TD 8 : Modélisation

Modélisation

2024-11-15

Avec solutions

- **L3 MIASHS/Ingémath**
- **Université Paris Cité**
- Année 2024-2025
- Course Homepage

- Moodle

Objectifs

L'objectif de cette séance est construire des modèles Entité-Association sur des problèmes miniatures.

Modélisation Entité-Association (E/A ou E/R)

Exercice (Supermarché)

Question

Produire un schéma E/R qui décrit des informations concernant les produits d'un supermarché.

Chaque *produit* a un *nom* et un *prix* et appartient à une *catégorie*.

Le supermarché a plusieurs *rayons*, un rayon étant caractérisé par un *étage* et un *numéro de rangée*. On veut maintenir l'emplacement des produits dans les rayons. Les produits d'une même catégorie sont placés dans le même rayon, mais un rayon peut contenir des produits de plusieurs catégories.

Question

Traduire le schéma EA dans le formalisme des pattes de corbeau

Question

Définir le schéma relationnel correspondant en SQL

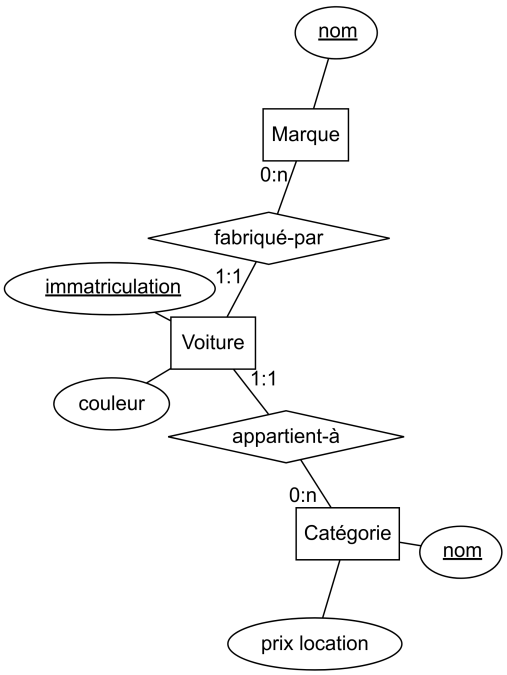
Exercice (Location de voitures)

Question

Produire un schéma E/R qui décrit des informations concernant des voitures à louer.

Chaque *voiture* a une plaque d'*immatriculation*, une *couleur* et une *marque*. Le *prix* de la location dépend de la *catégorie*, où chaque catégorie est identifiée par un nom.

Solution



Pas d'entités faibles car chaque entité a son propre identifiant.
À discuter : pourrait-on utiliser un lien `est-un` ( ) pour modéliser le fait qu'un véhicule relève d'une catégorie ?

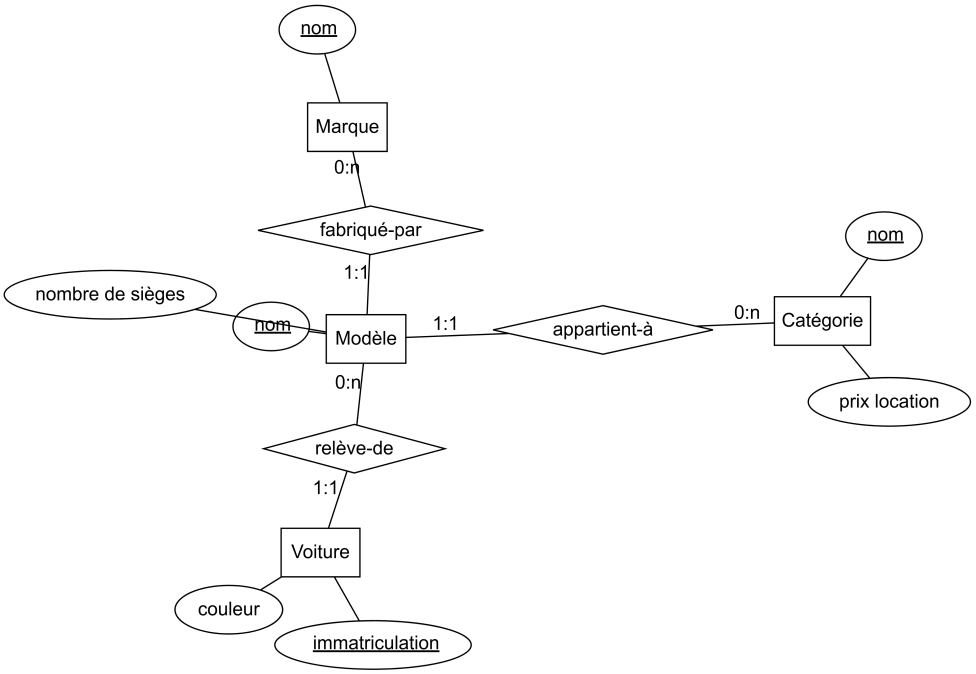
Question

Modifier ensuite le schéma pour représenter les modèles de voitures.

Un *modèle* a un *nom*, une *marque* et un *nombre de sièges*.

Toutes les voitures du même modèle doivent appartenir à la même catégorie de prix.

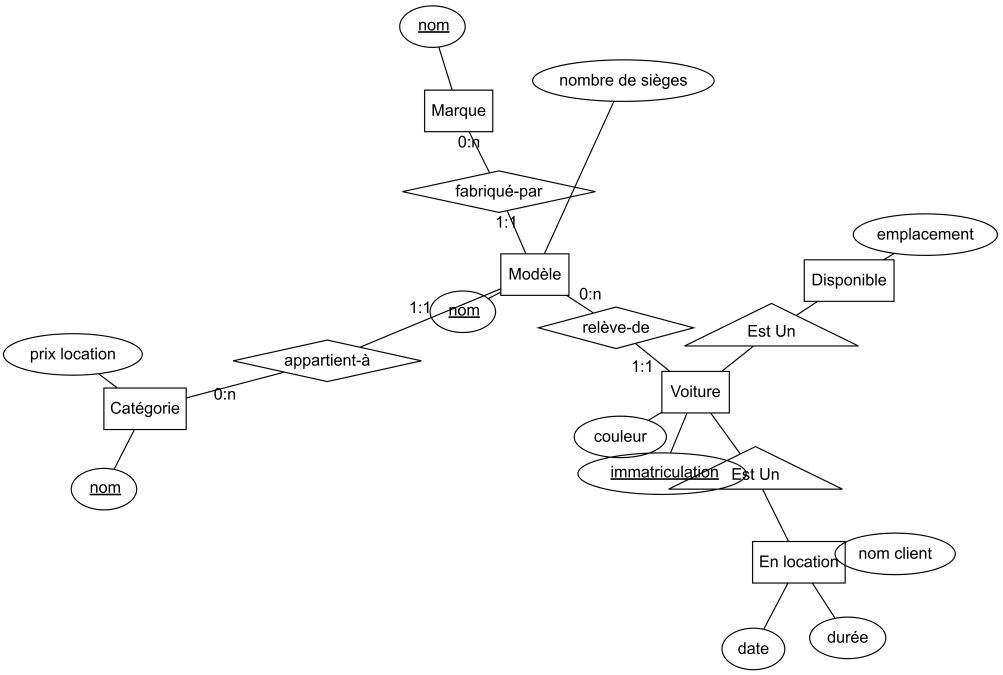
Solution



Question

De plus, on veut distinguer les voitures *disponibles* des voitures en *location*. Pour les voitures disponibles on représente l'*emplacement*. Pour les voitures en location on représente la *date* et la *durée de la location*, ainsi que le *nom du client*.

Solution

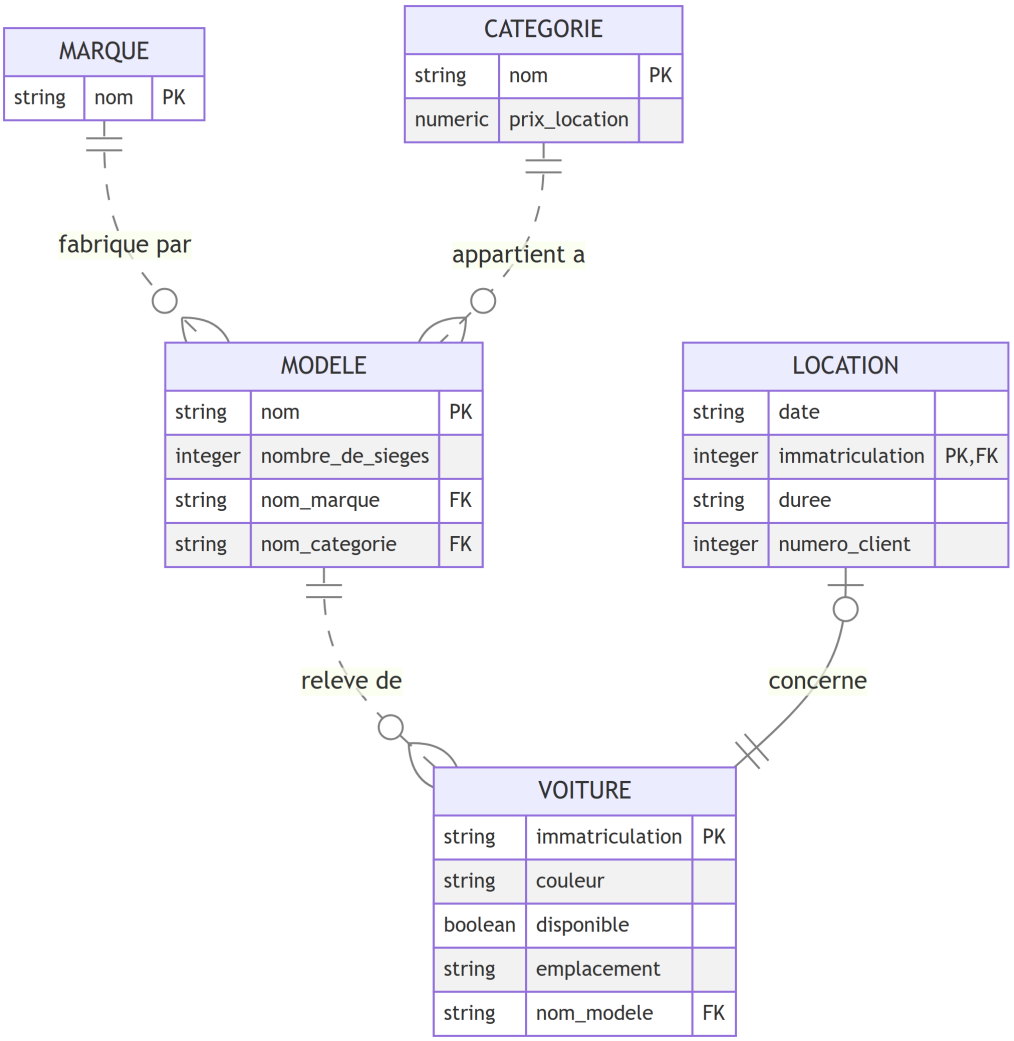


On utilise ici les liens `Est Un` pour décrire le statut des voitures (spécialisation).
Il faudrait ajouter une contrainte d'exclusion totale : une voiture est soit en location, soit disponible.
On pourrait aussi passer par des attributs `statut`, `emplacement` et une entité faible `Location`.

Question

Traduire le schéma EA dans le formalisme des pattes de corbeau

Solution



Contraintes externes :

- Dans `VOITURE`, `disponible` si et seulement si `emplacement` est `NOT NULL`
- Dans `VOITURE` et `LOCATION`, `NOT disponible` si et seulement si dans `LOCATION`, il existe une instance qui réfère à l'instance de `VOITURE`.

Trouver une meilleure modélisation pour la spécialisation `DISPONIBLE/EN LOCATION`.

Solution avec dbSchema

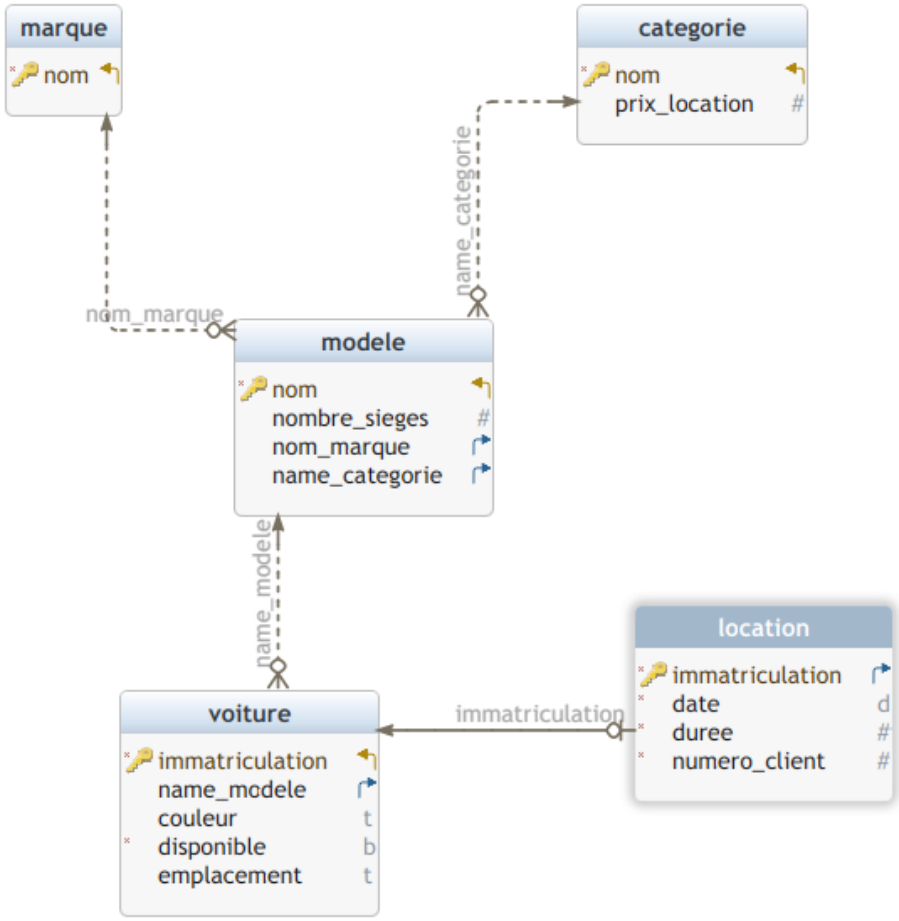


FIG. 1 : Schema Avis d'après dbSchema

Question

Définir le schéma relationnel correspondant en SQL

Solution

```
CREATE SCHEMA IF NOT EXISTS schema_avis;

CREATE  TABLE schema_avis.categorie (
    nom                 text  NOT NULL  ,
    prix_location       numeric    ,
    CONSTRAINT pk_categorie PRIMARY KEY ( nom )
 );

CREATE  TABLE schema_avis.marque (
    nom                 text  NOT NULL  ,
    CONSTRAINT pk_marque PRIMARY KEY ( nom )
 );

CREATE  TABLE schema_avis.modele (
    nom                 text  NOT NULL  ,
    nombre_sieges       bigint    ,
    nom_marque          text    ,
    name_categorie      text    ,
    CONSTRAINT pk_modele PRIMARY KEY ( nom )
 );
```

suite

```
CREATE  TABLE schema_avis.voiture (
    immatriculation     bigint  NOT NULL  ,
    name_modele         text    ,
    couleur             text    ,
    disponible          boolean  NOT NULL  ,
    emplacement         text    ,
    CONSTRAINT pk_voiture PRIMARY KEY ( immatriculation )
 );

CREATE  TABLE schema_avis.location (
    immatriculation     bigint  NOT NULL  ,
    "date"              date  NOT NULL  ,
    duree               bigint  NOT NULL  ,
    numero_client       bigint  NOT NULL  ,
    CONSTRAINT pk_location PRIMARY KEY ( immatriculation )
 );
```

suite

```
ALTER TABLE schema_avis.location
ADD CONSTRAINT
  fk_location_voiture
FOREIGN KEY ( immatriculation )
REFERENCES schema_avis.voiture( immatriculation );

ALTER TABLE schema_avis.modele
ADD CONSTRAINT
  fk_modele_marque
FOREIGN KEY ( nom_marque )
REFERENCES schema_avis.marque( nom )
ON DELETE CASCADE ON UPDATE CASCADE;

ALTER TABLE schema_avis.modele
ADD CONSTRAINT
  fk_modele_categorie
FOREIGN KEY ( name_categorie )
REFERENCES schema_avis.categorie( nom );

ALTER TABLE schema_avis.voiture
ADD CONSTRAINT
  fk_voiture_modele
FOREIGN KEY ( name_modele )
REFERENCES schema_avis.modele( nom )
ON DELETE CASCADE ON UPDATE CASCADE;
```

Exercice (Gestion du personnel d'une entreprise)

Dans une entreprise, chaque *employé* (identifié par un *numéro*) est attaché à un *département* de l'entreprise. Il occupe un *bureau* et participe à un ou plusieurs *projets* développés par l'entreprise.

De chaque employé, on connait : le *nom*, le *prénom*, les *emplois* qu'il a occupés à différentes dates et les *salaires* qu'il a *perçus* dans ces emplois.

Chaque *département* est identifié par un *numéro*, a son *budget* propre et est dirigé par un *directeur* faisant partie du personnel de l'entreprise.

Chaque *bureau* est identifié par un *numéro*, est rattaché à un *département* et est caractérisé par sa *surface* en mètres carrés. Il possède un numéro de *téléphone* associé.

Chaque *projet* est identifié par un *numéro*, possède un certain *budget* et *emploie* plusieurs *personnes* appartenant à différents *départements*. Chaque employé est *affecté* pour un *certain nombre d'heures* à un projet.

Question

Donner un modèle entité-association correspondant à la description ci-dessus.

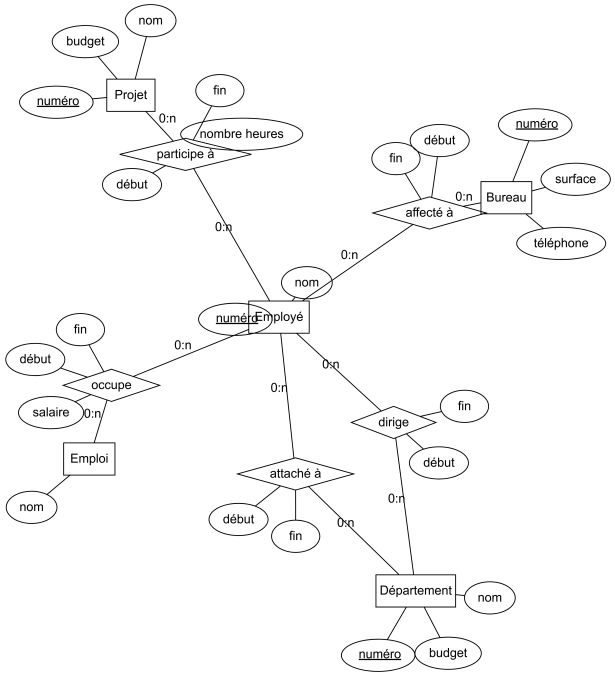
Solution

Question

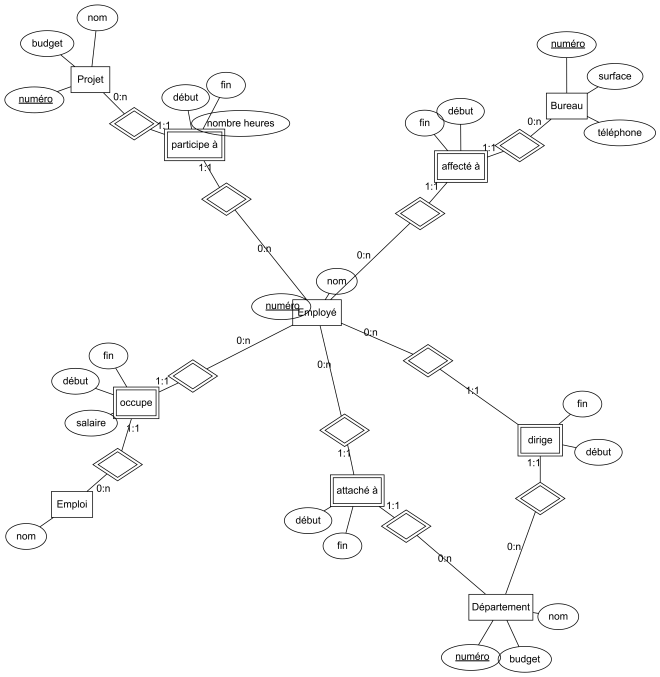
Modifier votre modélisation pour tenir compte de l'évolution dans le temps de la vie de l'entreprise : les projets ont des durées de vie limitées, chaque employé est affecté à un projet (mais aussi un département) pendant une certaines durée, etc

Solution

On munit les associations d'attributs `début` et `fin`.
👉 Il faut aussi changer les cardinalités `0:1` (respectivement `1:1`)
en `0:n` (respectivement `1:n`). Les relations *un-plusieurs* deviennent des relations *plusieurs-plusieurs*. La traduction vers les modèles en pattes de corbeau sera plus compliquée. Il faudra transformer les associations *plusieurs-plusieurs* en entité faible, relier ces entités faibles aux entités fortes participant aux associations *plusieurs-plusieurs* par des associations faibles *plusieurs-un*.



Solution



☛ Parmi les contraintes externes, il faut ajouter quelques contraintes de non-recouvrement :

- un employé ne peut pas occuper plusieurs emplois simultanément.
- un employé ne peut pas être affecté dans plusieurs bureaux simultanément.
- un employé ne peut pas être attaché à plusieurs départements simultanément.
- un employé ne peut pas participer à un même projet plusieurs fois simultanément.

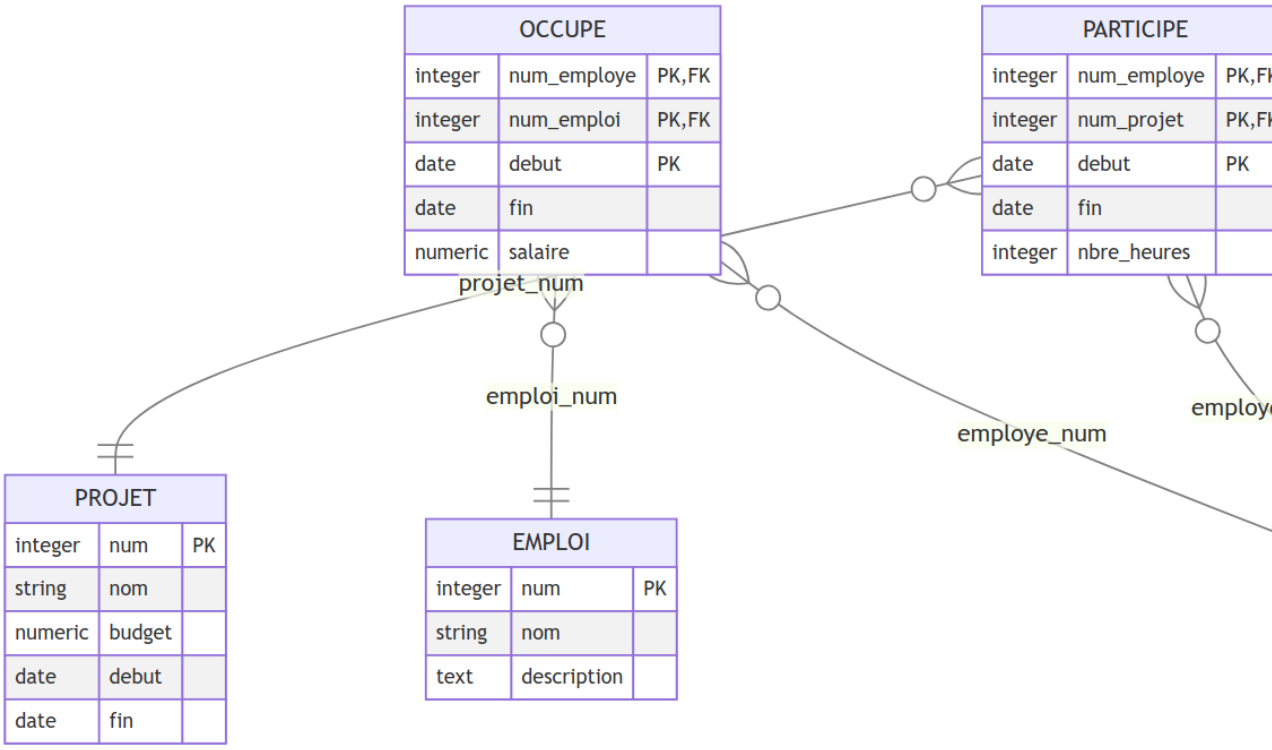
☛ À un instant donné, la somme des nombres d'heures de participation d'un même employé à différents projets ne devrait pas dépasser la durée légale du travail.

☛ À un instant donné, le nombre d'occupants d'un bureau ne devrait pas excéder une borne déterminée par la surface du bureau.

Question

Traduire le schéma EA dans le formalisme des pattes de corbeau

Solution



`MEMBRE` et `DIRIGE` ont en apparence le même schéma. Ce sont les contraintes qui distinguent les deux tables.

Solution

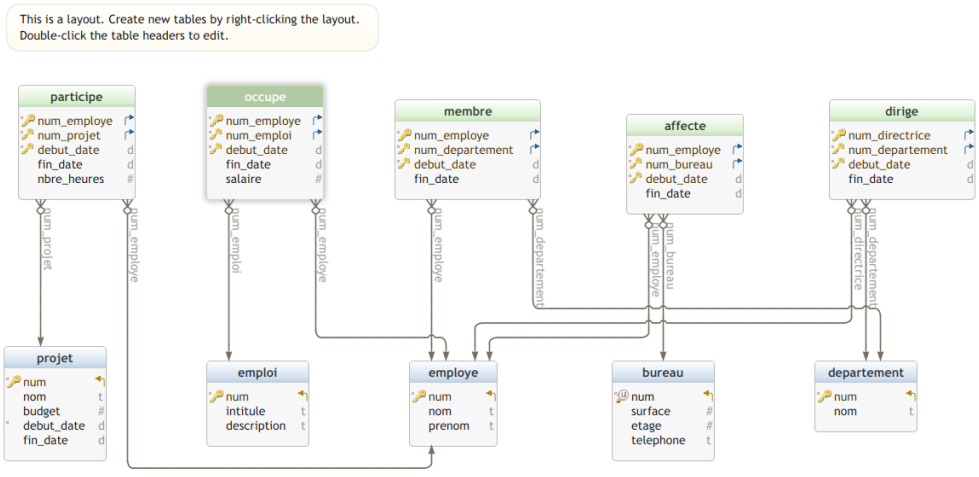


FIG. 2 : Schema entreprise

Question

Définir le schéma relationnel correspondant en SQL

Solution

```
CREATE SCHEMA IF NOT EXISTS cho;

CREATE  TABLE cho.bureau (
    num                 bigint  NOT NULL  ,
    surface             numeric    ,
    etage               integer    ,
    telephone           text    ,
    CONSTRAINT unq_bureau_num UNIQUE ( num )
 );

CREATE  TABLE cho.departement (
    num                 bigint  NOT NULL  ,
    nom                 text    ,
    CONSTRAINT pk_departement PRIMARY KEY ( num )
 );

CREATE  TABLE cho.emploi (
    num                 bigint  NOT NULL  ,
    intitule            text    ,
    description         text    ,
    CONSTRAINT pk_emploi PRIMARY KEY ( num )
 );

CREATE  TABLE cho.employe (
    num                 bigint  NOT NULL  ,
    nom                 text    ,
    prenom              text    ,
    CONSTRAINT pk_employe PRIMARY KEY ( num )
 );
```

Solution

```
-- tables intermédiaires
--
CREATE  TABLE cho.membre (
    num_employe          bigint  NOT NULL  ,
    num_departement      bigint  NOT NULL  ,
    debut_date           date DEFAULT CURRENT_DATE NOT NULL  ,
    fin_date             date DEFAULT NULL   ,
    CONSTRAINT pk_membre PRIMARY KEY ( num_employe, num_departement, debut_date )
 );

CREATE  TABLE cho.occupe (
    num_employe          bigint  NOT NULL  ,
    num_emploi           bigint  NOT NULL  ,
    debut_date           date DEFAULT CURRENT_DATE NOT NULL  ,
    fin_date             date DEFAULT NULL   ,
    salaire              money    ,
    CONSTRAINT pk_occupe PRIMARY KEY ( num_employe, num_emploi, debut_date )
 );

CREATE  TABLE cho.projet (
    num                  bigint  NOT NULL  ,
    nom                  text    ,
    budget               money    ,
    debut_date           date DEFAULT CURRENT_DATE NOT NULL  ,
    fin_date             date DEFAULT NULL   ,
    CONSTRAINT pk_projet PRIMARY KEY ( num )
 );

CREATE  TABLE cho.affecte (
    num_employe          bigint  NOT NULL  ,
    num_bureau           bigint  NOT NULL  ,
    debut_date           date DEFAULT CURRENT_DATE NOT NULL  ,
    fin_date             date DEFAULT NULL   ,
    CONSTRAINT pk_affecte PRIMARY KEY ( num_employe, num_bureau, debut_date )
 );

CREATE  TABLE cho.dirige (
    num_directrice       bigint  NOT NULL  ,
    num_departement      bigint  NOT NULL  ,
    debut_date           date DEFAULT CURRENT_DATE NOT NULL  ,
    fin_date             date DEFAULT NULL   ,
    CONSTRAINT pk_dirige PRIMARY KEY ( num_directrice, num_departement, debut_date )
 );

CREATE  TABLE cho.participe (
    num_employe          bigint  NOT NULL  ,
    num_projet           bigint  NOT NULL  ,
    debut_date           date DEFAULT CURRENT_DATE NOT NULL  ,
    fin_date             date DEFAULT NULL   ,
    nbre_heures          bigint DEFAULT 0   ,
    CONSTRAINT pk_participe PRIMARY KEY ( num_employe, num_projet, debut_date )
 );
```

Solution

```
-- contraintes de tuple

ALTER TABLE cho.participe
ADD CONSTRAINT cns_participe
CHECK (fin_date IS NULL OR fin_date >= debut_date );

ALTER TABLE cho.affecte
ADD CONSTRAINT cns_affecte
CHECK (fin_date IS NULL OR fin_date >= debut_date );

ALTER TABLE cho.dirige
ADD CONSTRAINT cns_dirige
CHECK (fin_date IS NULL OR fin_date >= debut_date );

ALTER TABLE cho.membre
ADD CONSTRAINT cns_membre
CHECK (fin_date IS NULL OR fin_date >= debut_date );

ALTER TABLE cho.occupe
ADD CONSTRAINT cns_occupe
CHECK (fin_date IS NULL OR fin_date >= debut_date );
```

Solution

```
-- contraintes referentielles
--
ALTER TABLE cho.affecte
ADD CONSTRAINT fk_affecte_employe
FOREIGN KEY ( num_employe ) REFERENCES cho.employe( num );

ALTER TABLE cho.affecte
ADD CONSTRAINT fk_affecte_bureau
FOREIGN KEY ( num_bureau ) REFERENCES cho.bureau( num );

ALTER TABLE cho.dirige
ADD CONSTRAINT fk_dirige_employe
FOREIGN KEY ( num_directrice ) REFERENCES cho.employe( num );

ALTER TABLE cho.dirige
ADD CONSTRAINT fk_dirige_departement
FOREIGN KEY ( num_departement ) REFERENCES cho.departement( num );

ALTER TABLE cho.membre
ADD CONSTRAINT fk_membre_employe
FOREIGN KEY ( num_employe ) REFERENCES cho.employe( num );

ALTER TABLE cho.membre
ADD CONSTRAINT fk_membre_departement
FOREIGN KEY ( num_departement ) REFERENCES cho.departement( num );

ALTER TABLE cho.occupe
ADD CONSTRAINT fk_occupe_employe
FOREIGN KEY ( num_employe ) REFERENCES cho.employe( num );

ALTER TABLE cho.occupe
ADD CONSTRAINT fk_occupe_emploi
FOREIGN KEY ( num_emploi ) REFERENCES cho.emploi( num );

ALTER TABLE cho.participe
ADD CONSTRAINT fk_participe_employe
FOREIGN KEY ( num_employe ) REFERENCES cho.employe( num );

ALTER TABLE cho.participe
ADD CONSTRAINT fk_participe_projet
FOREIGN KEY ( num_projet ) REFERENCES cho.projet( num );
```

Question

Essayer de coder les contraintes externes (exclusion, vérification, unicité, ...)
On s'intéresse d'abord aux contraintes qui pèsent sur les associations entre `employe` et `departement`.

- Un employé ne peut être membre de plusieurs départements simultanément
- Un département ne peut pas être dirigé simultanément par plusieurs employés
- Un employé ne peut pas diiger plusieurs départements simultanément

Solution

```
-- Un employé ne peut être membre de plusieurs  départements simultanément
ALTER TABLE cho.membre
ADD CONSTRAINT exc_membre_1 EXCLUDE USING gist (
  num_employe WITH =,
  num_departement WITH <>,
  daterange(debut_date, fin_date) WITH &&
) ;
```

```
-- Un département ne peut pas être dirigé simultanément par plusieurs employés
ALTER TABLE cho.dirige
ADD CONSTRAINT exc_dirige_1 EXCLUDE USING gist (
  num_directrice WITH <>,
  num_departement WITH =,
  daterange(debut_date, fin_date) WITH &&
) ;
```

```
-- Un employé ne peut pas diriger simultanément par plusieurs  départements
ALTER TABLE cho.dirige
ADD CONSTRAINT exc_dirige_2 EXCLUDE USING gist (
  num_directrice WITH =,
  num_departement WITH <>,
  daterange(debut_date, fin_date) WITH &&
) ;
```

Question

Essayer de coder les contraintes externes (exclusion, vérification, unicité, …)
On s'intéresse maintenant aux contraintes qui pèsent sur les associations entre `employe` et `projet`.

- Un employé ne peut participer à un projet que pendant la durée de vie du projet
- Un employé ne peut pas travailler plus de 50 heures par semaine

La première contrainte concerne deux tables `projet` et `participe` : il faut que l'intervalle spécifié par `debut_date, fin_date` dans `participe` soit inclus dans l'intervalle `debut_date, fin_date` de l'instance de `projet` désignée par `num_projet`. Cette vérification devrait être effectuée lors des insertions/mises à jour dans `participe` mais aussi lors des mises à jour dans `projet`.
Pour mettre en place de genre de contraintes, `SQl` et `PostgreSQL` offre un cadre : celui des gachettes (`TRIGGER`). Cela va au delà de ce cours. Nous allons essayer de faire avec les moyens dont nous disposons : les fonctions `SQL` et les contraintes `CHECK`.

```
CREATE FUNCTION cho.chk_participation_in_project_range(
  p_num_projet bigint,
  p_debut_date date,
  p_fin_date date
)
RETURNS integer
LANGUAGE SQL AS
$$
SELECT
  COUNT(*)
FROM
  cho.projet pr
WHERE
  p_num_projet=pr.num
  AND
  daterange(p_debut_date, p_fin_date) <@ daterange(pr.debut_date, pr.fin_date) ;
$$ ;
```

```
ALTER TABLE cho.participe
ADD CONSTRAINT cns_participe_2
CHECK (
  1 = cho.chk_participation_in_project_range(
        num_projet,
        debut_date,
        fin_date
  )
)
;
```

Il faudrait créer une fonction et une contrainte `CHECK` du côté `projet`.
Il faudrait aussi vérifier que la directrice d'un département est membre du département …