IPEFA Sup Seraing - Verviers

Projet de développement SGDB

Notes de cours

Georgette Collard

2025-2026

Table des matières

Chapitre 1 : Introduction

Depuis la conception jusqu'à l'usage d'une base de données, plusieurs étapes doivent être considérées :

1) **La création du modèle entité-association** : représentation graphique des concepts du monde réel qu'on veut mémoriser

2) **La création du modèle relationnel** : le modèle entité-association est converti en un modèle relationnel, un modèle constitué d'un ensemble de tables.

3) **La création de la base de données** : à partir du modèle relationnel, la base de données et les tables sont créées dans un système de gestion de bases de données (SGDB).

4) **L'utilisation de la base de données** : une application va transmettre des requêtes SQL au SGDB en vue d'ajouter, de consulter, de modifier et de supprimer des lignes dans les tables des bases de données.

Chapitre 2 : Le modèle entité-relation (ou entité – association)

Le modèle entité-relation (*entity-relationship*) est un cadre de définition général du type de contenu potentiel d'une base de données. Il utilise deux types génériques de base :
Les ensembles d'entités : ensemble d'objets de même structure (enregistre les mêmes informations)
Les relations (ou associations) entre ces ensembles.

Les entités

Une entité (*entity*) est un objet (abstrait ou concret) au sujet duquel on conserve de l'information dans la base de données. Il faut qu'une entité soit individualisable, en d'autres mots qu'il puisse être distingué d'une autre entité.

Exemples :

- Entité concrètes : une personne, une voiture
- Entité abstraites : un trajet, un horaire

Note : Du point de vue d'une base de données, une entité n'a de sens que si l'on peut identifier les informations qui seront conservé à son sujet.

Les ensembles d'entités (*entity sets*)

Les entités sont regroupées en ensembles d'entités semblables (de même type), c'est-à-dire au sujet desquelles on veut conserver la même information.

Exemples :

- Les acteurs d'un film
- Les employés d'une firme
- Les cours de la promotion sociale de Seraing

Rôle

La participation d'un ensemble d'entités à une relation est son **rôle.** On donne un **nom** au rôle d'un ensemble d'entités dans une relation.

Exemples :

- Dans la relation *ENSEIGNE* : *EMPLOYE_PROV_LIEGE, COURS_PROSOC*
  - Le rôle de l'ensemble d'entités *EMPLOYE_PROV_LIEGE* est « *enseignant* »
  - Le rôle de l'ensemble d'entités *COURS_PROSOC* est « *cours enseigné* »
- Dans la relation PATERNITE : *PERSONNE, PERSONNE*
  - L'ensemble d'entités PERSONNE a deux rôles distincts : « *père* » et « *enfant* »

Contraintes de cardinalité

Pour chaque ensemble d'entités participant à une relation, c'est-à-dire pour chaque rôle d'un ensemble d'entités, on précise dans combien de tuples de la relation chaque entité peut apparaître.

En fait, on ne donne pas un nombre mais on donne un intervalle : (*min, max*) où

- *min* est en général 0 ou 1
- *max* est en général 1 ou N (*)

Exemple :

> Quelle est la participation de l'ensemble d'entités *EMPLOYE_PROV_LIEGE* dans la relation *ENSEIGNE* ?
>
> En d'autres termes, quelles est la cardinalité du rôle « *enseignant* » dans la relation *ENSEIGNE*?
>
> Ou encore, combien de fois un employé particulier de la province de Liège peut-il apparaître dans une quelconque extension de la relation *ENSEIGNE*
>
> - *min* : 0
> - *max* : N

Diagrammes entité-relation (*entity-relationship diagrams*)

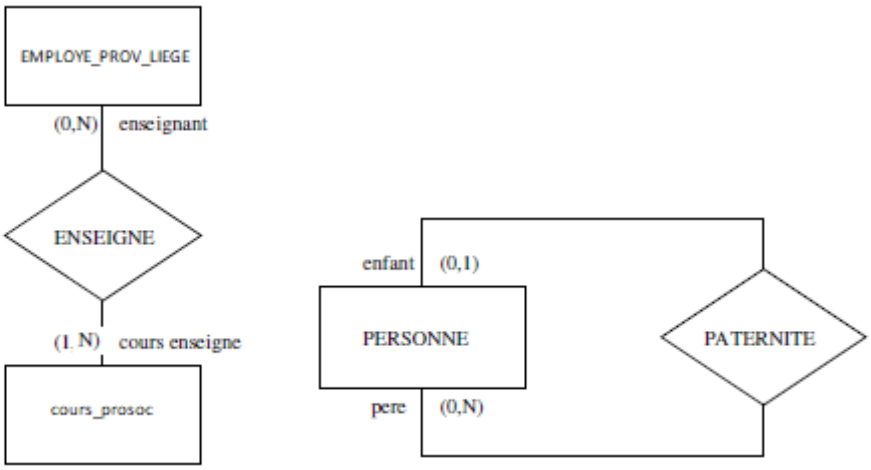
Un diagramme entité-relation est une description graphique des ensembles d'entités des relations qui seront représentés dans une base de données.

Ces diagrammes représentent donc le schéma des données qui seront représentées, par opposition à leur extension qui sera le contenu de la base de données.

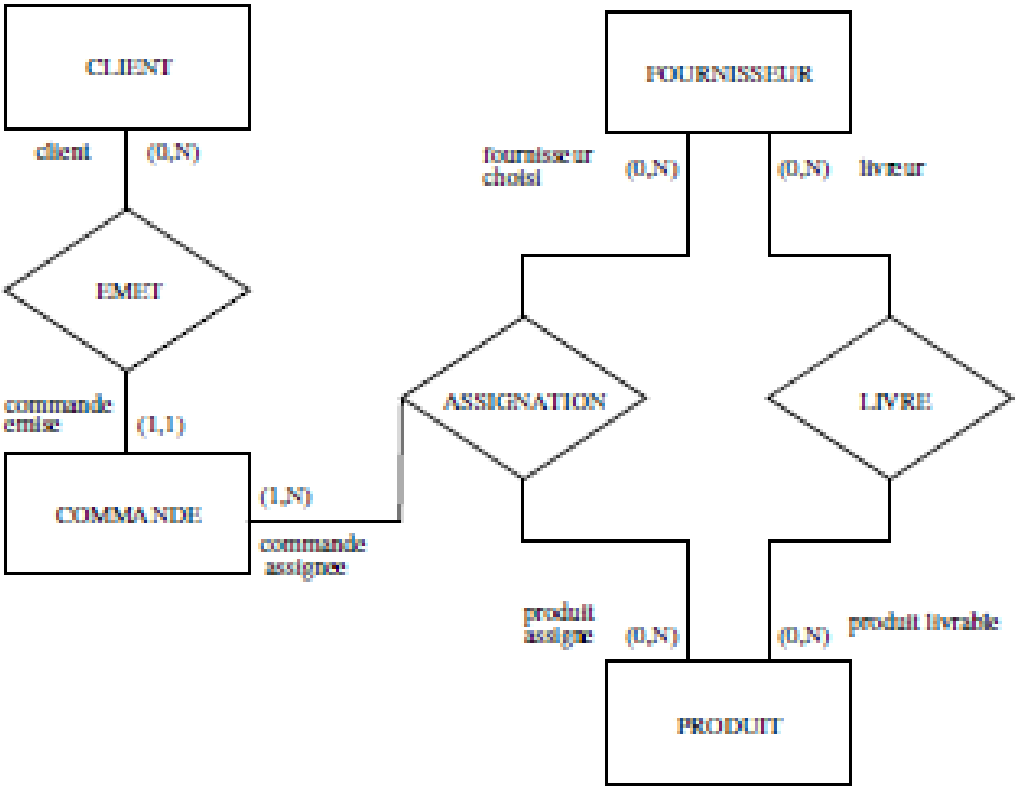
Notation :

- Ensemble d'entité : nom dans un rectangle
- Relation : nom dans un losange ou un hexagone avec traits vers les ensembles d'entités participant à la relation (chaque trait est étiqueté par le nom du rôle et la contrainte de cardinalité correspondants)

Exemples



On veut modéliser le fait que les clients peuvent passer des commandes à une centrale d'achat qui a des fournisseurs habituels pour les différents produits possibles et qui décide par quel fournisseur un produit commandé doit être livré.



Les attributs

Les informations conservées au sujet des entités d'un ensemble sont leurs **attributs**.

Exemple :

- Le nom, l'adresse, la date de naissance d'une personne
- L'intitulé, la charge horaire d'un cours

Toutes les entités d'un ensemble ont exactement les mêmes attributs.

Chaque attribut d'un ensemble d'entités

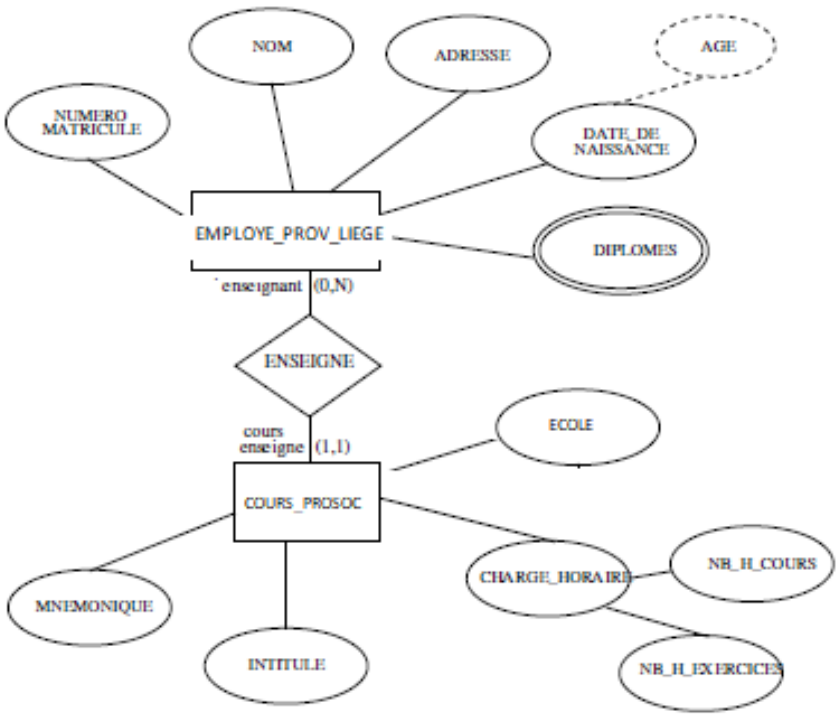
- a un *nom* unique dans le contexte de cet ensemble d'entités,
- prend ses valeurs dans un domaine spécifié (*type de l'attribut*).

Différentes catégorie d'attributs

Un attribut peut être

- **Simple** (atomique) ou **composite** (décomposable en plusieurs attributs plus simples)
  Exemple : l'attribut *ADRESSE* peut être constitué des attributs plus simples :
  RUE_ET_NUMERO, BOITE, VILLE, CODE_POSTAL, PAYS.
- **Obligatoire** (une entité doit avoir une valeur pour cet attribut) ou **facultatif**
- À **valeur unique** ou à **plusieurs valeurs**
  Exemple : l'attribut *DIPLOMES* (d'une personne) peut avoir plusieurs valeurs pour une même personne.
- **Enregistré** ou **dérivé** (d'un autre attribut)

  Exemple : l'attribut *AGE* (d'une personne) peut être dérivé de l'attribut enregistré *DATE_DE_NAISSANCE*.



Clé

Une **clé** d'un ensemble d'entités est un ensemble minimum d'attributs qui identifient de façon unique une entité parmi cet ensemble.

Donc, deux entités distinctes d'un ensemble ne peuvent jamais avoir les mêmes valeurs pour les attributs de la clé.

Exemples :

- Le numéro de matricule d'un enseignant
- La marque et le numéro de série d'une voiture

Notation : les attributs de la clé sont soulignés.

Ensembles d'entités faibles

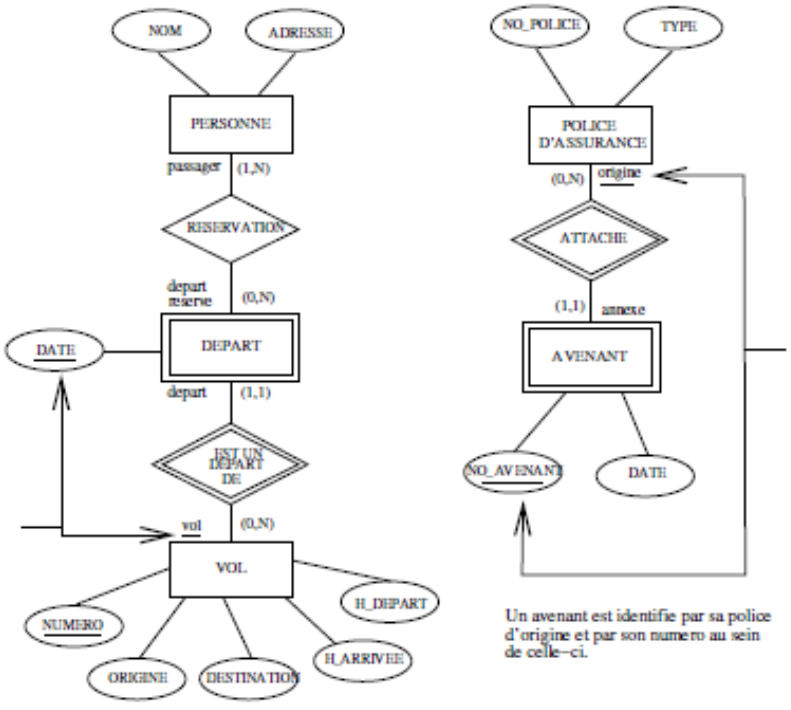
Il peut arriver qu'un ensemble d'entité ne dispose pas d'attributs constituant une clé : c'est un **ensemble d'entités faibles**

Les entités d'un ensemble faible se distinguent les unes des autres par

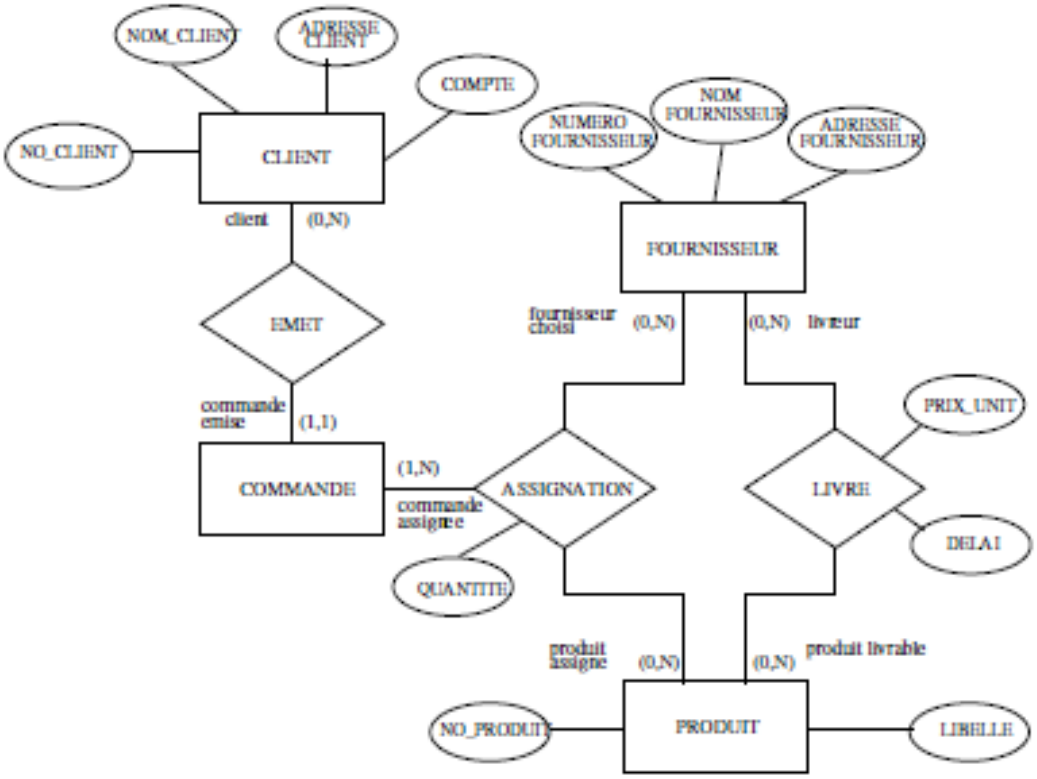
- (éventuellement) certains de ses attributs,
  et
- Le fait qu'elles sont en relation avec d'autres ensembles d'entités.

On doit donc généraliser la notion de clé : la clé d'un ensemble d'entités est constituée

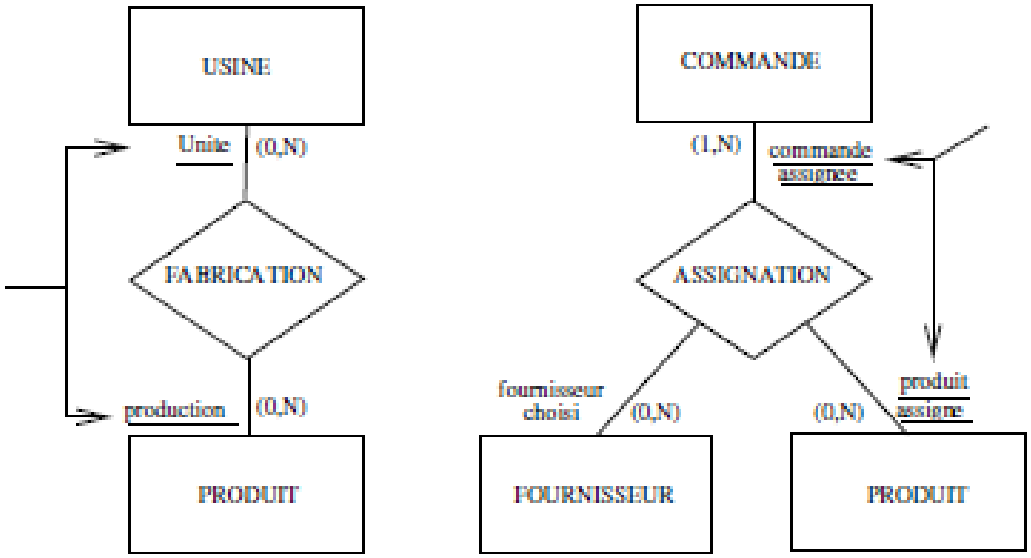
- D'attributs de l'ensemble d'entités
  et/ou
- De rôles joués par d'autres ensembles d'entités dans leur relation avec cet ensemble d'entités (**relations identifiantes**).



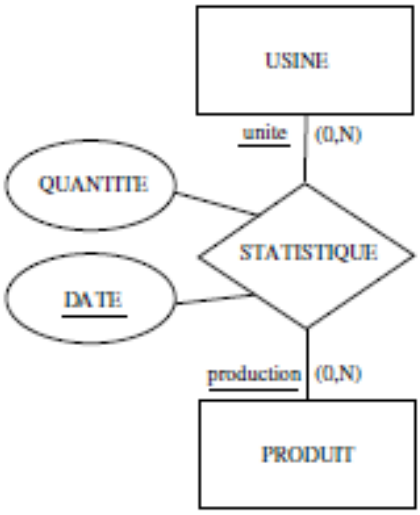
Exemple (attributs de relations) :

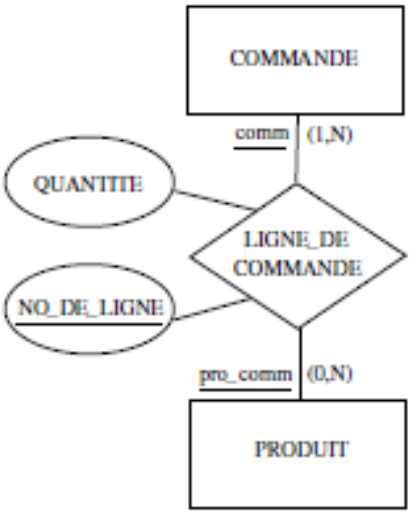


Exemples (clés de relations) :



Exemples (clés de relations) :





Contraintes d'intégrité

Les **contraintes d'intégrité** sont des contraintes que doivent satisfaire les données d'une base de données à tout instant. Elles sont exprimées au niveau du schéma.

- Les contraintes de cardinalité (*min, max*)
- Les clés
- Contraintes sur les attributs
- Contraintes référentielles (on ne fait référence qu'à une entité existante)
- etc

Une extension du modèle entité-relation : Relation IS-A

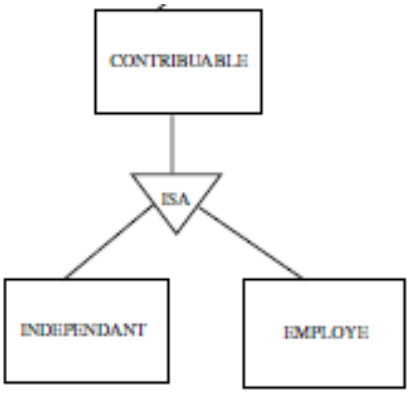
Il est parfois intéressant de définir des ensembles d'entités de façon hiérarchique, la hiérarchie correspondant à une structure d'inclusion.

Ensemble d'entité inclus : **sous-classe** (ou ensemble d'entités spécifique)

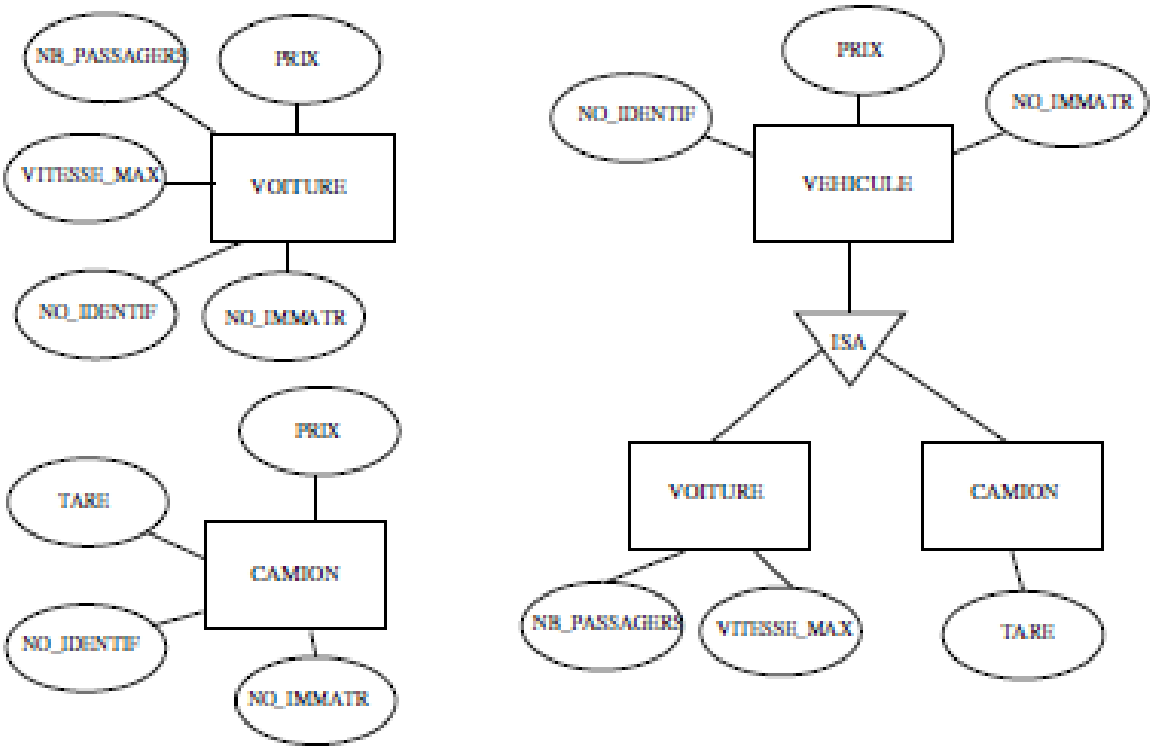
Ensemble d'entités en comprenant d'autres : **super-classe** (ou ensemble d'entités génériques)

Exemple : **spécialisation** : division d'un ensemble d'entités en **sous-classes**

Un contribuable peut être soit indépendante, soit salariée



Exemple : **généralisation** : regroupement de plusieurs ensembles d'entités en une **super-classe.**



Une classe peut être divisée selon plusieurs critères indépendants, donnant lieu à plusieurs spécialisations différentes.

Exemple : L'ensemble d'entité *EMPLOYE* peut donner lieu à deux taxonomies basées sur des critères distincts :

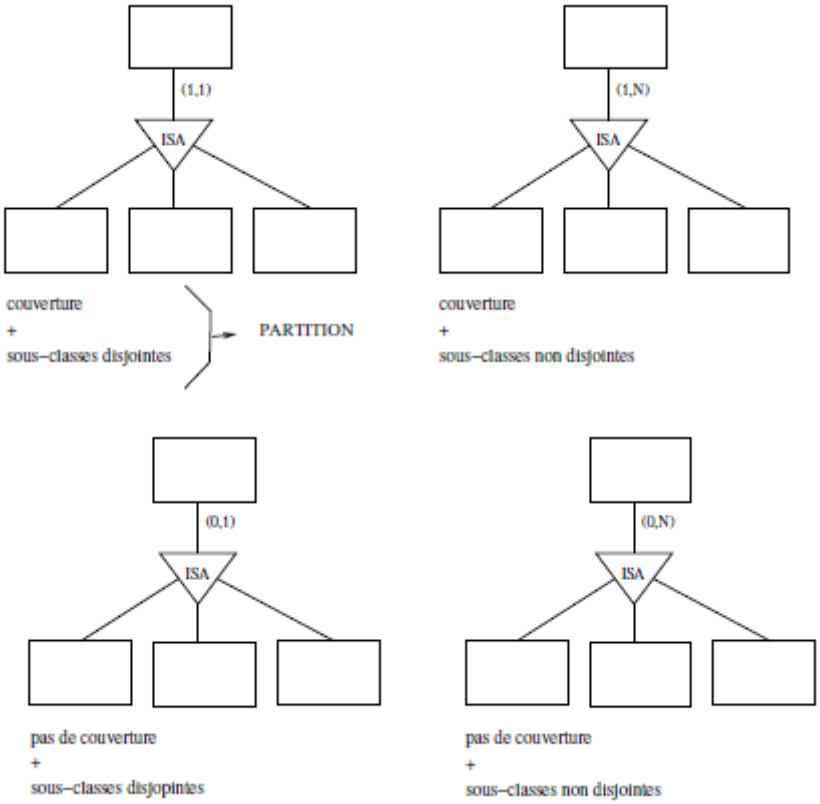
- La fonction de l'employé donne lieu à une spécialisation en 3 sous-classes : *SECRETAIRE, TECHNICIEN, CADRE*
- Le type de numération donne lieu à une autre spécialisation, complètement indépendante de la première, en 2 sous-classes : *SALARIE, CONSULTANT*

Une classe peut être la sous-classe de plusieurs superclasses, participant ainsi à plusieurs généralisations.

Contraintes sur les spécialisations/généralisations

Deux catégories de contraintes indépendantes :

- Quand l'union des sous-classes donne la super-classe, les sous-classes constituent une **couverture** de la super-classe. Dans ce cas, la cardinalité minimum du rôle de la super-classe dans la relation IS-A est 1 s'il y a couverture, et est 0 sinon.
- Les sous-classes d'une super-classe peuvent être **disjointes** ou non. Dans ce cas, la cardinalité maximum du rôle de la super-classe dans la relation IS-A est 1 si les sous-classes sont disjointes, et N sinon.



Héritage

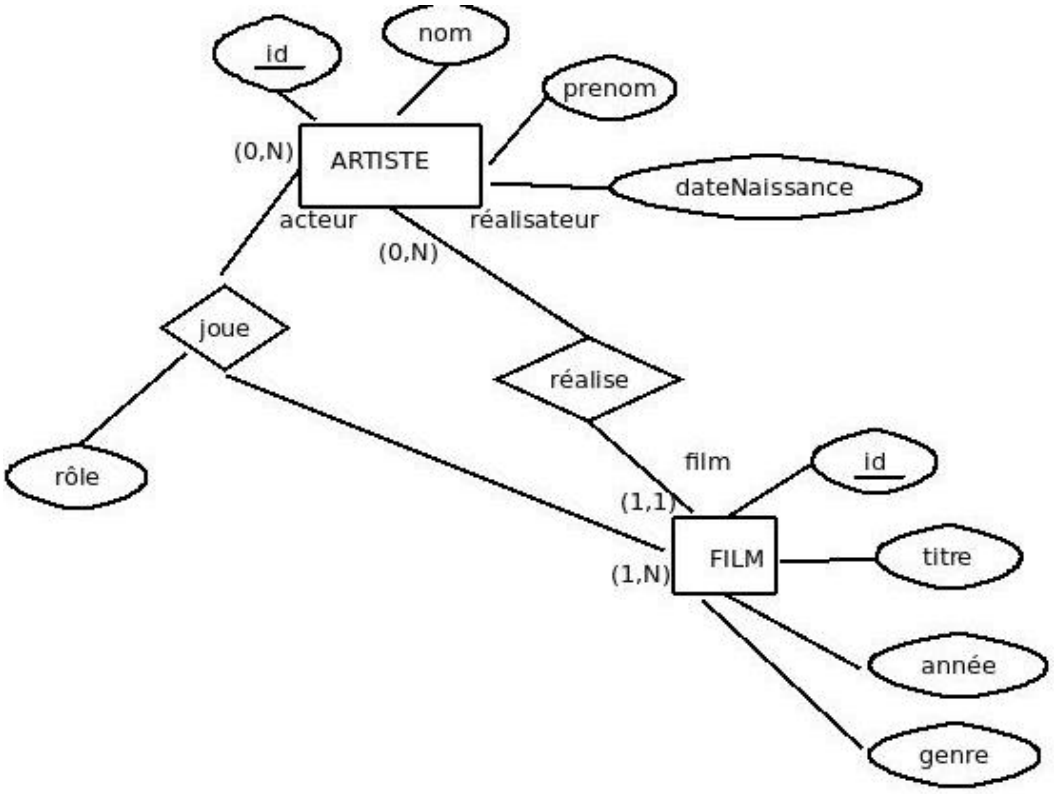
A l'inclusion entre ensembles d'entités correspond **l'héritage des propriétés** (attributs et rôles).

Lorsqu'un ensemble d'entités est une sous-classe de deux classes différentes, on parle **d'héritage multiple.**

Exercices

Exercice résolu

Soit une base de données décrivant des films avec leur réalisateur et leurs acteurs. Un film est réalisé que par un seul réalisateur. Un acteur ne joue qu'un rôle dans un film donné. Les réalisateurs et les acteurs sont des artistes. Chaque artiste est caractérisé par un numéro, un nom, un prénom et une date de naissance. Un film est caractérisé par un titre, une année de sortie, un genre. Le réalisateur peut être acteur dans ses films réalisés.



Processus d'élaboration d'un modèle entité-association

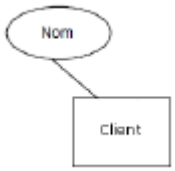
1. On décompose l'énonce en **propositions élémentaires (propositions binaires)**. Une proposition binaire affirme l'existence de 3 types de faits: 2 concepts et 1 lien entre eux.

2. Les types de faits contenus dans chaque proposition binaire sont ajoutés au modèle entité-association s'ils y sont absents et s'ils n'entrent pas en contradiction avec les informations déjà présentes dans le modèle.

   Chaque concept d'une proposition binaire est représenté dans le modèle entité-association soit par un ensemble d'entités, soit par un attribut d'un ensemble d'entités.

   - Quand un concept est une propriété de l'autre concept, alors ce concept-là devient un attribut de l'autre concept qui lui devient un ensemble d'entités.

     Exemple. Pour la proposition Tout client a un nom, les concepts sont *client* et *nom*. Le lien est *a*. *nom* devient un attribut de l'ensemble d'entités *Client*:

   - Quand les 2 concepts sont indépendants, c'est-à-dire quand aucun des 2 concepts n'est une propriété de l'autre, alors ils deviennent chacun un ensemble d'entités.

     Exemple. Pour la proposition Une commande est passée par un client, les concepts sont *commande* et *client*. Le lien est *est passée par*. Les concepts *commande* et *client* deviennent 2 ensembles d'entités et le lien *est passée par* devient une association:

Remarque: il importe de considérer les deux types suivants de propositions non binaires:

- **Une proposition est non binaire réductible** quand elle contient plus de 2 concepts et peut être décomposée directement en plusieurs propositions binaires.

  Exemple. Un trajet organise entre une ville de départ et une ville d'arrivée peut se réduire en:
  - un trajet a une ville de départ.
  - un trajet a une ville d'arrivée.

  Exemple. Les clients ont un nom et une adresse peut se réduire en:
  - les clients ont un nom.
  - les clients ont une adresse.

- **Une proposition est non binaire non réductible** quand elle contient plus de 2 concepts et ne peut être décomposée directement en plusieurs propositions binaires.

Exemple. Les clients achètent des produits chez des fournisseurs. On ne peut, sans perdre de l'information, réduire cette proposition en les 3 propositions binaires suivantes:

- Les clients achètent des produits.
- Les fournisseurs vendent des produits.
- Les clients achètent chez des fournisseurs.

Pour illustrer ce problème, supposons les clients Jean et Paul, les produits P1 et P2, et les fournisseurs F1 et F2 ainsi que la proposition initiale *Les clients achètent des produits chez des fournisseurs*. Supposons aussi les 3 faits suivants :

- Jean achète P1 chez F1.
- Jean achète P2 chez F2.
- Paul achète P1 chez F2.

En réduisant ces 3 faits non binaires en des faits binaires, on obtient les 9 faits binaires suivants :

- Jean achète P1.
- Jean achète P2.
- Paul achète P1.
- F1 vend P1.
- F2 vend P1.
- F2 vend P2.
- Jean achète chez F1.
- Jean achète chez F2.
- Paul achète chez F2.

Ces 9 faits binaires ne sont pas équivalents aux 3 faits non binaires donnés plus haut, car, à partir de ces faits binaires, on peut générer, par exemple, le fait *Jean achète P1 chez F2* lequel dans la réalité est faux !!!

La solution à ce problème est d'ajouter le **concept central** *achat* dans la proposition initiale Les clients achètent des produits chez des fournisseurs. Celle-ci devient alors les clients font des achats de produits chez des fournisseurs. Cette proposition non binaire peut maintenant être réduite en les 3 propositions binaires suivantes:

- Un achat est effectué par un client.
- Un achat concerne un produit.
- Un achat s'effectue chez un fournisseur.

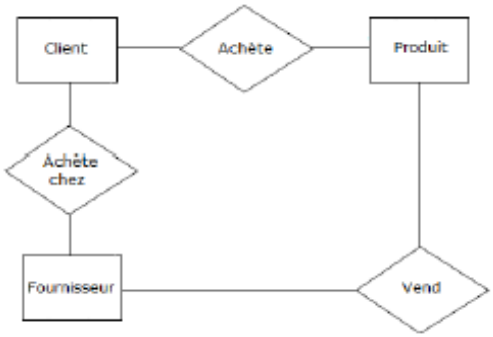
Les 3 faits non binaires originaux peuvent alors être exprimes ainsi:

- l'achat X est effectué par Jean.
- l'achat Y est effectué par Jean.
- l'achat Z est effectué par Paul.
- l'achat X concerne P1.
- l'achat Y concerne P2.
- l'achat Z concerne P1.
- l'achat X s'effectue chez F1.
- l'achat Y s'effectue chez F2.
- l'achat Z s'effectue chez F2.

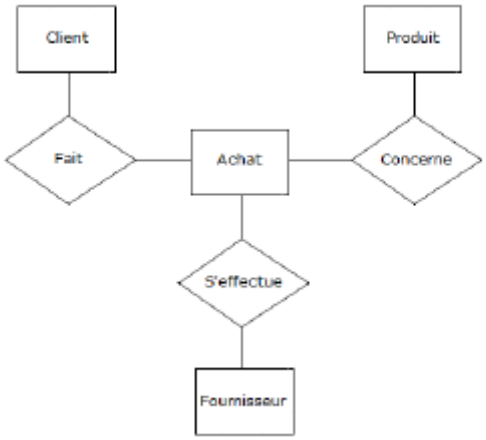
Grâce au concept d'achat, on retrouve les 3 faits originaux:

- l'achat X concerne Jean, P1 et F1.
- l'achat Y concerne Jean, P2 et F2.
- l'achat Z concerne Paul, P1 et F2.
-

Modèle entité-association avant l'ajout du concept Achat :



Modèle entité-association après l'ajout du concept Achat :



Exercices

Exercice 1

Produire un modèle entité-association qui décrit les 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.

Exercice 2

Un tournoi de tennis est constitué de joueurs, de terrains et de rencontres (matchs de simple). Le tournoi comprend plusieurs rencontres. Un joueur participe à une rencontre, laquelle se joue sur un terrain. Après chaque rencontre, seul un joueur est déclaré vainqueur. Le joueur est décrit par un matricule unique, un nom et un classement. La rencontre est décrite par un numéro unique, une date et une heure précise. Un terrain est décrit par un numéro unique et une surface (terre battue extérieur, terre battue intérieur, french court).

Exercice 3

La sécurité sociale veut développer un système informatique permettant de suivre les prescriptions médicales effectuées par les médecins. Chaque médecin consulte un patient à la fois. Au terme de la consultation, le médecin peut prescrire des médicaments aux patients. Un médecin est caractérisé par un matricule, un nom, un prénom, une adresse et sa spécialité. Un patient est caractérisé par un numéro de sécurité sociale, un nom, un prénom, une adresse et sa mutuelle. Le médicament est décrit par un code, par un nom, par une description.

Exercice 4

Un journaliste est identifié par son matricule, par un nom et par une date de naissance. Un article est décrit par un numéro et un contenu. Un article est rédigé que par un seul journaliste et traite que d'un seul sujet. Un sujet donné peut être traité par plusieurs articles. Un article n'apparaît que dans une seule édition de journal. L'édition d'un journal se caractérise par un numéro et une date de parution. Un journaliste peut travailler pur plusieurs journaux. Un journal est caractérisé par un titre et par une adresse.

Exercice 5

Un livre appartient à une seule catégorie. Un livre est identifié par son ISBN et est caractérisé par un titre, auteur et la maison d'édition. Une catégorie est identifiée par un numéro unique et est caractérisé par un nom obligatoire et éventuellement une sous-catégorie. Il peut exister plusieurs exemplaires pour un livre donné. Chaque exemplaire est identifié par un numéro unique.
Une personne (emprunteur) est identifiée par un identifiant unique, un nom, prénom, adresse, GSM. Une personne peut emprunter à un moment donné au maximum 10 livres, et peut éventuellement prolonger un exemplaire emprunté mais qu'une seule fois. Un livre emprunté peut être prolongé endéans les 28 jours de l'emprunt. Un livre emprunté ou prolongé doit être restitué endéans les 28 jours après la date de sortie ou la date de prolongation.

Chapitre 3 : Le modèle relationnel

Une base de données permet de centraliser, stocker, rechercher, supprimer, consulter tous type d'informations. C'est un ensemble organisé d'informations traitant d'un même sujet.

Les principaux avantages d'une base de données sont

- **Non redondance d'informations** : afin d'éviter les problèmes lors des mises à jour, chaque donnée ne doit être présente qu'une seule fois dans la base

- **Cohérence des données** : Les données sont soumises à un certain nombre de contraintes d'intégrité qui définissent un état cohérent de la base. Elles doivent pouvoir être exprimées simplement et vérifiées automatiquement à chaque insertion, modification ou suppression des données. La cohérence implique qu'il n'y a pas de redondance de données (un acteur ne peut pas jouer un rôle dans un film qui n'est pas connu du système ou l'âge d'une personne doit être positif).

Un système de gestion de base de données (SGBD) est un logiciel qui permet de manipuler les informations stockées dans une base de données.

Le modèle relationnel (**SGBDR**, *Système de gestion de bases de données relationnelles*) : les données sont enregistrées dans des tableaux à deux dimensions (lignes et colonnes).

Une ligne représente un **enregistrement**. Les colonnes représentent les **champs**. Un enregistrement correspond à une entrée complète comparable à une fiche. Chaque enregistrement contient des champs, chacun des champs comprend une partie d'information de l'enregistrement comme par exemple le nom, le prénom, l'adresse, … Une colonne (un champ, un attribut) prends sa valeur parmi un **domaine de valeurs**. Celui-ci représente des valeurs possibles pour la colonne. Un domaine de valeurs correspond à un type de données.

La **clé primaire** d'une table est un ensemble minimum de colonnes qui identifie de façon unique un enregistrement (une ligne de la table). On ne peut enlever aucun attribut à la clé primaire sans lui faire perdre son statut de clé primaire. Deux lignes distinctes d'une table ne peuvent jamais avoir la même valeur pour la clé primaire.

Une **clé étrangère** dans une table est un ensemble de colonnes qui correspond à la copie de la clé primaire d'une autre table.

Contrainte d'unicité : aucun composant d'une clé PRIMAIRE ne peut être nul.

Contrainte référentiel : Une clé étrangère est soit nulle, soit égale à une valeur de la clé primaire de la table qu'elle référence.

Passage du modèle entité-association au modèle relationnel

Exemple : Réalisateur, comédiens de film

ARTISTES

Id	nom	prenom	anneeNaissance
200	Veber	Francis	28/7/1937
201	Cameron	James	16/8/1954
202	Spielberg	Steven	18/12/1946
203	Villeret	Jacques	6/2/1951
204	Prévost	Daniel	20/10/1939

FILMS

id	titre	annee	genre	idRealisateur
10	Le diner de cons	1998	Comédie	200
11	Titanic	1997	Romance dramatique	201
12	La liste de Schindler	1993	Drame historique	201

ROLES

idFilm	idArtiste	role
10	203	François Pignon
10	204	Lucien Cheval

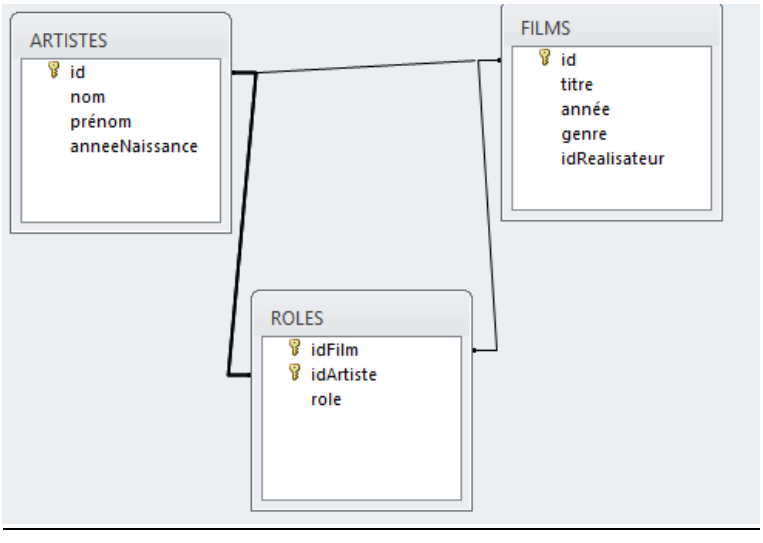
Schéma

ARTISTES(id, nom, prenom, anneeNaissance)

FILMS(id, titre, annee, genre, idRealisateur)

ROLES(idFilm, idArtiste, role)

Base de données relationnelles



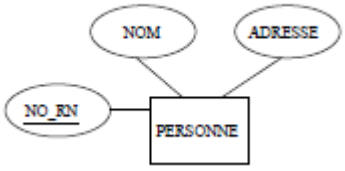
Comment convertir un modèle entité-association en un modèle relationnel ?

Un **ensemble d'entité** (non faible) est représenté par une relation dont les attributs sont les attributs simples de l'ensemble d'entités.

Les **attributs composites** sont remplacés par leurs composantes.

Les **attributs multivalués** : voir plus loin

Exemple :

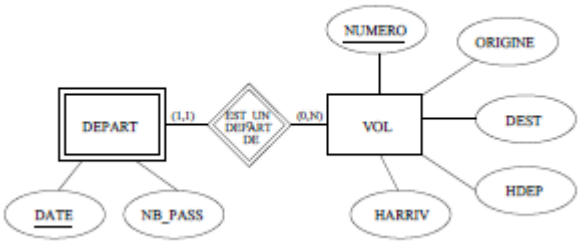


PERSONNE(no_rn, nom, adresse)

Un **ensemble d'entité faible** est représenté par une relation dont les attributs sont

- Les attributs de l'ensemble d'entité plus
- Les attributs de la clé de chacun des ensembles d'entités participant aux relations identifiantes de l'ensemble d'entité faible.

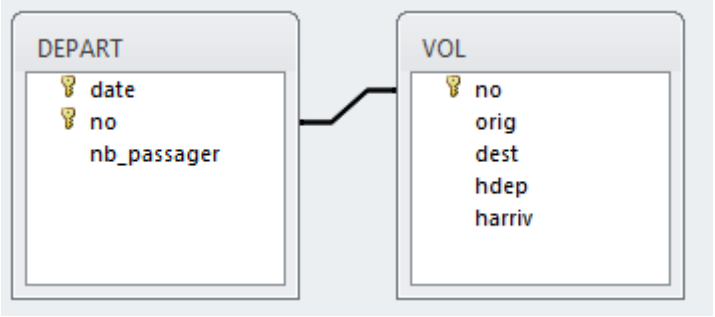
Exemple :



VOL(no, orig, dest, hdep, harriv)

DEPART(date, no, nb_passagers)

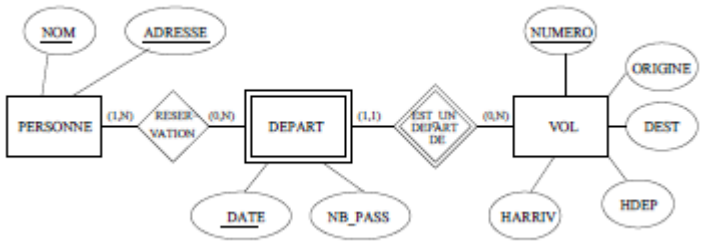
Sous forme de tables:



Une **association entre plusieurs ensembles d'entités** est représentée par une relation dont les attributs sont

- Les attributs de l'association plus
- Les attributs de la clé de chacun des ensembles d'entités participant à la relation.

Exemple :



RESERVATION(nom, adresse, date, no)

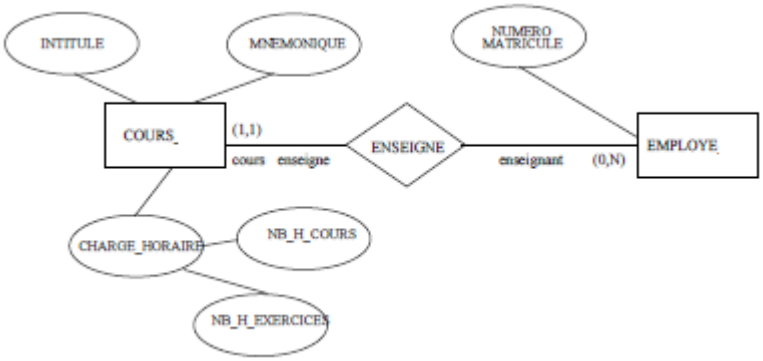
Cas particulier : rôle (1,1)

Soit E_1 et E_2 représenté pat T_1 et T_2 et une association R

Plutôt que de représenter R, on peut ajouter à T_1

- Les attributs de la clé de T_2
- Les attributs de la relation R

Exemple :



COURS(mmémo, intitulé, h_cours, h_ex, h_tp, no_matr_enseignant)

Les **attributs multivalués** d'un ensemble d'entité sont représentés à l'aide d'une nouvelle relation dont les attributs sont

- Les attributs de la clé de la relation correspondant à E
- Un attribut correspondant à l'attribut multivalué (s'il est composite => ses composantes)

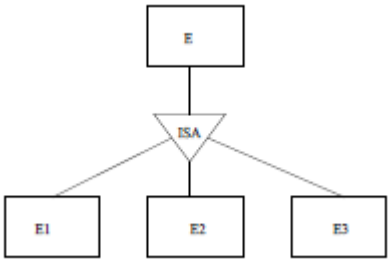
Exemple :

ETUDIANT(no, nom, adresse)

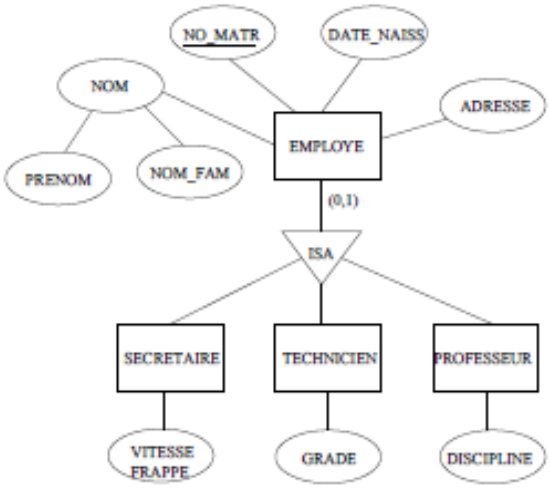
RESULTAT(no_etudiant, cours, note)

Spécialisation/généralisation : Un ensemble d'entités étant une sous classe d'un autre ensemble d'entités est représenté par une relation dont les attributs sont

- Les attributs de la clé de la super-classe
- Les attributs de la sous-classe



Exemple :



EMPLOYE(no_matr, prenom, nom, date_naiss, adresse)

SECRETAIRE(no_matr, vitesse)

TECHNICIEN(no_matr, grade)

PROFESSEUR(no_matr, discipline)

Redondance et normalisation

Soit la relation :

IMPLANTATION	NOM	ADRESSE	CODE POSTAL	LOCALITE	N° Distributeur	Date Vente	Quantite Vendue	CODE PRODUIT	DESIGNATION	Prix Achat	Prix Vente
1	HELP	Quai du Barbou, 2	4000	Liège	2	18-10-17	6	bo0023	coca can	0,20	0,51
1	HELP	Quai du Barbou, 3	4000	Liège	2	18-10-17	32	bo0027	EAU	0,22	0,51
1	HELP	Quai du Barbou, 4	4000	Liège	2	19-10-17	4	bo0023	coca can	0,20	0,51
2	IPEPS Seraing Sup	Rue Colard Trouillet, 48	4100	Seraing	5	18-10-17	5	bo0023	coca can	0,20	0,51
2	IPEPS Seraing Sup	Rue Colard Trouillet, 48	4100	Seraing	5	18-10-17	8	bo0027	EAU	0,22	0,51
2	IPEPS Seraing Sup	Rue Colard Trouillet, 48	4100	Seraing	5	19-10-17	2	bo0023	coca can	0,20	0,51
3	IPES	Avenue Delchambre, 13	4500	Huy	7	18-10-17	7	bo0023	coca can	0,20	0,51
3	IPES	Avenue Delchambre, 14	4500	Huy	7	18-10-17	5	bo0027	EAU	0,22	0,51
3	IPES	Avenue Delchambre, 15	4500	Huy	8	18-10-17	8	bo0023	coca can	0,20	0,51
3	IPES	Avenue Delchambre, 16	4500	Huy	7	19-10-17	12	bo0023	coca can	0,20	0,51
3	IPES	Avenue Delchambre, 17	4500	Huy	8	19-10-17	3	bo0023	coca can	0,20	0,51
4	IPEPS Verviers tech	rue aux laines, 69	4800	Verviers	3	18-10-17	3	bo0023	coca can	0,20	0,51
4	IPEPS Verviers tech	rue aux laines, 70	4801	Verviers	3	18-10-17	5	bo0027	EAU	0,22	0,51
4	IPEPS Verviers tech	rue aux laines, 71	4802	Verviers	3	19-10-17	3	bo0023	coca can	0,20	0,51

- Proposez une clé primaire pour cette relation
- Cette relation contient-elles des redondances ? Si oui, lesquels ? Justifiez brièvement.
- Si la relation contient des redondances, proposez une solution contenant exactement la même information, mais sans redondance.

Imaginons qu'on souhaite représenter des personnes, identifiées par leur numéro de sécurité sociale, caractérisées par leur nom, leur prénom, les véhicules qu'elles ont achetés à une certaine date et à un certain prix. Un véhicule est caractérisé par un type, une marque et une puissance.

On peut aboutir à la représentation relationnelle suivante :

NSS	Nom	Prénom	Marque	Type	Puiss	Date	Prix
16607...	Dupont	Paul	Renault	Clio	5	1/1/96	60000
16607...	Dupont	Paul	Peugeot	504	7	2/7/75	47300
24908...	Martin	Marie	Peugeot	504	7	1/10/89	54900
15405...	Durand	Olivier	Peugeot	504	7	8/8/90	12000
15405...	Durand	Olivier	Renault	Clio	5	7/6/98	65000
15405...	Durand	Olivier	BMW	520	10	4/5/2001	98000

Des redondances sont présentes et conduiront à des problèmes de contrôle de la cohérence de l'information de mise à jour (changement de nom à reporter dans de multiples tuples), de la perte d'information lors de la suppression de données (disparition des informations concernant un type de véhicule) et de difficulté à représenter certaines informations (un type de véhicule sans propriétaire).

Principes de la normalisation

La théorie de la normalisation est une théorie destiné à concevoir un bon schéma d'une BD sans redondance d'information et sans risques d'anomalie de mise à jour.

La théorie de la normalisation est fondée sur deux concepts principaux :

- **Les dépendances fonctionnelles** : Elles traduisent des contraintes sur les données
- **Les formes normales** : Elles définissent des relations bien conçues

Première forme normale (1FN)

> Une relation est en 1NF si elle possède au moins une clé et si tous ses attributs sont atomiques.

Un attribut est atomique s'il ne contient qu'une seule valeur pour un tuple donné, et donc s'il ne regroupe pas un ensemble de plusieurs valeurs.

Exemple :

Soit la relation PERSONNE : *PERSONNE(id, nom, prenom, date_naissance, diplôme)*

Id	Nom	Prenom	Date_naissance	diplôme
1	Dupont	Martine	14/7/1980	Master en informatique – agrégation AESS
2	Durant	Simon	28/2/1970	Master en économie

Une personne peut avoir plusieurs diplômes => l'attribut diplôme n'est pas atomique

Pour que la relation soit en 1FN, on pourrait ajouter l'attribut diplôme à la clé et avoir ainsi deux tuples pour Martine Dupont.

PERSONNE(id, diplôme ,nom, prenom, date_naissance)

Id	Nom	Prenom	Date_naissance	diplôme
1	Dupont	Martine	14/7/1980	Master en informatique
1	Dupont	Martine	14/7/1980	agrégation AESS
2	Durant	Simon	28/2/1970	Master en économie

Deuxième forme normale (2FN)

La deuxième forme normale permet d'éliminer les dépendances entre des parties de clé et des attributs n'appartenant pas à cette clé.

> Une relation est en 2FN si elle est en 1FN et si tout attribut qui n'est pas dans une clé ne dépend pas d'une partie seulement d'une clé.
> (attention : vérifier pour la clé primaire et pour toutes les clés candidates)

Exemple :

Soit la relation EMPLOYE : *EMPLOYE(nom, profession, salaire)*

Soit les dépendances fonctionnelles profession -> salaire

À partir de la profession, on peut déterminer le salaire, donc la clé n'est pas élémentaire. Donc, cette relation n'est pas en 2FN

Pour avoir un schéma relationnel en 2FN, il faut alors décomposer la relation EMPLOYE en deux relations :

- *EMPLOYE(nom, profession)*
- *PROFESSION(profession, salaire)*

Reprenant la relation précédente PERSONNE(id, diplôme ,nom, prenom, date_naissance)

Cette relation n'est pas en 2FN car id détermine le nom (*id -> nom*), id détermine le prénom (*id -> prenom*), id détermine la date de naissance (*id -> date_naissance*).

Ces relations le sont :

- *PERSONNE(id, nom, prenom, date_naissance)*
- *DIPLÔME(id, diplôme)*

Troisième forme nomale (3FN)

La troisième forme normale permet d'éliminer les dépendances entre les attributs n'appartenant pas à une clé

> Une relation est en 3FN si elle est en 2FN et si tout attribut n'appartenant pas à une clé ne dépend pas d'un autre attribut n'appartenant pas à une clé. (concerne la clé primaire et toutes les clés candidates)

Exemple :

Soit la relation Profession

PROFESSION(profession, salaire, prime) et les dépendances fonctionnelles sur cette relation :

- profession-> salaire
- salaire -> prime

Cette relation n'est pas en 3FN car salaire, qui n'est pas la clé, détermine prime.

Pour avoir un schéma relationnel en 3FN, il faut décomposer Profession

- PROFESSION(profession, salaire)
- SALAIRE(salaire, prime)

La forme normale de Boyce-Codd (BNCF)

La forme normale de Boyce-Codd permet d'éliminer les dépendances entre les attributs n'appartenant pas à une clé vers les parties de clé.

> Une relation est en BCFN si elle est en 3FN et si tout attribut qui n'appartient pas à la clé n'est pas source d'une dépendance fonctionnelle vers une partie d'une clé.

Exemple :

Soit la relation PERSONNE : PERSONNE(numSS, pays, nom, region) et les dépendances fonctionnelles :

- numSS Pays -> nom
- numSS pays -> region
- region -> pays

Il existe une dépendance fonctionnelle qui n'est pas issue d'une clé et qui détermine un attribut appartenant à une clé. Cette relation est en 3FN, mais pas en BCNF car en BCNF, toutes les dépendances fonctionnelles sont issues de la clé.

Pour avoir un schéma en BCNF, il faut décomposer cette relation :

- PERSONNE(numSS, nom, region)
- REGION(region, pays)

Attention : une décomposition en BCFN ne préserve pas toujours toutes les dépendances fonctionnelles.

Conclusion

La normalisation permet de décomposer un schéma relationnelle afin d'obtenir des relations non redondantes.

La 3FN est souhaitable car il est toujours possible de l'obtenir sans perte d'informations et sans perte de dépendances fonctionnelles. La BCNF est également indiquée car elle est un peu plus puissante et plutôt plus simple que la 3FN.

La BCNF n'est pas encore suffisante pour éliminer toutes les redondances. Il existe pour cela les 4FN et 5FN qui ne sont pas abordées dans ce cours. Notons également que les cas de non-4FN et de non-5FN sont assez rares dans la réalité.

Exercices

Exercice 1

Convertir les modèles entité-association des 5 exercices du chapitre précédent en les modèles relationnels équivalents.

Chapitre 4 – PostgreSQL

PostgreSQL est un système de gestion de base de données relationnelle et objet (***SGBDRO***).

PostreSQL est un logiciel libre, disponible selon les termes d'une licence de type BSD (*Berkeley Software Distribution License)* Comme les projets libres Apache, Linux, PostgreSQL n'est pas contrôlé par une seule entreprise, mais est fondé sur une communauté mondiale de développeurs et d'entreprises.

Histoire :

- Ingres (**In**telligent **G**raphic **RE**lational **S**ystem) développée à Berkeley par Michaël Stonebrake
- 1985 : Michaël Stonebrake recommence le développement de Ingres et le nomme PostgreSQL(Post-ingres)
- 1995 : ajout des fonctionnalités SQL => Postgres95
- 1996 : changement de nom => PostgreSQL

Caractéristiques

- PostgreSQL utilise des type de données simples (entiers, caractères, …) et de types de données composés (créé par l'utilisateur). L'utilisateur peut créer des types, utiliser l'héritage de type.
- PostgreSQL est pratiquement conforme aux normes ANSI SQL 89, SQL92, SQL99, SQL2003 et SQL2008.
- Il fonctionne sur LINUX et toute sorte d'UNIX. Depuis, la version Windows 8, il fonctionne nativement sur Windows.

Interfaces utilisateurs

- ***psql*** : permet la saisie de requêtes SL, directement ou par l'utilisation de procédures stockées.
- ***pgadmin*** : outil d'administration graphique pour PostgreSQL distribué selon les termes de la licence PostgreSQL
- ***phpPgAdmin*** : interface web d'administration pour PostgreSQL, écrit en PHP

pilote pour c#

Npgsql :

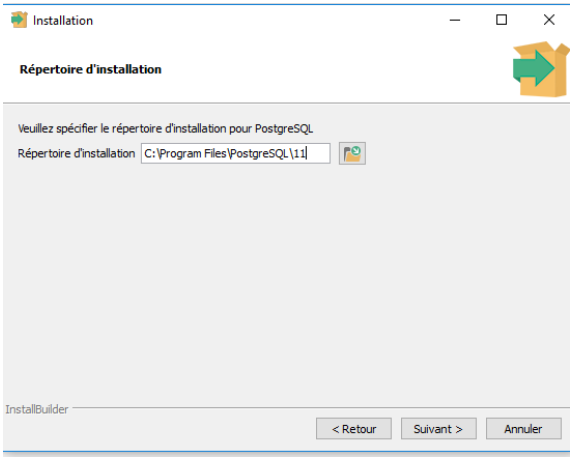
- dataprovider ADO.net,
- open source,
- écrit en C#,
- permet d'accèder au serveur de base de données PostgreSQL

Installation de PostgreSQL

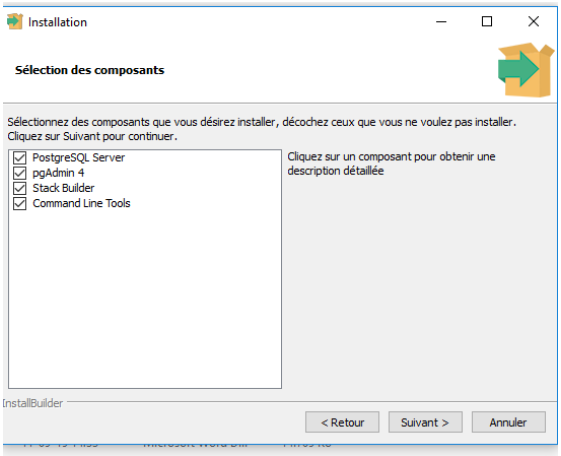
https://www.postgresql.org/

Une fois l'installateur *Interactive installer by EnterpriseDB* téléchargé, lancer-le.

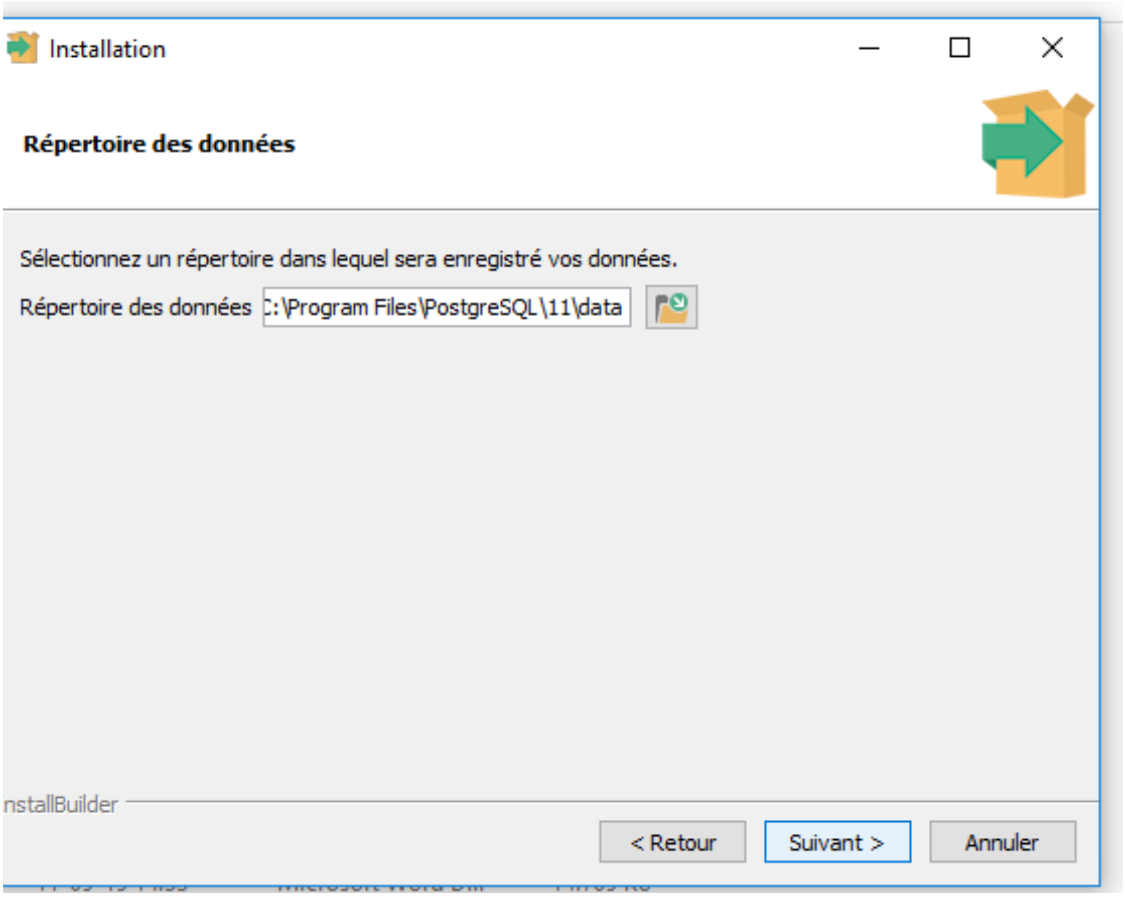
Cliquer sur *suivant*



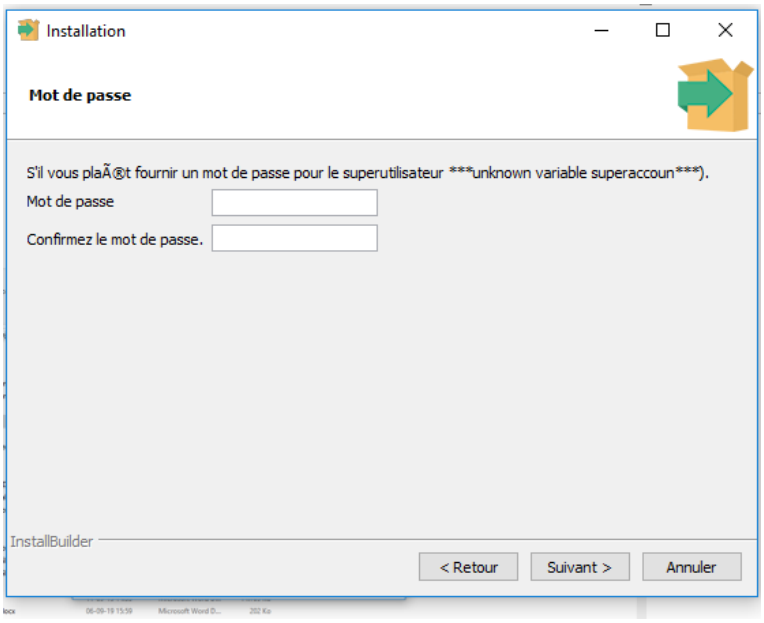
Définir le répertoire d'installation



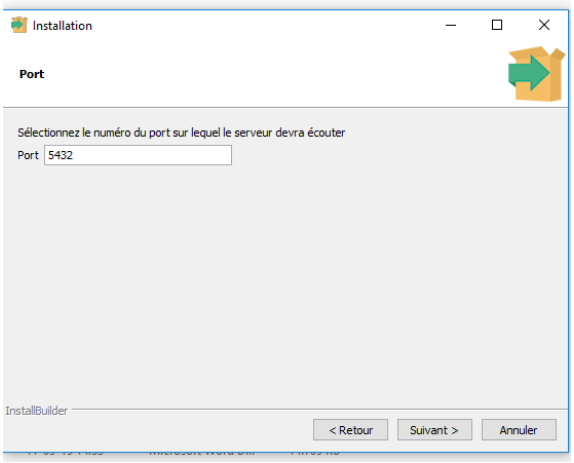
Cliquer sur *suivant*



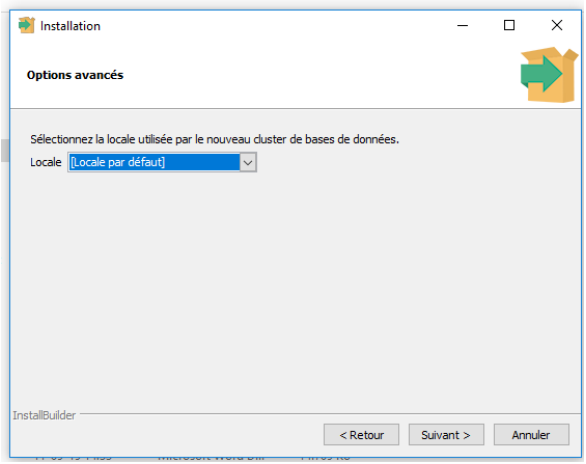
Définir le répertoire de données



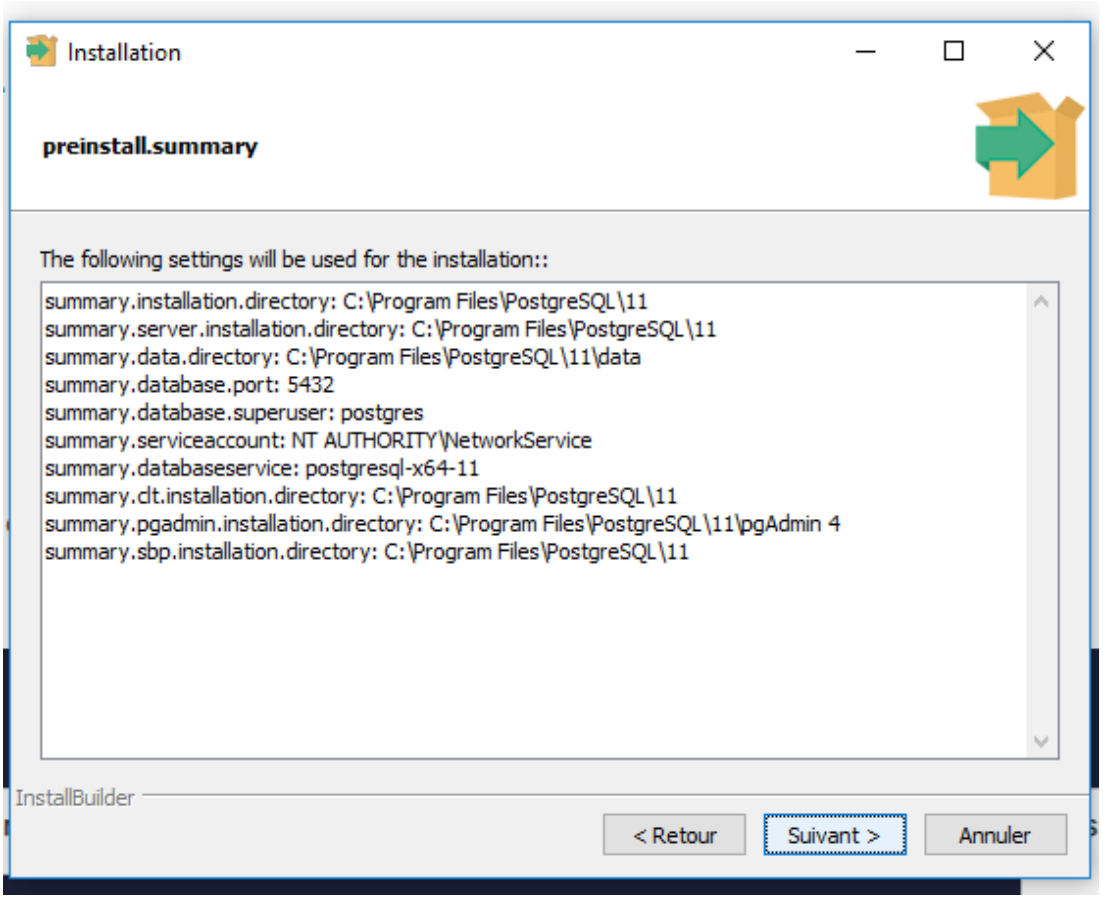
Encoder le mot de passe pour le superuser *postres*



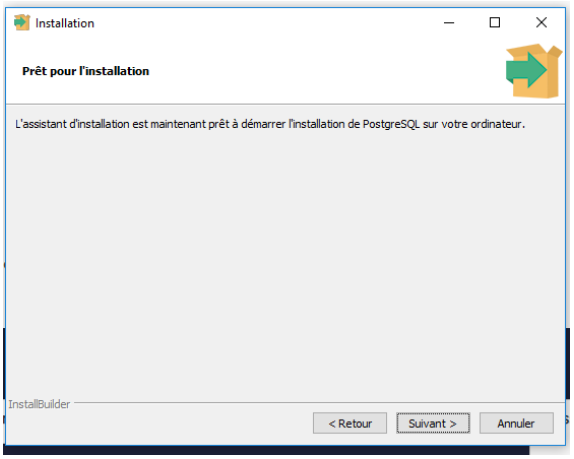
Laisser le port par défaut

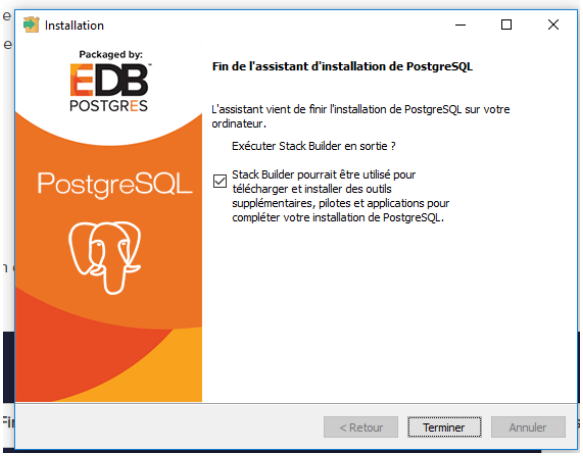


Laisser la locale par défaut

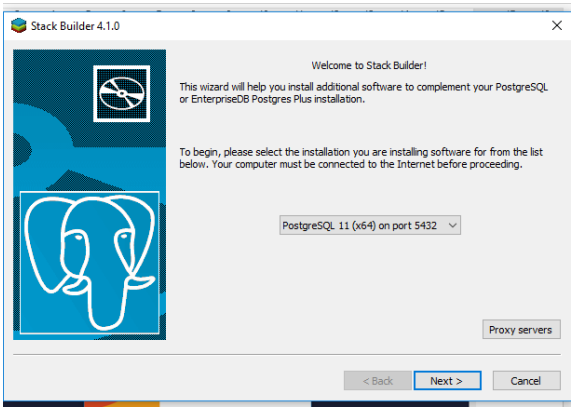


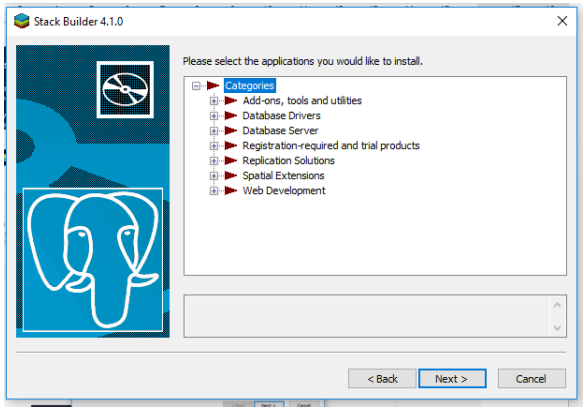
Cliquer sur *Suivant* pour démarrer l'installation

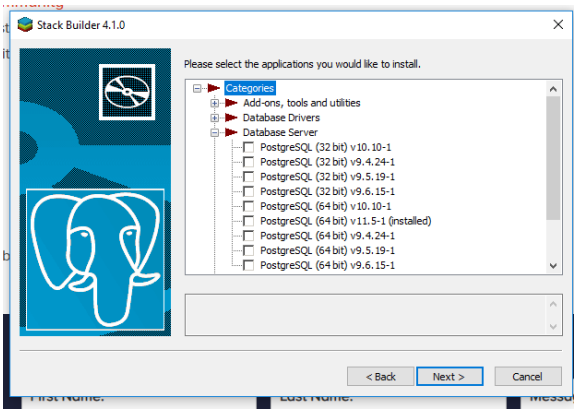


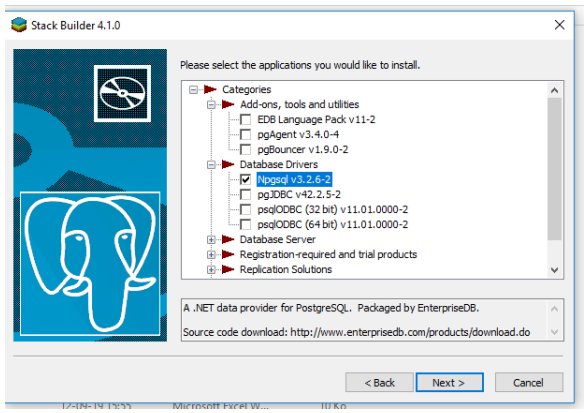


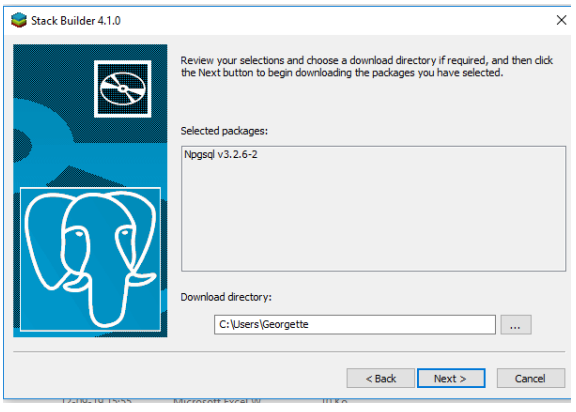
stack Builder permet d'installer des outils supplémentaires

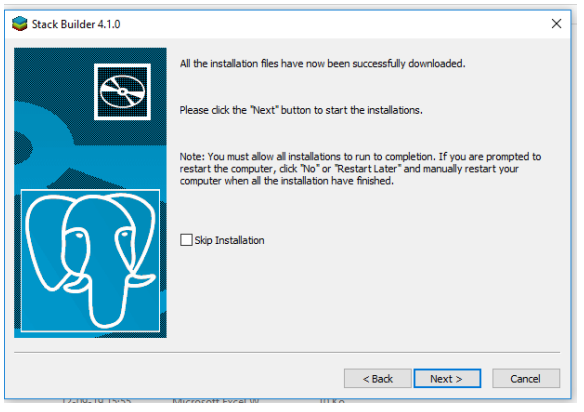


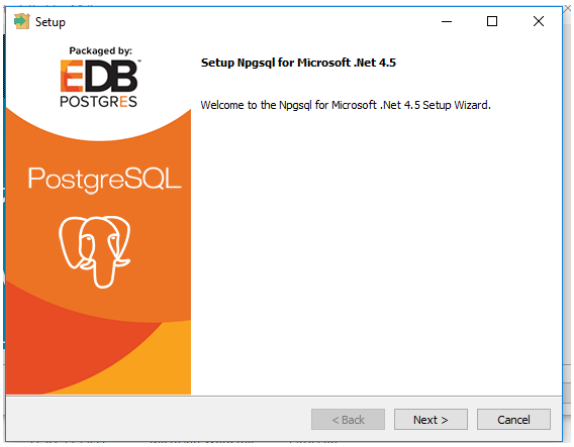


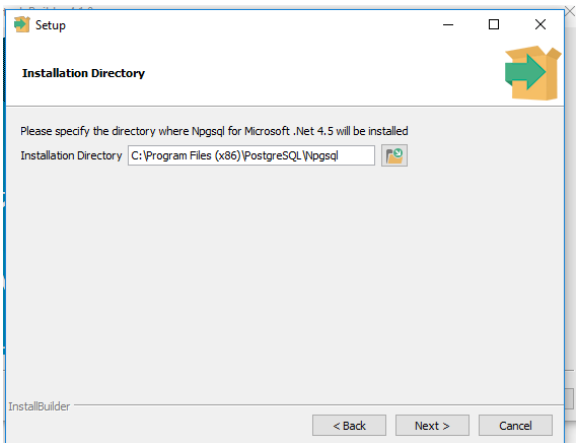




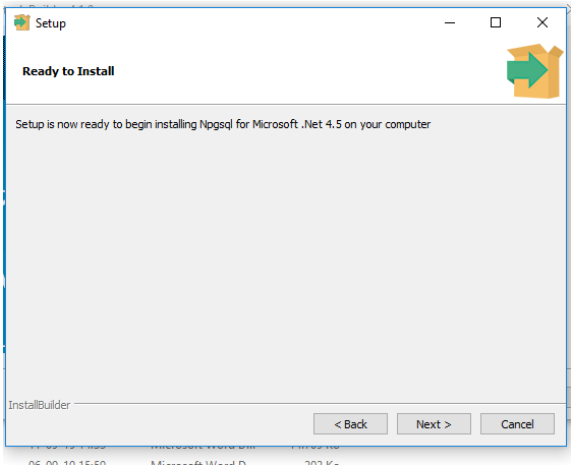


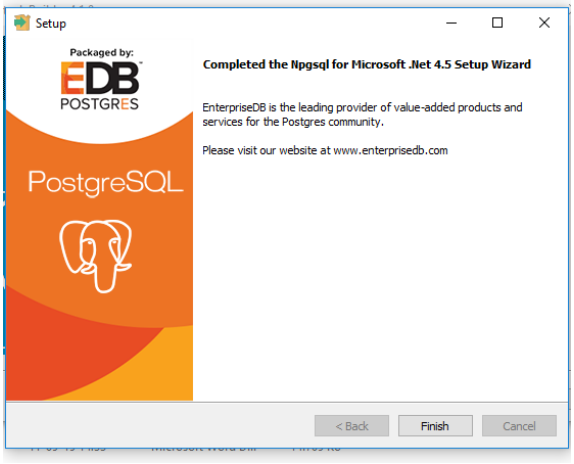


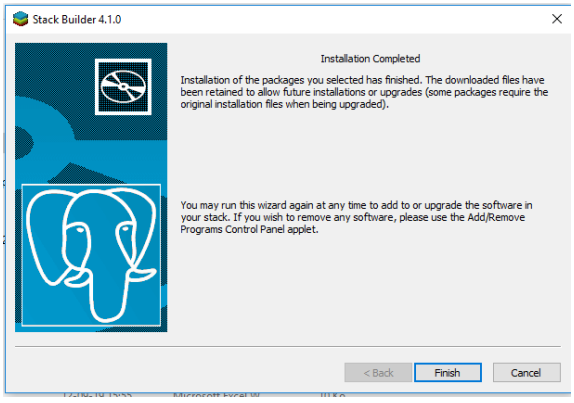




installation du pilote ngpSQL

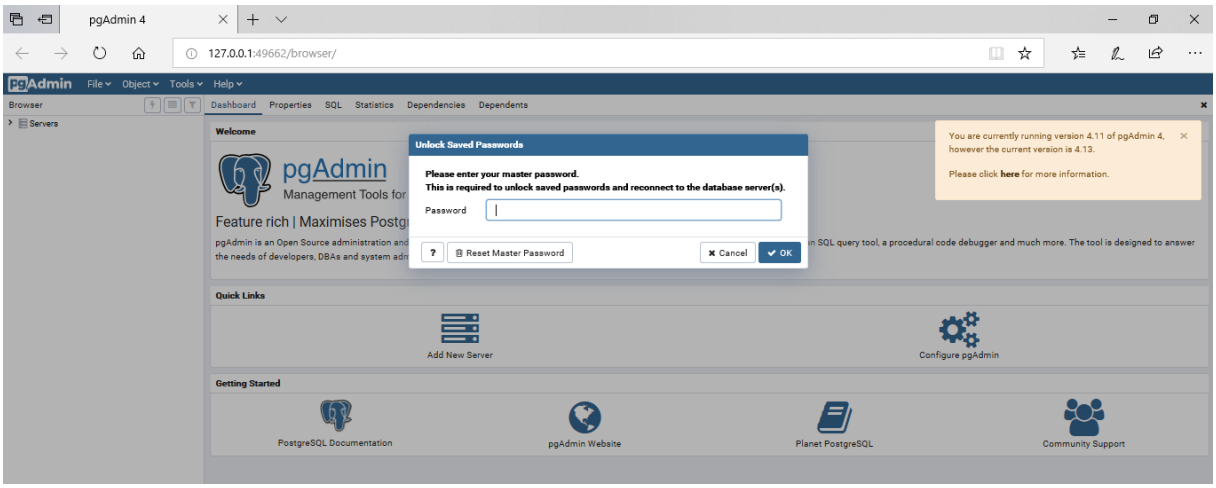




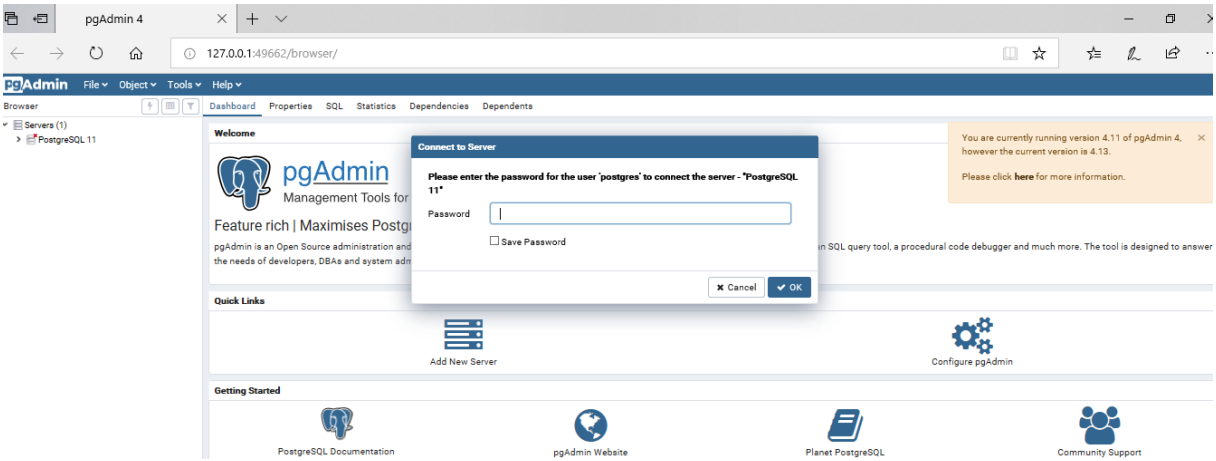


Création d'une base de donnée

- Lancer l'application *pgadmin*

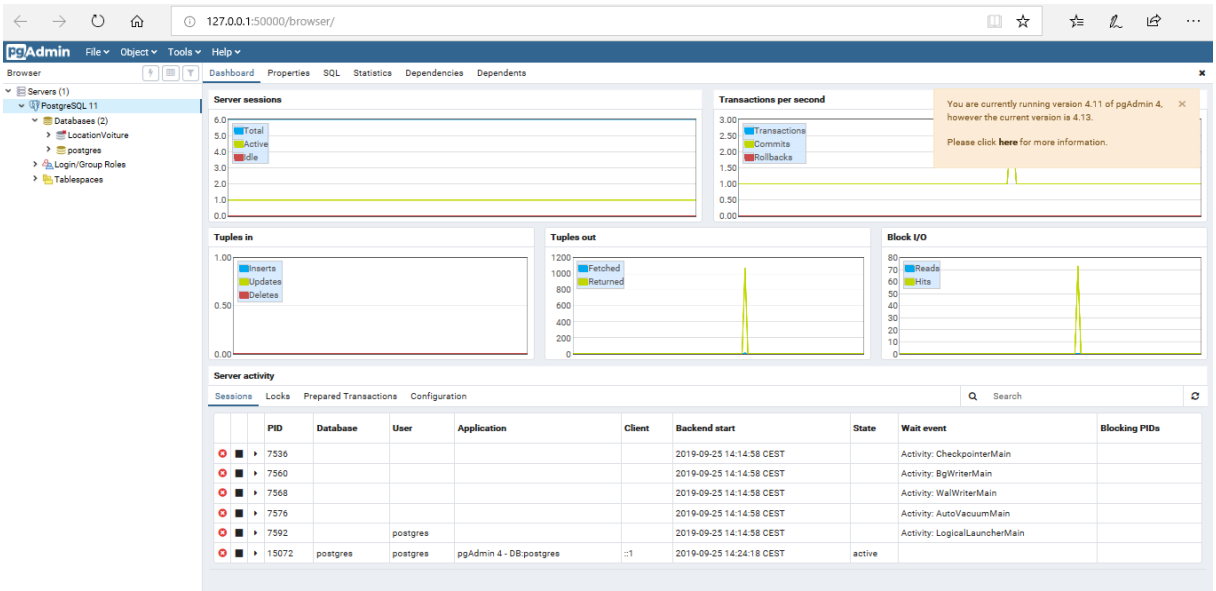


- Entrer le mot de passe du superuser

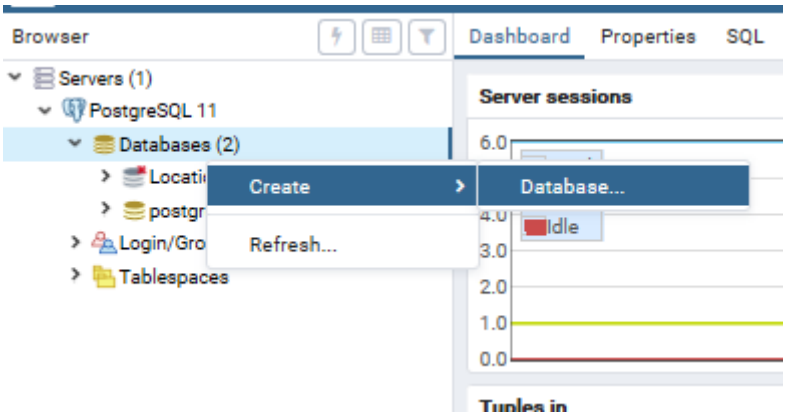


- Cliquer sur server et entrer le mot de passe du superuser *postgres*

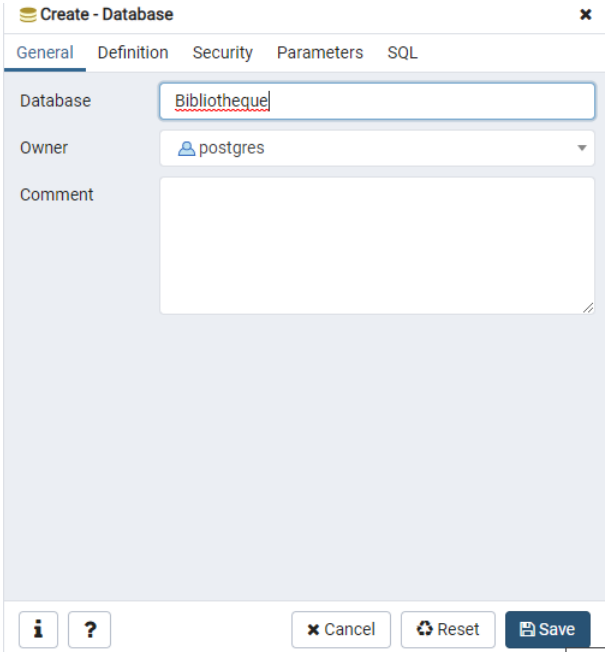
La fenêtre suivante apparaît



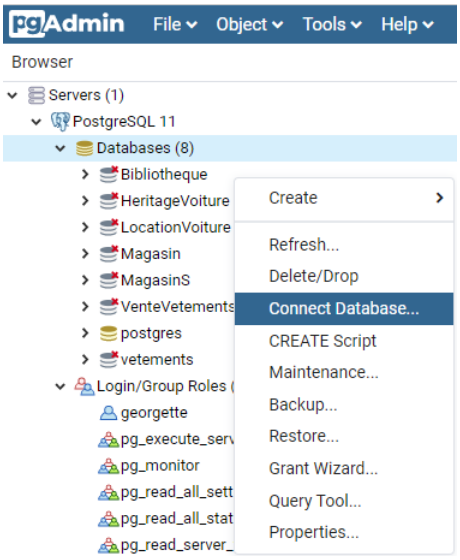
➢ Pour créer une nouvelle base de données, cliquer sur *Databases – create – Database* …



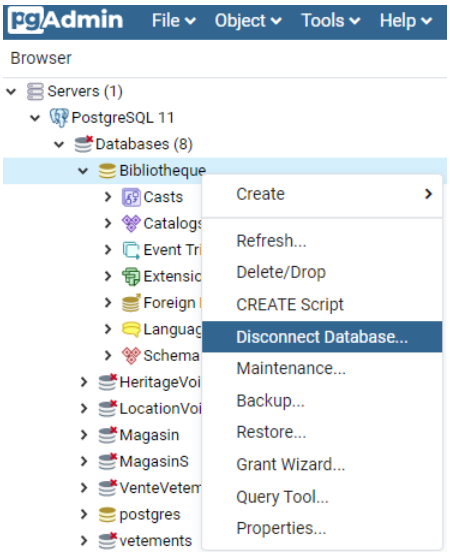
1. Créer la base de données Bibliotheque

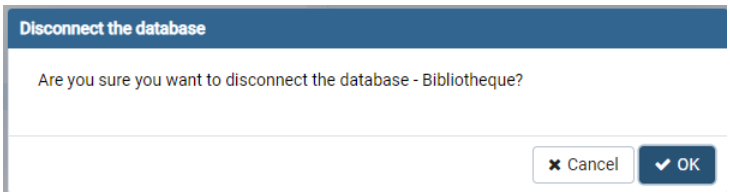


2. Se connecter à la base de données *Bibliotheque*

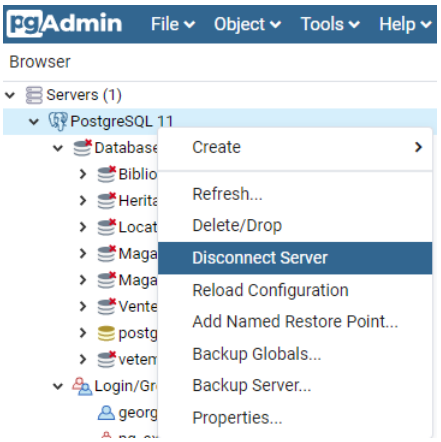


3. Se déconnecter de la base de données *Bibliotheque*





4. Se déconnecter du serveur



5. quitter pgadmin

Se connecter à la base de données *Bibliotheque* à partir du terminal de commande

1. Ouvrir un terminal de commande et taper *cd " c:\Program Files\PostgreSQL\11\bin"*
2. Se connecter au serveur de base de donnée en tapant : *psql –U postgres –d Bilbliotheque*
3. Entrer le mot de passe

```
C:\Program Files\PostgreSQL\11\bin>psql -U postgres -d Bibliotheque
Mot de passe pour l'utilisateur postgres :
psql (11.5)
Attention : l'encodage console (850) diffère de l'encodage Windows (1252).
            Les caractères 8 bits peuvent ne pas fonctionner correctement.
            Voir la section « Notes aux utilisateurs de Windows » de la page
            référence de psql pour les détails.
Saisissez « help » pour l'aide.
```

- Pour quitter taper *\q*
- Pour ne plus voir apparaître ce message, taper la commande *chcp 1252*

```
Bibliotheque=# \q

C:\Program Files\PostgreSQL\11\bin>chcp 1252
Page de codes active : 1252

C:\Program Files\PostgreSQL\11\bin>psql -U postgres -d Bibliotheque
Mot de passe pour l'utilisateur postgres :
psql (11.5)
Saisissez « help » pour l'aide.

Bibliotheque=#
```

Type de données

PostgreSQL a un large type de données disponibles. Les utilisateurs peuvent ajouter de nouveaux types à PostgreSQL en utilisant la commande **CREATE TYPE**.

Type numérique

Nom	Taille de stockage	Description	Étendue
smallint	2 octets	entier de faible étendue	de -32768 à +32767
integer	4 octets	entiers les plus courants	de -2147483648 à +2147483647
bigint	8 octets	grands entiers	de -9223372036854775808 à 9223372036854775807
decimal	variable	précision indiquée par l'utilisateur, valeurs exactes	pas de limite
numeric	variable	précision indiquée par l'utilisateur, valeurs exactes	pas de limite
real	4 octets	précision variable, valeurs inexactes	précision de 6 décimales
double precision	8 octets	précision variable, valeurs inexactes	précision de 15 décimales
serial	4 octets	entier à incrémentation automatique	de 1 à 2147483647
bigserial	8 octets	entier de grande taille à incrémentation automatique	de 1 à 9223372036854775807

tableau numérique - https://docs.postgresql.fr/8.1/datatype.html

Type de caractères

Nom	Description
character varying(*n*), varchar(*n*)	Longueur variable avec limite
character(*n*), char(*n*)	longueur fixe, comblé avec des espaces
text	longueur variable illimitée

tableau de caractères - https://docs.postgresql.fr/8.1/datatype-character.html

Type Date – Heure

Nom	Taille de stockage	Description	Valeur minimale	Valeur maximale	Résolution
timestamp [(*p*)] [without time zone]	8 octets	date et heure	4713 avant JC	5874897 après JC	1 microseconde / 14 chiffres
timestamp [(*p*)] with time zone	8 octets	date et heure, avec fuseau horaire	4713 avant JC	5874897 après JC	1 microseconde / 14 chiffres
interval [(*p*)]	12 octets	intervalle de temps	-178000000 années	178000000 années	1 microseconde / 14 chiffres
date	4 octets	date seulement	4713 avant JC	5874897 après JC	1 jour
time [(*p*)] [without time zone]	8 octets	heure seulement	00:00:00.00	24:00:00	1 microseconde / 14 chiffres
time [(*p*)] with time zone	12 octets	heure seulement, avec fuseau horaire	00:00:00+1359	24:00:00-1359	1 microseconde / 14 chiffres

tableau date-heure - https://docs.postgresql.fr/8.1/datatype-datetime.html

Type boolean

Ce type utilise un octet de stockage.

Les valeurs booléennes sont affichées en utilisant les lettres ***t*** *(true)* et ***f*** *(false)*

Valeur littéral valide pour true	Valeur littéral valide pour false
TRUE	FALSE
'true'	'false'
't'	'f'
'1'	'0'
'y'	'n'
'yes'	'no'

Il est recommandé d'utiliser **TRUE** et **FALSE** qui sont compatible avec la norme SQL

Type géométrique

Nom	Taille de stockage	Représentation	Description
point	16 octets	Point du plan	(x,y)
line	32 octets	Ligne infinie (pas entièrement implémenté)	((x1,y1),(x2,y2))
lseg	32 octets	Segment de droite fini	((x1,y1),(x2,y2))
box	32 octets	Boite rectangulaire	((x1,y1),(x2,y2))
path	16+16n octets	Chemin fermé (similaire à un polygone)	((x1,y1),...)
path	16+16n octets	Chemin ouvert	[(x1,y1),...]
polygon	40+16n octets	Polygone (similaire à un chemin fermé)	((x1,y1),...)
circle	24 octets	Cercle	<(x,y),r> (centre et rayon)

type géométrique - https://docs.postgresql.fr/8.1/datatype-geometric.html

ProgreSQL a un ensemble de fonctions et d'opérateurs permettant d'effectuer différentes opérations géométriques comme l'agrandissement, la translation, la rotation, la détermination des intersections

Type d'adresse réseau

Nom	Taille de stockage	Description
cidr	12 ou 24 octets	réseaux IPv4 et IPv6
inet	12 ou 24 octets	hôtes et réseaux IPv4 et IPv6
macaddr	6 bytes	adresses MAC

type d'adresse réseau - https://docs.postgresql.fr/8.1/datatype-net-types.html

Type tableau

Voir https://docs.postgresql.fr/8.1/arrays.html

Ceci pour des variables multi-valeurs.

Attention : première forme normale n'est plus respectée, redondance des informations !!!

Type composite

Soit une adresse composée d'une rue, d'une localité et d'un numéro de code postal

```
create type adresse as (
    rue varchar(35),
    cp char(4),
    localite varchar(20));
```

```
create type couleur as enum('rouge', 'bleu', 'noir', 'blanc','gris');
```

```
create table PERSONNE(
    idClient SERIAL  primary key ,
    nom varchar(15),
    prenom varchar(15),
    gsm varchar(12),
    adressePersonne adresse);
```

Insérer dans une table

```
insert into PERSONNE(nom, prenom, gsm, adressePersonne)
    values('Durant', 'Alice', '012-34-45-56', ('rue du Commerce, 5', '4100', 'Seraing'));
```

Accéder à une variable composite

```
select nom, prenom, (adressePersonne).rue, (adressePersonne).localite
from PERSONNE
where (adressePersonne).cp like '4%';
```

ou en utilisant le nom de la table

```
select PERSONNE.nom, PERSONNE.prenom, (PERSONNE.adressePersonne).rue, (PERSONNE.adressePersonne).localite
from PERSONNE
where (PERSONNE.adressePersonne).cp like '4%';
```

```
update PERSONNE
set adressePersonne.rue = 'Avenue du Chien, 39'
where nom='Durant' and prenom='Alice';
```

Il ne faut pas placer des parenthèses autour des noms de colonnes apparaissant après SET, mais il faut les mettre lors de la référence à la même colonne à droite du signe d'égalité.

Comptabilité

Les types suivants sont conformes à la norme SQL : bit, bit varying, boolean, char, character varying, character, varchar, date, double précision, integer, interval, decimal, real, smallint, time, timestamp.

Créer une table (CREATE TABLE)

(pas les mettes tous comme exemples)

```
create table PERSONNE(
    idPersonne SERIAL  primary key ,
    nom varchar(15),
    prenom varchar(15),
    date_naissance date,
    gsm varchar(12),
    adressePersonne adresse);

create table LIVRE(
    isbn varchar(30) primary key,
    idCategorie integer,
    titre varchar(100),
    auteur varchar(50),
    edition varchar(30),
    foreign key (idCategorie) references CATEGORIE(idCategorie));
```

clé primaire qui s'auto-incrémente

type utilisateur

clé primaire

clé étrangère

Modifier une table (ALTER TABLE)

Ajouter une colonne

```
alter table personne add column idPersonne serial primary key;
```

Supprimer une colonne

```
alter table personne drop column idClient;
```

Ajouter une contrainte

```
alter table personne alter column GSM set not null;

alter table personne add check (char_length(nom) > 3);

alter table EMPRUNT add foreign key (isbn) references livre(isbn);

alter table EMPRUNT add foreign key (idPersonne) references personne(idPersonne);

alter table reservation add foreign key (isbn) references livre(isbn),
                        add foreign key (idPersonne) references personne(idPersonne);
```

Supprimer une contrainte

```
alter table personne drop constraint personne_nom_check;
```

Les contraintes non NULL n'ont pas de nom

```
alter table personne alter column GSM drop not null;
```

Pour voir la liste des contraintes :

```
select * from information_schema.Table_constraints;
```

Pour voir le nom des contraintes de la table *personne*

```
select tc.constraint_name
from information_schema.table_constraints as tc
where tc.table_name='personne';
```

Modifier des valeurs par défaut

```
alter table emprunt alter column date_sortie set default current_date;
```

Supprimer une valeur par défaut, revient à modifier la valeur par défaut à NULL

```
alter table emprunt alter column date_sortie drop default ;
```

Cela n'affecte pas les lignes existantes de la table, mais uniquement la valeur par défaut pour les futures commandes **INSERT**.

Modifier les types de valeur d'une colonne

```
ALTER TABLE Categorie ALTER COLUMN prixLocation TYPE numeric(10,2);
```

Elle ne peut réussir que si chaque valeur de la colonne peut être convertie dans le nouveau type par une conversion implicite. Si une conversion plus complexe est nécessaire, une clause **USING** peut être ajoutée qui indique comment calculer les nouvelles valeurs à partir des anciennes.

Renommer des colonnes

```
ALTER TABLE CLIENT RENAME COLUMN id TO idClient;
```

Renommer une table

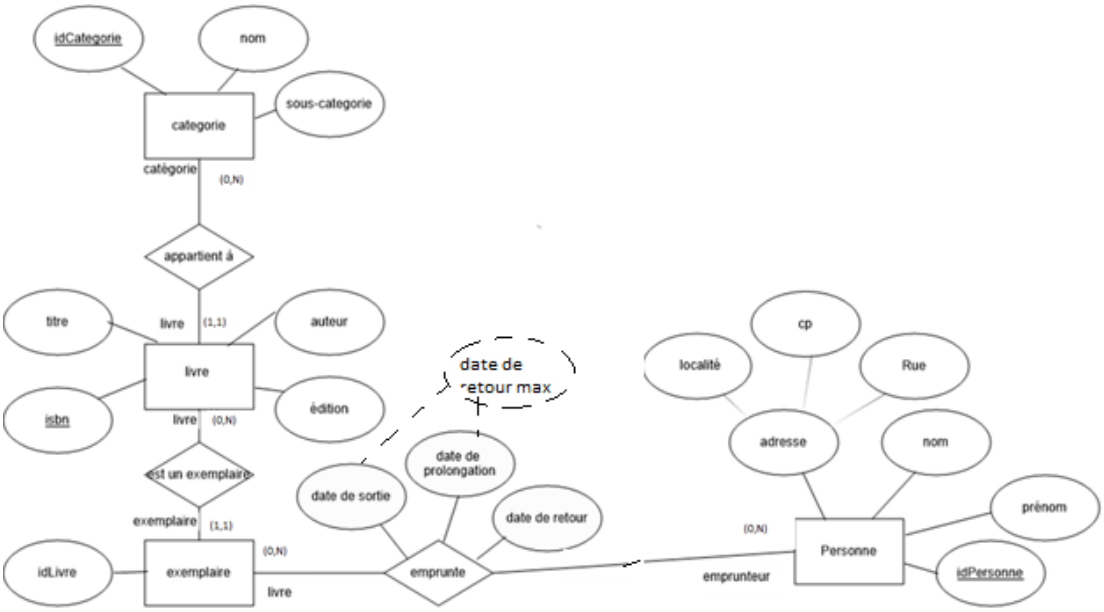
```
alter table CLIENT rename to locataire;
```

Supprimer une table

```
drop table CLIENT;
```

Créer les tables de la base de données Bibliothèques

Voici le modèle entité-association



Le modèle relationnel est :

PERSONNE(idPersonne, nom, prenom, GSM, (rue, cp, localite))

CATEGORIE(idCategorie, nom, sous_categorie)

LIVRE(isbn, titre, auteur, edition,#idCategorie)

EXEMPLAIRE(idLivre, #isbn)

EMPRUNT(#idPersonne, #iidLivre, date_sortie, date_retour, date_prolongation)

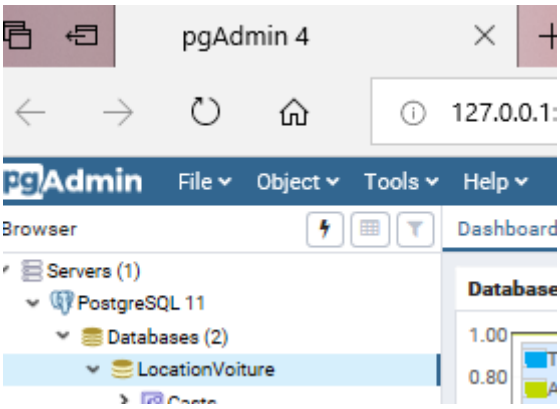
Les contraintes sont

- Nom et prénom doivent au moins contenir 3 caractères
- Le champ nom de la table catégorie ne peut être NULL
- La date de prolongation doit être comprise entre la date de sortie et la date de sortie + 28 jours
- Si la date de prolongation est non null alors la date de retour doit être supérieure à la date de prolongation sinon la date de retour doit être supérieure à la date de sortie
- La date de sortie a comme valeur par défaut la date courante.

Insérer des enregistrements dans les tables

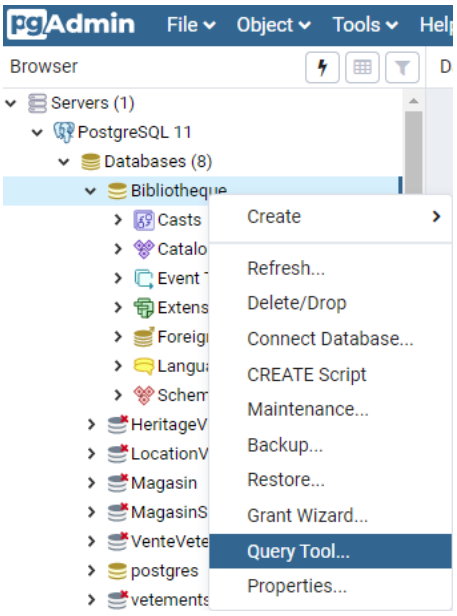
Via pgadmin

- Lancer pgadmin et se connecter

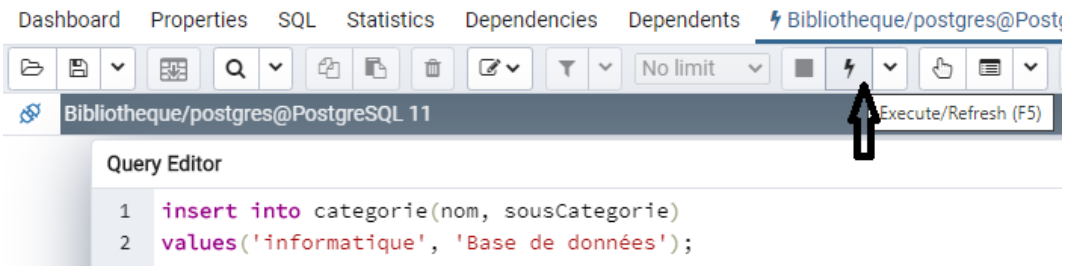


- Se connecter au serveur PostgreSQL 11 (entrer de nouveau le mot de passe)
- Se connecter à la base de données en cliquant sur la base de données

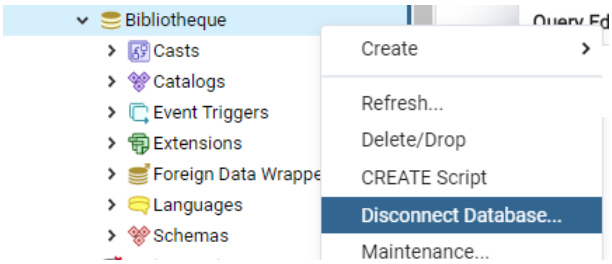
Via le terminal de commande de pgAdmin



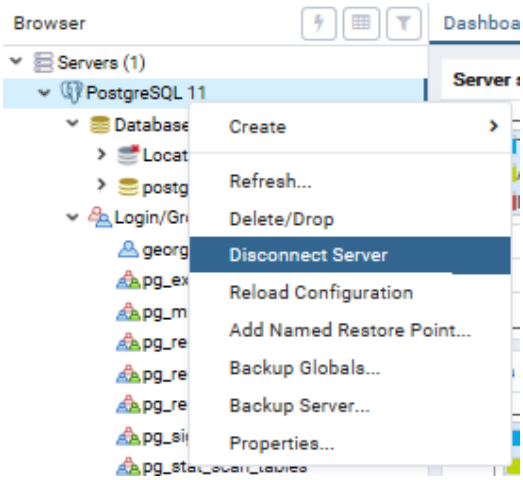
- Cliquer droit sur la base de données – cliquer sur *Query Tool*



- Se déconnecter de la base de données *Bibliotheque*



- Se déconnecter du serveur



- Quitter pgadmin

Insertion d'enregistrements dans une table. (INSERT)

```
insert into categorie(nom, sousCategorie)
values('informatique', 'Programmation');

insert into categorie(nom)
values('Geographie');

insert into categorie(nom, sousCategorie)
values('informatique', 'Reseau'),
      ('roman', 'thriller');
```

Modifier des enregistrements (UPDATE)

```
update personne
    set adressePersonne.rue = (adressePersonne).rue || 'A'
    where nom = 'Durant' and prenom='Alice';


update personne
    set prenom = 'Aline'
    where nom = 'Durant' and prenom='Alice';
```

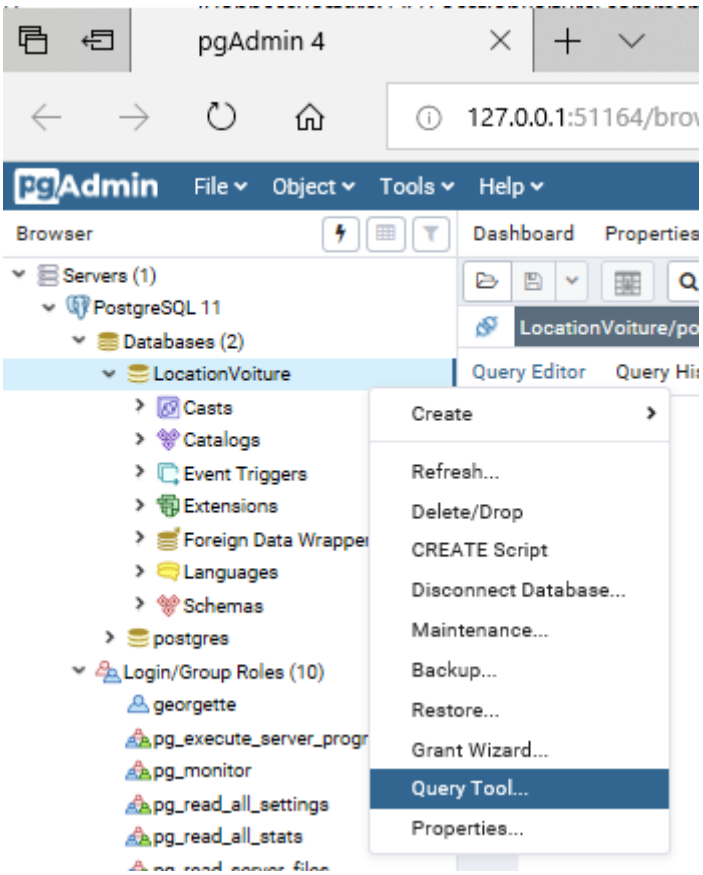
```
update emprunt
set date_sortie = current_date
where idPersonne = 6 and idLivre = 1 and date_sortie='02/09/2020';
```

Supprimer un(des) enregistrement(s) (DELETE)

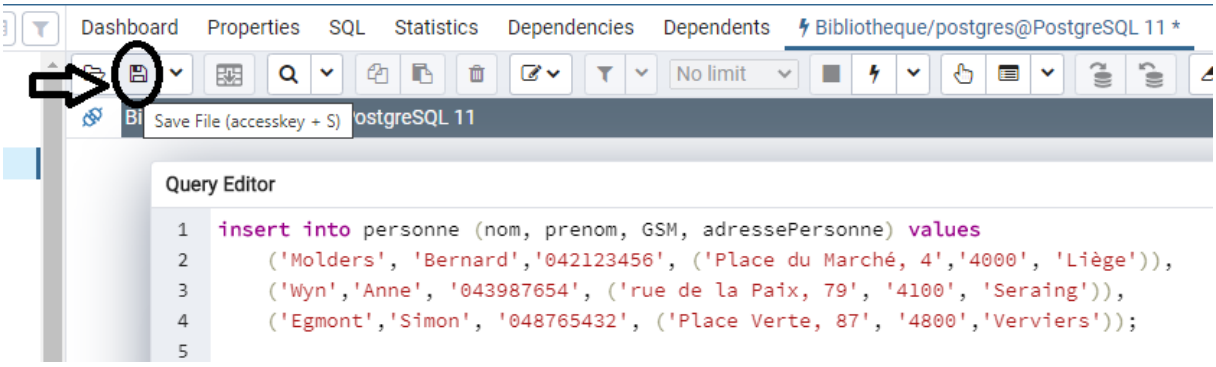
```
delete from livre
where auteur='inconnu';
```

Application : Bibliotheque

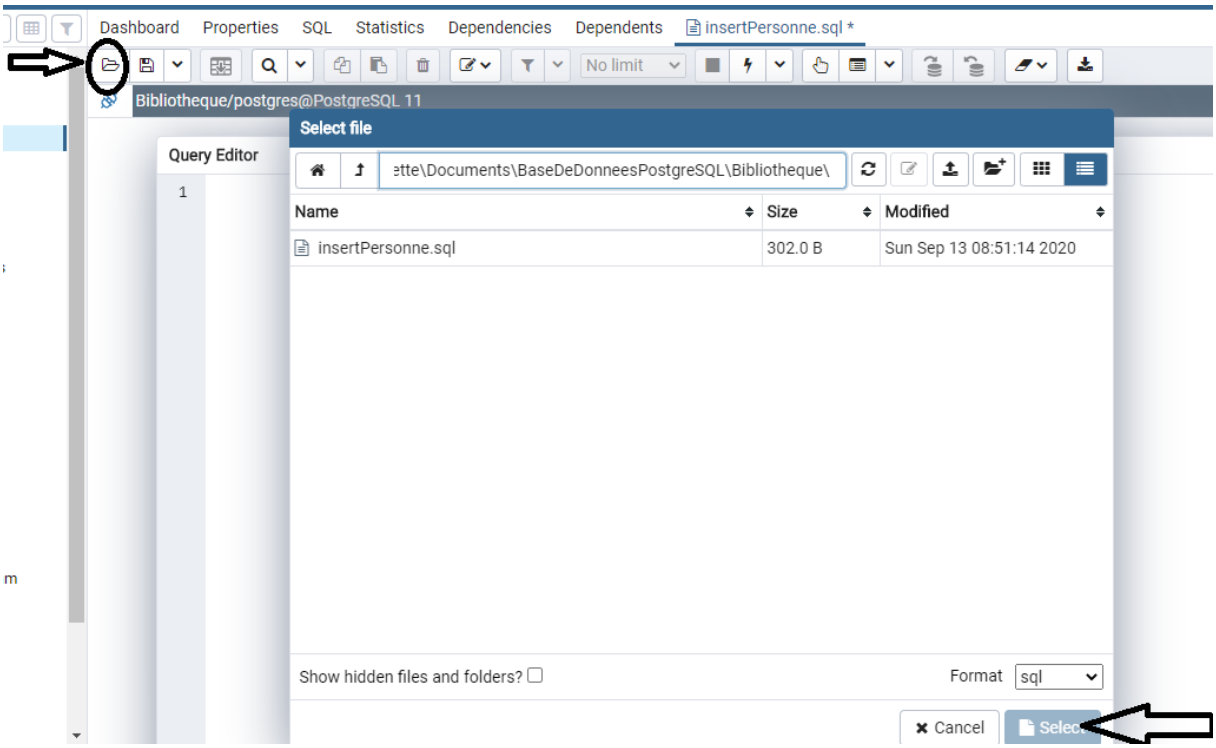
- Lancer pgadmin et se connecter
- Ouvrir une fênetre de commande (Query tool)



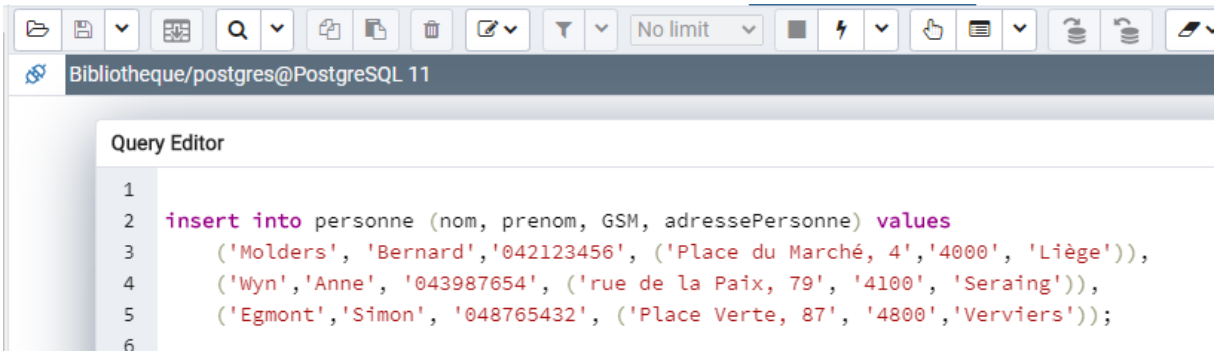
- Encoder les insertions suivantes et le sauver dans un fichier



- Quitter l'éditeur.
- Ouvrir un query tools



Ouvrir le fichier *insertPersonne.sql*



Exécuter le fichier.

Consulter les données (SELECT)

Tester les commandes suivantes :

- Obtenir toutes les lignes de la table PERSONNE

```
select * from PERSONNE;
```

- Lister les nom et prénom des personnes habitant Seraing

```
select nom, prenom
from PERSONNE
where (adressePersonne).localite='Seraing';
```

- Afficher les différentes localités des personnes

```
select distinct (adressePersonne).localite
from PERSONNE;
```

- Afficher les nom et prénom des personnes habitant Seraing et dont le nom commence par la lettre C

```
select nom, prenom
from PERSONNE
where (((adressePersonne).localite = 'Seraing') and (nom like 'C%'));
```

- Afficher le nom et prénom des personnes n'habitant pas Seraing et dont le nom commence par C, et obtenir également les personnes habitant Verviers.

```
select nom, prenom
from PERSONNE
where (((adressePersonne).localite != 'Seraing') and (nom like 'C%'))
    or ((adressePersonne).localite = 'Verviers');
```

- Afficher les personnes pour lesquels on ne connait pas l'adresse

```
select idPersonne, nom, prenom
from PERSONNE
where adressePersonne is null;
```

- Classer par ordre croissant les noms des personnes

```
select nom, prenom from PERSONNE
    order by nom asc;
```

- Classer par ordre décroissant les noms et prénoms des personnes

```
select nom, prenom from PERSONNE
    order by nom desc, prenom desc;
```

- Donner le nombre de livres empruntés

```
select count(idlivre) as nombreLivresEmprunte
from EMPRUNT
where date_retour is null;
```

- Donner le nombre de livres

```
select count(*) as nombreLivres
from LIVRE;
```

- Donner les nombres de livres par auteur

```
select auteur, count(*)
from LIVRE
group by auteur;
```

- Donner le nombre de livres empruntés par personne

```
select PERSONNE.idPersonne, nom, prenom, count(*) as nombreLivresEmprunte
from EMPRUNT inner join PERSONNE on EMPRUNT.idPersonne = PERSONNE.idPersonne
where date_sortie = null
group by PERSONNE.idPersonne, nom, prenom;
```

- Donner le nombre de livres empruntés pour les personnes ayant emprunté plus de 3 livres

```
select PERSONNE.idPersonne, nom, prenom, count(*) as nombreLivresEmprunte
from EMPRUNT inner join PERSONNE on EMPRUNT.idPersonne = PERSONNE.idPersonne
where date_sortie = null
group by PERSONNE.idPersonne, nom, prenom
having count(*) > 3;
```

- Donner la moyenne des durées des emprunts

```
select round(avg(date_retour - date_sortie))
from EMPRUNT
where date_sortie is not null;
```

- Donner le nombre d'emprunts dont la durée est supérieure à la moyenne des durées des emprunts

```
select count(*)
from emprunt
where (date_sortie is not null) and
      ((date_retour - date_sortie) > (select (avg(date_retour - date_sortie))
                                      from EMPRUNT
                                      where date_sortie is not null));
```

- Quels sont les personnes qui ont emprunté un livre entre 10 et 20 jours

```
select distinct PERSONNE.idPersonne, nom, prenom
from PERSONNE inner join EMPRUNT on PERSONNE.idPersonne = EMPRUNT.idPersonne
where (date_retour is not null) and ((date_retour - date_sortie) between 10 and 20)
```

- Quels sont les personnes qui ont emprunté le livre Couleur d'Europe de Jean-Philippe Lerclos ?

```
select PERSONNE.idPersonne, nom, prenom
from (PERSONNE inner join EMPRUNT on PERSONNE.idPersonne = EMPRUNT.idPersonne)
    inner join (LIVRE inner join EXEMPLAIRE on LIVRE.isbn = EXEMPLAIRE.isbn)
    on EXEMPLAIRE.idLivre = EMPRUNT.idLivre
where titre = 'Couleur d''Europe' and auteur = 'Jean-Philippe Lerclos';
```

- Obtenir le(s) personne(s) qui ont actuellement empruntés dont le délai est dépassé

```
select PERSONNE.idPersonne, nom, prenom
from PERSONNE inner join EMPRUNT on PERSONNE.idPersonne = EMPRUNT.idPersonne
where (date_retour is null) and
      ((date_prolongation is null and (date_sortie + 28) < current_date) or
       (date_prolongation is not null and (date_prolongation +28) < current_date));
```

- Rechercher la date du premier emprunt de la personne dont l'id est 2

```
select min(date_sortie)
from EMPRUNT inner join PERSONNE on EMPRUNT.idPersonne = PERSONNE.idPersonne
where PERSONNE.idPersonne = 2;
```

Attribut dérivé

Pour créer un attribut dérivé dans PostgreSQL, il y a deux possibilités

- soit une fonction calculée ou une expression dans une requête, dans ce cas la valeur de l'attribut n'est pas stockée de manière permanente dans la table
  - expression dans une requête

```
select (case when date_prolongation isnull then date_sortie + 28
        else date_prolongation + 28 end) as date_retour_max,
        date_sortie, date_prolongation
from emprunt;
```

  - via une fonction

```
create function calculer_date_retour_max(date_sortie date, date_prolongation date)
returns date
as
$$
begin
    if (date_prolongation isnull)
    then
        return date_sortie + 28;
    else
        return date_prolongation + 28;
    end if;
end
$$ language plpgsql;

select date_sortie, date_prolongation,
        calculer_date_retour_max(date_sortie, date_prolongation) as date_retour_max
from emprunt;
```

- soit utiliser une colonne calculée ou mettre à jour manuellement (ou trigger) lors de l'insertion ou la mise à jour. Dans ce cas, la valeur de l'attribut dérivé est stockée de manière permanente dans la table.

- Trigger lors de l'insertion ou la mise à jour

```
CREATE FUNCTION trigger_date_retour() RETURNS trigger
as $$
BEGIN
    if (new.date_prolongation isnull)
    then
        new.date_retour_max=new.date_sortie+28;
    else
        new.date_retour_max=new.date_prolongation+28;
    end if;
    return new;
end;
$$ language plpgsql
```

```
create trigger calculer_retour_max
before insert or update of date_sortie,  date_prolongation
on emprunt
for each row
execute function trigger_date_retour();
```

- Colonne calculée (à partie de la version 12 de postgreSQL)

```
create table EMPRUNT    (
    idPersonne integer,
    idLivre integer,
    date_sortie date,
    date_prolongation date,
    date_retour date,
    date_retour_max date generated always as
         (case when date_prolongation is null then date_sortie+28 else date_prolongation+28 end)
        stored,
    primary key (idPersonne, idLivre, date_sortie))
```

Transactions

Les transactions sont un concept fondamental de tous les systèmes de bases de données. Le point essentiel d'une transaction est qu'il assemble plusieurs étapes en une seule opération tout-ou-rien.

Les états intermédiaires entre les étapes ne sont pas visibles par les autres transactions concurrentes, et si un échec survient empêchant la transaction de bien se terminer, alors aucune des étapes n'affecte la base de données.

Une transaction est réalisée en entourant les commandes SQL de la transaction avec les commandes **BEGIN** et **COMMIT**.

Si, au cours de la transaction, on décide qu'on ne veut pas valider, on exécute la commande **ROLLBACK** au lieu de **COMMIT**, et toutes nos mises à jour jusqu'à maintenant seront annulées.

En fait, PostgreSQL traite chaque instruction SQL comme étant exécutée dans une transaction.

Si on ne lance pas une commande **BEGIN**, alors chaque instruction individuelle se trouve enveloppée avec un **BEGIN** et (en cas de succès) un **COMMIT** implicites. Un groupe d'instructions entouré par un **BEGIN** et un **COMMIT** est quelque fois appelé un bloc transactionnel.

Exemple :

Supposons qu'on puisse réserver un livre à la bibliothèque, dès que le livre réservé est emprunté, on supprime la réservation de la table *RESERVATION* et on ajoute un enregistrement dans la table *EMPRUNT*. Si une de ces deux commande SQL (*delete, insert*) échoue, aucune des deux ne doit être exécutée.

```
BEGIN;

delete from RESERVATION
where idPersonne = 3 and idlivre = 9

insert into EMPRUNT(idPersonne, idlivre, date_sortie)
values(3,9, current_date);

rollback;
```

```
BEGIN;

delete from RESERVATION
where idPersonne = 3 and idlivre = 9

insert into EMPRUNT(idPersonne, idlivre, date_sortie)
values(3,9, current_date);

commit;
```

Héritage

PostgreSQL implémente l'héritage des tables

Soit la table *PERSONNE* qui contient les patients, soit la table *MEDECIN* qui contient les médecins, cette table hérite de la table *PERSONNE* et contient deux champs supplémentaires : *matricule* et *specialite*.

```
create type adresse as (
    rue varchar(35),
    cp char(4),
    localite varchar(20));


create table PERSONNE(
    onss char(12)  primary key ,
    nom varchar(15),
    prenom varchar(15),
    date_naissance date,
    mutualite varchar(25),
    adressePersonne adresse);

create table MEDECIN(
    matricule char(12) primary key,
    specialite varchar(20)
) inherits (PERSONNE);
```

On utilise le mot-clé ***inherits*** pour l'héritage.

- Insérons quelques enregistrements

```
insert into MEDECIN (matricule, specialite, onss, nom,
                     prenom, date_naissance, mutualite, adressePersonne)
values ('123456789012', 'generaliste', '987654321098', 'Toubib', 'Jean',
        '30/1/1970','Libre',('Rue de l''Industrie, 65', '4100','Seraing')),
        ('223456789012', 'generaliste', '887654321098', 'Malade', 'Marie',
        '28/2/1980','Libre',('Rue de l''école, 65', '4100','Seraing'));
```

```
insert into PERSONNE (ONSS, nom, prenom, date_naissance, mutualite, adressePersonne)
    values('456789012344', 'Anspach', 'Jules', '23/2/1958', 'Libre', ('rue du Bourg, 6', '4000', 'Liege')),
    ('556789012344', 'Buls', 'Charles', '23/3/1978', 'Libre', ('rue du Temple, 16', '4000', 'Liege'));
```

- Pour lister toutes les personnes :

```
select * from personne
```

Data Output Notifications

	onss [PK] character (12)	nom character varying (15)	prenom character varying (15)	date_naissance date	mutualite character varying (25)	adressepersonne adresse
1	456789012344	Anspach	Jules	1958-02-23	Libre	("rue du Bourg, 6",40...
2	987654321098	Toubib	Jean	1970-01-30	Libre	("Rue de l'Industrie, 6...
3	887654321098	Malade	Marie	1980-02-28	Libre	("Rue de l'école, 65",...

- Pour lister tous les médecins :

```
1  select * from MEDECIN;
```

Data Output Notifications

	onss character (12)	nom character varying (15)	prenom character varying (15)	date_naissance date	mutualite character varying (25)	adressepersonne adresse	matricule character (12)	speci chara
1	987654321098	Toubib	Jean	1970-01-30	Libre	("Rue de l'Industrie, 6...	123456789012	gener
2	887654321098	Malade	Marie	1980-02-28	Libre	("Rue de l'école, 65",...	223456789012	gener

- Pour lister les personnes qui ne sont pas médecin (mot-clé **only**)

```
select * from only personne
```

Data Output Notifications

	onss [PK] character (12)	nom character varying (15)	prenom character varying (15)	date_naissance date	mutualite character varying (25)	adressepersonne adresse
1	456789012344	Anspach	Jules	1958-02-23	Libre	("rue du Bourg, 6",40...

- Pour lister toutes les personnes et savoir s'ils font partie des tables enfants

```
select pg.relname, per.nom, per.prenom
from PERSONNE per, pg_class pg
where per.tableoid=pg.oid;
```

Data Output Notifications

	relname name	nom character varying (15)	prenom character varying (15)
1	personne	Anspach	Jules
2	medecin	Toubib	Jean
3	medecin	Malade	Marie

- Insérons dans la table MEDECIN, un médecin avec le même ONSS que le médecin Toubib Jean

```
insert into MEDECIN (matricule, specialite, onss, nom,
                     prenom, date_naissance, mutualite, adressePersonne)
values ('323456789012', 'pediatrie', '987654321098', 'Lepetit', 'Luc',
        '30/4/1976','Libre',('Rue de l''Acier, 44', '4100','Seraing'));
```

Messages

```
Query returned successfully in 56 msec.
```

Query Editor

```
1  select * from medecin;
```

ta Output Notifications

onss character (12)	nom character varying (15)	prenom character varying (15)	date_naissance date	mutualite character varying (25)	adressepersonne adresse	matricule [PK] character (12)	speci charac
987654321098	Toubib	Jean	1970-01-30	Libre	("Rue de l'Industrie, 6...	123456789012	genera
887654321098	Malade	Marie	1980-02-28	Libre	("Rue de l'école, 65",...	223456789012	genera
987654321098	Lepetit	Luc	1976-04-30	Libre	("Rue de l'Acier, 44",4...	323456789012	pediat

L'insertion s'est bien déroulée, malgré que ONSS est une clé primaire de la table PERSONNE.

- Insérons dans la table MEDECIN, un médecin avec le même ONSS que Anspach Jules qui n'est pas médecin.

```
Insert into PERSONNE(onss, nom,
                     prenom, date_naissance, mutualite, adressePersonne)
values( '987654321098','Impossible', 'Lucas', '24/5/1975', 'Libre',
       ('Rue de la Paix, 40', '4100', 'Seraing')) ;
```

```
select * from PERSONNE;
```

)ata Output Notifications

	onss [PK] character (12)	nom character varying (15)	prenom character varying (15)	date_naissance date	mutualite character varying (25)	adressepersonne adresse
1	987654321098	Impossible	Lucas	1975-05-24	Libre	("rue de la Paix, 7",40...
2	987654321098	Toubib	Jean	1970-01-30	Libre	("Rue de l'Industrie, 6...
3	887654321098	Malade	Marie	1980-02-28	Libre	("Rue de l'école, 65",...
4	987654321098	Lepetit	Luc	1976-04-30	Libre	("Rue de l'Acier, 44",4...

L'insertion s'est bien déroulée, malgré un ONSS identique. Or ONSS est la clé primaire de la table PERSONNE.

- Insérons dans la table PERSONNE, un enregistrement avec le même ONSS qu'une autre personne qui n'est pas médecin.

  L'insertion est impossible car viole la clé primaire.

La contrainte UNIQUE du champ *ONSS* n'a pas été héritée par la table *MEDECIN*.

- Supprimons tous les enregistrements des tables *PERSONNE* et *MEDECIN*
- Ajoutons une contrainte d'unicité du champ *onss* à la table *MEDECIN*

```
alter table MEDECIN add constraint onss_unique unique (onss);
```

- Ensuite, réinsérons les enregistrements dans la table MEDECIN

```
insert into MEDECIN (matricule, specialite, onss, nom,
                     prenom, date_naissance, mutualite, adressePersonne)
values ('123456789012', 'generaliste', '987654321098', 'Toubib', 'Jean',
        '30/1/1970','Libre',('Rue de l''Industrie, 65', '4100','Seraing')),
        ('223456789012', 'generaliste', '887654321098', 'Malade', 'Marie',
        '28/2/1980','Libre',('Rue de l''école, 65', '4100','Seraing'));
```

Maintenant, nous ne pouvons plus ajouter un médecin ou une personne ayant le même ONSS qu'un autre médecin.

Les contraintes d'unicité ne sont pas héritées, il faut les redéfinir dans les tables enfants.

- Ajoutons une contrainte de vérification à la table PERSONNE

```
alter table PERSONNE add check (char_length(nom) > 3) ;
```

```
insert into personne (onss, nom, prenom, date_naissance, mutualite, adressePersonne)
values('192837465012','U', 'Prenom', '02/02/2000', 'Libre',('rue d, 3', '4000','Liege'));
```

Messages

```
ERROR:  ERREUR:  la nouvelle ligne de la relation « personne » viole la
contrainte de vérification « personne_nom_check »
DETAIL:  La ligne en échec contient (192837465012, U, Prenom, 2000-02-02,
Libre, ("rue d, 3",4000,Liege))
```

```
insert into medecin (matricule, specialite,
                     onss, nom, prenom, date_naissance, mutualite, adressePersonne)
values('239846122334','pediatrie','192837465012','U',
       'Prenom', '02/02/2000', 'Libre',('rue d, 3', '4000','Liege'));
```

Messages

```
ERROR:  ERREUR:  la nouvelle ligne de la relation « medecin » viole la
contrainte de vérification « personne_nom_check »
DETAIL:  La ligne en échec contient (192837465012, U, Prenom, 2000-02-02,
Libre, ("rue d, 3",4000,Liege), 239846122334, pediatrie)
```

Par contre, **les contraintes de vérification sont héritées.**

En conclusion, les contraintes peuvent être définies sur les tables de la hiérarchie d'héritage. Toutes les contraintes de vérification d'une table parent sont automatiquement héritées par tous ses enfants. Néanmoins, les autres types de contraintes (unicité, clé étrangère, …) ne sont pas hérités.