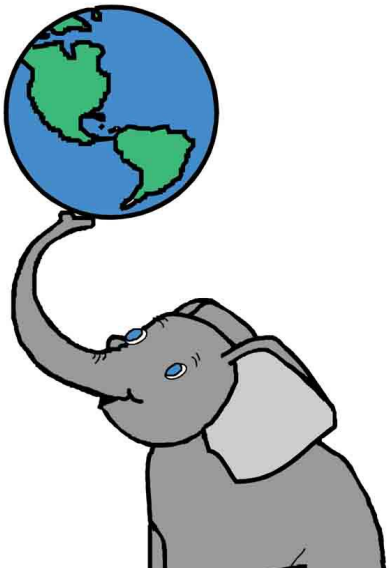
Rappels et concepts de base



Version 1.4

Janvier 2022

Ministère de la Transition Ecologique
Licence ETALAB

Table des matières

Objectifs

Les objectifs du module sont de :

- Réviser les notions sur les bases relationnelles et la modélisation
- découvrir les avantages et inconvénients de PostgreSQL
- Réviser des concepts de PostGIS.

Introduction

La durée de ce module est estimée à 2h30.
L'exercice de fin vous demandera de participer à un fil de discussion sur la liste.

I

Rappels sur les bases de données relationnelles

A. Définitions et types de bases de données

Définition : Systèmes de Gestion de Bases de Données (SGBD)

Un Système de Gestion de Base de Données (SGBD) est un logiciel permettant de stocker de la donnée dans une base de données en garantissant la qualité, la pérennité et la confidentialité des informations. La complexité des opérations de traitement des données ne nécessite pas d'être totalement connue par les utilisateurs.

Types de bases de données

Selon leur construction et les possibilités qu'ils offrent les SGBD peuvent être dit hiérarchiques, relationnels, orientés objet, objet-relationnel, XML/RDF ou mixte. Ils peuvent être distribués, centralisés ou embarqués et peuvent être spatiaux (source Wikipedia).

Depuis 2010, la majorité des SGBD sont de type relationnel (SGBDR) : ils manipulent des bases de données conformément au modèle de données relationnel.

Selon ce modèle, les données sont placées dans des tables avec lignes et colonnes et toute donnée contenue dans la base de données peut être retrouvée à l'aide du nom de la table, du nom de la colonne et de la clé primaire.

Le modèle relationnel est destiné à assurer l'indépendance des données et à offrir les moyens de contrôler la cohérence et d'éviter la redondance.

Les règles de cohérence qui s'appliquent aux bases de données relationnelles garantissent l'unicité et l'interdiction de nullité des clés primaires, et l'intégrité référentielle (nous reviendrons sur ces notions)

PostgreSQL est un système gestion de base de données relationnelle et objet (SGBDRO) multiplate-formes (Windows, Solaris, SunOS, Mac OS X, HP-UX, AIX, Linux, IRIX, Digital Unix, BSD, NetBSD, FreeBSD, OpenBSD, SCO unix, NeXTSTEP, UnixWare et toutes sortes d'Unix).

C'est un outil libre sous licence *BSD*[1].

L'aspect Objet dans PostgreSQL est une extension au modèle relationnel avec par exemple le support des *tableaux*[2], de l'*héritage* [3] (voir les limitations actuelles concernant l'héritage en bas de page) et des *fonctions*[4].

PostgreSQL peut être étendu par l'utilisateur de multiples façons :

- De nouveaux types de données
- De nouvelles fonctions
- De nouveaux opérateurs
- De nouvelles fonctions d'agrégat
- De nouvelles méthodes d'indexation
- De nouveaux langages de procédure

PostGIS apporte à PostgreSQL la capacité de gestion et de traitement des données géographiques.

B. Acteurs et rôles

Les SGBD sont des outils très puissants. Leur usage n'est pas immédiat.

Fondamental

La mise en place d'une base de données nécessite des compétences variées qui doivent être réparties selon plusieurs rôles (une même personne pouvant avoir plusieurs rôles).

Analyste : Il prend en charge la modélisation des activités et produit à partir du monde réel, un Modèle Conceptuel des Données (MCD) et un Modèle Conceptuel des Traitements (MCT) par exemple avec la méthode Merise ou par la création des différents diagrammes structurels et comportementaux de l'UML (langage de modélisation unifié). Voir par exemple une introduction à UML *ici*[5].

Concepteur de la base : Il traduit le modèle conceptuel et un modèle logique exploitable par un SGBD donné (ici PostgreSQL). Il prépare les tables, les vues, les Schémas d'accès (nous détaillerons ces notions plus tard).

Administrateur de la base (DBA ou DataBase Administrator) : Il a la responsabilité du fonctionnement du SGBD. Il crée les bases, gère les droits d'accès et les rôles dans postgreSQL.

Administrateur Système : Il gère l'installation du système. Il dispose des droits de plus haut niveau sur la base (superuser) et sur le système d'exploitation. Il gère les

1 - https://fr.wikipedia.org/wiki/Licence_BSD
2 - http://docs.postgresql.fr/12/arrays.html
3 - http://docs.postgresql.fr/12/ddl-inherit.html
4 - http://docs.postgresql.fr/12/xfunc.html
5 - http://laurent-audibert.developpez.com/Cours-UML/?page=introduction-modelisation-objet#L1-4

droits d'accès au serveur (fichiers de configuration). Il met en œuvre la politique de sauvegarde en liaison avec l'administrateur de la base. Il met en œuvre la supervision des serveurs et remet en route le système en cas de problème. Il vérifie le bon fonctionnement (ressource disque, mémoire, ...).

Utilisateurs : Ils ont accès au système d'information défini dans le SGBD grâce aux schémas, tables et vues définies par le concepteur. Ils accèdent au SGBD au travers de clients logiciels leur permettant de charger les couches (ex : QGIS) et/ou de formuler des requêtes SQL (ex :QGIS ou PgAdminIII)

Fondamental

Une préparation de tout travail dans un nouveau domaine à prendre en compte est donc indispensable en amont de l'outil informatique ; ceci s'apparente à l'analyse conceptuelle d'un problème :

- le champ d'application et la problématique
- les données, leur utilité et leurs attributs
- le traitement des données
- les relations et les cardinalités entre les éléments

Dans cette formation nous n'abordons pas en profondeur la conception des bases de données. Il conviendra donc si le besoin s'en fait sentir de compléter la formation par une formation de type 'Concevoir et structurer une base de données géographiques' ou en consultant, par exemple, *les supports mis en ligne par Stéphane Crozat*[6].

La formation sur les *bases de données proposée par l'INSA*[7] pourra également être consultée.

Nous proposons toutefois ci-après un quelques rappels sur la démarche à mettre en œuvre.

C. Rappels sur la modélisation

Le **monde réel** doit être modélisé sous une forme abstraite et simplifiée. Le modèle adopté permet de réduire la complexité d'un phénomène en éliminant les détails inutiles du point de l'objectif que l'on se fixe. Il existe *différentes méthodes de modélisation*[8], et une notation qui est assez généralement adoptée ; *UML*[9].

Dans le cadre de l'utilisation d'un SGBD ce qui nous intéresse le plus est la *modélisation des données*[10] (pas des traitements comme dans le cas de la réalisation d'une application informatique). On s'intéressera donc avant tout à ce qui sous *MERISE*[11] ou dans le modèle *entité-association*[12] s'appelle le **modèle conceptuel des données** (indépendant de la base du SGBD) et qui est décliné ensuite en **modèle logique**, puis en **modèle physique** qui est le modèle directement exploitable dans la base de données (ex : PostgreSQL) suivant un *modèle relationnel*[13] (avec parfois

6 - https://stph.scenari-community.org/bdd/0/co/bdd_1.html
7 - http://scenari.insa-rouen.fr/utop/table_matieres/version_2015-02/cours/co/utop_web.html
8 - https://fr.wikipedia.org/wiki/Méthodes_d'analyse_et_de_conception
9 - https://fr.wikipedia.org/wiki/UML_(informatique)
10 - https://fr.wikipedia.org/wiki/Modèle_de_données
11 - https://fr.wikipedia.org/wiki/Merise_(informatique)
12 - https://fr.wikipedia.org/wiki/Modèle_entité-association
13 - https://fr.wikipedia.org/wiki/Modèle_relationnel

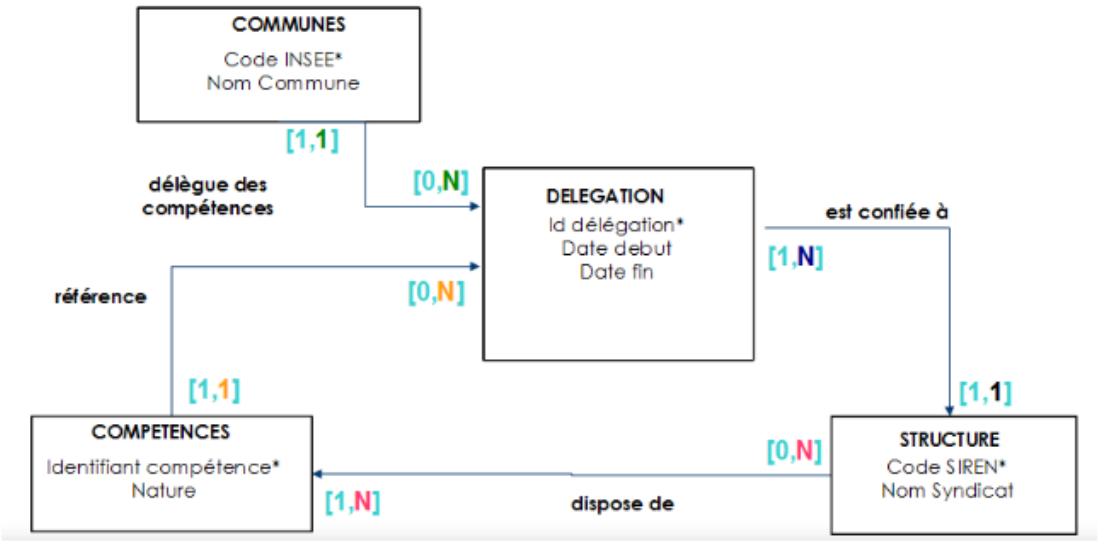
utilisation de l'héritage venant du modèle objet, puisque PostgreSQL est un SGBD *relationnel-objet*[14]). Avec UML on s'intéressera avant tout au *diagramme de classe*[15].

Pour en savoir plus il est conseillé de suivre un stage sur la conception des bases de données. Voir par exemple *ce cours sur MERISE*[16]. A noter qu'il existe des ateliers logiciels (« postgreSQL workbench ») plus ou moins sophistiqués permettant de modéliser les bases de données. Par exemple *pgmodeler*[17] qui est un outil éventuellement à tester mais dont les binaires sont devenus payants.

Looping-mcd[18] est également un outil qui peut permettre de s'initier à la modélisation conceptuelle.

Dans ce cours nous serons amenés à prendre pour exemple la problématique de la gestion des délégations de compétences des communes.

Les objets principaux sont les *communes*, les *compétences* et les *organismes de gestion*. Avec le formalisme adopté dans la formation 'concevoir et structurer une base de données géographiques', le schéma conceptuel que nous allons utiliser est le suivant :



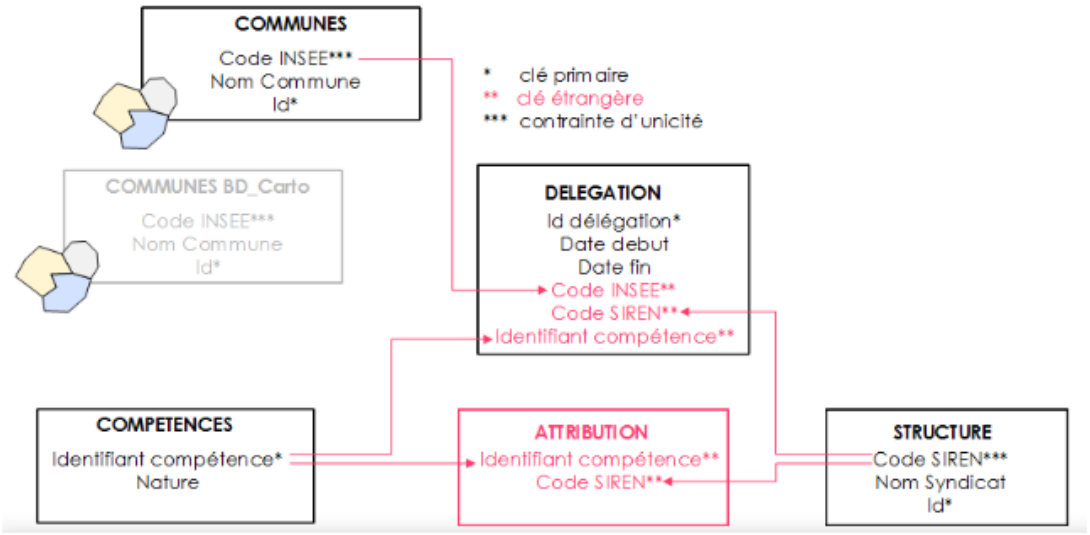
délégation de compétences - schéma conceptuel avec cardinalités

Les [19] (ou multiplicité) sont une notion fondamentale dans la modélisation des données. Ici nous pouvons lire le modèle :

- Une commune délègue de 0 à N compétences (date de début, date de fin).
- Une délégation est confiée à une et une seule structure.
- Une structure gère de 1 à N délégations
- ...

14 - https://fr.wikipedia.org/wiki/Base_de_données_orientée_objet
15 - http://laurent-audibert.developpez.com/Cours-UML/?page=diagramme-classes
16 - http://ineumann.developpez.com/tutoriels/merise/initiation-merise/
17 - http://www.pgmodeler.com.br/
18 - https://www.looping-mcd.fr/
19 - http://laurent-audibert.developpez.com/Cours-UML/?page=diagramme-classes#L3-3-4

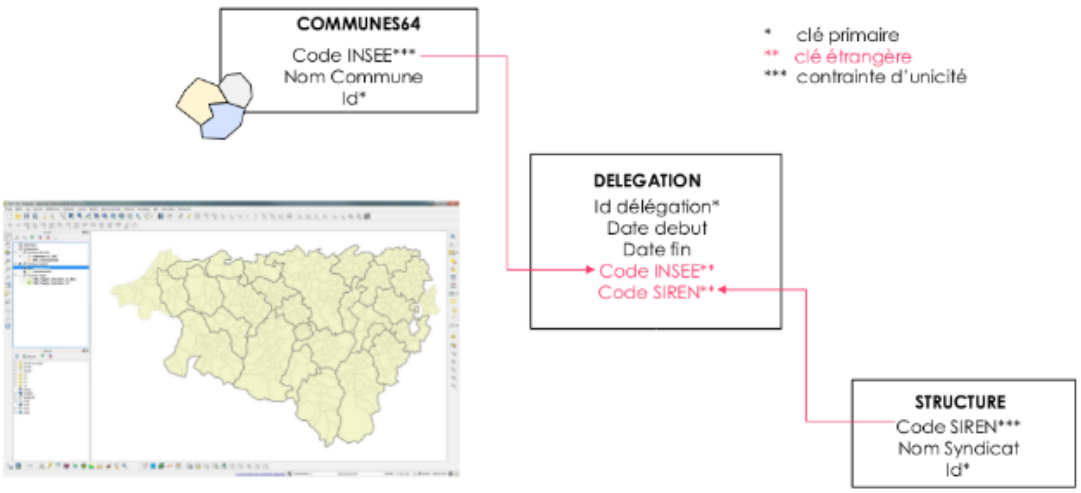
Ce schéma conceptuel peut-être décliné dans le modèle physique suivant :



modèle physique

Les notions fondamentales à connaître sont celle de ***clef primaire***[20] et [21]. Qui vont traduire les relations dans le modèle logique. Il existe des *règles de passage*[22] d'un modèle conceptuel à un modèle physique. Dans le domaine de la géomatique il est également important d'identifier les tables qui seront porteuses d'un (ou plusieurs) type de géométrie. Dans cet exemple seul, la table commune est porteuse d'une géométrie de type POLYGON.

Pour les besoins de ce cours nous utiliserons un modèle physique simplifié :



modèle simplifié

20 - https://fr.wikipedia.org/wiki/Clé_primaire
21 - https://fr.wikipedia.org/wiki/Clé_étrangère
22 - http://sqlpro.developpez.com/cours/modelisation/merise/?page=passage#L5.1

D. Clef primaire et clefs étrangères

Une clef primaire sert à identifier une ligne d'une table de façon unique. Dans un SGBDR la clef primaire est identique à une contrainte d'unicité et une contrainte NOT NULL, composée de une ou plusieurs colonnes.

Exemple en SQL :

```
CREATE TABLE exemple (a integer, b integer, c integer, PRIMARY KEY(a,b)) ;
```

Une clé primaire indique qu'une colonne ou un groupe de colonnes (dans l'exemple ci-dessus a et b) peut être utilisé(e) comme identifiant unique des lignes de la table. L'ajout d'une clé primaire créera automatiquement un index unique sur la colonne ou le groupe de colonnes utilisé dans la clé primaire. Une table a, au plus, une clé primaire.

Le choix d'une clé primaire est une étape importante dans la conception d'une table. Il peut être nécessaire d'ajouter une colonne qui soit auto-incrémentée.

Si on considère pour les besoins de l'exemple, que l'on souhaite pouvoir gérer un propriétaire d'un véhicule même s'il ne possède pas de permis, alors le numéro de permis ne peut être la clef primaire. On va donc définir une nouvelle colonne *id* qui sera incrémentée automatiquement.

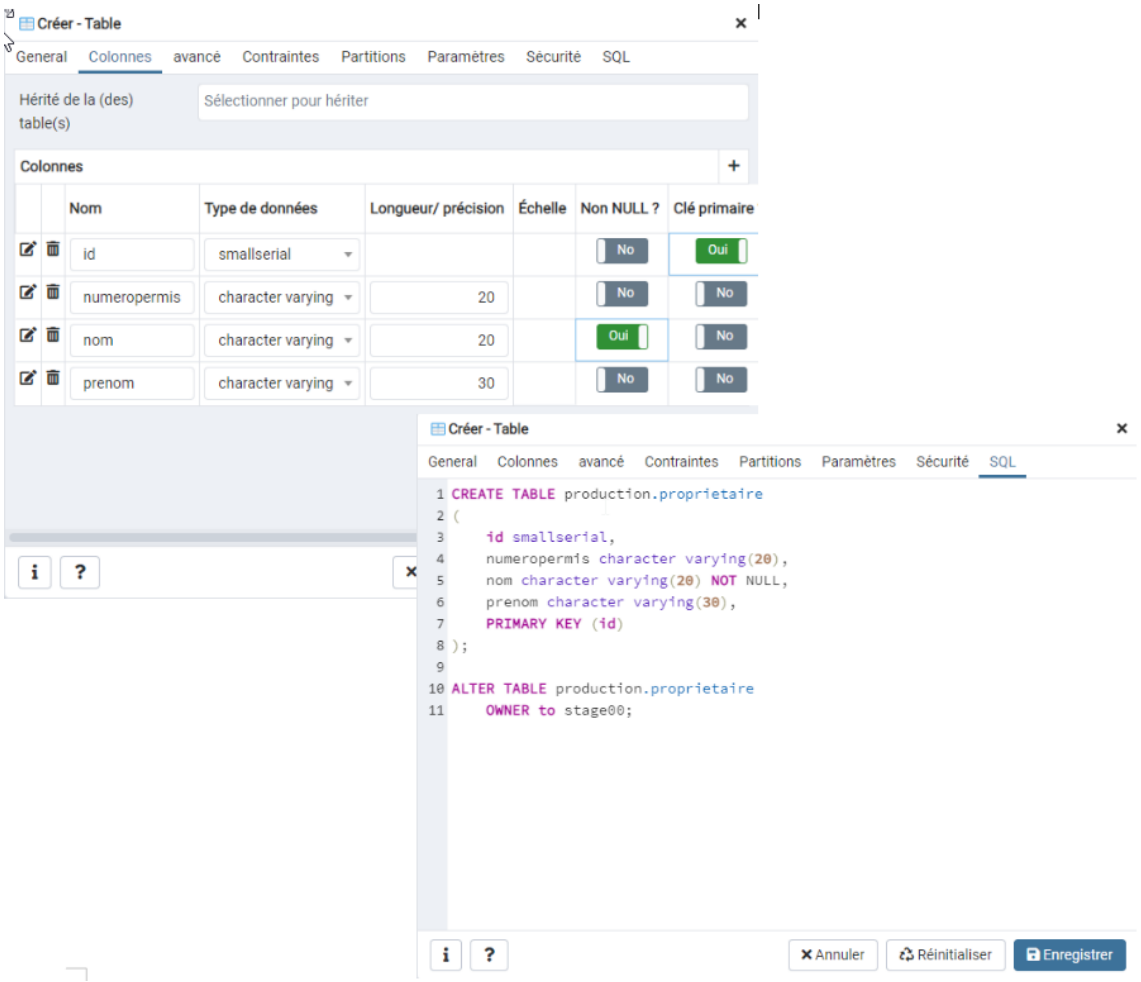
(On pourra consulter en complément *cet extrait*[23] de la formation de Stéphane Crozat sur les clefs artificielles et les clefs signifiantes)

Le SQL pour créer la table s'écrira :

```
CREATE TABLE proprietaire (
id SMALLINT AUTO_INCREMENT,
numeropermis VARCHAR(20),
nom VARCHAR(30) NOT NULL,
prenom VARCHAR(30),
PRIMARY KEY (id)
)
```

23 - https://stph.scenari-community.org/bdd/rel1/co/relUC022.html

Nous verrons qu'il est possible de réaliser les requêtes SQL à l'aide des assistants sous PgAdminIII et vérifier la syntaxe SQL dans le dernier onglet de l'assistant :



on notera que *smallserial* et *smallint AUTO_INCREMENT* sont équivalent, de même que *VARCHAR* et *character varying*.

Complément : Les séquences et type sérié

Sous PostgreSQL on peut utiliser les *séquences*[24] pour générer une clef primaire. Voir également *les types sériés*[25].

Une contrainte de clé étrangère stipule que les valeurs d'une colonne (ou d'un groupe de colonnes) doivent correspondre aux valeurs qui apparaissent dans les lignes d'une autre table. On dit que cela maintient l'**intégrité référentielle** entre les deux tables.

Si nous reprenons notre exemple Propriétaire / véhicule.

Si on considère qu'un véhicule doit toujours avoir un propriétaire préalablement existant, on peut déclarer :

```
CREATE TABLE vehicule (
numeroserie VARCHAR(20) PRIMARY KEY,
type VARCHAR(20),
```

24 - http://dgriessinger.developpez.com/postgresql/sequences/
25 - https://docs.postgresql.fr/12/datatype-numeric.html#DATATYPE-SERIAL

```
marque VARCHAR(20),
id_proprietaire SMALLINT REFERENCE proprietaire(id)
)
```

Lors de la saisie d'un nouveau véhicule, le système vérifiera que l'id saisie dans id_proprietaire existe bien dans la table proprietaire.

Une table peut contenir plusieurs contraintes de clé étrangère. Les relations n-m entre tables sont implantées ainsi (voir par exemple l'extrait de la formation de Stéphane Crozat à partir d'*ici*[26]).

voir également *la documentation de PostgreSQL*[27].

E. Les contraintes

Les types de données sont un moyen de restreindre la nature des données qui peuvent être stockées dans une table. Pour beaucoup d'applications, toutefois, la contrainte fournie par ce biais est trop grossière. Par exemple, une colonne qui contient le prix d'un produit ne doit accepter que des valeurs positives, mais il n'existe pas de type de données standard qui n'accepte que des valeurs positives. Un autre problème peut provenir de la volonté de contraindre les données d'une colonne par rapport aux autres colonnes ou lignes. Par exemple, dans une table contenant des informations de produit, il ne peut y avoir qu'une ligne par numéro de produit.

Pour cela, SQL permet de définir des contraintes sur les colonnes et les tables. Les contraintes donnent autant de contrôle sur les données des tables qu'un utilisateur peut le souhaiter. Si un utilisateur tente de stocker des données dans une colonne en violation d'une contrainte, une erreur est levée.

Les types de contraintes :

- **NOT NULL** : Impose que la colonne soit renseignée lors d'un UPDATE ou INSERT (mise à jour ou insertion d'enregistrement).
- **UNIQUE** : les valeurs de la colonne doivent être toutes différentes
- **CHECK** : conditions sur les valeurs acceptées (contrainte de vérification). Elle permet d'indiquer que la valeur d'une colonne particulière doit satisfaire une expression booléenne (valeur de vérité).

Par exemple, pour obliger les prix des produits à être positifs, on peut utiliser :

```
CREATE TABLE produits (
no_produit integer,
nom text,
prix numeric CHECK (prix > 0)
);
```

Une contrainte de vérification s'utilise avec le mot clé CHECK suivi d'une expression entre parenthèses.

Autres contraintes :

- **PRIMARY KEY** : clef primaire. Une clef primaire est une contrainte qui combine une contrainte d'unicité et une contrainte NOT NULL.
- **FOREIGN KEY** : clef étrangère.

Une commande SQL ALTER TABLE (que nous verrons plus tard) peut être utilisé pour ajouter ou supprimer une contrainte.

26 - https://stph.scenari-community.org/bdd/0/co/rel2a2.html
27 - http://docs.postgresql.fr/12/tutorial-fk.html

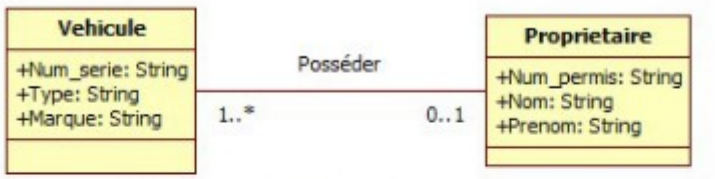
Pour en savoir plus sur les *contraintes*[28].

F. Modélisation UML ; les standards du CNIG

Les standards du CNIG dont on trouvera un recensement par exemple sur *geo2france.fr*[29] sont de bons exemples de modélisation de problématiques métiers.

Ils utilisent le langage de modélisation unifié, de l'anglais Unified Modeling Language (UML), qui est un langage de modélisation graphique à base de pictogrammes conçu pour fournir une méthode normalisée pour visualiser la conception d'un système. Comme indiqué, dans les bases de données on utilise essentiellement le diagramme de classe.

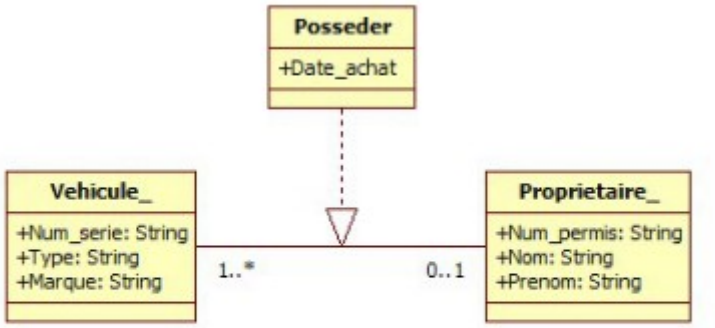
Voici d'autres exemples de diagrammes UML avec leur interprétation :



Description de la perception rédigée pour être compris…

Un VEHICULE a 0 ou 1 propriétaire.
Il est reconnu par un Num_serie (unique pour chaque véhicule).
Il porte comme information : le Type et la marque du véhicule.

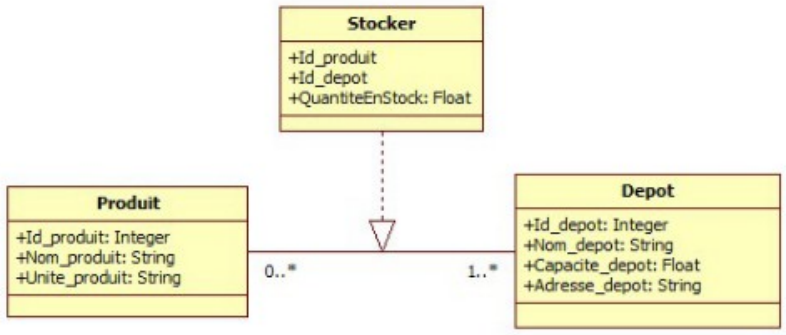
Un PROPRIETAIRE posséde 1 ou plusieurs véhicules.
Il est reconnu par son Num_permis (unique pour chaque propriétaire).
Il porte comme informations : le Nom et le prénom du propriétaire.



Description de la perception rédigée pour être compris…

On créé une table de relation porteuse d'un attribut (Date_achat).

28 - http://docs.postgresql.fr/12/ddl-constraints.html
29 - https://www.geo2france.fr/portail/recensement-des-modeles-de-donnees-nationaux



Description de la perception rédigée pour être compris...

Un PRODUIT représente les matières utilisées par l'entreprise (sable, colle, briques, clous, etc..)
Il est reconnu par un Code-produit (unique pour chaque produit)
Il porte comme information : le Nom-produit et l'Unité de stockage du produit

Un DEPOT représente les dépôts de l'entreprise qui sont répartis dans le département
Il est reconnu par son Code-Dépôt (unique pour chaque dépôt)
Il porte comme informations : le Nom-dépôt, la capacité (en m3), l'adresse du dépôt.

Un Produit peut être stocké dans un ou plusieurs Dépôts
Un Dépôt peut être vide ou contenir plusieurs produits

Il est intéressant de conserver la Qté en stock des produits dans chaque dépôt.
SI je place la propriété dans PRODUIT, elle devient la Qté-en-stock du produit, et pose problème si le produit est stocké dans plusieurs dépôts,
si je la place dans DEPOT, elle devient la quantité en stock du dépôt, ce qui pose problème s'il y a plusieurs produits

Complément

Pour en savoir plus on pourra, par exemple, consulter la formation en ligne élaborée par *Stéphane Crozat*[30] sur UML.

G. Quiz sur les concepts de base

Exercice 1

[*Solution n°1 p 35*]

PostgreSQL est un SGBD de type

30 - https://stph.scenari-community.org/bdd/mod1/co/mod1.html

- Hiérarchique
- orienté objet
- XML/RDF
- mixte
- objet-relationnel

Exercice 2

[*Solution n°2 p 35*]

Quel est la signification du sigle MCD ?

- Modèle Constitutif des Données
- Modèle Conceptuel des Données
- Modèle Contrainte Dépendance
- Modèle Coût Dépense
- Modèle Commun de Données

Exercice 3

[*Solution n°3 p 36*]

Une contrainte d'intégrité référentielle interdit

- une valeur de clé étrangère qui ne correspond à aucune clé primaire de la table référencée.
- des doublons dans une clef primaire
- des doublons dans une clef étrangère

Exercice 4

[Solution n°4 p 36]

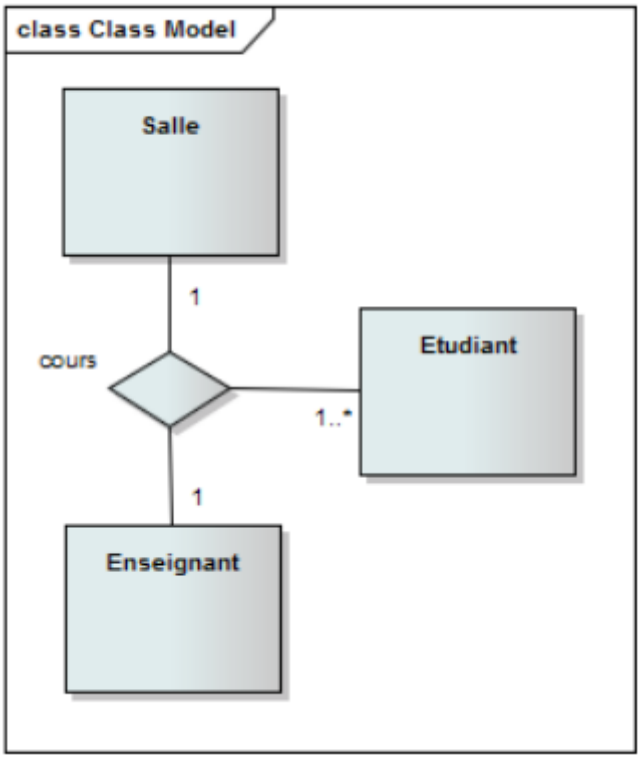
Indiquez quelles sont les affirmations suivantes sur les cardinalités qui sont vraies :

☐	1 = un et un seul
☐	0 .. * = 0 à plusieurs
☐	0..1 = facultatif
☐	1 .. * = obligatoire
☐	2 .. 4 = de deux à quatre

Exercice 5

[Solution n°5 p 36]

A partir du diagramme de classe ci-dessous indiquez quelles sont les affirmations suivantes qui sont vraies :



☐	Un enseignant fait ses cours dans une seule classe à au moins un étudiant
☐	Une salle accueille les cours pour un seul enseignant
☐	Un cours peut ne pas avoir d'étudiant
☐	Un étudiant suit des cours dans une seule salle avec un seul enseignant

II

Avantages et inconvénients de PostgreSQL

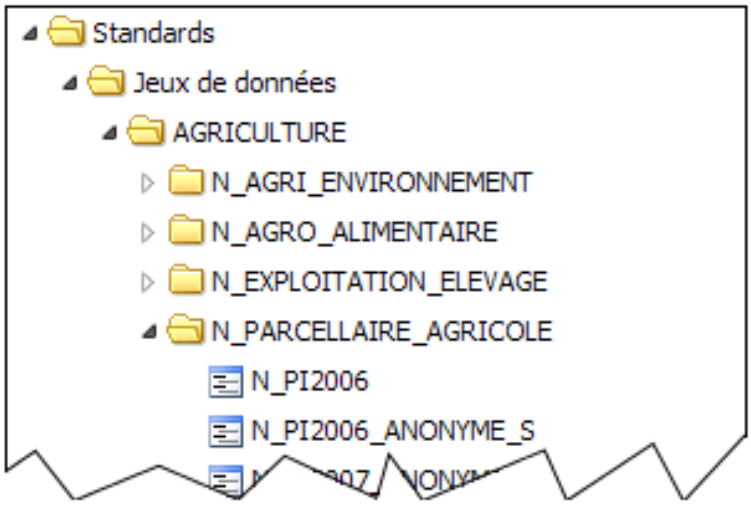
A. Avantages et inconvénients de PostgreSQL

Un SGBDRO comme PostgreSQL permet de stocker une information beaucoup plus structurée en termes de relations et de contraintes qu'avec une arborescence de fichiers 'à plat' comme des fichiers SHP. Il permet en outre d'outrepasser les limites inhérentes à ces formats propriétaires (comme la limite à 10 caractères des noms de champs en SHP) et peut servir de format 'pivot' pour générer ensuite en export et import les différents formats.

En contrepartie il n'est pas possible d'y stocker une structure arborescente de fichiers comme celle de la *GéoBASE*[31]

Voir cependant l'encadré sur **ASGARD** en fin de page...

31 - http://www.geoinformations.developpement-durable.gouv.fr/nouvelle-arborescence-geobase-et-patrimoine-de-a994.html



Les avantages de l'utilisation PostgreSQL/PostGIS dans le domaine de la géomatique sont :

- Meilleure prise en charge des données volumineuses (ex : fichiers MAJICS, RPG, référentiels vecteurs,...).
- Exploitation de modèle de données complexes. Gestion des relations entre données géographiques et données attributaires.
- Gestion fine possible des droits d'accès.
- Possibilité de disposer de 'vues' (tables virtuelles résultantes d'une requête SQL de type SELECT...)
- Gestion des accès concurrentiels.
- Lien fort avec QGIS et fonctions spatiales plus étendues que sous QGIS natif.
- Format de données 'pivot' outrepassant les limites des logiciels propriétaires et indépendant des solutions propriétaires.
- Possibilité de développer des applications 'métiers' (même simples, par exemple avec des formulaires QGIS associés à des déclencheurs (triggers) dans PostgreSQL.

Les inconvénients sont :

- Déployer un SGBD est un projet de service et doit donc être planifié (charge de travail,...) et géré comme tel (implication d'un nombre varié d'acteurs).
- Les compétences à acquérir et maintenir,
- La charge de travail supplémentaire pour l'exploitation.
- La mobilisation de compétences 'systèmes' pour l'installation et le paramétrage.
- Le projet de migration des données et de reprise de l'existant (en particulier des 'projets' QGIS) en cas de décision de migration d'un patrimoine existant sous forme de fichiers 'à plat'.
- Les opérations de migration en cas de changement de version majeure de PostgreSQL.
- L'obligation de maintenir et de gérer les imports / exports vers les fichiers 'à plat' qui sont plus faciles à véhiculer pour les échanges avec les partenaires et une nécessité pour l'utilisation de certains progiciels.

Complément : Nomenclature nationale des schémas et suite ASGARD.

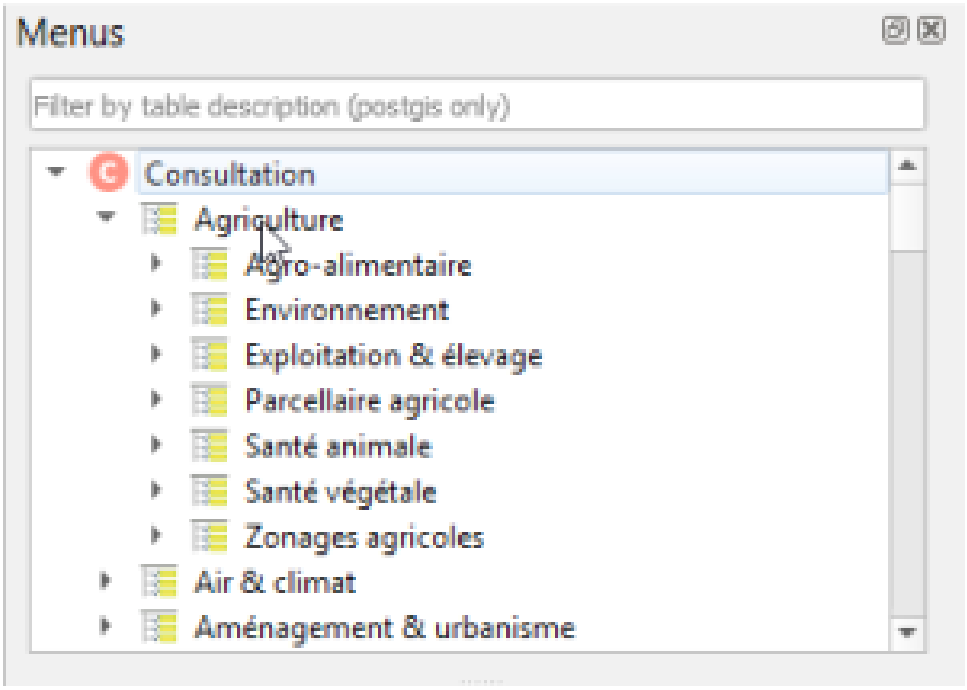
En 2020, un groupe de travail a produit *une recommandation nationale*[32] (accessible en extranet uniquement) sur la nomenclature des schémas pour les bases de données dans PostgreSQL.

Un des buts de la nomenclature est de permettre de simuler un niveau d'arborescence supplémentaire afin de pouvoir présenter le patrimoine de données dans une structure proche de celle de la GéoBASE à l'aide du plugin AsgardMenu.

En réalité, **ASGARD** est une suite répondant à plusieurs objectifs et comprenant :

- Une extension PostgreSQL : qui implémente un modèle de gestion des droits simplifiés, une classification des schémas en blocs fonctionnels et la nomenclature nationale des schémas.
- un plugin AsgardManager : qui propose une interface ergonomique pour les fonctionnalités courantes disponibles avec **ASGARD**.
- un plugin AsgardMenu : qui est un outil de présentation du patrimoine de données en base PostgreSQL, sous la forme d'un menu paramétrable.

Exemple de menu d'accès au patrimoine généré pas AsgardMenu :



ASGARD fait l'objet d'un *accompagnement*[33] (lien extranet) spécifique.

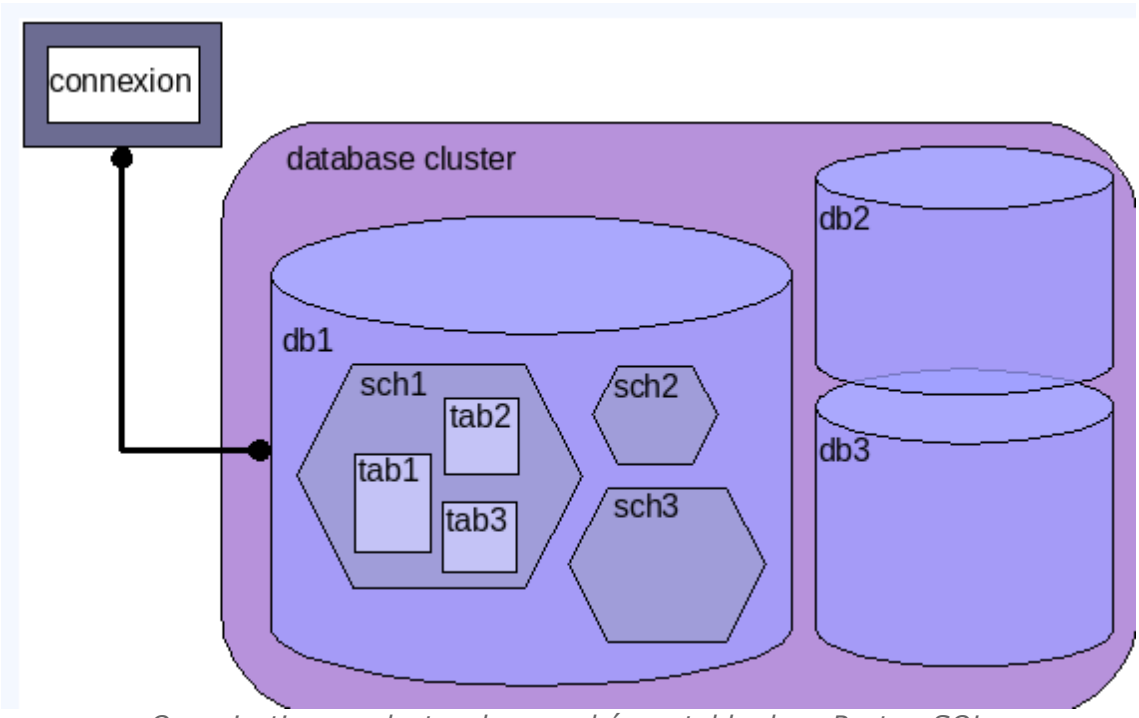
32 - http://geoinformations.metier.e2.rie.gouv.fr/groupe-de-travail-utilisation-de-postgis-dans-l-a3733.html
33 - http://geoinformations.metier.e2.rie.gouv.fr/ressources-asgard-a3758.html

III

Concepts de base de PostgreSQL / PostGIS

A. Bases et schémas

L'organisation générale d'un serveur PostgreSQL est la suivante.



Organisation en cluster, base, schéma, table dans PostgreSQL

A l'installation du serveur un emplacement de stockage pour les bases de données a été initialisé. Ceci est appelé 'groupe de bases de données' ou encore 'grappe de base de données' (database cluster en anglais). Un groupe de bases de données est une collection de bases données et est géré par une seule instance d'un serveur de bases de données en cours d'exécution. Plusieurs groupes de bases de données peuvent fonctionner en même temps sur le même serveur.

Une instance correspond à une exécution du serveur PostgreSQL. Un groupe de base de données correspond à un conteneur de bases de données sous la forme de répertoires et de fichiers. Chaque instance dispose de ses structures mémoires, ses processus, son arborescence de fichiers, ses fichiers de configuration, son port d'écoute,...(pour l'administrateur système voir la commande *INITDB*[34])

Un 'cluster' de bases de données contient une ou plusieurs base(s) nommée(s).

Conseil : Organisation en bases et schémas

Toute connexion cliente au serveur ne peut accéder qu'aux données d'une seule base, celle indiquée dans la requête de connexion. Il est donc important de bien organiser ses bases et de privilégier l'utilisation des schémas (voir ci-dessous) dans une même base de données pour les tables que l'on souhaite inter-opérer. Le 'cross-database' reste toutefois possible (avec Dblink ou Foreign data wrapper). Nous en verrons un exemple dans ce cours.

Une base de données contient un ou plusieurs **schéma**(s) nommé(s) qui, eux, contiennent des tables. Les schémas contiennent aussi d'autres types d'objets nommés (types de données, fonctions et opérateurs, par exemple). Le même nom d'objet peut être utilisé dans différents schémas sans conflit ; par exemple, schema1 et mon_schema peuvent tous les deux contenir une table nommée ma_table.

À la différence des bases de données, les schémas ne sont pas séparés de manière rigide : un utilisateur peut accéder aux objets de n'importe quel schéma de la base de données à laquelle il est connecté, sous réserve qu'il en ait le droit.

Il existe plusieurs raisons d'utiliser les schémas :

- autoriser de nombreux utilisateurs à utiliser une base de données sans interférer avec les autres ;
- organiser les objets de la base de données en groupes logiques afin de faciliter leur gestion ;
- les applications tiers peuvent être placées dans des schémas séparés pour éviter les collisions avec les noms d'autres objets.

Pour créer les objets d'un schéma ou y accéder, on écrit un nom qualifié constitué du nom du schéma et du nom de la table séparés par un point ; schema.table

exemple :

```
CREATE TABLE mon_schema.ma_table (...
```

Par défaut (si le nom du schéma n'est pas spécifié) les données sont créées dans le schéma *public*, mais il est de bonne pratique de mettre les données dans d'autres schémas car cela permet de séparer le cœur de PostGIS (tables systèmes, fonctions,...) des données elles-mêmes et facilite par exemple les sauvegardes / restaurations ainsi que la gestion des droits.

Complément : Chemin de parcours des schémas

L'écriture des noms qualifiés (MonSchema.MaTable) peut être contraignante dans les requêtes SQL. De ce fait, les tables sont souvent appelées par des **noms non-qualifiés**, soit le seul nom de la table. Le système détermine la table appelée en

34 - http://docs.postgresqlfr.org/12/app-initdb.html

suivant un **chemin de recherche**, liste de schémas dans lesquels chercher. La première table correspondante est considérée comme la table voulue. S'il n'y a pas de correspondance, une erreur est remontée, quand bien même il existerait des tables dont le nom correspond dans d'autres schémas de la base

Le premier schéma du chemin de recherche est appelé schéma courant. En plus d'être le premier schéma parcouru, il est aussi le schéma dans lequel les nouvelles tables sont créées si la commande **CREATE TABLE** ne précise pas de nom de schéma.

Le chemin de recherche courant est affiché à l'aide de la commande :

```
SHOW search_path;
```

Dans la configuration par défaut, ceci renvoie :

```
search_path
--------------
"$user",public
```

Le premier élément précise qu'un schéma de même nom que l'utilisateur courant est recherché. En l'absence d'un tel schéma, l'entrée est ignorée. Le deuxième élément renvoie au schéma public précédemment évoqué.

Par défaut, les utilisateurs ne peuvent pas accéder aux objets présents dans les schémas qui ne leur appartiennent pas. Pour le permettre, le propriétaire du schéma doit donner le droit **USAGE** sur le schéma

Par défaut, tout le monde bénéficie des droits CREATE et USAGE sur le schéma public. Ce privilège peut être révoqué :

```
REVOKE CREATE ON SCHEMA public FROM PUBLIC;
```

Le premier « public » est le schéma, le second «PUBLIC» signifie « tout utilisateur ». Dans le premier cas, c'est un identifiant, dans le second, un mot clé, d'où la casse différente.

Rappel : Convention de nommage

Une convention couramment utilisée revient à écrire les mots clés en majuscule et les noms en minuscule, exemple :

```
UPDATE ma_table SET a = 5;
```

Pour ajouter un schéma au chemin de recherche on écrit :

```
SET search_path TO mon_schema,public;
```

Si on omet *public*, le schéma public n'est plus accessible sans qualification explicite.

La modification des chemin de recherche par `SET search_path` n'est valide que pendant la session et sera perdue à la prochaine connexion.

Il est possible de modifier de façon permanente le chemin de recherche pour un utilisateur avec `ALTER USER`, Exemple :

```
ALTER USER alain SET search_path TO monschema, public ;
```

Conseil : Utilisation des schémas

Les schémas peuvent être utilisés avec différentes stratégies pour organiser les données

- si aucun schéma n'est créé, alors tous les utilisateurs ont implicitement accès au schéma public. Cette situation n'est recommandée que pour un usage de PostgreSQL avec un utilisateur unique (usage bureautique).
- pour chaque utilisateur, un schéma, de nom identique à celui de l'utilisateur,

peut être créé. Le chemin de recherche par défaut commence par $user, soit le nom de l'utilisateur. Si tous les utilisateurs disposent d'un schéma distinct, ils accèdent, par défaut, à leur propre schéma (et n'ont pas à utiliser les noms qualifiés). Dans cette configuration, il est possible de révoquer l'accès au schéma public (voire de supprimer ce schéma) pour confiner les utilisateurs dans leur propre schéma ;

- l'installation d'applications partagées (tables utilisables par tout le monde, fonctionnalités supplémentaires fournies par des applications tiers, etc) peut se faire dans des schémas distincts. Il faut alors accorder des privilèges appropriés pour permettre aux autres utilisateurs d'y accéder (nous y reviendrons). Les utilisateurs peuvent alors se référer à ces objets additionnels en qualifiant leur nom du nom de schéma ou ajouter les schémas supplémentaires dans leur chemin de recherche.

B. Tablespaces

Les 'tablespace' dans PostgreSQL™ permettent aux administrateurs de bases de données (ce paragraphe s'adresse donc en priorité à eux) de définir l'emplacement dans le système de fichiers où seront stockés les fichiers représentant les objets de la base de données. Une fois créé, un tablespace peut être référencé par son nom lors de la création d'objets.

En utilisant les tablespaces, un administrateur peut contrôler les emplacements sur le disque d'une installation PostgreSQL™. Ceci est utile dans au moins deux cas. Tout d'abord, si la partition ou le volume sur lequel le groupe a été initialisé arrive à court d'espace disque mais ne peut pas être étendu, un tablespace peut être créé sur une partition différente et utilisé jusqu'à ce que le système soit reconfiguré.

Deuxièmement, les tablespaces permettent à un administrateur d'utiliser sa connaissance des objets de la base pour optimiser les performances. Par exemple, un index qui est très utilisé peut être placé sur un disque très rapide et disponible, comme un périphérique mémoire. En même temps, une table stockant des données archivées et peu utilisée ou dont les performances ne portent pas à conséquence pourra être stockée sur un disque système plus lent, moins cher.

Exemple :

```
CREATE TABLESPACE espace_index OWNER alain LOCATION 'data/index' ;
```

OWNER alain indique que le propriétaire est alain. Il doit avoir les droits en écriture sur le répertoire 'data/index' (attention à utiliser / pour les noms de répertoires).

Une fois créé on pourra référencer le tablespace, par exemple lors de la création d'une table :

```
CREATE TABLE foo(i int) TABLESPACE montablespace;
```

Les commandes de modifications des objets, que nous verrons dans le cours, permettent de réaffecter a posteriori un objet à un tablespace particulier

```
ex : ALTER TABLE foo SET TABLESPACE nouveautablespace ;
```

Pour en savoir plus.[35]

Attention

Manipuler les Tablespaces est réservé aux administrateurs aguerris. Positionner un tablespace sur un système de fichier temporaire comme un disque RAM peut mettre en péril la fiabilité de l'instance entière.

35 - http://docs.postgresql.fr/9.4/manage-ag-tablespaces.html

C. Rappels sur les types de données

Pour les entiers le type integer est le plus utilisé.

Le type numeric peut stocker des nombres contenant un très grand nombre de chiffres et effectuer des calculs exacts. Il est spécialement recommandé pour stocker les montants financiers et autres quantités pour lesquelles l'exactitude est indispensable. Néanmoins, l'arithmétique sur les valeurs numeric est très lente comparée aux types entiers ou aux types à virgule flottante

Les types de données real et double precision sont des types numériques inexacts de précision variable. Tester une égalité entre 2 valeurs peut ne pas donner le résultat attendu.

Nom	Taille de stockage	Description	Étendue
smallint	2 octets	entier de faible étendue	de -32768 à +32767
integer	4 octets	entier habituel	de -2147483648 à +2147483647
bigint	8 octets	grand entier	de -9223372036854775808 à +9223372036854775807
decimal	variable	précision indiquée par l'utilisateur, valeur exacte	jusqu'à 131072 chiffres avant le point décimal ; jusqu'à 16383 après le point décimal
numeric	variable	précision indiquée par l'utilisateur, valeur exacte	jusqu'à 131072 chiffres avant le point décimal ; jusqu'à 16383 après le point décimal
real	4 octets	précision variable, valeur inexacte	précision de 6 décimales
double precision	8 octets	précision variable, valeur inexacte	précision de 15 décimales
smallserial	2 bytes	Entier sur 2 octets à incrémentation automatique	1 to 32767
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

types numériques

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

types caractères

On favorisera la varchar par rapport au char.

Les types de données sous PostgreSQL sont très complets. Voir la *documentation*[36].

D. PostGIS

PostGIS ajoute le support d'objets géographiques à la base de données PostgreSQL sous forme de géométries (points, lignes, polygones), conformément aux standards établis par l'Open Geospatial Consortium (OGC).

Il utilise de nombreuses bibliothèques du monde libre et en particulier gdal pour le raster, ogr pour le vecteur, proj.4 pour la gestion des projections et geos pour la géométrie.

36 - http://docs.postgresql.fr/12/datatype.html

PostGIS comporte beaucoup de fonctions :

- Fonctions de gestion (Postgis_version(), Postgis_geos_version(), Postgis_gdal_version(), ...)
- Fonctions de mesure (st_perimeter(geometry) , st_distance(geometry,geometry), ...)
- Fonctions d'extraction d'informations géométriques (st_astext(1) , st_assvg(1, integer, integer), st_asewkt(1) ...)
- Fonctions de constructeurs géométriques (st_centroid(geometry) , st_pointonsurface(geometry), ...)
- Fonctions d'éditeur géométriques
- ...

Quelques types de données géométriques de type vecteurs :

Elements	stockage PostGIS
ponctuels	POINT (44 798)
linéaires	LINESTRING(40 798, 72 808, 87 797, 122 812)
surfaciques	POLYGON((40 798, 72 808, 87 797, 122 812, 70 798))
ponctuels composés	MULTIPOINT(77 798, ...)
linéaires composés	MULTILINESTRING((40 798, 72 808, 87 797, 122 812),...)
Surfaciques composés	MULTIPOLYGON((40 798, 72 808, 87 797, 122 812, 70 798),...)
...	

Tableau 1 Les types de données vecteurs

Dans PostGIS les projections sont stockées dans la table *spatial_ref_sys* qui est une table standard de l'OGC.

Depuis la version 2.0 PostGIS gère également les données rasters.

Il permet d'utiliser des fonctions d'overlay (telles que ST_Intersects ou ST_Within) d'une manière transparente sans que l'utilisateur n'ait à se soucier si les couches en entrée sont en format vectoriel ou en format matriciel.

Les couches raster sont chargées en tant que données WKT (Well Know Text).

raster2pgsql est le pendant de shp2pgsql pour les données vecteurs de type SHP.

Nous n'aborderons pas plus le sujet du raster dans PostGIS. Le cas échéant on pourra consulter *ce site*[37] (en anglais).

37 - http://trac.osgeo.org/postgis/wiki/WKTRaster

IV

Exercice : Quels usages pour votre service ?

L'objectif de ce deuxième exercice est d'exposer sur la liste de discussion les attentes de votre service en matière de bases de données du type PostgreSQL/PostGIS

Question

Le fil de discussion « L2 : les attentes de votre service » a été créé sur la liste. Utilisez le bouton « répondre à la liste » pour faire part de vos réflexions.

Indiquez par exemple :

- si votre service utilise ou souhaite utiliser PostgreSQL/PostGIS ;
- pour quels types d'usage ;
- quelle organisation (acteurs et rôles ...) est envisagée.

N'hésitez pas à commenter les réflexions des autres apprenants ou à demander des éclaircissements.

Solution des exercices

> Solution n°1 *(exercice p. 18)*

○	Hiérarchique
○	orienté objet
○	XML/RDF
○	mixte
◉	objet-relationnel

PostgreSQL est un SGBD relationnel avec des extensions objet comme les classes, l'héritage, ...
En anglais *ORDBMS*[38].

> Solution n°2 *(exercice p. 19)*

○	Modèle Constitutif des Données
◉	Modèle Conceptuel des Données
○	Modèle Contrainte Dépendance
○	Modèle Coût Dépense
○	Modèle Commun de Données

38 - https://en.wikipedia.org/wiki/Object-relational_database

> Solution n°3 *(exercice p. 19)*

◉	une valeur de clé étrangère qui ne correspond à aucune clé primaire de la table référencée.
○	des doublons dans une clef primaire
○	des doublons dans une clef étrangère

> Solution n°4 *(exercice p. 20)*

☑	1 = un et un seul *1 est équivalent à 1..1*
☑	0 .. * = 0 à plusieurs *0 ou plus*
☑	0..1 = facultatif *la cardinalité indique de 0 à 1 donc au plus un.*
☑	1 .. * = obligatoire *au moins un*
☑	2 .. 4 = de deux à quatre

> Solution n°5 *(exercice p. 20)*

☑	Un enseignant fait ses cours dans une seule classe à au moins un étudiant
☑	Une salle accueille les cours pour un seul enseignant
☐	Un cours peut ne pas avoir d'étudiant
☑	Un étudiant suit des cours dans une seule salle avec un seul enseignant