la valeur **NULL/VIDE** devient possible en SQL.
  pratique mais peut provoquer des problèmes !

Vocabulaire du modèle physique

Abstraction mentale est **réalisation concrète** s'éloignent.

- Une **relation** : ensemble de **tuples non ordonnés** et **uniques** ;

Dans une **table** :

- les **lignes sont ordonnées** car placées dans des cases mémoires ordonnées ;
  mais l'ordre n'est pas fiable car changer au cours du temps !
- les **doublons sont possibles** par choix : vérifier serait long !
  mais les clés primaires permettent de limiter le problème ;
- la valeur **NULL/VIDE** devient possible en SQL.
  pratique mais peut provoquer des problèmes !

Vocabulaire du modèle physique

Abstraction mentale est **réalisation concrète** s'éloignent.

- Une **relation** : ensemble de **tuples non ordonnés** et **uniques** ;

Dans une **table** :

- les **lignes sont ordonnées** car placées dans des cases mémoires ordonnées ;
  mais l'ordre n'est pas fiable car changer au cours du temps !
- les **doublons sont possibles** par choix : vérifier serait long !
  mais les clés primaires permettent de limiter le problème ;
- la valeur **NULL/VIDE** devient possible en SQL.
  pratique mais peut provoquer des problèmes !

Vocabulaire du modèle physique

Abstraction mentale est **réalisation concrète** s'éloignent.

- Une **relation** : ensemble de **tuples non ordonnés** et **uniques** ;

Dans une **table** :

- les **lignes sont ordonnées** car placées dans des cases mémoires ordonnées ;
  mais l'ordre n'est pas fiable car changer au cours du temps !
- les **doublons sont possibles** par choix : vérifier serait long !
  mais les clés primaires permettent de limiter le problème ;
- la valeur **NULL/VIDE** devient possible en SQL.
  pratique mais peut provoquer des problèmes !

Vocabulaire : conceptuel - logique - physique

Modèle	Entité-Association	Relationnel	**SQL**
Classe d'éléments	Classe d'entité ou d'association	Relation	Table
Propriétés	Attributs	Attributs	Attributs ou Colonnes
Liens	Association	Système clé primaire et clé étrangère	Système clé primaire et clé étrangère
Elément précis	Entité ou association	n-uplet Tuple	Ligne, row enregistrement

CREATE TABLE
logique vers physique

A Modèle logique vers Modèle physique

Soit la relation **Vin** (idV:ENTIER, cru:TEXTE, millesime:ANNEE)

- Passage en **SQL** (Structured Query Language) version **postGreSQL** :

```sql
CREATE TABLE Vin (
    idV INT,
    cru VARCHAR(40) NOT NULL,
    millesime INT CHECK (millesime BETWEEN 1700 AND 2100)
    );
```

- Différence entre **Relation** et **Table** :

	Relation	**Table**
Modèle	Logique (mathématique)	Physique (informatique)
Doublon ?	Non (ensemble)	Oui (multiensemble) sans PK
		Non (multiensemble) si PK
Ordonné ?	Non (ensemble)	Non (multiensemble)
Valeur inconnue ?	Non car typée	Oui (typée mais NULL possible)

B Table SANS clé primaire ? doublons possibles

```sql
CREATE TABLE Vin (
    idV INT,
    cru VARCHAR(40) NOT NULL,
    millesime INT CHECK (millesime BETWEEN 1700 AND 2100)
);
INSERT INTO Vin
VALUES
(1, 'Grand Cru', 2005), (2, 'Grand Cru', 2010),
(3, 'Qualité supérieure', 2020),
(4, 'Qualité supérieure', 2022);
```

```
CREATE TABLE
INSERT 0 4                -- > Aucun problème
```

```sql
INSERT INTO Vin VALUES (3, 'Qualité supérieure', 2020);  -- doublon ?
```

```
INSERT 0 1                -- > doublon ok !
```

B Table SANS clé primaire ? doublons possibles

```sql
CREATE TABLE Vin (
    idV INT,
    cru VARCHAR(40) NOT NULL,
    millesime INT CHECK (millesime BETWEEN 1700 AND 2100)
);
INSERT INTO Vin
VALUES
(1, 'Grand Cru', 2005), (2, 'Grand Cru', 2010),
(3, 'Qualité supérieure', 2020),
(4, 'Qualité supérieure', 2022);
```

```
CREATE TABLE
INSERT 0 4                -- > Aucun problème
```

```sql
INSERT INTO Vin VALUES (3, 'Qualité supérieure', 2020);  -- doublon ?
```

```
INSERT 0 1                -- > doublon ok !
```

B Table SANS clé primaire : doublons possibles

Visualisation des données enregistrées :

```
SELECT *
FROM Vin;
```

```
+------+----------------------+-----------+
| idV  | cru                  | millesime |
+------+----------------------+-----------+
|    1 | Grand Cru            |      2005 |
|    2 | Grand Cru            |      2010 |
|    3 | Qualité supérieure   |      2020 |
|    4 | Qualité supérieure   |      2022 |
|    3 | Qualité supérieure   |      2020 |
+------+----------------------+-----------+
(5 rows)
```

B Table SQL sans clé primaire → doublon possible !

C Table AVEC clé primaire : doublon impossible

```sql
DROP TABLE IF EXISTS Vin ;  -- suppression de l'ancienne
CREATE TABLE Vin (
    idV INT,
    cru VARCHAR(40) NOT NULL,
    millesime INT CHECK (millesime BETWEEN 1700 AND 2100),
    PRIMARY KEY (idV)
    );
INSERT INTO Vin
VALUES
(1, 'Grand Cru', 2005),
(2, 'Grand Cru', 2010),
(3, 'Qualité supérieure', 2020),
(4, 'Qualité supérieure', 2022);
```

```
DROP TABLE    CREATE TABLE     INSERT 0 4 ---> Aucun problème
```

```sql
INSERT INTO Vin VALUES (3, 'Grand Cru', 2005);  -- doublon ?
```

```
ERROR:  duplicate key value violates unique constraint "vin_pkey"
DETAIL:  Key (idv)=(3) already exists.
```

C Table AVEC clé primaire : doublon impossible

```sql
DROP TABLE IF EXISTS Vin ;  -- suppression de l'ancienne
CREATE TABLE Vin (
    idV INT,
    cru VARCHAR(40) NOT NULL,
    millesime INT CHECK (millesime BETWEEN 1700 AND 2100),
    PRIMARY KEY (idV)
    );
INSERT INTO Vin
VALUES
(1, 'Grand Cru', 2005),
(2, 'Grand Cru', 2010),
(3, 'Qualité supérieure', 2020),
(4, 'Qualité supérieure', 2022);
```

```
DROP TABLE    CREATE TABLE     INSERT 0 4 ---> Aucun problème
```

```sql
INSERT INTO Vin VALUES (3, 'Grand Cru', 2005);  -- doublon ?
```

```
ERROR:  duplicate key value violates unique constraint "vin_pkey"
DETAIL:  Key (idv)=(3) already exists.
```

C Déclaration de clé primaire : version alternative

```sql
DROP TABLE IF EXISTS Vin ;  -- suppression de l'ancienne
CREATE TABLE Vin (
    idV INT PRIMARY KEY,
    cru VARCHAR(40) NOT NULL,
    millesime INT CHECK (millesime BETWEEN 1700 AND 2100)

    );
INSERT INTO Vin
VALUES
(1, 'Grand Cru', 2005),
(2, 'Grand Cru', 2010),
(3, 'Qualité supérieure', 2020),
(4, 'Qualité supérieure', 2022);
```

Convention pour cette année :
N'utilisez pas cette façon de faire : elle ne fonctionne que si la clé primaire n'est composée que d'une colonne.

C Déclaration de clé primaire composite

```sql
DROP TABLE IF EXISTS Guide ;
CREATE TABLE Guide (
    nomG VARCHAR(40),
    prenomG VARCHAR(40),
    diplome VARCHAR(40),
    niveau VARCHAR(40) CHECK (niveau IN ('débutant', 'moyen', 'avancé')),
    PRIMARY KEY (nomG, prenomG)
    );
```

```sql
    PRIMARY KEY (nomG, prenomG)
```

La syntaxe SQL impose cette façon de déclarer une clé primaire composite (couple, triplet...) : en plaçant deux **PRIMARY KEY**, une sur les lignes de nomG et prenomG, vous auriez une erreur.

CHECK est ici utilisé pour ne permettre qu'un ensemble limité de valeurs.

```sql
    niveau VARCHAR(40) CHECK (niveau IN ('débutant', 'moyen', 'avancé'))
```

C Déclaration de clé primaire composite

```
DROP TABLE IF EXISTS Guide ;
CREATE TABLE Guide (
    nomG VARCHAR(40),
    prenomG VARCHAR(40),
    diplome VARCHAR(40),
    niveau VARCHAR(40) CHECK (niveau IN ('débutant', 'moyen', 'avancé')),
    PRIMARY KEY (nomG, prenomG)
    );
```

```
    PRIMARY KEY (nomG, prenomG)
```

La syntaxe SQL impose cette façon de déclarer une clé primaire composite (couple, triplet...) : en plaçant deux **PRIMARY KEY**, une sur les lignes de nomG et prenomG, vous auriez une erreur.

CHECK est ici utilisé pour ne permettre qu'un ensemble limité de valeurs.

```
    niveau VARCHAR(40) CHECK (niveau IN ('débutant', 'moyen', 'avancé'))
```

D Déclaration de clé étrangère

```sql
DROP TABLE IF EXISTS Vin ;
CREATE TABLE Vin (
    idV INT,
    cru VARCHAR(40) NOT NULL,
    millesime INT CHECK (millesime BETWEEN 1700 AND 2100),
    prix DECIMAL(8, 2)
    quantite INT,
    idR INT,
    PRIMARY KEY (idV),
    FOREIGN KEY (idR) REFERENCES Region(idR)
);
```

peut être la Table SQL implémentant cette Relation :

Vin (idV:ENTIER, cru:TEXTE, millesime:ANNEE, prix:PRIX, quantite:ENTIER, #idR:ENTIER)

Obligatoire : idR créée **PUIS** déclarée comme FK vers la PK idR de Region.
Obligatoire : **Region doit donc avoir été créée AVANT Vin.**
Convention : **PRIMARY KEY** puis **FOREIGN KEY** ensuite.

D Déclaration de clé étrangère composite

Randonnee(idR, nomR, niveau, denivele, **#nomG, #prenomG**)

```
CREATE TABLE Randonnee (
    idR INT,
    nomR VARCHAR(40),
    niveau VARCHAR(20)
      CHECK (niveau IN ('facile', 'moyen', 'difficile', 'très difficile')),
    denivele INT,
    nomG VARCHAR(40) NOT NULL,
    prenomG VARCHAR(40) NOT NULL,
    PRIMARY KEY (idR),
    FOREIGN KEY (nomG, prenomG) REFERENCES Guide(nomG, prenomG)
    );
```

Au vu du TD, la FK est composite car vers la PK (nomG,prenomG) de Guide.

Mais on pourrait croire avoir DEUX clés étrangères d'une colonne non ?

Une solution pour règler cette ambiguïté de notation :

Randonnee(idR, nomR, niveau, denivele, **#(nomG, prenomG)**)

Mais cette convention n'est pas obligatoire, juste une écriture moins ambigüe.

D Déclaration de clé étrangère composite

Randonnee(idR, nomR, niveau, denivele, **#nomG, #prenomG**)

```
CREATE TABLE Randonnee (
    idR INT,
    nomR VARCHAR(40),
    niveau VARCHAR(20)
      CHECK (niveau IN ('facile', 'moyen', 'difficile', 'très difficile')),
    denivele INT,
    nomG VARCHAR(40) NOT NULL,
    prenomG VARCHAR(40) NOT NULL,
    PRIMARY KEY (idR),
    FOREIGN KEY (nomG, prenomG) REFERENCES Guide(nomG, prenomG)
    );
```

Au vu du TD, la FK est composite car vers la PK (nomG,prenomG) de Guide.
Mais on pourrait croire avoir DEUX clés étrangères d'une colonne non ?
Une solution pour règler cette ambiguïté de notation :
Randonnee(idR, nomR, niveau, denivele, **#(nomG, prenomG)**)
Mais cette convention n'est pas obligatoire, juste une écriture moins ambigüe.

D Déclaration de clé étrangère composite

Randonnee(<u>idR</u>, nomR, niveau, denivele, **#nomG, #prenomG**)

```
CREATE TABLE Randonnee (
    idR INT,
    nomR VARCHAR(40),
    niveau VARCHAR(20)
      CHECK (niveau IN ('facile', 'moyen', 'difficile', 'très difficile')),
    denivele INT,
    nomG VARCHAR(40) NOT NULL,
    prenomG VARCHAR(40) NOT NULL,
    PRIMARY KEY (idR),
    FOREIGN KEY (nomG, prenomG) REFERENCES Guide(nomG, prenomG)
    );
```

Au vu du TD, la FK est composite car vers la PK (nomG,prenomG) de Guide.
Mais on pourrait croire avoir DEUX clés étrangères d'une colonne non ?
Une solution pour règler cette ambiguïté de notation :
Randonnee(<u>idR</u>, nomR, niveau, denivele, **#(nomG, prenomG)**)
Mais cette convention n'est pas obligatoire, juste une écriture moins ambigüe.

D Déclaration de clé étrangère composite

Randonnee(<u>idR</u>, nomR, niveau, denivele, **#nomG, #prenomG**)

```
CREATE TABLE Randonnee (
    idR INT,
    nomR VARCHAR(40),
    niveau VARCHAR(20)
      CHECK (niveau IN ('facile', 'moyen', 'difficile', 'très difficile')),
    denivele INT,
    nomG VARCHAR(40) NOT NULL,
    prenomG VARCHAR(40) NOT NULL,
    PRIMARY KEY (idR),
    FOREIGN KEY (nomG, prenomG) REFERENCES Guide(nomG, prenomG)
    );
```

Au vu du TD, la FK est composite car vers la PK (nomG,prenomG) de Guide.
Mais on pourrait croire avoir DEUX clés étrangères d'une colonne non ?
Une solution pour règler cette ambiguïté de notation :
Randonnee(<u>idR</u>, nomR, niveau, denivele, **#(nomG, prenomG)**)
Mais cette convention n'est pas obligatoire, juste une écriture moins ambigüe.

D ATTENTION : une clé étrangère DOIT référencer une colonne UNIQUE

Randonnee(idR, nomR, niveau, denivele, #nomG, #prenomG)

```
CREATE TABLE Randonnee (
    idR INT,
    nomR VARCHAR(40),
    niveau VARCHAR(20) CHECK (niveau IN (...)),
    denivele INT,
    nomG VARCHAR(40) NOT NULL,
    prenomG VARCHAR(40) NOT NULL,
    PRIMARY KEY (idR),
    FOREIGN KEY (nomG) REFERENCES Guide(nomG),
    FOREIGN KEY (prenomG) REFERENCES Guide(prenomG)
    );
```

ATTENTION : ceci **est refusé** car la clé étrangère **nomG** référence **nom de Guide** qui ne possède pas la propriété UNIQUE : un nom peut apparaître 2 fois !

A SAVOIR : une clé étrangère doit référencer

- une colonne déclarée UNIQUE ou
- une clé primaire (qui est automatiquement UNIQUE + NOT NULL)

E Clé primaire composite : à éviter !

En conclusion, l'usage de clé primaire double ou triple est plutôt à éviter.
Quelque cas où on a pas trop le choix mais, globalement, il aurait mieux fallu faire ceci :

```
CREATE TABLE Guide (
    idG INT,
    nomG VARCHAR(40),
    prenomG VARCHAR(40),
    diplome VARCHAR(40),
    niveau VARCHAR(40)
        CHECK (niveau IN ('débutant', 'moyen', 'avancé')),
    PRIMARY KEY (idG),
    UNIQUE (nomG, prenomG)    -- pour coller au TD ?
    );
```

F Compléments contraintes 1/3

Les contraintes d'attributs (à part NOT NULL) peuvent s'écrire sur une ligne indépendante :

```
DROP TABLE IF EXISTS Vin ;
CREATE TABLE Vin (
    idV INT,
    nom VARCHAR(40),
    cru VARCHAR(40) NOT NULL,
    millesime INT,
    prix DECIMAL(8, 2),
    quantite INT,
    idR INT,
    UNIQUE (nom),
    CHECK (millesime BETWEEN 1700 AND 2100),
    CHECK (prix >= 0.0),
    CHECK (quantite >= 0.0),
    PRIMARY KEY (idV),
    FOREIGN KEY (idR) REFERENCES Region(idR)
);
```

F Compléments contraintes 2/3

Il est possible de donner un nom aux contraintes que vous rajouter. Si vous ne les faites pas, c'est le sgbd qui le fait pour vous...

```
DROP TABLE IF EXISTS Vin;

CREATE TABLE Vin (
    idV INT,
    nom VARCHAR(40),
    cru VARCHAR(40) NOT NULL,
    millesime INT,
    prix DECIMAL(8, 2),
    quantite INT,
    idR INT,
    CONSTRAINT uq_vin_nom       UNIQUE (nom),
    CONSTRAINT ck_vin_millesime CHECK (millesime BETWEEN 1700 AND 2100),
    CONSTRAINT ck_vin_prix      CHECK (prix >= 0.0),
    CONSTRAINT ck_vin_quantite  CHECK (quantite >= 0.0),
    CONSTRAINT pk_vin           PRIMARY KEY (idV),
    CONSTRAINT fk_vin_region    FOREIGN KEY (idR) REFERENCES Region(idR)
);
```

F Compléments contraintes 3/3

Pourquoi nommer ? Pour supprimer et/ou modifier facilement !

```
DROP CONSTRAINT ck_vin_prix;
```

Maintenant qu'elle est supprimée, on peut en recréer une un peu différente :

```
ALTER TABLE Vin
ADD CONSTRAINT ck_vin_prix
CHECK (prix >= 0 AND prix <= 10000);
```

F Compléments contraintes 3/3

Pourquoi nommer ? Pour supprimer et/ou modifier facilement !

```
DROP CONSTRAINT ck_vin_prix;
```

Maintenant qu'elle est supprimée, on peut en recréer une un peu différente :

```
ALTER TABLE Vin
ADD CONSTRAINT ck_vin_prix
CHECK (prix >= 0 AND prix <= 10000);
```

F Compléments contraintes 3/3

Pourquoi nommer ? Pour supprimer et/ou modifier facilement !

```
DROP CONSTRAINT ck_vin_prix;
```

Maintenant qu'elle est supprimée, on peut en recréer une un peu différente :

```
ALTER TABLE Vin
ADD CONSTRAINT ck_vin_prix
CHECK (prix >= 0 AND prix <= 10000);
```

G Gestion automatique des PK

On peut demander au SGBD de gérer lui-même la numérotation des id :

```sql
DROP TABLE IF EXISTS Vin;
CREATE TABLE Vin (
    idV INT GENERATED ALWAYS AS IDENTITY,
    nom VARCHAR(40),
    cru VARCHAR(40) NOT NULL,
    millesime INT,
    prix DECIMAL(8, 2),
    quantite INT,
    idR INT,                    ... /* suite inchangé */
);
```

```sql
/* remarquez qu'on omet idV */
INSERT INTO Vin (nom, cru, millesime, prix, quantite, idR)
VALUES
('Château Rouge', 'Bordeaux', 2018, 25.50, 12, 1),
('Vieilles Vignes', 'Bourgogne', 2020, 38.00, 8, 2),
('Les Terrasses', 'Rhône', 2019, 18.90, 20, 3),
('Grand Cru', 'Bordeaux', 2016, 75.00, 5, 1)
;
```

TD Question 3
Passage du modèle relationnel maximaliste à SQL

CREATE TABLE

Transformation en modèle relationnel

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Session** (idS, tarif, date)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Encadrement** (#nomG, #prenomG, #idS)
- **Reservation** (#idS, #idV, nbPlaces)
- **Proposition** (#idR, #idS)

En attendant d'avoir un SGBD installé sur votre machine :
https://www.db-fiddle.com/

CREATE TABLE Visiteur v1 v2

- **Visiteur** (idV, nomV, prenomV, departement)

```sql
CREATE TABLE Visiteur (
    idV INT GENERATED ALWAYS AS IDENTITY,
    nomV VARCHAR(40),
    prenomV VARCHAR(40),
    departement INT NOT NULL,
    PRIMARY KEY (idV)
);
```

```sql
CREATE TABLE Visiteur (
    idV INT GENERATED ALWAYS AS IDENTITY,
    nomV VARCHAR(40),
    prenomV VARCHAR(40),
    departement INT NOT NULL,
    CONSTRAINT pk_visiteur PRIMARY KEY (idV)
);
```

CREATE TABLE Visiteur v1 v2

- **Visiteur** (idV, nomV, prenomV, departement)

```sql
CREATE TABLE Visiteur (
    idV INT GENERATED ALWAYS AS IDENTITY,
    nomV VARCHAR(40),
    prenomV VARCHAR(40),
    departement INT NOT NULL,
    PRIMARY KEY (idV)
);
```

```sql
CREATE TABLE Visiteur (
    idV INT GENERATED ALWAYS AS IDENTITY,
    nomV VARCHAR(40),
    prenomV VARCHAR(40),
    departement INT NOT NULL,
    CONSTRAINT pk_visiteur PRIMARY KEY (idV)
);
```

CREATE TABLE Visiteur MySQL / MariaDB

- **Visiteur** (idV, nomV, prenomV, departement)

```sql
CREATE TABLE Visiteur (
    idV INT AUTO_INCREMENT,
    nomV VARCHAR(40),
    prenomV VARCHAR(40),
    departement INT NOT NULL,
    PRIMARY KEY (idV)
);
```

CREATE TABLE Visiteur

Clé primaire auto-incrémentée : pas besoin de fournir la valeur de la clé primaire :

```
INSERT INTO Visiteur(nomV, prenomV, departement)
VALUES
('A', 'Alice', 2),
('B', 'Bob', 59),
('C', NULL, 62);

SELECT * FROM Visiteur;
```

```
+-----+------+---------+-------------+
| idV | nomV | prenomV | departement |
+-----+------+---------+-------------+
|   1 | A    | Alice   |           2 |
|   2 | B    | Bob     |          59 |
|   3 | C    | NULL    |          62 |
+-----+------+---------+-------------+
(3 rows)
```

CREATE TABLE Visiteur

Notez que l'attribut departement a été déclaré :

- Sans accent ;
- Avec NOT NULL : on doit le fournir, sinon cela déclenche un refus comme ci-dessous :

```
INSERT INTO Visiteur(nomV, prenomV, departement)
VALUES
('D', 'Didier', NULL);
```

```
Query Error: null value in column "departement" of
relation "visiteur" violates not-null constraint
```

CREATE TABLE Guide v1 v2

- **Guide**(nomG, prenomG, diplome, niveau)

```sql
CREATE TABLE Guide (
    nomG VARCHAR(40),
    prenomG VARCHAR(40),
    diplome VARCHAR(40),
    niveau VARCHAR(20),
    PRIMARY KEY (nomG, prenomG)
);
```

```sql
    CONSTRAINT pk_guide PRIMARY KEY (nomG, prenomG)    -- v2
```

Clé primaire composite : sur une ligne à part

On DOIT déclarer une clé primaire composite en fin de bloc

Notez la présence des parenthèses sur la ligne 6 : la clé primaire est un couple.

CREATE TABLE Session v1 v2

- **Session**(idS, tarif, date)

```
CREATE TABLE Session (
    idS INT GENERATED ALWAYS AS IDENTITY,
    tarif DECIMAL(8,2),
    date DATE,
    PRIMARY KEY (idS)
);
```

ou

```
CREATE TABLE Session (
    idS INT GENERATED ALWAYS AS IDENTITY,
    tarif DECIMAL(8,2),
    date DATE,
    CONSTRAINT pk_session PRIMARY KEY (idS)
);
```

CREATE TABLE Randonnee v1

- **Randonnee**(idR, nomR, niveau, denivele, #nomG, #prenomG)

```
CREATE TABLE Randonnee (
    idR INT  GENERATED ALWAYS AS IDENTITY,
    nomR VARCHAR(40),
    niveau VARCHAR(20),
    denivele INT,
    nomG VARCHAR(40) NOT NULL,
    prenomG VARCHAR(40) NOT NULL,
    PRIMARY KEY (idR),
    FOREIGN KEY (nomG, prenomG) REFERENCES Guide(nomG, prenomG)


);
```

NOT NULL car le schéma E/A explicite qu'une randonnée est associée impérativement à un et un seul guide.

CREATE TABLE Randonnee v2

- **Randonnee**(idR, nomR, niveau, denivele, #nomG, #prenomG)

```
CREATE TABLE Randonnee (
    idR INT  GENERATED ALWAYS AS IDENTITY,
    nomR VARCHAR(40),
    niveau VARCHAR(20),
    denivele INT,
    nomG VARCHAR(40) NOT NULL,
    prenomG VARCHAR(40) NOT NULL,
    CONSTRAINT pk_randonnee
        PRIMARY KEY (idR),
    CONSTRAINT fk_randonnee_guide
        FOREIGN KEY (nomG, prenomG)
            REFERENCES Guide(nomG, prenomG)
);
```

CREATE TABLE Encadrement v1

- **Encadrement**(#nomG, #prenomG, #idS)

```
CREATE TABLE Encadrement (
    nomG VARCHAR(40),
    prenomG VARCHAR(40),
    idS INT,
    PRIMARY KEY (nomG, prenomG, idS),
    FOREIGN KEY (nomG, prenomG) REFERENCES Guide(nomG, prenomG),
    FOREIGN KEY (idS) REFERENCES Session(idS)
);
```

Le schéma EA indique qu'une Session doit être associée avec, au moins un guide. Mais pas besoin de noter **NOT NULL** car les attributs sont des clés primaires : elles sont par définition **UNIQUE NOT NULL**.

CREATE TABLE Encadrement v2

- **Encadrement**(#nomG, #prenomG, #idS)

```
CREATE TABLE Encadrement (
    nomG VARCHAR(40),
    prenomG VARCHAR(40),
    idS INT,
    CONSTRAINT pk_encadrement
        PRIMARY KEY (nomG, prenomG, idS),
    CONSTRAINT fk_encadrement_guide
        FOREIGN KEY (nomG, prenomG)
            REFERENCES Guide(nomG, prenomG),
    CONSTRAINT fk_encadrement_session
        FOREIGN KEY (idS)
            REFERENCES Session(idS)
);
```

CREATE TABLE Reservation v1 v2

- **Reservation**(#idS, #idV, nbPlaces)

```sql
CREATE TABLE Reservation (
    idS INT,
    idV INT,
    nbPlaces INT,
    PRIMARY KEY (idS, idV),
    FOREIGN KEY (idS) REFERENCES Session(idS),
    FOREIGN KEY (idV) REFERENCES Visiteur(idV)
);
```

```sql
CREATE TABLE Reservation (
    idS INT,
    idV INT,
    nbPlaces INT,
    CONSTRAINT pk_reservation PRIMARY KEY (idS, idV),
    CONSTRAINT fk_reservation_session
        FOREIGN KEY (idS) REFERENCES Session(idS),
    CONSTRAINT fk_reservation_visiteur
        FOREIGN KEY (idV) REFERENCES Visiteur(idV)
);
```

CREATE TABLE Proposition v1 v2

- **Proposition**(#idR, #idS)

```sql
CREATE TABLE Proposition (
    idR INT,
    idS INT,
    PRIMARY KEY (idR, idS),
    FOREIGN KEY (idR) REFERENCES Randonnee(idR),
    FOREIGN KEY (idS) REFERENCES Session(idS)
);
```

```sql
CREATE TABLE Proposition (
    idR INT,
    idS INT,
    CONSTRAINT pk_proposition
        PRIMARY KEY (idR, idS),
    CONSTRAINT fk_proposition_randonnee
        FOREIGN KEY (idR) REFERENCES Randonnee(idR),
    CONSTRAINT fk_proposition_session
        FOREIGN KEY (idS) REFERENCES Session(idS)
);
```

Passage du schéma relationnel à la table SQL

CREATE TABLE

Transformation en modèle relationnel minimaliste

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Session** (idS, tarif, date, #idR, #nomG, #prenomG)
- **Reservation** (#idS, #idV, nbPlaces)

CREATE TABLE

Transformation en modèle relationnel minimaliste

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Session** (idS, tarif, date, #idR, #nomG, #prenomG)
- **Reservation** (#idS, #idV, nbPlaces)

CREATE TABLE

Transformation en modèle relationnel minimaliste

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Session** (idS, tarif, date, #idR, #nomG, #prenomG)
- **Reservation** (#idS, #idV, nbPlaces)

CREATE TABLE

Transformation en modèle relationnel minimaliste

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Session** (idS, tarif, date, #idR, #nomG, #prenomG)
- **Reservation** (#idS, #idV, nbPlaces)

CREATE TABLE

Transformation en modèle relationnel minimaliste

- **Visiteur** (idV, nomV, prenomV, departement)
- **Guide** (nomG, prenomG, diplome, niveau)
- **Randonnee** (idR, nomR, niveau, denivele, #nomG, #prenomG)
- **Session** (idS, tarif, date, #idR, #nomG, #prenomG)
- **Reservation** (#idS, #idV, nbPlaces)

Bilan

Les points à mettre en valeur

- Vocabulaire des 3 modèles ;
- texte $\rightarrow$ modèle Entité Association ;
- Modèle E/A $\rightarrow$ modèle relationnel ;
  - One to many
  - Many to many
- Modèle relationnel $\rightarrow$ vers SQL (TP) ;
  - Syntaxe CREATE TABLE nom (attri1 TYPE...,)
  - Importance des virgules de séparation pour les INSERT
  - Syntaxe INSERT INTO table(tuple) VALUES (), (), () ;
  - Importance du point-virgule pour cloturer une requête
  - Guillemets simple ', types CHAR(40), VARCHAR(40), TEXT
  - DATE sous forme "AAAA-MM-JJ"
  - Signification de NOT NULL et NULL
  - Rajout d'une clé primaire simple (non composite)
  - Rajout d'une clé primaire composite
  - GENERATED ALWAYS AS IDENTITY
  - Rajout d'une clé étrangère : FOREIGN KEY (fk) REFERENCES table(pk)

Les points à mettre en valeur

- Vocabulaire des 3 modèles ;
- texte $\rightarrow$ modèle Entité Association ;
- Modèle E/A $\rightarrow$ modèle relationnel ;
  - One to many
  - Many to many
- Modèle relationnel $\rightarrow$ vers SQL (TP) ;
  - Syntaxe CREATE TABLE nom (attri1 TYPE...,)
  - Importance des virgules de séparation pour les INSERT
  - Syntaxe INSERT INTO table(tuple) VALUES (), (), () ;
  - Importance du point-virgule pour cloturer une requête
  - Guillemets simple ', types CHAR(40), VARCHAR(40), TEXT
  - DATE sous forme "AAAA-MM-JJ"
  - Signification de NOT NULL et NULL
  - Rajout d'une clé primaire simple (non composite)
  - Rajout d'une clé primaire composite
  - GENERATED ALWAYS AS IDENTITY
  - Rajout d'une clé étrangère : FOREIGN KEY (fk) REFERENCES table(pk)

Les points à mettre en valeur

- Vocabulaire des 3 modèles ;
- texte $\rightarrow$ modèle Entité Association ;
- Modèle E/A $\rightarrow$ modèle relationnel ;
  - One to many
  - Many to many
- Modèle relationnel $\rightarrow$ vers SQL (TP) ;
  - Syntaxe CREATE TABLE nom (attri1 TYPE...,)
  - Importance des virgules de séparation pour les INSERT
  - Syntaxe INSERT INTO table(tuple) VALUES (), (), () ;
  - Importance du point-virgule pour cloturer une requête
  - Guillemets simple ', types CHAR(40), VARCHAR(40), TEXT
  - DATE sous forme "AAAA-MM-JJ"
  - Signification de NOT NULL et NULL
  - Rajout d'une clé primaire simple (non composite)
  - Rajout d'une clé primaire composite
  - GENERATED ALWAYS AS IDENTITY
  - Rajout d'une clé étrangère : FOREIGN KEY (fk) REFERENCES table(pk)

Les points à mettre en valeur

- Vocabulaire des 3 modèles ;
- texte $\rightarrow$ modèle Entité Association ;
- Modèle E/A $\rightarrow$ modèle relationnel ;
  - One to many
  - Many to many
- Modèle relationnel $\rightarrow$ vers SQL (TP) ;
  - Syntaxe CREATE TABLE nom (attri1 TYPE...,)
  - Importance des virgules de séparation pour les INSERT
  - Syntaxe INSERT INTO table(tuple) VALUES (), (), () ;
  - Importance du point-virgule pour cloturer une requête
  - Guillemets simple ', types CHAR(40), VARCHAR(40), TEXT
  - DATE sous forme "AAAA-MM-JJ"
  - Signification de NOT NULL et NULL
  - Rajout d'une clé primaire simple (non composite)
  - Rajout d'une clé primaire composite
  - GENERATED ALWAYS AS IDENTITY
  - Rajout d'une clé étrangère : FOREIGN KEY (fk) REFERENCES table(pk)

Les points à mettre en valeur

- Vocabulaire des 3 modèles ;
- texte $\rightarrow$ modèle Entité Association ;
- Modèle E/A $\rightarrow$ modèle relationnel ;
  - One to many
  - Many to many
- Modèle relationnel $\rightarrow$ vers SQL (TP) ;
  - Syntaxe CREATE TABLE nom (attri1 TYPE...,)
  - Importance des virgules de séparation pour les INSERT
  - Syntaxe INSERT INTO table(tuple) VALUES (), (), () ;
  - Importance du point-virgule pour cloturer une requête
  - Guillemets simple ', types CHAR(40), VARCHAR(40), TEXT
  - DATE sous forme "AAAA-MM-JJ"
  - Signification de NOT NULL et NULL
  - Rajout d'une clé primaire simple (non composite)
  - Rajout d'une clé primaire composite
  - GENERATED ALWAYS AS IDENTITY
  - Rajout d'une clé étrangère : FOREIGN KEY (fk) REFERENCES table(pk)

Les points à mettre en valeur

- Vocabulaire des 3 modèles ;
- texte $\rightarrow$ modèle Entité Association ;
- Modèle E/A $\rightarrow$ modèle relationnel ;
  - One to many
  - Many to many
- Modèle relationnel $\rightarrow$ vers SQL (TP) ;
  - Syntaxe CREATE TABLE nom (attri1 TYPE...,)
  - Importance des virgules de séparation pour les INSERT
  - Syntaxe INSERT INTO table(tuple) VALUES (), (), () ;
  - Importance du point-virgule pour cloturer une requête
  - Guillemets simple ', types CHAR(40), VARCHAR(40), TEXT
  - DATE sous forme "AAAA-MM-JJ"
  - Signification de NOT NULL et NULL
  - Rajout d'une clé primaire simple (non composite)
  - Rajout d'une clé primaire composite
  - GENERATED ALWAYS AS IDENTITY
  - Rajout d'une clé étrangère : FOREIGN KEY (fk) REFERENCES table(pk)