Institut Notre-Dame Beauraing

UAA7

Base de données

Théorie et exercices

Technique de transition

–

Option Informatique

–

Titulaire du cours : L. Embrechts

Table des matières

1 Introduction

Une **Base de Données** (BD) est un ensemble de données structuré.

Les BD sont nées à la fin des années 1960 pour combler les lacunes des systèmes de fichiers et faciliter la gestion qualitative et quantitative des données informatiques.

En effet, un système de fichiers pose de nombreux problèmes :

- **Dépendance** : chaque application possède ses propres fichiers et leur exploitation ne peut se faire que par l'application qui les a créé
- **Redondance** : deux applications qui utilisent les mêmes informations stockent ces dernières dans deux fichiers différents
- **Incohérence** : le fait de dupliquer les données dans des fichiers différents entraîne un risque d'incohérence si la modification des données n'est pas répercutée dans tous les fichiers
- **Impossibilité d'interroger les données** : les données sont dispersées dans différents fichiers qui n'ont aucun lien entre eux
- **Pas de sécurité** : chaque utilisateur sauve ses propres fichiers avec une fréquence qui lui est propre
- **Structure non modifiable** : une fois un fichier créé, il est impossible de modifier la définition des types d'enregistrement
- **Pas de relation entre les fichiers** : il est difficile de mettre en relation les données de deux fichiers différents

Un **Service de Gestion de Base de Données** (SGBD) est un logiciel qui prend en charge la structuration, le stockage, la mise à jour et la maintenance d'une base de données.

On utilise des SGBDR (Services de Gestion de Base de Données Relationnelles) pour les implémenter. Le langage SQL est le langage commun à tous les SGBDR, ce qui permet de concevoir des BD relativement indépendamment des systèmes utilisés.

Les usages de BD se sont aujourd'hui généralisés pour entrer dans tous les secteurs de l'entreprise, depuis les petites bases utilisées par quelques personnes dans un service pour des besoins de gestion de données locales, jusqu'aux bases qui gèrent de façon centralisée des données partagées par tous les acteurs de l'entreprise.

Les conséquences de cette généralisation et de cette diversification des usages se retrouvent dans l'émergence de solutions conceptuelles et technologiques nouvelles, les bases de données du mouvement NoSQL particulièrement utilisées par les grands acteurs du web.

2 Méthodologie générale de conception de BD

2.1 Les quatre étapes

On distingue quatre étapes dans la conception d'une base de données :

1. **Analyse de la situation existante et des besoins (clarification).**

   Elle consiste à étudier le problème et à consigner dans un document, la note de clarification, les besoins, les choix, les contraintes.

2. **Création d'un modèle conceptuel qui permet de représenter tous les aspects importants du problème.**

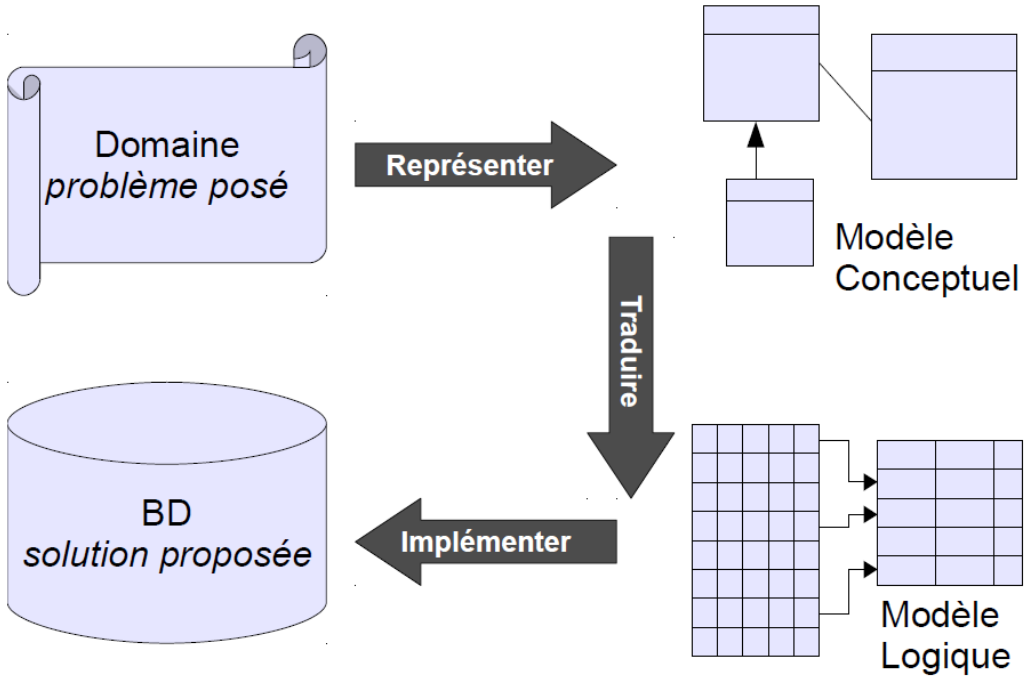
   Elle permet de décrire le problème posé, de façon non-formelle (généralement en graphique), en prenant des hypothèses de simplification. Ce n'est pas une description du réel, mais une représentation simplifiée d'une réalité.

3. **Traduction du modèle conceptuel en modèle logique (et normalisation de ce modèle logique).**

   Elle permet de décrire une solution, en prenant une orientation informatique générale (type de SGBD typiquement), formelle, mais indépendamment des choix d'implémentation spécifiques.

4. **Implémentation d'une base de données dans un SGBD, à partir du modèle logique (et optimisation)**

   Elle correspond aux choix techniques, en terme de SGBD choisi et à leur mise en œuvre (programmation, optimisation...).



2.2 L'importance de l'étape d'analyse

Si la première étape est fondamentale dans le processus de conception, elle est aussi la plus délicate. En effet, tandis que des formalismes puissants existent pour la modélisation conceptuelle puis pour la modélisation logique, la perception de l'existant et des besoins reste une étape qui repose essentiellement sur l'expertise d'analyse de l'ingénieur.

2.3 L'importance de l'étape de modélisation conceptuelle

Étant donnée une analyse des besoins correctement réalisée, la seconde étape consiste à la traduction selon un modèle conceptuel. Le modèle conceptuel étant formel, il va permettre de passer d'une spécification en langage naturel, et donc soumise à interprétation, à une spécification non ambiguë. Le recours aux formalismes de modélisation tels que l'E-A (Entités-Associations) ou l'UML (*Unified Modeling Language*) est donc une aide fondamentale pour parvenir à une représentation qui ne sera plus liée à l'interprétation du lecteur.

2.4 Les étapes de traduction logique et d'implémentation

Des logiciels spécialisés (par exemple Objecteering) sont capables à partir d'un modèle conceptuel d'appliquer des algorithmes de traduction qui permettent d'obtenir directement le modèle logique, puis les instructions pour la création de la base de données dans un langage de données tel que SQL.

L'existence de tels algorithmes de traduction montre que les étapes de traduction logique et d'implémentation sont moins complexes que les précédentes, car plus systématiques. Néanmoins ces étapes exigent tout de même des compétences techniques pour optimiser les modèles logiques (normalisation), puis les implémentations en fonction d'un contexte de mise en œuvre matériel, logiciel et humain.

3 Modélisation conceptuelle

3.1 Présentation du modèle entités-associations

Le modèle entité-association constitue la méthode de référence pour modéliser conceptuellement un domaine d'application. Contrairement aux approches ad-hoc, il offre un formalisme rigoureux et des règles précises.

3.2 Les entités

Une **entité** représente un objet concret ou abstrait ayant une existence propre et présentant un intérêt pour le système d'information.

3.2.1. Type d'entité vs. Occurrence d'entité

Il est crucial de bien distinguer :

- **Type d'entité** : la classe générale (ex: ÉTUDIANT)
- **Occurrence d'entité** : un élément particulier (ex: Marie Dupont, étudiante)

Exemples d'entités : PERSONNE, VÉHICULE, PRODUIT, BÂTIMENT, COMMANDE, COURS, PROJET, CONTRAT, RÉSERVATION, TRANSACTION, CONSULTATION,...

Contre-exemples : "Nom" ou "Âge" ne sont pas des entités mais des propriétés d'entités.

3.2.2. Règles d'identification des entités

1. L'entité doit avoir une **existence indépendante**
2. Elle doit posséder des **propriétés propres**
3. **Plusieurs occurrences** doivent exister ou être envisageables
4. Elle doit être **pertinente** pour le domaine étudié

3.3 Les attributs

Un **attribut** décrit une propriété élémentaire d'une entité ou d'une association.

Un attribut peut être :

- **atomique** : non décomposable
- **composé** : décomposable en sous-parties

Un attribut peut également être :

- **monovalué** : une seule valeur par occurrence (par défaut)
- **multivalué** : plusieurs valeurs possibles (noté avec `[0-N]`)

- **facultatif** : une seule valeur facultative par occurrence (noté avec `[0-1]`)

Voici un exemple pour chaque possibilité :

	Atomique	*Composé*
Monovalué	Numéro de registre national	Adresse résidence principale (numéro, rue, ville, pays)
Multivalué	Liste des langues parlées	Liste des permis obtenus (type et année d'obtention)
Facultatif	Numéro de téléphone	Adresse résidence secondaire (numéro, rue, ville, pays)

3.4 Les identifiants

Un **identifiant** est un ensemble minimal d'attributs permettant de distinguer de façon unique chaque occurrence d'une entité.

3.4.1. Propriétés

Voici les propriétés requises pour qu'un attribut puisse jouer le rôle d'identifiant :

1. **Unicité** : deux occurrences ne peuvent avoir la même valeur d'identifiant
2. **Minimalité** : aucun sous-ensemble de l'identifiant ne suffit à assurer l'unicité
3. **Stabilité** : la valeur ne doit pas changer au cours du temps
4. **Existence** : la valeur doit toujours être connue

3.4.2. Types d'identifiants

Un identifiant peut être :

- **naturel** : basé sur des propriétés naturelles (ex: numéro de registre national)
- **artificiel** : créé spécifiquement (ex: numéro de client auto-généré)
- **composé** : formé de plusieurs attributs (ex: numéro + bâtiment)

3.4.3. Exemples et contre-exemples

✅ Numéro d'immatriculation d'un véhicule

✅ Code ISBN d'un livre

✅ Couple (nom, prénom, date de naissance) pour une personne

❌ Nom seul d'une personne (pas unique)

❌ Âge (peut changer)

❌ Adresse email (peut changer, peut être inconnue)

3.5 Les associations

Une **association** modélise un lien perçu dans le réel entre deux ou plusieurs entités.

Une association n'a pas d'existence propre mais tire sa raison d'être des entités qu'elle relie.

3.5.1. Dimension des associations

Une association peut être :

- **binaire** : relie 2 entités (cas le plus fréquent)
- **ternaire** : relie 3 entités
- **n-aire** : relie n entités (rare en pratique)

Voici quelques exemples d'association binaire :

- ELEVE *est inscrit dans* ECOLE
- PROFESSEUR *donne* MATIERE
- PERSONNE *possède* ANIMAL

3.5.2. Associations réflexives

Une entité peut être associée à elle-même.

Voici quelques exemples :

- PERSONNE *est mariée avec* PERSONNE
- EMPLOYÉ *supervise* EMPLOYÉ
- COURS *est prérequis de* COURS

3.5.3. Associations porteuses de données

Une association peut posséder ses propres attributs, qui dépendent de toutes les entités participantes.

Voici quelques exemples :

- ÉTUDIANT *suit* COURS avec l'attribut NOTE OBTENUE
- EMPLOYÉ *travaille sur* PROJET avec les attributs DATE DE DÉBUT et DURÉE
- CLIENT *passe* COMMANDE avec l'attribut DATE COMMANDE

3.6 Les cardinalités

Les **cardinalités** expriment le nombre d'occurrences d'une entité pouvant participer aux occurrences d'une association.

3.6.1. Notation

Les cardinalités sont notées `(min,max)` où :

- `min` exprime le nombre minimum de participations
- `max` exprime le nombre maximum de participations

3.6.2. Valeurs courantes

Voici la liste des cardinalités souvent rencontrées :

- `(0,1)` : participation optionnelle, au plus une fois
- `(1,1)` : participation obligatoire, exactement une fois
- `(0,n)` : participation optionnelle, plusieurs fois possibles
- `(1,n)` : participation obligatoire, au moins une fois
- `(p,q)` : entre p et q participations

3.6.3. Détermination des cardinalités

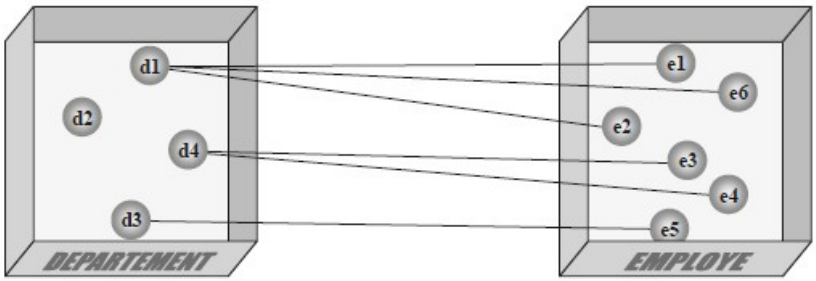
Pour déterminer la cardinalité d'une entité E par rapport à une association A :

1. Prendre une occurrence quelconque de E
2. Se demander : "Combien d'occurrences de A cette occurrence de E peut-elle avoir ?"
3. Déterminer le minimum et le maximum sur tous les cas possibles

3.7 Les types d'associations

3.7.1. Type d'associations un-à-plusieurs

Le schéma ci-dessous représente un ensemble de 4 départements d'une entreprise, symboliquement représentés par les pastilles étiquetées d1, d2, d3 et d4, ainsi qu'un ensemble de 6 employés, désignés par e1, e2, e3, e4, e5 et e6.



Entre ces ensembles existe une relation qui exprime qu'un département occupe des employés et qu'un employé est occupé par un département. Un arc tracé entre les entité d4 et e3 indique que le département d4 occupe l'employé e3.

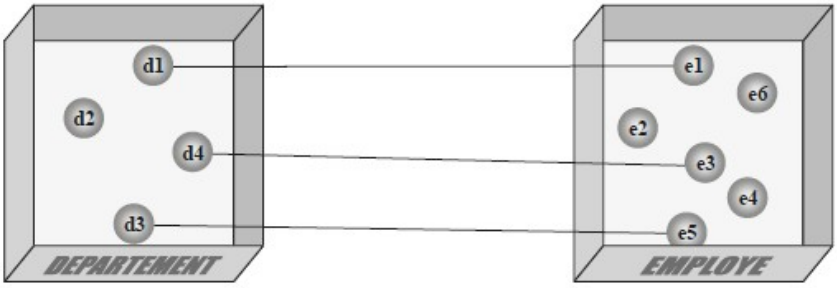
Il s'agit d'un type d'associations un-à-plusieurs.

On représente ce fait en indiquant, sur chacune des branches (ou rôles) du type d'associations, le nombre maximum d'associations dans lesquelles une entité peut apparaître, c'est-à-dire le nombre d'arcs issus de cette entité. On admet deux valeurs : un et plusieurs (qu'on notera respectivement 1 et N).

3.7.2. Type d'associations un-à-un

Le schéma ci-dessous représente les mêmes ensembles de départements et d'employés.

Entre ces ensembles existe une autre relation qui exprime qu'un département a un directeur qui est un employé et qu'un employé peut être directeur d'un département. Il s'agit d'un type d'associations un-à-un.

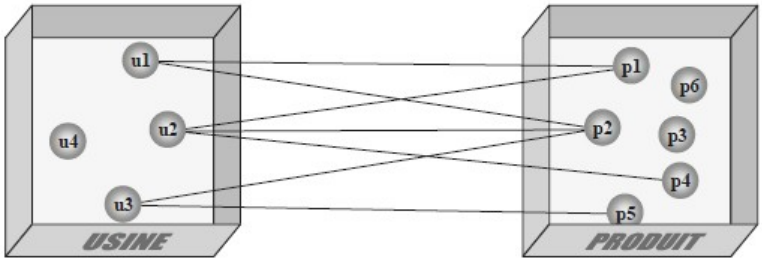


A chaque entité DEPARTEMENT correspond une seule entité EMPLOYE (dans le rôle de directeur), et à chaque entité EMPLOYE ne peut correspondre qu'une seule entité DEPARTEMENT.

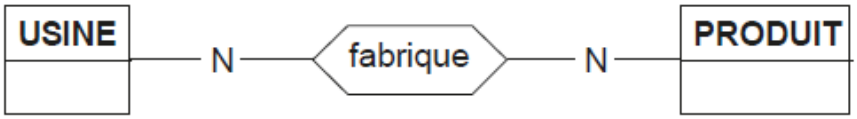
On indiquera le symbole 1 sur chacune des branches du type d'associations.

3.7.3. Type d'associations plusieurs-à-plusieurs

Le schéma ci-dessous décrit un ensemble d'usines et un ensemble de produits. Entre ces ensembles existe une relation qui exprime qu'une usine fabrique plusieurs produits et qu'un produit peut être fabriqué par plusieurs usines. Il s'agit d'un type d'associations plusieurs-à-plusieurs.



A chaque entité USINE peuvent correspondre plusieurs entités PRODUIT, et à chaque entité PRODUIT peuvent correspondre plusieurs entités USINE. On indiquera donc le symbole `N`, représentant le nombre plusieurs, sur chaque branche du type d'associations.

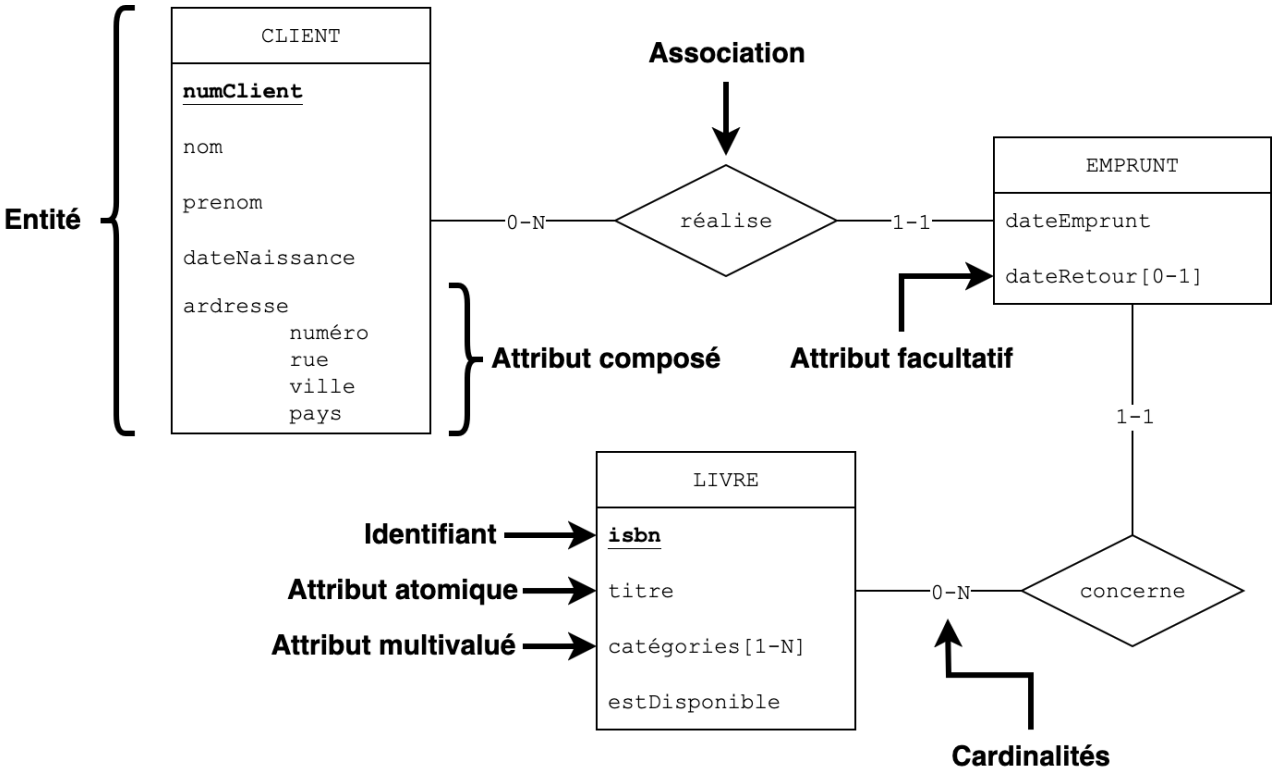


3.7.4.Type d’associations obligatoire ou facultatif

On peut imposer qu’un type d’associations soit obligatoire pour un type d’entités qui y participe. On exprime par là que tout employé est occupé par un département. De même qu’on a indiqué, par `1` ou `N`, le nombre maximum d’arcs associés, on indiquera par `0` ou `1`, le nombre minimum d’arcs issus de toute entité.

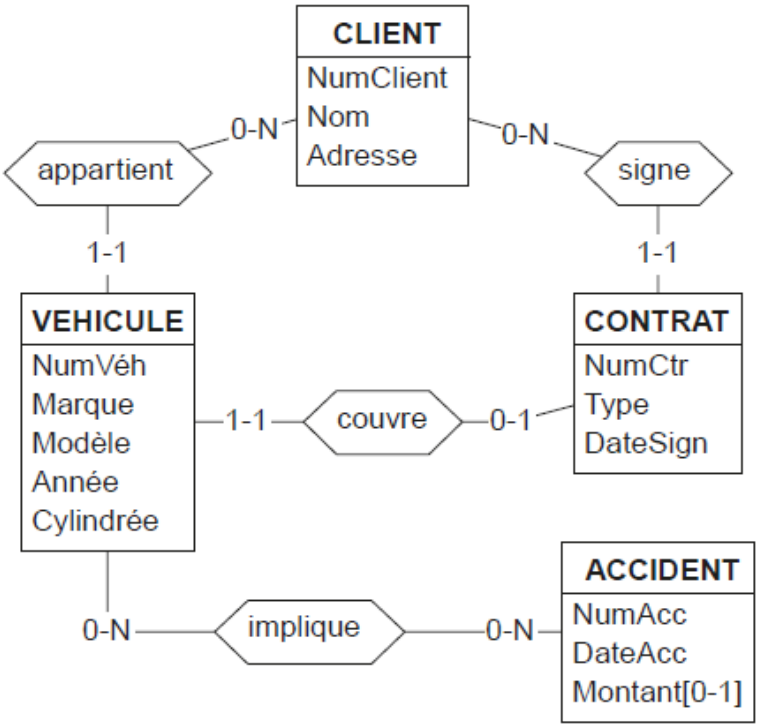
La figure suivante peut alors s’interpréter comme suit : toute entité DEPARTEMENT est associée, via occupe, à un nombre quelconque (de `0` à `N`) d’entités EMPLOYE, et toute entité EMPLOYE est associée, via occupe, à exactement une (de `1` à `1`) entité DEPARTEMENT.

3.8 Récapitulatifs des concepts vus et exemple



Exemple : contrat d'assurance

Un contrat est lié au client qui l'a signé; il existe donc une association entre ce contrat et ce client. On dira que toutes les associations de cette nature appartiennent au type d'associations signe entre les types d'entités CLIENT et CONTRAT. On définira également un type d'associations appartient entre CLIENT et VEHICULE, indiquant qu'un véhicule appartient à un client, ainsi qu'un type d'associations couvre entre CONTRAT et VEHICULE, indiquant qu'un contrat couvre les risques d'un véhicule. De même, en admettant qu'un accident implique un nombre quelconque de véhicules, on définira un type d'associations implique entre ACCIDENT et VEHICULE.



3.9 Exercices

3.9.1. Identification des entités

1. Un employé est caractérisé par un numéro d'employé (unique au sein de l'entreprise), un nom, une adresse, une date de naissance ainsi qu'un numéro de téléphone.

2. Un médecin généraliste s'identifie (auprès de l'INAMI) par un numéro. Il possède un nom ainsi qu'un numéro de téléphone. Un médecin est spécialisé dans un domaine particulier. Il ausculte régulièrement plusieurs patients ; ces derniers étant caractérisé par un nom et une date de naissance. Un patient possède également un numéro unique.

3. Un médicament est prescrit par un médecin pour un patient. Il se caractérise par un libellé et un prix. Le médecin renseigne toujours au patient la posologie préconisée par le fabricant.

3.9.2. Identification des associations

4. Un employé est caractérisé par un numéro d'employé (unique au sein de l'entreprise), un nom, une adresse, une date de naissance ainsi qu'un numéro de téléphone.

5. Un médecin généraliste s'identifie (auprès de l'INAMI) par un numéro. Il possède un nom ainsi qu'un numéro de téléphone. Un médecin est spécialisé dans un domaine particulier. Il ausculte régulièrement plusieurs patients ; ces derniers étant caractérisé par un nom et une date de naissance. Un patient possède également un numéro unique.

6. Un médicament est prescrit par un médecin pour un patient. Il se caractérise par un libellé et un prix. Le médecin renseigne toujours au patient la posologie préconisée par le fabricant.

7. À combien d'occurrences de CONSULTATION peut être reliée une occurrence de PATIENT ?

8. À combien d'occurrences de CONSULTATION peut être reliée une occurrence de MEDECIN ?

3.9.3.Identification des entités, des associations ainsi que des cardinalités

9. Un médecin donne plusieurs consultations ; la base de données doit gérer le cas des nouveaux médecins n'ayant encore jamais exercé. Un patient peut consulter autant de médecin qu'il le souhaite mais la base de données doit gérer le cas des patients n'ayant encore consulté aucun médecin.

10. Une école est dirigée par un employé (le directeur). Evidemment, tous les employés d'une école ne sont pas directeurs. Cependant, un employé peut être affecté à plusieurs écoles (c'est le cas notamment des enseignants) et une école peut compter plusieurs employés. Il arrive, lors de la création d'une nouvelle école, que l'on connaisse le directeur mais pas encore les employés qui y seront affectés.

11. Une usine fabrique plusieurs produits (en tout cas, au moins un). Un produit est généralement toujours fabriqué dans la même usine mais, dans le cas où la demande est forte, une autre usine de la même entreprise peut l'aider à augmenter la production. La base de données doit comprendre également certains produits naturels (ex. l'eau).

12. Un élève est inscrit dans une section (ex. informatique) qu'il doit obligatoirement choisir en début d'année scolaire. Une section peut compter plusieurs élèves. On ne souhaite pas conserver les éventuels changements d'option.

13. Même énoncé que le précédent mais on souhaite cette fois ci retenir l'ensemble des changements d'option

14. Une compagnie d'assurances demande à un de ses informaticiens de créer le schéma conceptuel (e-a) de sa base de données. Un client est domicilié dans une localité caractérisée par un libellé et un code postal. Lorsqu'un client signe un contrat, le numéro de contrat ainsi que la date de signature est retenue. La compagnie d'assurance propose plusieurs types de contrat, dont un permettant de couvrir un véhicule. Dans ce cas, le numéro de châssis du véhicule ainsi que sa date de fabrication est retenue. L'intérêt d'assurer sa voiture est d'être couvert en cas de dommages causés par un accident. Lorsqu'un tel évènement se produit, la date, le lieu ainsi que les témoins de l'accident sont conservés.

3.9.4. Identification des attributs facultatifs et booléens

15. Le touriste peut payer par carte visa ou par un autre moyen.

16. Le touriste peut avoir réservé son séjour à l'hôtel via une agence de voyage

17. On ne sait pas toujours si le touriste a réservé via une agence de voyage ou non

18. Un touriste peut venir avec son propre véhicule. Dans ce cas, un montant supplémentaire lui sera facturé pour la place de parking.

19. Un touriste peut venir avec son propre véhicule. Dans ce cas, le numéro de plaque de la voiture est mémorisé.

20. Le touriste peut être accompagné d'un animal domestique. En cas d'incendie dans le bâtiment, le propriétaire de l'hôtel doit être capable de connaître le nombre de touristes accompagnés d'un animal.

21. Le touriste peut être accompagné d'un animal domestique. Dans ce cas, on mémorise de quelle espèce il s'agit.

22. Le touriste peut être accompagné de plusieurs animaux domestiques. Dans ce cas, on mémorise la liste des espèces accompagnants le touriste.

3.9.5. Modélisation d’un schéma entités-associations

Ces exercices sont à réaliser sur l’outil DB-MAIN[1].

23. On te demande de concevoir une base de données pour la gestion administrative d'une école. Le système doit gérer les élèves, les professeurs, les matières, les classes et les notes. Un **élève** est caractérisé par son numéro d'élève, nom, prénom, date de naissance, adresse. Un **professeur** possède un numéro d'employé, nom, prénom, spécialité. Une **matière** a un code, un libellé, un coefficient. Une **classe** est identifiée par son nom, son niveau. Un élève appartient à une classe. Un professeur enseigne plusieurs matières. Un professeur peut enseigner à plusieurs classes. Une matière est enseignée dans plusieurs classes. Un élève a des notes dans différentes matières.

24. Un groupe de projet NF17 a réalisé un modèle pour constituer une base de données des consommations des clients d'un restaurant italien. Voici les informations extraites du document de synthèse fourni par les étudiants :
Une table est caractérisée par un numéro et possède toujours une capacité maximale théorique et un type. Un client, en réalité celui qui paye l'addition, possède un nom, un prénom et une date de naissance et peut avoir en plus une adresse courriel ou un numéro de téléphone (tous les deux facultatifs). Un ticket concerne un client, et chaque ticket possède un numéro unique et toujours renseigné, un nombre de couvert et le montant de l'addition réglé. Nous n'avons pas utilisé la date du ticket comme clé primaire car il est possible d'avoir la génération de deux tickets exactement au même moment. Un met est une entrée, un plat ou un dessert ; et il possède un prix.

25. Un aquarium souhaite gérer ses petites bêtes. Il dispose pour cela de plusieurs bassins, répartis dans plusieurs pièces. Des animaux de différentes espèces sont achetés, immatriculés, et disposent d'un suivi médical personnalisé - on garde donc la trace de la date et de la nature des soins dont ils bénéficient. Les animaux sont mélangés dans les bassins, et il arrive qu'on les déplace - là encore, on souhaite savoir à quelle date un animal donné a quitté tel bassin pour être placé dans tel autre. Les biologistes classent les animaux selon une arborescence à quatre niveaux. De plus général au particulier : ordre, famille, genre, espèce. Il va de soi que chaque animal de l'aquarium doit être correctement identifié dans cette arborescence.

26. Les patients d’un hôpital sont répartis dans les services (caractérisés chacun par un nom identifiant, sa localisation, sa spécialité) de ce dernier. A chaque patient peuvent être prescrits des remèdes. Un remède est identifié par son nom et caractérisé par son type, son fabricant et l’adresse de ce dernier. Chaque prescription d’un remède à un patient est faite par un médecin à une date donnée pour une durée déterminée. On ne peut rédiger plus d’une prescription d’un remède déterminé pour un même patient le même jour. Chaque patient est identifié par un numéro d’inscription. On en

1 https://www.db-main.eu

connaît le nom, l'adresse et la date de naissance. Chaque médecin appartient à un service. Il est identifié par son nom et son prénom. On sera particulièrement attentif à la notion de prescription.

27. La Défense belge vous contacte pour construire sa base donnée regroupant l'ensemble des espions qu'elle envoie sur le terrain.
Lorsqu'un agent secret est engagé par la Défense, il reçoit un matricule qui permet de l'identifier lors de ses missions. Son nom, son prénom ainsi que son numéro de téléphone doivent également être enregistrés. Il peut parfois être délicat d'envoyer des agents féminins dans certains pays (notamment au Moyen-Orient) : il faut donc pouvoir distinguer un agent masculin d'un agent féminin. De même, certaines missions plus physiques doivent être confiées à des agents plus jeunes.
La Défense doit pouvoir retrouver l'ensemble des langues que parle un agent. Chaque langue est identifiée par un code et caractérisée par un libellé en français, en néerlandais et en allemand. La Défense doit pouvoir également retrouver le pourcentage d'individus dans le monde qui parlent cette langue.
Un agent est toujours assigné à un et un seul Quartier Général (QG). Chaque QG est identifié par un nom et caractérisé par une adresse et un numéro de téléphone. La Défense doit pouvoir générer une liste de tous les agents affectés dans un QG particulier.
Le job d'un agent secret est de participer à des missions. Lorsque la Défense met en place une mission, cette dernière reçoit un code unique, une date de début, une durée estimée et une durée effective (lorsqu'elle est terminée) ainsi qu'une description. Il existe plusieurs types de mission : protection de témoin, exfiltration, renseignements, etc. Un type de mission et identifié par un code et caractérisée par un libellé. À tout moment, la Défense doit pouvoir extraire la liste des agents qui participent à une mission particulière.
Une mission peut se dérouler dans plusieurs localités. Il arrive que les lieux où se déroule la mission soit donnés lors du briefing de manière orale uniquement. Un code est associé à chacune des localités : deux localités différentes ne reçoivent jamais le même code. La Défense doit pouvoir retrouver le nom de la localité, son code postal ainsi que le nombre d'habitants.
La Défense souhaite pouvoir identifier les différents pays où se déroule la mission. Elle souhaite connaître son nom, son dirigeant, sa devise ainsi que le type de gouvernement en place (république, monarchie, dictature, etc.). De plus, elle souhaite retrouver les langues officielles de chaque pays.
Une fois envoyé sur le terrain, un agent secret peut contacter des fixeurs en place dans certains pays qui l'aideront à mener à bien la mission. Pour prendre contact avec un fixeur, l'agent a besoin de connaître le matricule (unique), le pseudonyme ainsi que le numéro de téléphone du fixeur en question. Un fixeur est disponible dans certaines localités bien précises : il refusera d'aider l'agent si ce dernier est en dehors de son champ d'action. Il arrive donc très souvent qu'un agent doive se débrouiller seul si aucun fixeur n'est disponible dans la localité où se déroule la mission. Afin

d'anticiper une telle mésaventure, l'agent a besoin d'obtenir, par localité où se déroule sa mission, la liste des pseudonymes et des numéros de téléphones de chaque fixeur.

28. Dans cet exercice, on vous demande de constituer une base de données de Bdphiles.
Un album de BD est identifié par un code ISBN et caractérisé par un titre et une année de parution. Un album fait partie d'une série : chaque série possède un nom (ex. *Les Aventures de Tintin*) ainsi qu'une catégorie (ex. *Aventure*). On doit pouvoir retrouver le prénom et le nom du dessinateur, le prénom et le nom du scénariste ainsi que le nombre d'albums que compte la série (ex. la série *Les Aventures de Tintin* compte 24 albums).
Une série de BD est éditée par une ou plusieurs maison(s) d'édition. Pour chaque série, on doit pouvoir retrouver le nom, l'adresse et le numéro de téléphone de chaque maison d'édition.
Un vrai BDphile connaît parfaitement les personnages principaux de chaque série. Ainsi, on souhaite pouvoir retrouver, par personnage, son nom (ex. *Milou*), son type (ex. *Animal*), une description de ses traits de caractères ainsi que le ou les albums dans lesquels il apparaît. On part du principe que deux personnages ne portent pas le même nom.
Un album peut être traduit dans plusieurs langues. Une langue est identifiée par un code. On doit pouvoir retrouver le libellé en anglais correspondant à un code de langue donné.
Les BDphiles peuvent s'enregistrer dans la base de données afin d'acheter ou de mettre en vente des albums. Lorsqu'un BDphile s'enregistre, il reçoit un numéro unique et spécifie son nom, son pseudo, son adresse mail ainsi que la liste des séries qu'il préfère (afin que le système puisse lui faire des recommandations lorsqu'un nouvel album est mis en vente).
Les BDphiles peuvent vendre des albums qu'ils possèdent à d'autres BDphiles. Lorsqu'une vente est conclue, on doit pouvoir retrouver les caractéristiques du propriétaire ainsi que les caractéristiques de l'album vendu.
La base de données doit permettre la production des listings suivants :

    a. le nom des séries qui contiennent au moins un personnage non-humain ;
    b. le nom de la maison d'édition qui a édité la série qui compte le plus d'albums ;
    c. la catégorie de la série la plus appréciée ;
    d. les titres des albums qui ont été traduits dans une langue donnée ;
    e. l'année de parution et le nombre de pages de chaque album mis en vente

29. On vous demande de créer la base de données permettant la gestion d'une bibliothèque.
Chaque livre est un exemplaire d'un ouvrage. A un ouvrage peut correspondre plusieurs exemplaires.
Un ouvrage est caractérisé par une référence unique, un titre, et, éventuellement, une liste de mots-clés (max 10 mots-clés).
Un ouvrage doit pouvoir être associé à son ou ses auteur(s). Un répertoire des auteurs est prévu. Un auteur ne sera enregistré dans la base de données que s'il a écrit au moins un ouvrage repris dans la bibliothèque. Pour chaque auteur, on mémorise le nom, le prénom, la date de naissance, la nationalité et la date de décès (si applicable).
Chaque exemplaire d'un ouvrage porte un numéro unique et a une localisation déterminée dans les rayonnages. Il y a 10 rayonnages numérotés de A à J. Actuellement, chaque rayonnage est constitué de 8 étagères numérotées de 1 à 8, chacune divisée en 4 planches. La localisation d'un exemplaire est donc la combinaison d'un numéro de rayonnage, d'un numéro d'étagère et d'un numéro de planche. On mémorise également pour chacun des exemplaires sa date d'achat et son état, à savoir, "disponible", "en réparation" ou "disparu". En effet, un exemplaire d'ouvrage peut être momentanément indisponible car en "réparation". De plus, on gardera trace dans la B.D des exemplaires éventuellement disparus (perte, vol, oubli...).

On doit pouvoir retrouver l'éditeur, l'année d'édition ainsi que le lieu d'édition de n'importe quel exemplaire.
Les exemplaires peuvent être empruntés. Tout emprunteur a un numéro qui l'identifie. Il est caractérisé en outre, par son nom, ses trois premiers prénoms (s'il en possède au moins trois), son adresse, son numéro de téléphone et sa date de naissance.

Il y a quatre types d'emprunteurs : extérieur, étudiant, assistant, professeur. Tout emprunteur ne peut être que d'un seul type à la fois. Les différents types d'emprunteur payent des cautions différentes. La durée maximum d'emprunt d'un exemplaire est de 1 semaine pour les extérieurs, 15 jours pour les étudiants et assistants, et 3 semaines pour les professeurs. Les emprunteurs ne peuvent emprunter qu'au maximum trois livres à la fois. Cette limite est portée à cinq pour les professeurs.

On mémorise dans la base de données uniquement les emprunts en cours. Par conséquent, dès qu'un exemplaire emprunté est rentré, l'emprunt est effacé de la base de données. Un emprunteur peut emprunter plusieurs exemplaires à la fois. On doit pouvoir envoyer une lettre de rappel aux emprunteurs en retard de plus d'une semaine. Il faut donc mémoriser la date d'emprunt.

30. On vous demande de concevoir une base de données pour la gestion d'une entreprise et notamment, le calcul des salaires des employés.
La société occupe des employés et est organisée en départements.

Un employé est identifié par un numéro unique au sein de la société. Il est également caractérisé par son numéro d'inscription dans le registre national, son nom, une liste de prénoms (4 max) dont le premier est obligatoire, sa date de naissance, son âge, l'adresse de son domicile, son sexe, le numéro du département dans lequel il travaille et son salaire. Dans le salaire, on distingue le "forfait" et les "frais de déplacement". Les frais de déplacement sont fonction des kilomètres qui séparent la ville du domicile de l'employé et la ville du département où il travaille. Le forfait est lui-même scindé en "fixe" et "bonus". Pour des facilités de comptabilité, une fois calculé, le dernier salaire de chaque employé est mémorisé dans la base de données. Le "forfait" du salaire est fonction de l'ancienneté de l'employé.

Un département est caractérisé par un numéro qui est unique au sein de l'entreprise. De plus, tout département a un nom qui est également identifiant. Comme les départements sont dispersés sur plusieurs sites (un seul site par département), l'adresse de chaque département est mémorisée. Un même département peut être composé de plusieurs services dont on ne retient que le code identifiant. Le département qui compte le plus de services est le département fabrication : il en compte 4.
Un département se voit affecter un ou plusieurs employés, alors qu'un employé ne peut travailler que dans un seul département. Certains employés sont "volants", càd qu'ils ne sont affectés à aucun département.

Un département peut contrôler des projets sur lesquels travaillent des employés. Un employé est affecté à un seul département mais peut collaborer à plusieurs projets. Par conséquent, un employé peut travailler sur un projet qui dépend d'un département autre que celui auquel il est affecté (ex : Dupont est affecté au département n°3 et travaille sur le projet 55 qui est contrôlé par le département n°7). Tout employé ne coopère pas forcément à un projet. Un projet a un nom et un budget. Deux projets ne portent pas le même nom.

Comme un employé peut partager son temps de travail entre différents projets, il est nécessaire de tenir compte de la répartition de sa charge. On mémorise donc, par employé, le nombre d'heures affectées à chaque projet auquel il participe.
Tout département est dirigé par un directeur unique, le directeur étant également un employé. On grade trace de la date de début de la fonction de chaque directeur.

Chaque employé, hormis le président directeur général, est supervisé par un autre employé. Tout employé n'est pas pour autant superviseur de quelqu'un.
Un des avantages de la société est de fournir gratuitement une assurance à l'employé et sa famille. Dans ce but, il faut connaître les enfants à charge des employés. Légalement, un enfant ne peut être déclaré personne à charge que d'un seul employé, et ce, même si les deux parents travaillent dans la même entreprise. Tout enfant à charge a un prénom, un sexe, et une date de naissance. On considère que deux enfants à la charge du même employé ne peuvent porter le même prénom.

4 Modélisation logique

4.1 Présentation du modèle entités-relations

Le modèle entité-association, bien qu'excellent pour la conception conceptuelle, présente une limite fondamentale : **il ne peut pas être directement implémenté dans un SGBD relationnel**.

En effet, les systèmes de gestion de bases de données comme MySQL, PostgreSQL, Oracle ou SQL Server ne comprennent que le langage des tables et des relations - ils ne savent pas interpréter directement les associations, cardinalités et autres concepts du modèle E-A.

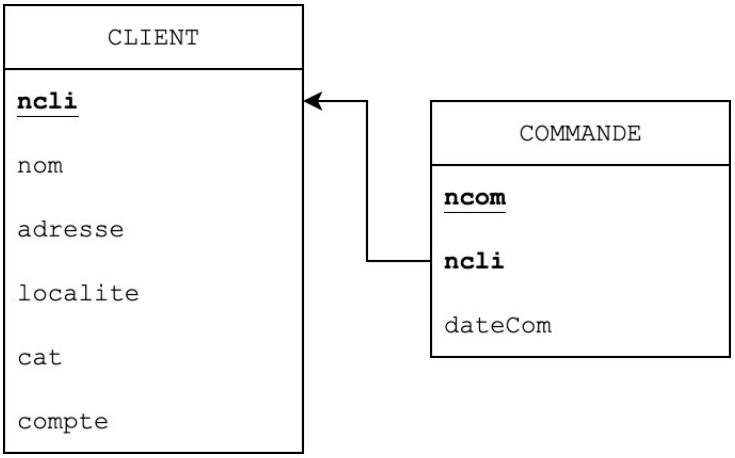
Il est donc nécessaire de transformer tout schéma E-A en un schéma relationnel avant de pouvoir le mettre en œuvre dans un SGBD. Cette transformation implique de représenter chaque entité par une *table*, chaque attribut par une *colonne*, et chaque association par une table supplémentaire ou des *clés étrangères* selon la nature de la relation. Les contraintes de cardinalité et d'intégrité doivent également être traduites à l'aide de *clés primaires*, de clés étrangères et de règles spécifiques dans le système relationnel.

Ainsi, le passage du modèle entité-association au modèle relationnel est une étape incontournable pour garantir que la structure des données imaginée lors de la phase conceptuelle soit correctement véhiculée et respectée dans la base de données effective.

Le **schéma relationnel** décrit formellement la structure d'une base de données. Il s'agit de la « fiche technique » complète de chaque table.

Composants d'un schéma de relation :

- Le nom de la table
- La liste des attributs (colonnes) avec leurs types
- La clé primaire (en gras soulignée)
- Les clés étrangères (en gras et notées avec →)



4.2 La table

Une **table** est la structure de base du modèle relationnel. Elle représente un ensemble d'informations organisées sous forme de tableau à deux dimensions composé de lignes et de colonnes.

Le modèle E-A utilise des "entités" et des "associations" (concepts abstraits) alors que le modèle relationnel ne connaît qu'un seul type de structure : la table.

Exemple : l'entité ÉTUDIANT du modèle E-A devient la table ÉTUDIANT dans le modèle relationnel.

4.3 La colonne

Une **colonne** représente un attribut dans le contexte relationnel. Chaque colonne a un nom unique dans la table et stocke des valeurs d'un type spécifique.

Alors que dans le modèle E-A, un attribut est une propriété d'une entité, dans le modèle relationnel, une colonne est un champ structuré de données dans une table.

Exemple : l'attribut NOM de l'entité ÉTUDIANT devient la colonne NOM de la table ÉTUDIANT.

4.4 La ligne

Une **ligne** représente une occurrence concrète des données.

C'est l'équivalent d'une occurrence d'entité dans le modèle E-A, mais organisée sous forme tabulaire.

Propriétés importantes :

- chaque ligne doit être unique dans la table (pas de doublons)
- l'ordre des lignes n'est pas significatif
- chaque ligne contient une valeur pour chaque colonne (ou NULL si la valeur est inconnue)

Exemple : client d'une banque

Le tableau suivant représente une table comportant six lignes décrivant chacune un client. On trouve dans chaque ligne quatre valeurs représentant respectivement le nom et l'adresse du client, la localité où il réside ainsi que l'état de son compte. L'une de ces lignes représente le fait suivant :

Il existe un client de nom « Avron », résidant 8,chaussée de la Cure à Toulouse et dont le compte est débiteur d'un montant de 1700.

Toutes les lignes d'une table ont le même format. Cette propriété signifie que dans la table client , toutes les lignes sont constituées d'une valeur de nom d'une valeur d'adresse,n d'une valeur de localité ainsi que d'une valeur de compte. L'ordre de lignes est indifférent.

L'ensemble des valeurs de même type correspondant à une même propriété des entités décrites s'appelle une colonne de la table. Une colonne est définie par son nom, le type de ses valeurs et leur longueur.

CLIENT			
nom	**adresse**	**localite**	**compte**
HANSENNE	23, avenue Dumont	Poitier	1250
MERCIER	25, rue le Maître	Namur	-3200
TOUSSAINT	14, rue de l'été	Poitier	0
VANBIST	40, rue Bransart	Lille	6800
AVRON	8, Chaussée de la Cure	Toulouse	-1700
NEUMAN	5, rue Godefroid	Poitier	60

4.5 La clé primaire

Une **clé primaire** est un attribut (ou un ensemble d'attributs) qui identifie de façon unique chaque ligne de la table.

C'est l'équivalent de l'identifiant dans le modèle E-A, mais avec des contraintes techniques plus strictes.

Parmi les identifiants d'une table, l'un est choisi comme le plus représentatif et sera dès lors déclaré identifiant primaire. Les autres sont alors des identifiants secondaires de la table. En règle générale, toute table possède un identifiant primaire et un nombre quelconque (éventuellement nul) d'identifiants secondaires.

Propriétés obligatoires :

- **Unicité** : aucune valeur ne peut se répéter
- **Non-nullité** : aucune valeur ne peut être `NULL`
- **Minimalité** : aucun attribut de la clé ne peut être retiré sans perdre l'unicité
- **Stabilité** : la valeur ne doit pas changer au cours du temps

4.6 La clé étrangère

Une **clé étrangère** est un attribut qui permet d'établir des liens physiques entre les tables en référençant la clé primaire d'une autre table.

La clé étrangère est un concept totalement nouveau qui n'existe pas dans le modèle E-A.

Caractéristiques :

- Elle implémente physiquement les associations du modèle E-A
- Elle garantit l'intégrité référentielle automatiquement
- Elle peut contenir des valeurs dupliquées (contrairement à la clé primaire)
- Sa valeur doit exister dans la table référencée ou être `NULL`

Exemple : Si COMMANDE fait référence à CLIENT, alors COMMANDE aura une clé étrangère ID_CLIENT qui référence la clé primaire ID de la table CLIENT.

4.7 La contrainte d'intégrité

Une **contrainte d'intégrité** est une règle automatiquement vérifiées par le SGBD pour garantir la cohérence des données.

Les contraintes d'intégrité n'existent pas explicitement dans le modèle E-A.

4.7.1.L'intégrité de domaine

Le **domaine** définit l'ensemble des valeurs possibles que peut prendre un attribut.

Autrement dit, le domaine définit le type et la nature des données que l'attribut peut contenir. C'est un concept plus précis et contraignant que dans le modèle E-A.

Il existe différents types de domaine :

- Domaines **numériques** :entiers, réels, avec contraintes
- Domaines **textuels** : chaînes de caractères, avec ou sans contraintes de format
- Domaines **temporels** : dates, heures, avec contraintes de cohérence
- Domaines **booléens** : vrai/faux

Voici quelques exemples :

Attribut	*Type*	*Contrainte*
`âge`	Entier	`âge ≥ 0 et âge ≤ 150`
`note`	Réel	`note ≥ 0.0 et note ≤ 20.0`
`nom`	Chaîne de caractères	longueur max de 50 caractères
`dateNaissance`	Date	comprise entre, "1900-01-01" et "2025-12-31"
`estPrésent`	Booléen	n.a
`catégorie`	Chaine de caractères	Uniquement « Haut », « Bas » ou « Accessoire »

L'**intégrité de domaine** garantit que chaque valeur stockée dans une colonne respecte strictement le domaine définis pour cette colonne.

4.7.2.L'intégrité d'entité

L'**intégrité d'entité** impose à chaque table d'avoir une clé primaire unique et non nulle permettant d'identifier sans ambiguïté chaque ligne.

Unicité **:** chaque valeur de la colonne (ou du groupe de colonnes formant la clé primaire) doit être différente dans toute la table. Il ne peut pas y avoir deux personnes avec le même numéro d'identifiant.

Non-nullité : La clé primaire ne peut jamais avoir la valeur `NULL`. Elle doit toujours être définie pour chaque enregistrement.

4.7.3. L'intégrité référentielle

L'**intégrité référentielle** assure la cohérence des liens entre les tables via les clés étrangères.

Toute valeur prise par une clé étrangère dans une table doit exister comme clé primaire dans la table référencée, ou bien être `NULL` (si la relation est optionnelle).

Principe : Une clé étrangère relie une table « enfant » (par exemple, `Commande`) à une table « parent » (par exemple, `Client`). On ne peut pas enregistrer une commande pour un client qui n'existe pas dans la table des clients.

Effets :

- Insertion: impossible d'ajouter une valeur de clé étrangère qui ne correspond pas à une clé primaire existante dans la table cible.
- Suppression/Modification : impossible de supprimer/modifier une clé primaire référencée tant que des enregistrements dans la table fille existent (sauf exceptions).

4.8 Règles de transformation

La transformation du modèle conceptuel vers le modèle logique suit des règles systématiques et automatisables :

4.8.1. Transformation des entités

- Chaque entité devient une relation (table)
- Les attributs de l'entité deviennent les colonnes de la table
- L'identifiant de l'entité devient la clé primaire

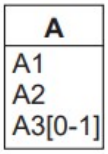
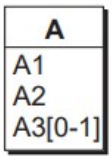
4.8.2. Associations `(1,N)`

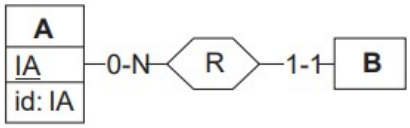
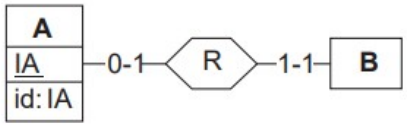
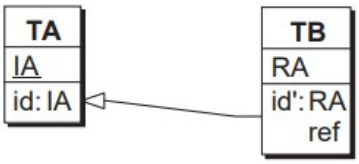
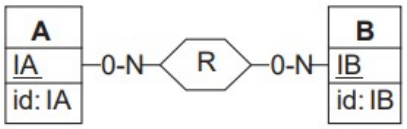
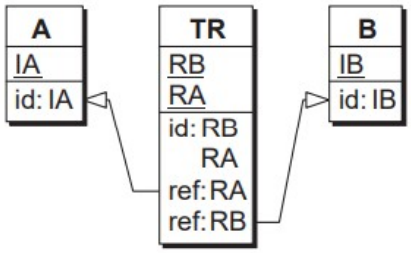
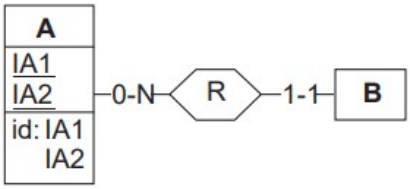
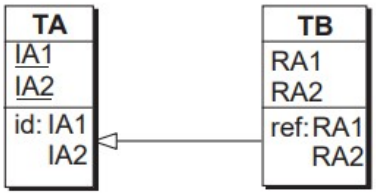
- Pas de nouvelle table pour l'association
- La clé primaire du côté "1" devient clé étrangère du côté "N"
- Les attributs de l'association rejoignent la relation du côté "N"

4.8.3. Associations `(N,N)`

- Création d'une nouvelle relation pour l'association
- Les clés primaires des entités participantes deviennent clés étrangères
- La clé primaire de la nouvelle relation est la concaténation des clés étrangères

Les tableaux ci-dessous constituent un récapitulatif des principales règles de traduction d'un schéma conceptuel en structures de tables.

Types d'entités et attibuts		
		

Types d'associations		
1 à N		
1 à 1		
N à N		
Id multi-composant		

4.8.4.Exercices

Transforme chaque schéma entités-associations en schéma entités-relations **sans utiliser DB-MAIN**.

Schéma 1 : série

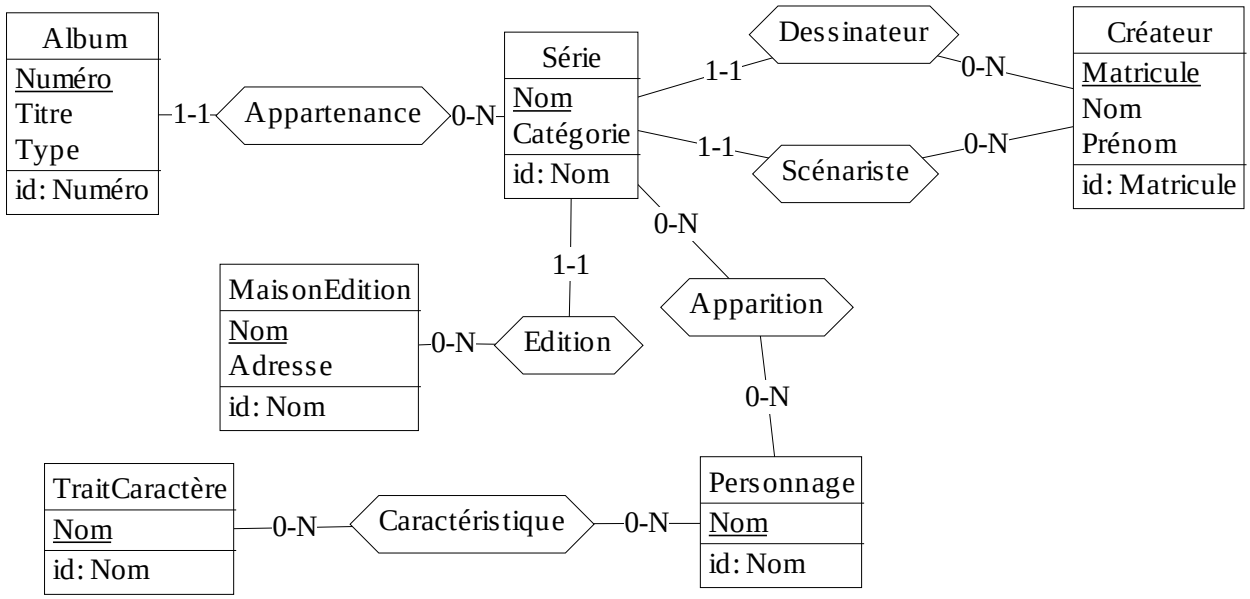


Schéma 2 : entreprise

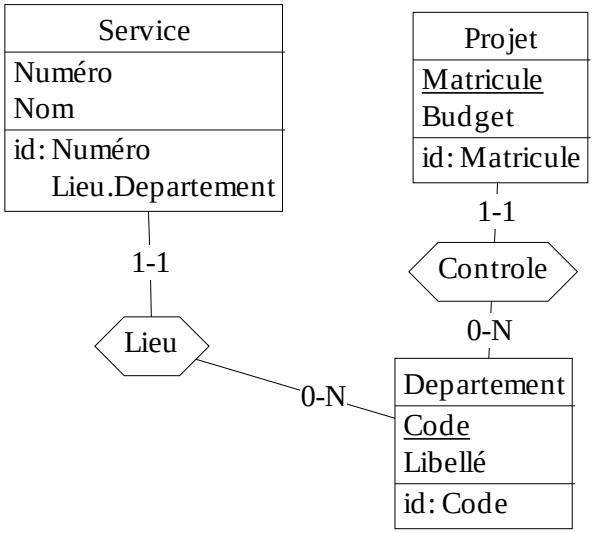


Schéma 3 : théâtre

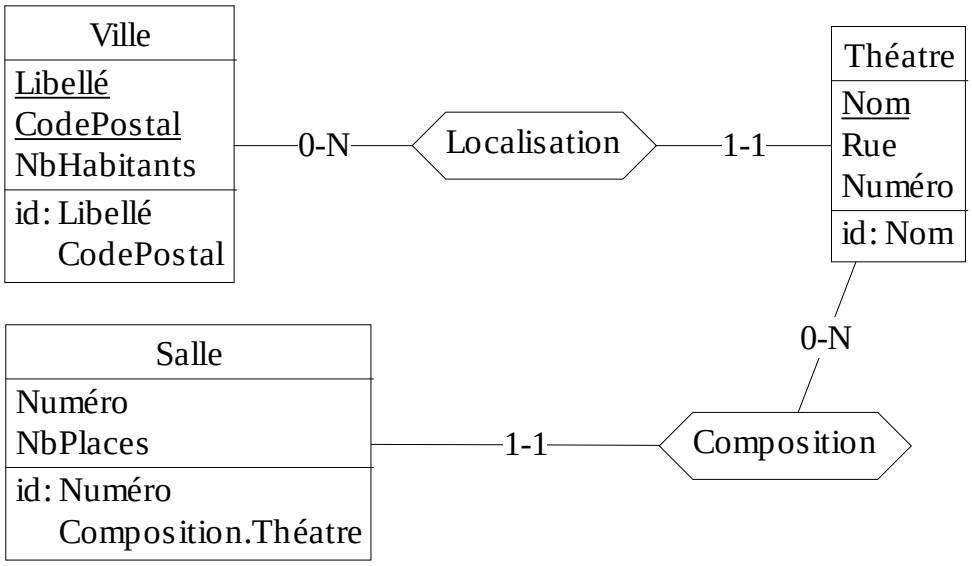
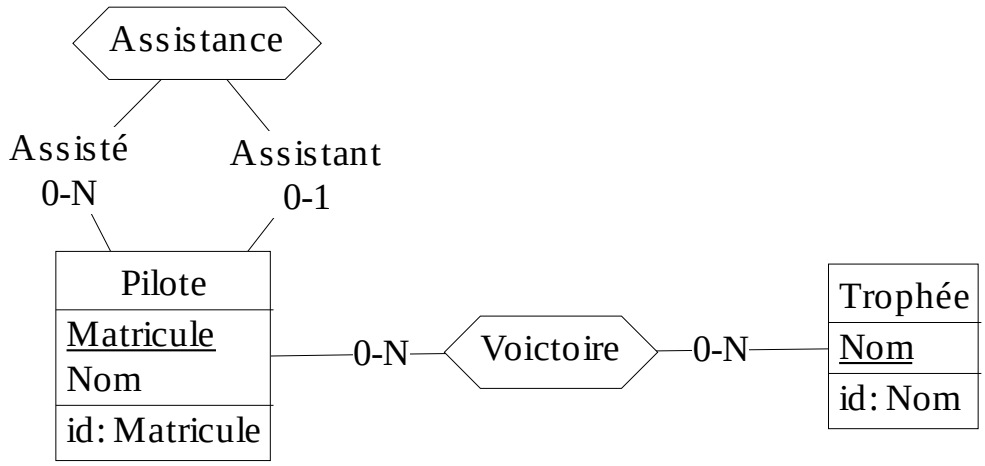


Schéma 4 : concours



5 Le langage SQL

Un **langage de données** est un langage informatique permettant de décrire et de manipuler les schémas et les données d'une base de données.

SQL est le langage de données consacré aux SGBD relationnels et relationnels-objet. Il permet de :

- créer des tables, en définissant le domaine de chaque colonne (DDL) ;
- insérer des lignes dans les tables (DML) ;
- lire les données entrées dans la base de données (Extraction).

5.1 Data Definition Language (DDL)

Le **DDL**, ou langage de définition des données, permet de créer, modifier ou supprimer les structures qui hébergent les données.

C'est grâce au DDL que l'on définit l'organisation et la structure des bases de données, en fixant les types de données, les contraintes d'intégrité et les relations entre les tables. Il s'agit d'une étape fondamentale pour assurer la cohérence et la robustesse du système d'information.

5.1.1. Création d'un schéma

Une base de données est est définie par son schéma. SQL propose donc de créer ce schéma (nouvelle DB) avant de définir ses différentes tables :

```
CREATE SCHEMA CLICOM;
```

Cette instruction sera souvent accompagnée de divers paramètres spécifiant notamment les conditions d'autorisation d'accès.

5.1.2. Création d'une table

Cette opération produit une table vide (sans lignes) dont le schéma est conforme aux indications données par la requête **CREATE TABLE**. On y spécifie le nom de la table et la description de ses colonnes. On spécifiera en outre pour chaque colonne son nom et le type de ses valeurs.

```
CREATE TABLE CLIENT (ncli       CHAR(10),
                     nom        CHAR(32),
                     adresse    CHAR(60),
                     localite   CHAR(30),
                     cat        CHAR(2),
                     compte     DECIMAL(9,2));
```

Les colonnes et leurs types

SQL offre divers types de données dans la déclaration d'une colonne. On citera les principaux :

`SMALLINT`	Entiers signés courts (par ex. 16 bits)
`INTEGER` `INT`	Entiers signés longs (par ex. 32 bits)
`NUMERIC(p,q)`	Nombre décimaux de p chiffres dont q après le point décimal ; si elle n'est pas mentionnée, la valeur de q est 0
`DECIMAL(p,q)`	Nombre décimaux d'au moins p chiffres dont q après le point décimal ; si elle n'est pas mentionnée, la valeur de q est 0
`FLOAT` `FLOAT(p)`	Nombre en virgule flottante d'au moins p bits significatifs
`CHAR` `CHARACTER(p)`	Chaîne de longueur fixe de p caractères
`CHARACTER VARYING` `VARCHAR(p)`	Chaîne de longueur variable d'au plus p caractères
`BIT`	0 ou 1
`DATE`	Date (année, moi et jour)
`TIME`	Instant (heure, minute, seconde, éventuellement 1000^{e} de seconde)
`TIMESTAMP`	Date + temps
`INTERVAL`	Intervalle en années/mois/jours entre dates ou en heures/minutes/secondes entre instants

Chaque SGBD ajoutera de sa propre initiative d'autres types de données. Par exemples les *Binary Large Objects* (`BLOB`), sorte de contenants génériques pouvant accueillir des chaînes

de bits de longueur illimitée telles que des images, séquences vidéo, séquences musicales, etc.

Les identifiants

On complétera la déclaration de la table par la clause **PRIMARY KEY** :

```
CREATE TABLE CLIENT (ncli       CHAR(10),
                     nom        CHAR(32),
                     adresse    CHAR(60),
                     localite   CHAR(30),
                     cat        CHAR(2),
                     compte     DECIMAL(9,2),
                     PRIMARY KEY (ncli));

CREATE TABLE DETAIL (ncom  CHAR(12),
                     npro   CHAR(15),
                     qcom  DECIMAL(8),
                     PRIMARY KEY (NCOM, NPRO));
```

Les autres identifiants seront déclarés via le prédicat **UNIQUE** :

```
CREATE TABLE ASSURE (
                num_affil  CHAR(10),
                num_ident  CHAR(15),
                nombreux   CHAR(35),
                PRIMARY KEY(num_affil),
                UNIQUE (num_ident));
```

Les clés étrangères

On déclarera la clé étrangère et la table référencée par une clause **FOREIGN KEY** :

```
CREATE TABLE DETAIL (     ncom CHAR(12),
                          npro CHAR(15),
                          qcom DECIMAL(8),
                          PRIMARY KEY (ncom, npro),
                          FOREIGN KEY(ncom) REFERENCES commande (ncom),
                          FOREIGN KEY(npro) REFERENCES produit (npro));
```

Remarque : dans cet exemple la clé primaire est multi-colonnes !

Par défaut, l'identifiant visé par la clé étrangère dans la table cible est l'identifiant primaire de celle-ci. Il est possible, mais ceci est déconseillé en toute généralité, de définir une clé étrangère visant un identifiant secondaire de la table cible.

La clé étrangère comporte autant de colonnes que l'identifiant cible et ces colonnes sont de même type, prises deux à deux.

Sans spécifier plus loin le sujet, il est possible de créer des clés étrangères dont une des colonnes est facultative. Cette option nécessite un traitement particulier, nous ne développerons pas ce point dans le cours.

Caractère obligatoire/facultatif d'une colonne

Attention (à l'inverse des schéma entités-association) ! Par défaut (si on ne spécifie rien) toute colonne est facultative.

Le caractère obligatoire d'une colonne se déclarera par la clause **`NOT NULL`** :

```
CREATE TABLE OFFRE ( numfl CHAR(10) NOT NULL,
                     numpl CHAR(15) NOT NULL,
                     prix  DECIMAL(8),
                     PRIMARY KEY(numfl, numpl),
                     FOREIGN KEY(numfl) REFERENCES fournisseur (numfl));
```

Forme synthétique des contraintes

Les contraintes impliquant une seule colonne peuvent être considérées comme des contraintes de colonnes (identifiants primaires et secondaires, clé étrangères). Elles pourront être alors déclarées comme complément de la définition de cette colonne :

```
CREATE TABLE COMMANDE (
                ncom CHAR(12) PRIMARY KEY,
                ncli CHAR(10) NOT NULL REFERENCES client (ncli),
                datecom DATE NOT NULL);
```

La contrainte CHECK

Une contrainte **`CHECK`** est une règle de validation associée à une colonne ou à un ensemble de colonnes. Elle impose que les valeurs insérées ou mises à jour dans la colonne respectent le domaine de cette colonne.

Exemple : imposer que la colonne `cat` ne puisse prendre que les valeurs « C1 », « C2 » ou « C3 » et que le compte ne peut pas être en négatif.

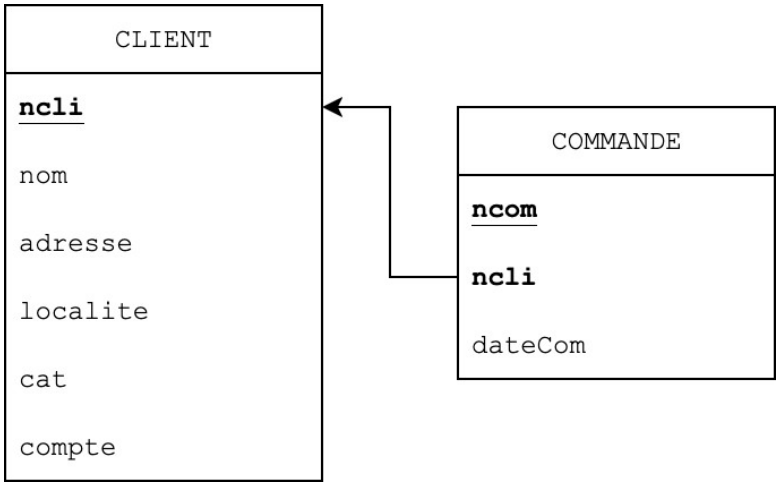
```sql
CREATE TABLE CLIENT (
    ncli      CHAR(10),
    nom       CHAR(32),
    adresse   CHAR(60),
    localite  CHAR(30),
    cat       CHAR(2),
    compte    DECIMAL(9,2),
    PRIMARY KEY (ncli),
    CONSTRAINT chk_cat CHECK (cat IN ('C1', 'C2', 'C3')),
    CONSTRAINT chk_compte CHECK (compte >= 0)
);
```

5.1.3. Suppression d'une table

Toute table peut-être supprimée. Elle est désormais inconnue et son contenu est perdu

```sql
DROP TABLE COMMANDE;
```

Attention ! Si au moment de la suppression, la table contenait des lignes, celles-ci sont préalablement supprimées.



De plus, cette opération est soumise aux contraintes référentielles qui concernent la table.

En effet, selon la structure donnée ci-dessus, le problème se pose lorsque ce client possède des commandes; il existe alors des lignes dans COMMANDE qui référencent la ligne de CLIENT à supprimer.

On définit trois comportements possibles qui laissent la base de données dans un état correct :

- **Blocage** (par défaut) : refuser la suppression de la ligne de CLIENT afin d'éviter de laisser dans la base de données des lignes de COMMANDE orphelines.

```
CREATE TABLE COMMANDE (
    ncom CHAR(12) PRIMARY KEY,
    ncli CHAR(10) NOT NULL REFERENCES client(ncli) ON DELETE RESTRICT,
    datecom DATE NOT NULL
);
```

- **Propagation ou cascade** : supprimer non seulement la ligne de CLIENT, mais aussi toutes les lignes de COMMANDE qui la référencent.

```
CREATE TABLE COMMANDE (
    ncom CHAR(12) PRIMARY KEY,
    ncli CHAR(10) NOT NULL REFERENCES client(ncli) ON DELETE CASCADE,
    datecom DATE NOT NULL
);
```

- **Indépendance** : le troisième est possible lorsque la clé étrangère est constituée de colonnes facultatives. Il consisterait ici, si `ncli` de COMMANDE avait été déclarée facultative, à effacer la valeur de la colonne `ncli` des lignes de COMMANDE qui référencent la ligne de CLIENT à supprimer. De la sorte, ces commandes n'appartiennent plus à aucun client après l'opération.

```
CREATE TABLE COMMANDE (
    ncom CHAR(12) PRIMARY KEY,
    ncli CHAR(10) NULL REFERENCES client(ncli) ON DELETE SET NULL,
    datecom DATE NOT NULL
);
```

Remarque : ce comportement est souvent utilisé lorsqu'on souhaite conserver un historique des données.

5.1.4. Ajout et retrait d'une colonne

La commande suivante ajoute la colonne `poids` à la table PRODUIT :

```
ALTER TABLE PRODUIT
ADD COLUMN poids SMALLINT
```

Après l'exécution de cette commande, la table possède une nouvelle colonne qui ne contient que des valeurs **NULL** pour toutes les lignes. On ne peut ajouter une colonne obligatoire que si la table est vide, ou si cette colonne possède une valeur par défaut.

L'élimination d'une colonne d'une table s'effectue à l'aide d'une commande similaire :

```
ALTER TABLE PRODUIT
DROP COLUMN prix
```

5.1.5. Ajout et retrait d'une contrainte

Nous avons rencontré jusqu'ici quatre types de contraintes :

- les identifiants primaires,
- les identifiants secondaires,
- les colonnes obligatoires
- les clés étrangères.

Ces contraintes, ou propriétés, sont généralement déclarées lors de la création de la table. Il est cependant possible de les ajouter et même de les retirer a posteriori par une commande **ALTER TABLE**.

Ajout d'un identifiant primaire

La demande d'ajout d'un identifiant sera refusée si les données que la table contient déjà violent cette propriété.

```
ALTER TABLE CLIENT ADD PRIMARY KEY (ncli);
```

Ajout d'un identifiant secondaire

Il en va de même pour les identifiants secondaires déclarés via une clause **UNIQUE** :

```
ALTER TABLE CLIENT ADD UNIQUE (nom, adresse, localite);
```

Transformation d'une colonne obligatoire en colonne facultative

Une colonne obligatoire peut être redéclarée facultative et inversement (si les données le permettent). En même temps que son type peut-être redéclaré. Attention également, à ce que le nouveau type n'altère pas les données déjà présentes dans la table (par exemple une chaîne de caractère plus courte).

```
ALTER TABLE CLIENT MODIFY adresse CHAR(255) NOT NULL;
ALTER TABLE CLIENT MODIFY cat CHAR(3) NULL;
```

Ajout d'une clé étrangère

Enfin, une clé étrangère peut être définie (si les données le permettent) ou retirée après création de la table source. Cette possibilité est indispensable lorsque le SGBD n'accepte pas la déclaration d'une clé étrangère vers une table non encore définie.

```
ALTER TABLE COMMANDE ADD FOREIGN KEY (ncli) REFERENCES client (ncli)
```

5.1.6. Exercices

Exercice 1

a) Créez les tables ci-dessous en respectant les contraintes. Insérez quelques lignes dans ces tables et affichez ensuite leur contenu afin de les tester.

Créez la table PAYS :

Nom de colonne	Type	Longueur	Obligatoire	Clé primaire	Clé étrangère vers
Numero	int	-	X	X	-
Nom	varchar	30	X	-	-

Créez la table LOCALITE :

Nom de colonne	Type	Longueur	Obligatoire	Clé primaire	Clé étrangère vers
NumLoc	int	-	X	X	-
CodePostal	int	4	X	-	-
Libelle	varchar	50	X	-	-
Pays	int	-	X	-	Pays

Créez la table PERSONNE :

Nom de colonne	Type	Longueur	Obligatoire	Clé primaire	Clé étrangère vers
NumCarteId	int	-	X	X	-
NomPrenom	varchar	100	X	-	-
NbEnfants	int	-	-	-	-
Salaire*	int	6	-	-	-
Localite	int	-	X	-	Localite

* Les salaires sont à mémoriser avec 2 chiffres après la virgule et contiennent maximum 6 chiffres significatifs.

b) Ajoutez la colonne suivante à la table PERSONNE :

Nom de colonne	Type	Longueur	Obligatoire	Clé primaire	Clé étrangère vers
Telephone	varchar	20	-	-	-

c) Créez les tables ci-dessous en respectant les contraintes. Insérez quelques lignes dans ces tables et affichez ensuite leur contenu afin de les tester.

Créez la table UTILISATEUR :

Nom de colonne	Type	Longueur	Obligatoire	Clé primaire	Clé étrangère vers
UserName	varchar	20	X	X	-
Nom*	varchar	30	X	-	-
Prenom*	varchar	50	X	-	-
Categorie*	varchar	20	X	-	-
DateCreation	date	-	X	-	-

*Contraintes supplémentaires :

- deux utilisateur ne peuvent avoir le même `Nom` et le même `Prenom`
- les seules valeurs permises dans la colonne `Categorie` sont : « Etudiant », « Professeur » ou « Administrateur »

Créez la table LOCAL :

Nom de colonne	Type	Longueur	Obligatoire	Clé primaire*	Clé étrangère vers
Etage	int	1	X	-	-
NomNumero	int	2	X	-	-
NbPlaces	int	3	-	-	-

*Contrainte supplémentaire : la clé primaire est constituée par la combinaison du numéro de l'étage et du numéro attribué au local au sein de l'étage ex : le local 3 situé au 2ième étage sera identifié par la combinaison 2 et 3.

Créez la table PC :

Nom de colonne	Type	Longueur	Obligatoire	Clé primaire	Clé étrangère vers
Matricule	varchar	10	X	X	-
?*	?	?	?	?	?

*Contrainte supplémentaire : prévoyez, dans l'instruction de création de la table, une clé étrangère vers la table « Local ». Cette clé étrangère permettra de retrouver, pour chaque PC, le local dans lequel il se trouve.

Créez la table SESSIONTRAVAIL permettant de mémoriser les sessions de travail des utilisateurs sur des PC :

Nom de colonne	Type	Longueur	Obligatoire	Clé primaire*	Clé étrangère vers
Login	varchar	20	X	-	Utilisateur
PC	varchar	10	X	-	PC
Debut	date	-	X	-	-
Fin	date	-		-	-

*Trouvez l'identifiant de la table « SessionTravail » et prévoyez-le dans l'instruction de création de la table.

Exercice 2

a) Crée les tables suivantes pour stocker les membres d'un réseau et des jeux vidéo disponibles.

Table JOUEUR :

Nom de colonne	Type	Longueur	Obligatoire	Clé primaire	Clé étrangère vers
Pseudo	varchar	20	X	X	-
Email*	varchar	60	X	-	-
MotDePasse	varchar	128	X	-	-
DateNais*	date	-	-	-	-
Ville	varchar	40	-	-	-
Niveau*	varchar	12	-	-	-

*Contraintes supplémentaires :

- Deux joueurs ne peuvent pas avoir la même adresse mail
- La date de naissance doit être antérieure à la date du jour (CURRENT_DATE)
- Les seules valeurs permises pour le niveau sont : « Débutant », « Amateur » et « Expert »

Table JEU :

Nom de colonne	Type	Longueur	Obligatoire	Clé primaire	Clé étrangère vers
Id	int	-	X	X	-
Nom	varchar	50	X	-	-
Catégorie*	varchar	25	-	-	-
NoteMoy*	decimal	2	-	-	-

*Contraintes supplémentaires :

- Les seules valeurs permises pour la catégorie sont : « FPS », « RPG », « Sport » et « Stratégie »
- La note moyenne est à mémoriser avec 1 chiffre après la virgule et contiennent maximum 2 chiffres significatifs.
- La note moyenne doit être comprise entre 0 et 10

b) Crée la table suivante pour modéliser les liens d'amitié entre les joueurs

Crée la table AMITIE :

Nom de colonne	Type	Longueur	Obligatoire	Clé primaire	Clé étrangère vers
Joueur1*	varchar	20	X	X	Joueur
Joueur2*	varchar	20	X	X	Joueur
DateAmitié	date	-	X	-	-

*Contraintes supplémentaire : arrange-toi pour que la suppression d'un des deux joueurs dans la base de données entraine la suppression de l'amitié.

c) Crée la table suivante pour enregistrer le meilleur score de chaque joueur sur un jeu en particulier

Crée la table SCORE :

Nom de colonne	Type	Longueur	Obligatoire	Clé primaire	Clé étrangère vers
`Joueur`	`varchar`	`20`	X	X	`Joueur`
`Jeu`	`int`	-	X	X	`Jeu`
`Score*`	`decimal`	`3`			

*Contraintes supplémentaires :

- Le score est à mémoriser avec 1 chiffre après la virgule et contiennent maximum 3 chiffres significatifs.
- Le score ne peut pas être négatif

d) Ajoute et supprime les colonnes nécessaires

Pour chaque joueur, on souhaite sauvegarder son avatar (lien d'une image). On souhaite également que chaque joueur spécifie obligatoirement sa ville lors de l'inscription.

e) Crée la table suivante permettant de stocker les clubs réunissants plusieurs joueurs

Nom de colonne	Type	Longueur	Obligatoire	Clé primaire	Clé étrangère vers
`NumLicence*`	`varchar`	`11`	X	X	-
`Nom`	`varchar`	`50`	X	-	-
`Créateur`	`varchar`	`50`	X	-	-
`Type*`	`varchar`	`15`	-	-	-

Contraintes supplémentaires :

- Le numéro de licence doit respecter le format suivant : `XXX-XXX-XXX`
- Les seules valeurs acceptées pour le type de club sont : « Casual », « Compétitif » et « Fan Club »

f) Ajoute les colonnes nécessaires dans les bonnes tables

On souhaite retenir le président de chaque club (un joueur). On souhaite également retenir, pour chaque joueur, le club auquel il appartient.

g) Quelques questions de réflexions…

→ Dans la table SCORE, pourquoi la clé primaire est-elle composée de (`joueur`, `jeu`) ?

→ Pourquoi la table AMITIE nécessite-t-elle une clé primaire composée des deux pseudos ? Que se passe-t-il si on supprime cette contrainte ?

→ Si on veut interdire à deux clubs d'avoir le même nom, quelle contrainte ajouterais-tu ?

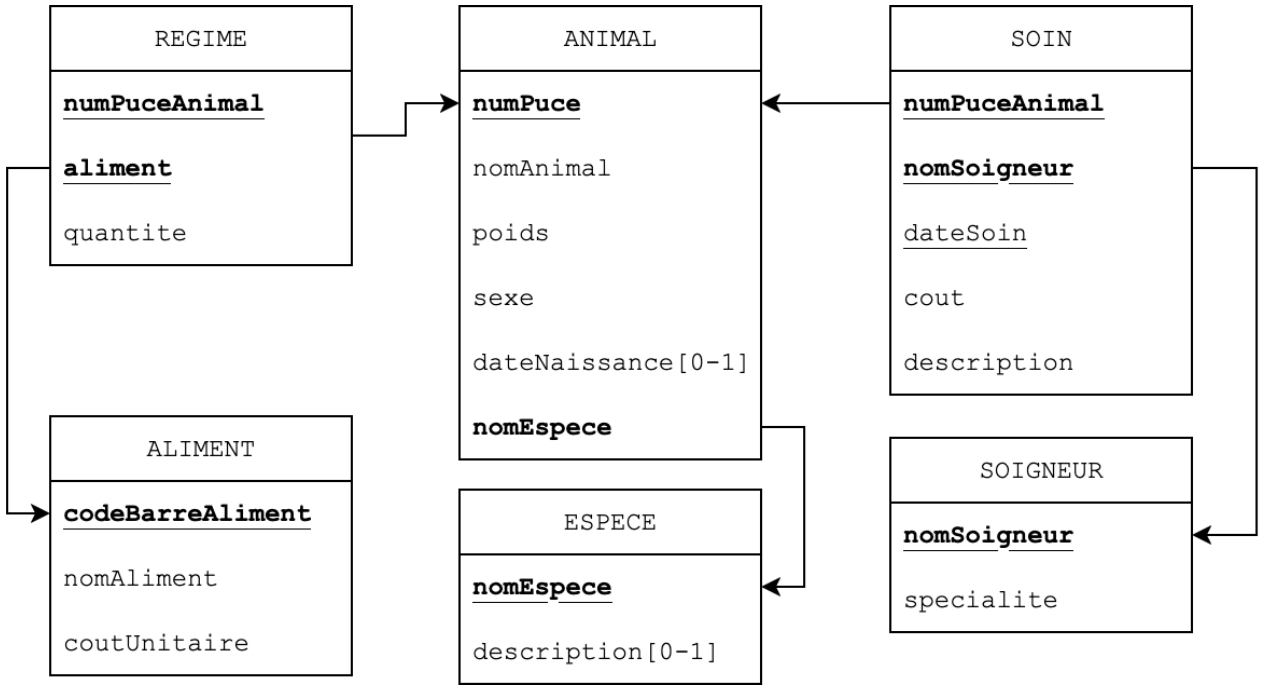
→ Les contraintes actuelles sur la table AMITIE empêchent-elles les boucles d'amitiés (A ami avec B, B ami avec A) ?

→ Que se passe-t-il si un président quitte la plateforme ?

Exercice 3

Tu travailles pour un zoo qui souhaite gérer ses animaux, leurs soigneurs, les soins administrés, et leur régime alimentaire.

Sur base du schéma E-R suivant, écris le code DDL permettant de créer la base de données du zoo en question :



Contraintes de domaine :

- `quantite`, `poids`, `coutUnitaire` et `cout` ne peuvent jamais être négatifs
- `quantite` ne peut pas dépasser 1000
- le poids de l'animal (`poids`) est à retenir avec 2 chiffres après la virgule et 8 chiffres significatifs
- `dateNaissance` ne peut pas se situer après la date du jour
- les seules valeurs acceptées pour `specialite` sont : « Vétérinaire », « Nutritionniste », « Dentiste » et « Comportementaliste »
- les seules valeurs acceptées pour `sexe` sont : « M » ou « F »
- deux animaux de la même espèce ne peuvent pas porter le même nom

Questions de réflexion :

Pourquoi avoir choisi l'identifiant (`numPuceAnimal`, `nomSoigneur`, `dateSoin`) pour la table `SOIN` ? L'identifiant (`numPuceAnimal`, `nomSoigneur`) serait-il suffisant ?

5.2 Data Manipulation Language (DML)

Le **DML**, ou langage de manipulation des données, regroupe l'ensemble des instructions permettant d'opérer sur les données déjà présentes dans une base de données.

Elle inclut notamment les instructions pour insérer de nouvelles lignes, modifier des informations existantes, supprimer des enregistrements ou encore interroger les données à l'aide de requêtes. Le DML joue donc un rôle central dans la gestion quotidienne des informations stockées.

5.2.1. Ajout de lignes

Si on désire ajouter une ligne constituée de valeurs nouvelles, on utilisera l'instruction `INSERT INTO ... VALUES ...`

Méthode implicite

L'ordre des valeurs est celui de la déclaration des colonnes lors de la création de la table.

```
INSERT INTO DETAIL VALUES ('30185','PA45',12)
```

Méthode explicite

On peut aussi, en particulier lorsque certaines valeurs seulement sont introduites, préciser le nom et l'ordre des colonnes concernées :

```
INSERT INTO CLIENT (ncli, nom, adresse, compte, localite)
VALUES ('C402','BERNIER','avenue de France, 28', -2500,'Lausanne')
```

Toute colonne non spécifiée prend la valeur **`NULL`**, ou la valeur par défaut si celle-ci a été déclarée comme propriété de la colonne. Toute colonne obligatoire (**`NOT NULL`**) doit recevoir une valeur, sauf si on lui a assigné une valeur par défaut (**`DEFAULT`**) lors de sa déclaration.

5.2.2. Suppression de lignes

L'ordre de suppression porte sur un sous-ensemble des lignes d'une table. Ces lignes sont désignées par une clause **`WHERE`**.

Cet ordre supprime de la table CLIENT la ligne qui décrit le client numéro K111 :

```
DELETE FROM CLIENT WHERE ncli = 'K111'
```

Cette instruction supprime de la table DETAIL les lignes, quel qu'en soit le nombre, qui spécifient un produit en rupture de stock :

```
DELETE FROM DETAIL
WHERE npro IN (SELECT npro
               FROM PRODUIT
               WHERE qstock ≤ 0)
```

Après l'opération, la base de données doit être dans un état qui respecte toutes les contraintes d'intégrité auxquelles elle est soumise, et en particulier les contraintes référentielles. Nous y reviendrons.

5.2.3.Modification de lignes

Ici encore, la modification sera effectuée sur toutes les lignes qui vérifient une condition de sélection :

```
UPDATE CLIENT
SET adresse = '29, av. de la Magne', localite = 'Niort'
WHERE ncli = 'F011'
```

Les nouvelles valeurs peuvent être obtenues par une expression arithmétique. La commande suivante enregistre une augmentation de prix de 5% pour les produits en sapin :

```
UPDATE PRODUIT
SET prix = prix * 1.05
WHERE libelle LIKE '%SAPIN%'
```

5.2.4.Exercices

Crée la table WORKER via le script *workers.sql* fourni. Prends le temps de comprendre la structure de la table.

Première partie

Insère dans la table WORKER les données suivantes en utilisant la méthode explicite (a) ou implicite (b).

ID	Lastname	Firstname	Salary	Benefitinkind	Birthdate	Hiredate	
100	Adams	Benoit	1800	carpackage	15/05/1978	15/10/2005	(a)
110	Dubois	Alain	1500		11/03/1985	02/02/2011	(b)
120	Radu	Jean	1500		25/10/1969		(a)
130	Trenet	Julien	2000				(a)
140	VanRoos	Marie					(a)
150		Renaud			21/11/1972		(b)
110	Ergot		1650			Date du jour	(b)
	Ramon	Pierre					(a)

Certaines insertions devraient produire une erreur. Effectue les modifications nécessaires au niveau des lignes à insérer pour que toutes les insertions puissent être effectuées.

Deuxième partie

Insère maintenant un nouveau travailleur qui vient d'être engagé aujourd'hui et un travailleur né le 21 novembre 1949. La table doit maintenant comporter 10 éléments.

Troisième partie

Effectue les mises à jour suivante :

- Attribuer un *carpackage* à tous les travailleurs qui ont été engagés entre le 1er janvier 2011 et le 31 décembre 2011
- Mettre la date d'aujourd'hui comme date d'engagement à tous les employés pour lesquels la date d'engagement n'a pas été définie
- Augmenter de 5% le salaire des employés qui touchent un salaire inférieur à 1800 euros
- Allouer un salaire de base de 1200 euros aux travailleurs qui ont un salaire inconnu
- Modifier le nom de famille du matricule 120 : il s'agit de Radut et non de Radu né le 25 octobre 1968

Quatrième partie

Effectue les suppressions de lignes suivantes :

- Supprimer les employés nés avant le 1ier janvier 1950
- Supprimer les employés dont l'identifiant est égal à 100, 110 ou 130
- Supprimer les employés dont le nom de famille commence par R

5.3 Extraction de données

L'**extraction** désigne l'ensemble des opérations visant à récupérer des données spécifiques depuis une base de données.

L'extraction de données est essentielle pour exploiter, analyser ou présenter l'information aux utilisateurs. L'extraction peut aller de simples interrogations à des analyses complexes intégrant filtres, regroupements, tris et jointures entre plusieurs tables. Elle transforme la donnée brute en connaissance utile.

5.3.1. Extraction simple

L'exécution d'une requête **SELECT** produit un résultat qui est une table. D'une manière générale, une requête simple contient trois parties principales :

- la clause **SELECT** précise les valeurs (nom des colonnes, valeurs dérivées) qui constituent chaque ligne du résultat,
- la clause **FROM** indique les tables desquelles le résultat tire ses valeurs,
- la clause **WHERE** donne la condition de sélection que doivent satisfaire les lignes qui fournissent le résultat.

La requête la plus simple consiste à demander d'afficher les valeurs de certaines colonnes de lignes d'une table.

La requête suivante demande les valeurs de `ncli`, `nom` et `localite` des lignes de la table CLIENT.

```
SELECT ncli, nom, localite FROM CLIENT
```

La réponse à cette requête se présenterait comme suit à l'écran.

NCLI	NOM	LOCALITE
B062	GOFFIN	Namur
B112	HANSENNE	Poitiers
B332	MONTI	Genève
B512	GILLET	Toulouse
C003	AVRON	Toulouse
C123	MERCIER	Namur
C400	FERARD	Poitiers
D063	MERCIER	Toulouse
F010	TOUSSAINT	Poitiers
F011	PONCELET	Toulouse
F400	JACOB	Bruxelles
K111	VANBIST	Lille

Si on demande les valeurs de toutes les colonnes, la clause **SELECT** peut se simplifier comme suit.

```
SELECT * FROM CLIENT
```

5.3.2.Extraire des lignes sélectionnées

Extrayons à présent les informations `ncli` et `nom` des lignes de la table CLIENT qui concernent les clients de Toulouse.

```
SELECT ncli, nom, localite
FROM CLIENT
WHERE localite = 'Toulouse'
```

Le résultat est la table suivante :

```
NCLI            NOM             LOCALITE
----            ---             --------
B512            GILLET          Toulouse
C003            AVRON           Toulouse
D063            MERCIER         Toulouse
F011            PONCELET        Toulouse
```

La relation d'égalité apparaissant dans la condition de sélection n'est qu'un exemple de comparateur; on dispose en fait des relations suivantes :

Relation	*Symbole*
égal à	=
plus grand que	>
plus petit que	<
différent de	! =
plus grand ou égal	>=
plus petit ou égal	<=

5.3.3.Ignorer les lignes dupliquées

En principe, le résultat d'une requête contient autant de lignes qu'il y a, dans la table de départ, de lignes vérifiant la condition de sélection. Il se peut donc, dès qu'aucun identifiant n'est repris entièrement dans la clause **SELECT**, que le résultat contienne plusieurs lignes identiques.

La requête suivante affiche les localités des clients repris dans la catégorie « *C1* » :

```
SELECT localite
FROM CLIENT
WHERE cat = 'C1'
```

Nous obtenons le résultat suivant :

```
LOCALITE
--------
Poitiers
Namur
Poitiers
Namur
Namur
```

La réponse contient autant de lignes que la table originale CLIENT en contient qui satisfont la sélection. On pourra éliminer les lignes en double par la clause **DISTINCT** comme suit :

```
SELECT DISTINCT localite
FROM CLIENT
WHERE cat = 'C1'
```

Nous obtenons le résultat suivant, qui ne contient plus que des lignes distinctes :

```
LOCALITE
--------
Poitiers
Namur
```

Il faut être très prudent lorsqu'on exploite de tels résultats.

Par exemple, la requête suivante donne les numéros des clients qui ont passé au moins une commande :

```
SELECT ncli FROM COMMANDE
```

Cependant, le nombre d'éléments du résultat n'est pas égal à celui des clients qui ont passé une commande, mais bien au nombre de commandes.

On écrira donc plutôt :

```
SELECT DISTINCT ncli FROM COMMANDE
```

5.3.4. Tester l'appartenance à une liste

Une condition peut aussi porter sur l'appartenance à une liste.

Par exemple, la requête suivante affiche les localités des clients repris dans la catégorie « *C1* », « C2 » ou « C3 » :

```
SELECT localite
FROM CLIENT
WHERE cat IN ('C1','C2','C3')
```

Autre exemple : afficher toutes les infos sur les clients qui n'habitent ni à Toulouse, ni à Namur, ni à Breda.

```
SELECT *
FROM CLIENT
WHERE localite NOT IN ('Toulouse','Namur','Breda')
```

5.3.5. Tester l'appartenance à un intervalle

Une condition peut également porter sur l'appartenance à un intervalle.

Par exemple, la requête suivante sélectionne les clients dont le compte est compris entre 1000 et 4000 (bornes incluses) :

```
SELECT *
FROM CLIENT
WHERE compte BETWEEN 1000 AND 4000
```

De même que le **`NOT IN`**, il existe également le **`NOT BETWEEN`**.

5.3.6. Tester la présence de certains caractères

On peut également tester la présence de certains caractères dans une valeur.

Ces conditions utilisent un masque qui décrit la structure générale des valeurs désignées. Dans ce masque, le symbole `_` (tiret du bas) désigne un caractère quelconque et le symbole `%` (pourcent) désigne toute suite de caractères, éventuellement vide. Tout autre caractère doit être présent là où il apparaît.

Par exemple, la première expression est satisfaite dans la table CLIENT pour les valeurs de `cat` constituées d'un caractère quelconque suivi du caractère **`1`**, soit l'une des valeurs "A1", "B1", "C1".

```
SELECT *
FROM CLIENT
WHERE cat LIKE '_1'
```

Celle-ci est satisfaite par toute valeur de `adresse` contenant le mot "Neuve".

```
SELECT *
FROM CLIENT
WHERE adresse LIKE '%Neuve%'
```

Un petit problème se pose : comment rechercher les caractères `%` et `_` dans les données ? Il suffit de les préfixer dans le masque par un caractère spécial qu'on définit dans une clause escape.

Dans l'exemple ci-dessous, on recherche la présence de « _CHENE » dans les valeurs de `libelle` :

```
SELECT *
FROM CLIENT
WHERE libelle LIKE '%$_CHENE%' ESCAPE '$'
```

Un masque peut aussi s'appliquer à une date. La forme suivante retient les dates de commande tombant en 2005

```
SELECT *
FROM CLIENT
WHERE datecom LIKE '%2005%'
```

Cette condition admet une forme négative **`NOT LIKE`**, dont l'interprétation est évidente :

```
SELECT *
FROM CLIENT
WHERE adresse NOT LIKE '%Neuve%'
```

5.3.7.Expressions composées

La condition de sélection introduite par la clause **`WHERE`** peut être constituée d'une expression booléenne de conditions élémentaires (opérateurs **`AND`**, **`OR`**, **`NOT`** et parenthèses).

Exemple : obtenir tous les clients qui habitent Toulouse et dont leur compte est en négatif.

```
SELECT nom, adresse, compte
FROM CLIENT
WHERE localite = 'Toulouse' AND compte < 0
```

Étant donné les conditions p et q relatives aux lignes d'une table, la clause

- `WHERE p AND q` sélectionne les lignes qui vérifient *simultanément* p et q;
- `WHERE p OR q` sélectionne les lignes qui vérifient p *ou* q *ou* les deux;
- `WHERE NOT p` sélectionne les lignes qui *ne vérifient pas* p.

L'usage de parenthèses permet de former des conditions plus élaborées encore :

```
SELECT nom, adresse, compte
FROM CLIENT
WHERE compte > 0
AND (cat = 'C1' OR localite = 'Paris')
```

Les parenthèses peuvent être utilisées même lorsqu'elles ne sont pas indispensables, par exemple pour rendre plus lisible une condition composée :

```
SELECT nom, adresse, compte
FROM CLIENT
WHERE (localite = 'Toulouse') AND (compte < 0)
```

5.3.8. Trier les résultats

En SQL, la clause `ORDER BY` permet de classer les résultats d'une requête selon une ou plusieurs colonnes, dans un ordre croissant ou décroissant. Cela facilite la lecture et l'analyse des données obtenues.

Exemple : trier une liste d'élèves par ordre alphabétique croissant sur une leur nom.

```
SELECT nom, prénom
FROM ELEVES
ORDER BY nom;
```

On peut également trier par ordre décroissant en utilisant le mot-clé **DESC** :

```
SELECT nom, prénom
FROM ELEVES
ORDER BY date_naissance DESC;
```

Enfin, il est possible de trier sur plusieurs colonnes.

Par exemple, la requête suivante trie d'abord par nom (ordre alphabétique croissant). Pour les élèves portant le même nom, elle trie à l'intérieur de ce groupe par note décroissante (du meilleur au moins bon).

```
SELECT nom, prénom, note
FROM ELEVES
ORDER BY nom ASC, note DESC;
```

Remarques :

- On peut trier sur n'importe quelle colonne du résultat, même si elle n'est pas affichée dans la sélection.
- L'ordre de tri par défaut est **ASC**, donc écrire **ORDER BY** nom est équivalent à **ORDER BY** nom **ASC**.

5.3.9. Tester l'absence d'une valeur

Pour rappel, en SQL, la valeur **NULL** signifie que la donnée est inconnue ou absente dans une colonne.

Ainsi, on ne peut pas utiliser les opérateurs classiques comme = ou ! = pour tester **NULL**, car **NULL** est une absence de valeur. Pour cela, SQL fournit une syntaxe spéciale :

- **IS NULL** : teste si la valeur est **NULL**
- **IS NOT NULL** : teste si la valeur n'est pas **NULL**

Par exemple, cette requête affiche tous les employés dont le numéro de téléphone est inconnu :

```
SELECT nom, prénom
FROM EMPLOYES
WHERE telephone IS NULL;
```

Cette requête ne garde que les clients dont l'email est connu :

```
SELECT client_id, email
FROM CLIENTS
WHERE email IS NOT NULL;
```

5.3.10. Grouper les résultats

Un groupement consiste à regrouper les lignes d'une table selon une ou plusieurs colonnes communes (ex : regrouper les élèves par classes).

La clause GROUP BY

Pour effectuer un groupement, on utilise le mot-clé **`GROUP BY`**. Lorsqu'on l'utilise, cela impose des règles strictes sur les colonnes que l'on peut mettre dans la clause **`SELECT`**.

En effet, dans une requête avec **`GROUP BY`**, toutes les colonnes sélectionnées doivent soit :

- Faire partie de la clause **`GROUP BY`**, c'est-à-dire être utilisées pour définir les groupes
- Ou bien être utilisées dans une fonction d'agrégation.

Les fonctions d'agrégation les plus utiles sont :

- **`COUNT`**`(*)` : compte le nombre de lignes dans chaque groupe
- **`COUNT`**`(`**`DISTINCT`**` colonne)` : compte le nombre de valeurs distinctes
- **`MIN`**`(colonne)` : valeur minimale
- **`MAX`**`(colonne)` : valeur maximale
- **`AVG`**`(colonne)` : moyenne
- **`SUM`**`(colonne)` : somme

Voici un premier exemple dans lequel on souhaite compter combien d'employés il y a dans chaque département :

```
SELECT department_id, COUNT(*)
FROM EMPLOYES
GROUP BY departement_id;
```

Pour cette requête, un exemple de résultat pourrait être :

departement_id	count(*)
1	16
2	90
3	7

Remarque : les départements qui n'ont aucun employé (c'est-à-dire les `departement_id` absents de la table EMPLOYES) ne figurent pas du tout dans le résultat. En effet, **`COUNT(*)`** ne retourne pas de ligne pour un département qui n'a pas d'employés, car ce département n'apparaît pas dans la table EMPLOYES tant qu'aucun employé n'y est lié. Le **`GROUP BY`** fait un regroupement seulement sur les `departement_id` présents dans la table EMPLOYES.

Supposons que l'on tente cette requête incorrecte :

```
SELECT classe_id, nom, COUNT(*)
FROM ELEVES
GROUP BY classe_id;
```

Le SGBD refuse car la colonne `nom` n'est ni dans la clause **`GROUP BY`**, ni une fonction d'agrégation.

Parfois, on veut regrouper les données selon deux critères (ou plus) à la fois, pour avoir un regroupement plus précis. Pour cela, on liste toutes les colonnes de regroupement dans le **`GROUP BY`**, séparées par des virgules.

Imaginons une table VENTES contenant les colonnes suivantes :

- `produit_id` : identifiant du produit vendu
- `magasin_id` : identifiant du magasin dans lequel le produit a été vendu
- `quantite` : nombre d'exemplaires du produit vendu

Chaque ligne représente une vente d'un produit dans un magasin donné.

Pour retrouver combien d'exemplaires de chaque produit ont été vendus dans chaque magasin, on écrit la requête suivante :

```
SELECT magasin_id, produit_id, SUM(quantite) AS total_vendu
FROM VENTES
GROUP BY magasin_id, produit_id;
```

Ainsi, si la base de données contient ces lignes :

magasin_id	produit_id	quantite
1	101	3
1	101	2
1	102	1
2	101	4
2	102	2
2	102	2

La requête donne ce résultat :

magasin_id	produit_id	total_vendu
1	101	5
1	102	1
2	101	4
2	102	4

La clause WHERE avant GROUP BY

Si on veut ne regrouper qu'un sous-ensemble de données, on utilise **`WHERE`** avant **`GROUP BY`**.

Exemple : afficher le nombre de filles par classe.

```sql
SELECT classe_id, COUNT(*) AS nb_filles
FROM ELEVES
WHERE sexe = 'F'
GROUP BY classe_id;
```

Remarques : dans cet exemple, seules les classes avec au moins une fille apparaîtront dans ce résultat.

La clause HAVING

La clause **`HAVING`** permet de filtrer des groupes.

Exemple : afficher le nombre de garçons par classe et ne garder que les classe qui comptent au moins 10 garçons.

```sql
SELECT classe_id, COUNT(*) AS nb_garcons
FROM ELEVES
WHERE sexe = 'H'
GROUP BY classe_id
HAVING COUNT(*) >= 10;
```

Autre exemple : lister les classes où la moyenne des notes des élèves arrivés après une certaine date se situe entre 12 et 16.

```sql
SELECT classe_id, AVG(note) AS moyenne
FROM eleves
WHERE date_arrivee > '2024-09-01'
GROUP BY classe_id
HAVING AVG(note) BETWEEN 12 AND 16;
```

Ce type de requête permet, par exemple, d'identifier les classes avec de bons résultats parmi les nouveaux élèves arrivés récemment.

Attention ! La clause **HAVING** ne doit pas être confondue avec **WHERE** :

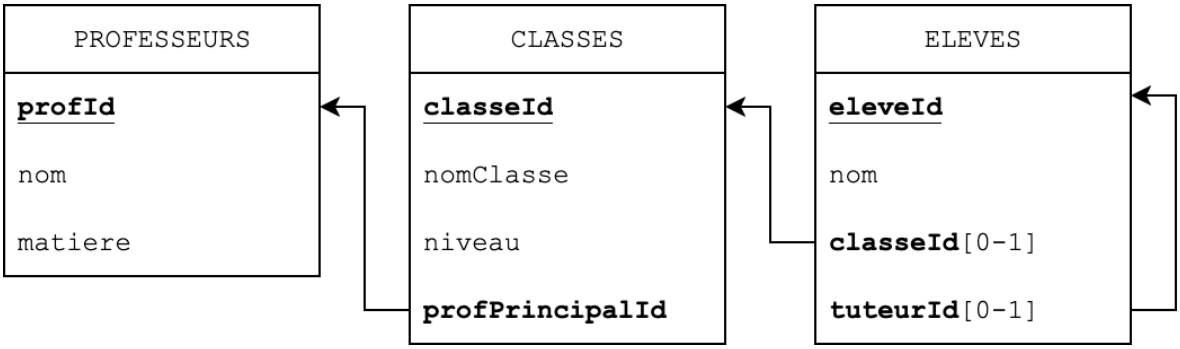
- La clause **WHERE** s'applique sur les données individuelles dans la table, ligne par ligne, avant que la base effectue un regroupement (**GROUP BY**) et toute agrégation. Elle sert à filtrer les enregistrements bruts selon des conditions classiques, sans utiliser de fonctions d'agrégation.
- La clause **HAVING** intervient après le regroupement des données par **GROUP BY**. Elle sert à filtrer les groupes créés, souvent en fonction de résultats d'agrégation (comme **COUNT**, **AVG**, **SUM**, etc.).

5.3.11. Jointures

Une **jointure** permet d'associer plusieurs tables dans une même requête.

Autrement dis, les jointures servent à relier des tables entre elles pour croiser les données et répondre à des questions complexes.

Pour cette section, nous allons utiliser l'exemple suivant :



Jointure interne

Une jointure interne (aussi appelée **INNER JOIN**) combine les lignes de deux tables lorsque la condition de jointure est satisfaite, c'est-à-dire quand il existe une correspondance entre les clés dans les deux tables. Seules les lignes ayant une correspondance dans les deux tables sont conservées.

Exemple : lister les élèves avec leur classe associée.

```
SELECT e.nom, c.nomClasse
FROM ELEVES e
JOIN CLASSES c ON e.classeId = c.classeId;
```

Remarque : seuls les élèves ayant une classe valide apparaissent.

Jointure avec condition supplémentaire

On peut ajouter des filtres dans la clause `WHERE` ou dans la jointure pour affiner les résultats, par exemple en sélectionnant uniquement un sous-ensemble des enregistrements liés.

Exemple : lister les classes dont le professeur principal est un professeur de mathématique.

```
SELECT c.nomClasse
FROM CLASSES c
JOIN PROFESSEURS p ON c.profPrincipalId = p.profId
WHERE p.matiere = 'Mathématiques';
```

Jointure entre plusieurs tables

Plusieurs tables peuvent être jointes simultanément pour agréger des informations complexes. La relation entre les tables doit être bien comprise, pour enchaîner plusieurs conditions de jointure.

Exemple : lister les élèves, leur classe et le nom du professeur principal pour des classes du secondaire.

```
SELECT e.nom, c.nomClasse, p.nom AS professeur
FROM ELEVES e
JOIN CLASSES c ON e.classeId = c.classeId
JOIN PROFESSEURS p ON c.profPrincipalId = p.profId
WHERE c.niveau = 'Secondaire';
```

Jointure externe gauche

Une jointure externe gauche conserve toutes les lignes de la table de gauche même si elles n'ont pas de correspondance dans la table de droite. Les colonnes de la table droite sont alors nulles pour ces lignes. Cela évite de perdre des données.

Exemple : lister tous les élèves, même ceux sans classe affectée.

```
SELECT e.nom, c.nom_classe
FROM ELEVES e
LEFT JOIN CLASSES c ON e.classeId = c.classeId;
```

Remarques : tous les élèves seront présents dans le résultat ; pour les élèves sans classe, la colonne `nom_classe` affichera `NULL`.

Jointure externe droite

Une jointure externe droite permet de conserver toutes les lignes de la table de droite, même si elles n'ont aucune correspondance dans la table de gauche.

Tout comme avec la jointure externe gauche, pour les lignes de la table de droite sans correspondance, les colonnes de la table de gauche auront la valeur `NULL`.

Exemple : lister toutes les classes, avec le nom des élèves inscrits dans chaque classe en incluant les classes qui n'ont aucun élève.

```
SELECT c.nomClasse, e.nom AS nomEleve
FROM ELEVES e
RIGHT JOIN CLASSES c ON e.classeId = c.classeId;
```

Remarques : toutes les classes seront présentes dans le résultat ; pour les classes sans élève, la colonne `nom_eleve` affichera `NULL`.

Auto-jointure

Une table peut être jointe à elle-même, par exemple pour représenter une relation hiérarchique ou de parrainage. On crée deux alias distincts pour différencier les rôles dans la jointure.

Exemple : afficher, pour chaque élève, son nom ainsi que le nom de son tuteur (qui lui aussi est un élève).

```
SELECT e.nom AS eleve, t.nom AS tuteur
FROM ELEVES e
JOIN ELEVES t ON e.idTuteur = t.eleveId;
```

Remarque : les élèves sans tuteur ne seront pas affichés (sinon, utiliser `LEFT JOIN`) .

Jointure externe complète

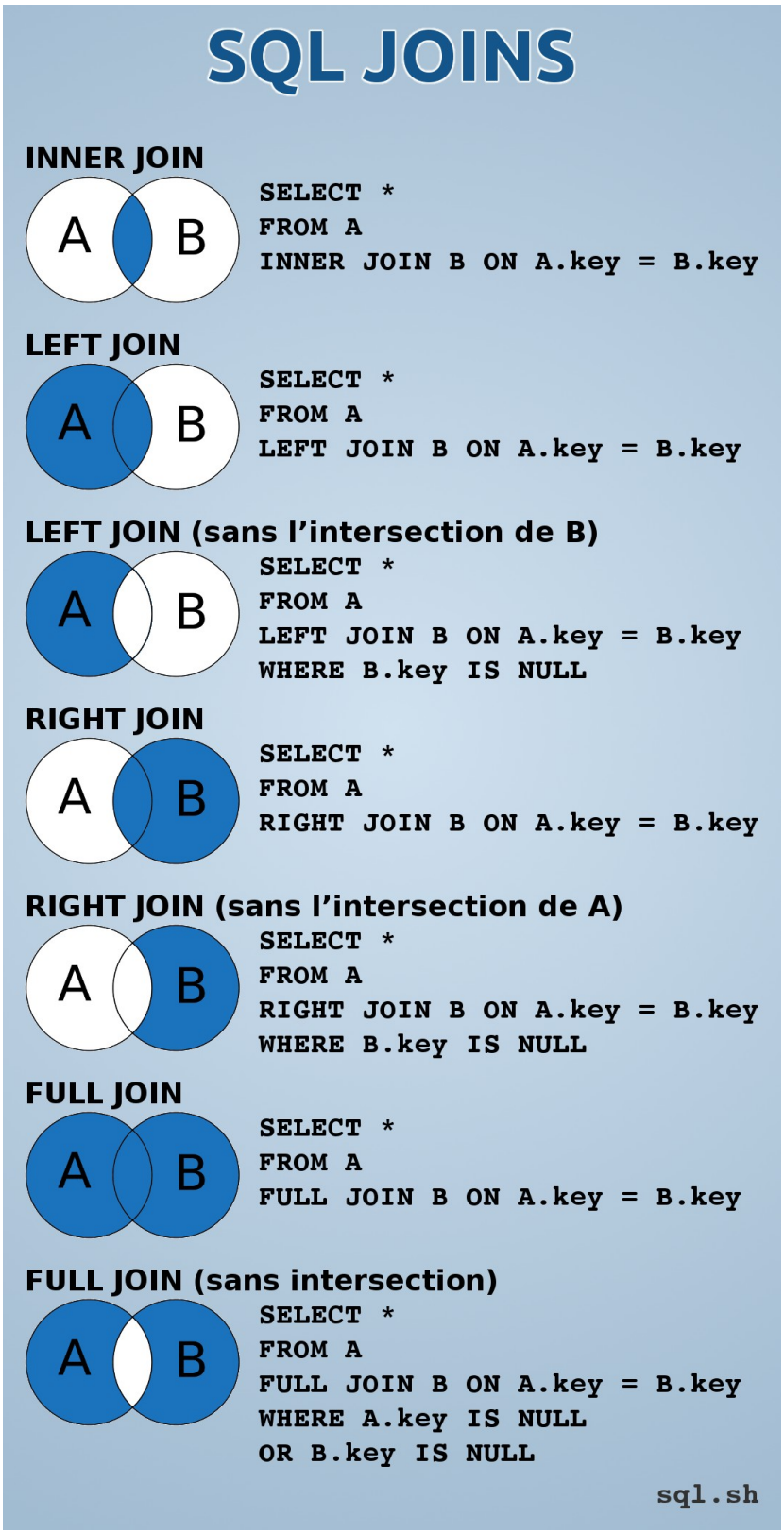
Une jointure externe complète affiche toutes les lignes des deux tables, y compris celles qui n'ont pas de correspondance dans l'autre table. Là où la correspondance n'existe pas, les colonnes sont `NULL`.

Exemple : lister toutes les classes, avec le nom des élèves inscrits dans chaque classe en incluant les classes qui n'ont aucun élève mais aussi les élèves qui n'ont pas de classe.

```
SELECT e.nom AS eleve, c.nomClasse
FROM ELEVES e
FULL OUTER JOIN CLASSES c ON e.classeId = c.classeId;
```

Remarque : toutes les classes et tous les élèves seront présents dans le résultat.

Récapitulatif des types de jointure



5.3.12. Sous-requêtes

Une **sous-requête** est une requête imbriquée à l'intérieur d'une autre.

Une sous-requête permet d'extraire des données à partir du résultat d'une autre requête.

Comprendre l'utilité des sous-requêtes

Pour illustrer l'utilité des sous-requêtes, considérons les clients qui habitent dans une localité donnée. Il est possible d'en retrouver les numéros en posant la requête suivante :

```
SELECT ncli
FROM CLIENT
WHERE localite = 'Namur'
```

dont l'exécution nous donnerait :

```
NCLI
----
B062
C123
L422
S127
```

Il est alors aisé de retrouver les commandes de ces clients de Namur :

```
SELECT ncom, datecom
FROM COMMANDE
WHERE ncli IN ('C123','S127','B062','L422')
```

Cette procédure n'est évidement pas très pratique. Il serait plus judicieux de remplacer cette liste de valeurs par l'expression qui a permis de les extraire de la table CLIENT.

On peut en effet écrire :

```
SELECT ncom, datecom
FROM COMMANDE
WHERE ncli IN (SELECT ncli
               FROM CLIENT
               WHERE localite = 'Namur')
```

Cette structure emboîtée correspond à l'association des tables COMMANDE et CLIENT sur la base de valeurs identiques de `ncli`.

Ainsi, une structure **`SELECT`**-**`FROM`** qui intervient dans une forme **`WHERE`** est appelée une sous-requête.

Sous-requêtes retournant une seule valeur

Une sous-requête dans une clause **WHERE** peut retourner une valeur unique. C'est ce qu'on appelle une sous-requête scalaire.

Exemple : lister les élèves inscrits dans la même classe qu'Alice.

```
SELECT nom
FROM ELEVES
WHERE classeId = (
    SELECT classeId
    FROM ELEVES
    WHERE nom = 'Alice'
)
```

Remarques :

- Puisque la sous-requête renvoie une seule valeur, il est correct d'utiliser l'égalité comme comparateur
- D'une certaine manière, Alice est dans la même classe qu'Alice (elle-même) : comment faire pour l'exclure du résultat ?

Les opérateurs de comparaison usuels (égal, différent, supérieur, inférieur) sont utilisables avec des sous-requêtes scalaire.

Sous-requêtes retournant plusieurs valeurs

Parfois, la sous-requête retourne plusieurs valeurs. Il faut alors utiliser un opérateur qui teste la présence d'une valeur de la requête principale dans ce résultat multiple.

(a) L'opérateur **IN** teste si une valeur est dans une liste produite par la sous-requête.
Exemple : lister les élèves inscrits dans l'une des classes de secondaire.

```
SELECT nom
FROM ELEVES
WHERE classeId IN (
    SELECT classeId
    FROM ELEVES
    WHERE niveau = 'Secondaire'
);
```

(b) L'opérateur **ANY** teste si une condition est vraie pour au moins une valeur de la sous-requête.
Exemple : afficher les élèves dont la note est supérieure à au moins une note des élèves en primaire.

```sql
SELECT nom, note
FROM ELEVES
WHERE note > ANY (
    SELECT note
    FROM ELEVES
    WHERE niveau = 'Primaire'
);
```

(c) L'opérateur **`ALL`** teste si une condition est vraie pour toutes les valeurs retournées. Exemple : afficher les élèves ayant une note supérieure à toutes les notes des élèves en primaire.

```sql
SELECT nom, note
FROM ELEVES
WHERE note > ALL (
    SELECT note
    FROM ELEVES
    WHERE niveau = 'Primaire'
);
```

Sous-requêtes corrélées

Une sous-requête est corrélée lorsqu'elle fait référence à une colonne de la requête principale. La sous-requête est donc évaluée pour chaque ligne de la requête extérieure.

Exemple : lister les élèves dont la note est supérieure à la moyenne des notes de leur propre classe.

```sql
SELECT nom, note
FROM ELEVES e
WHERE note > (
    SELECT AVG(note)
    FROM ELEVES
    WHERE classeId = e.classeId
);
```

Remarque : ici, la sous-requête dépend de la classe de chaque élève (`e.classeId`).

Sous-requêtes avec plusieurs colonnes (tuples)

Certaines sous-requêtes retournent plusieurs colonnes, et on peut comparer un ensemble de colonnes via l'opérateur **`IN`** avec des tuples.

Exemple : trouver les élèves qui ont la même classe et la même note qu'un élève particulier (ici, `eleveId = 123`).

```
SELECT nom
FROM eleves
WHERE (classeId, note) IN (
    SELECT classeId, note
    FROM ELEVES
    WHERE eleveId = 123
);
```

Une sous-requête peut elle-même contenir une sous-requête. La requête suivante donne les produits qui ont été commandés par au moins un client de Namur.

```
SELECT *
FROM PRODUIT
WHERE npro IN
(   SELECT npro
    FROM DETAIL
    WHERE ncom IN
    (     SELECT ncom
          FROM COMMANDE
          WHERE ncli IN
                (     SELECT ncli
                      FROM CLIENT
                      WHERE localite ='Namur')))
```

5.3.13. Exercices

Sur base des scripts DDL et DML mis à ta disposition, réalise les requêtes d'extraction suivantes.

L'entreprise

1. Affichez les noms de famille, prénoms et numéros de département de tous les employés, le tout trié par département, et en cas de département commun, par ordre alphabétique inverse sur le nom de famille.
   (40 lignes)

2. Affichez le nom de famille de tous les employés ainsi que leur salaire augmenté de l'indexation de 2% (renommer la colonne : Indexed Salary).
   (40 lignes)

3. Affichez toutes les informations sur les employés dont l'identifiant est compris entre 100 et 200, bornes comprises.
   (34 lignes)

4. Affichez les noms de famille, prénoms et numéros de téléphone des employés dont on connaît le pourcentage de commission (càd qui touchent un pourcentage de commission non null).
   (30 lignes)

5. Affichez toutes les informations sur les employés dont le nom de famille est King, Ernst, Greenberg ou Chen. Utilisez l'opérateur de condition IN (...).
   (4 lignes)

6. Affichez uniquement les numéros des départements et des jobs (job_id) de tous les employés. Evitez les duplicatas à l'affichage.
   (19 lignes)

7. Affichez l'adresse mail et le numéro de téléphone de tous les employés ayant la fonction "President" (job_id = 4).
   (1 ligne)

8. Affichez les noms et prénoms des employés dont le nom de famille commence par D ou dont le nom de famille contient la lettre a ou la lettre e en seconde position.
   (14 lignes)

9. Affichez toutes les informations sur les employés dont le matricule du manager est 100, 103 ou 124, et qui n'ont pas un salaire compris entre 2500 et 4000. Triez le tout par manager, et en cas de manager commun, par salaire (du plus haut salaire au plus

bas salaire).
(18 lignes)

10. Listez les matricules des employés qui ne sont affectés à aucun département.
(0 ligne)

11. Affichez toutes les informations sur les employés dont le numéro de téléphone commence par 650 ou 590 et qui ne touchent pas de commission. Triez le tout du plus bas salaire au plus haut salaire.
(4 lignes)

12. Affichez par département, le numéro du département, le nombre d'employés ainsi que le plus petit et le plus grand des salaires des employés travaillant dans le département.
(11 lignes)

13. Affichez par type de job, l'identifiant du job et le nombre de salaires différents.
(19 lignes)

14. Affichez par département, le numéro du département ainsi que la moyenne des salaires des employés engagés après le 10 juillet 1988 qui ne touchent pas de commission.
(5 lignes)

15. Affichez l'identifiant des départements qui comptent plus de 2 employés.
(6 lignes)

16. Comptez combien il y a d'employés par type de job au sein de chaque département. Affichez à chaque fois les identifiants du département et du type de job.
(19 lignes)

17. Affichez par manager, le numéro du manager ainsi que le nombre d'employés dont il est responsable. N'affichez que les managers qui sont responsables de moins de 5 employés.
(7 lignes)

18. Affichez par département, le numéro du département ainsi que la moyenne des salaires des employés engagés après le 1er janvier 1980. N'affichez que les départements dont le salaire moyen est compris entre 4000 et 10000.
(9 lignes)

19. Affichez par département, l'identifiant du département ainsi que la différence entre le plus grand et le plus petit salaire dans ce département (rebaptisez cette colonne "Salary difference").
(11 lignes)

20. Affichez par manager, l'identifiant du manager ainsi que la moyenne des salaires des employés sous sa responsabilité. Excluez les employés qui n'ont pas de manager. Écartez les managers qui n'ont pas au moins 2 personnes sous leur responsabilité. Triez le tout par ordre décroissant sur le salaire moyen.
(6 lignes)

21. Affichez le total des salaires à payer par département au sein de chaque type de job. Affichez à chaque fois les identifiants du type de job et du département. N'affichez que les groupes pour lesquels il y a au moins deux employés.
(8 lignes)

33. Affichez les noms des employés ainsi que les numéros et noms des départements auxquels ils sont affectés.
(40 lignes)

34. Afficher les noms des employés qui touchent une commission non nulle ainsi que les libellés de leur job (job_title).
(30 lignes)

35. Affichez les emails des employés des départements 2, 5 et 10 ainsi que les identifiants des localités des départements auxquels ils sont affectés.
(15 lignes)

36. Affichez les noms des employés, les noms des départements auxquels ils sont affectés ainsi que les libellés des localités où se situent ces départements.
(40 lignes)

37. Afficher les noms des employés avec les noms de leurs départements.

38. Assurez-vous que tous les employés soient affichés même s'ils ne sont affectés à aucun département !
(20 lignes)
5.2 Assurez-vous que tous les départements soient affichés même si aucun employé n'y est affecté !
(20 lignes)
5.3 Assurez-vous que tous les employés et tous les départements soient affichés au moins une fois !
(21 lignes)

39. Afficher les noms des employés engagés après le 31 décembre 1995 ainsi que la ville où ils travaillent et les libellés de leur job.
(25 lignes)

39.1 Afficher les noms et prénoms des employés avec les noms et les job_id des supérieurs
directs. Rebaptisez les colonnes respectivement 'Nom de l'employé', 'Prénom de

l'employé', 'Nom du manager' et 'Job du manager'.
(39 lignes)

39.2 Idem mais assurez-vous que tous les employés sont affichés (même ceux qui n'ont aucun supérieur direct) !
(40 lignes)

40. Afficher les noms des employés qui travaillent à Toronto ou Seattle.
(20 lignes)

41. Afficher les noms des employés ainsi que les noms des départements, villes, pays et régions correspondantes.
(40 lignes)

42. Idem mais assurez-vous que tous les employés sont affichés (même ceux qui ne sont affectés à aucun département) !
(20 lignes)

43. Afficher les noms des employés qui ont été embauchés après leurs supérieurs directs respectifs. Affichez également les noms des managers avec les dates d'embauche respectives.
Rebaptisez les colonnes.
(33 lignes)

44. Affichez le nom de famille et la date d'embauche des employés qui ont le même job que l'employé 103, en excluant toutefois l'employé 103. (4 lignes)

45. Affichez toutes les données (dont vous disposez dans la table des employés) sur les employés qui travaillent dans un département où au moins un employé contient la lettre "u" dans son nom de famille. (24 lignes)

46. Affichez le nom de famille et le salaire des employés qui sont sous la responsabilité directe de Hunold. (4 lignes)

47. Affichez le nom de famille des employés qui travaillent dans le même département que l'employé Hunold ou l'employé King (n'affichez ni Hunold ni King). (6 lignes)

48. Affichez toutes les données (dont vous disposez dans la table des employés) des employés qui touchent plus que le salaire moyen de tous les employés. Le tout trié par ordre croissant sur le salaire. (17 lignes)

49. Affichez le nom de famille, le salaire et le numéro du manager des employés qui ont le même manager que l'employé 108 tout en touchant une commission au moins égale à celle de l'employé 123. (2 lignes)

50. Affichez toutes les données (dont vous disposez dans la table des employés) sur les employés qui touchent un salaire supérieur au salaire de tous les employés dont le job est « Public Accountant » (job_id = 1). (14 lignes)

51. Affichez toutes les données (dont vous disposez dans la table des employés) sur les employés qui travaillent dans le même département que l'employé 104 tout en ayant le même manager que lui. (4 lignes)
52. Affichez le nom de famille, le numéro du job et le numéro du manager des employés qui ont le même job et le même manager qu'un employé du département 4 ou 9. (3 lignes)
53. Affichez les noms de famille des employés qui ont été embauchés avant au moins un employé du département 2. (24 lignes)
54. Affichez toutes les données (dont vous disposez dans la table des employés) sur les employés qui travaillent dans le même département tout en touchant le même salaire qu'un employé qui touchent un pourcentage de commission. (31 lignes)
55. Affichez les noms et prénoms des employés qui touchent moins que le salaire moyen de leur département. (23 lignes)
56. Affichez les noms et prénoms des employés, ainsi que les noms des départements où ils travaillent, de tous les employés qui touchent le même salaire et le même pourcentage de commission qu'un employé embauché après 1989. (27 lignes)
57. Affichez toutes les données (dont vous disposez dans la table des employés) sur les employés qui ont été embauchés avant leur manager direct (5 lignes)
58. Affiche les emails et les numéros de téléphone de tous les employés, le tout trié par salaire, et en cas de salaire commun, par pourcentage de commission dans l'ordre décroissant.
59. Affiche les noms et prénoms de tous les employés dans une seule colonne nommée « Nom et prénom ». Renseigne-toi sur la fonction `CONCAT` ici : https://sql.sh/fonctions/concat
60. Affiche toutes les infos sur les employés qui gagnent entre 2000 et 4000 euros.
61. Affiche les numéros des départements des employés qui ne touchent pas de commission et qui ont la fonction de programmeur (`job_id = 9`). Évite les duplicata à l'affichage. Trie les résultats du plus bas au plus haut salaire.
62. Affiche, par fonction (`job_id`), l'identifiant de la fonction ainsi que la moyenne des salaires des employés. Trie les résultats de la plus haute moyenne à la plus basse.
63. Calcule le montant total des salaires des employés des départements 2, 5 et 10. Renomme la colonne « Total des salaires ».
64. Affiche les numéros des départements qui comptent plus de 2 employés qui touchent au moins 4000 euros.

65. Affiche le numéro de département ainsi que le nom de la ville (table LOCATIONS) de tous les départements.
66. Affiche le nom des départements qui sont situés dans la ville ayant pour code postal (`postal_code`) `98199`. Fait attention au type de la colonne `postal_code` !
67. Affiche, par ville, le numéro de la ville (`location_id`) ainsi que le nombre d'employés qui y travaillent (en fonction de leur département). Ne prends en compte que les employés qui touchent un salaire de plus de 4000 euros. Exclue les villes qui ne comptent pas au moins deux employés. Trie le tout par ordre décroissant sur le nombre d'employés.
68. Affiche le noms des employés qui gagnent plus que leur manager respectif. Affiche également la différence de salaire (nom de colonne « Différence ») et trie les résultats sur base de cette différence (ordre croissant).

Le zoo

1. Afficher tous les aliments dont le prix unitaire est supérieur à 3 €
2. Afficher le poids en kg et en grammes de tous les animaux
3. Afficher tous les animaux du plus lourd au plus léger
4. Lister toutes les spécialités de soigneur disponibles (*sans doublons !*)
5. Lister les aliments dont le nom contient le mot “frais”
6. Afficher les femelles pesant plus de 100 kg ainsi que les mâles nés après 2020
7. Afficher les lions, tigres et éléphants nés entre 2015 et 2023
8. Afficher les soins triés par coût du plus cher au moins cher puis, à coût égal, par date la plus récente
9. Afficher tous les animaux dont on ne connaît pas la date de naissance et qui on déjà fait un « Bilan santé »
10. Afficher le coût total des soins par soigneur
11. Afficher, pour chaque soin, le nom de l'animal, l'espèce, le nom du soigneur et la spécialité du soigneur
12. Afficher la liste des soigneurs avec le nombre total de soins qu'ils ont effectués (inclure soigneurs sans soin). Trie le tout sur base du nombre total de soins le plus élevé au moins élevé.
13. Afficher tous les aliments, ainsi que pour chaque aliment, le ou les animaux qui les consomment et la quantité (même si aucun animal ne consomme cet aliment)
14. Afficher les soigneurs ayant prodigué au moins 3 soins

15. Afficher tous les animaux ainsi que tous les soins qui leur ont été prodigués. Assure-toi d'afficher également les animaux qui n'ont jamais reçu de soins. Trie le tout par animal puis par date de soin
16. Afficher les animaux qui ont reçu plus de 2 soins en 2023. Trie le tout par ordre croissant du nombre de soins
17. Afficher les aliments qui ne sont dans le régime d'aucun animal
18. Lister tous les aliments et les animaux qui les consomment (y compris ceux sans régime associé)
19. Afficher les espèces dont la consommation totale d'aliments dépasse 1,000 unités
20. Affiche, par espèce, la consommation totale d'aliments
21. Affiche, par espèce, la consommation totale de chaque aliment
22. Afficher le nom des animaux ayant déjà bénéficié d'un soin
23. Lister les soigneurs n'ayant jamais prodigué de soins
24. Afficher les animaux dont le poids est supérieur à tous ceux des lions
25. Dernier soin reçu par chaque animal (date max)
26. Afficher le nom et le poids moyen des animaux par espèce
27. Afficher pour chaque animal son poids et le nombre total d'alimentations dans son régime
28. Afficher pour chaque soigneur et pour chaque mois de 2024, le nombre de soins effectués. Trie les résultats d'abord par soigneur, puis par mois.
29. Afficher les animaux qui n'ont jamais reçu aucun soin (donc aucune entrée correspondante dans la table Soin)
30. Afficher les animaux qui ne mangent jamais “Viande de boeuf”

6 Conclusion et ressources complémentaires

Ce cours a posé les bases indispensables de la structuration, de la modélisation et de la manipulation de bases de données relationnelles, en insistant sur la méthodologie, l'identification des entités, des associations, des cardinalités, la conception logique, les contraintes d'intégrité, ainsi que sur la pratique du langage SQL pour la gestion et l'exploitation quotidienne des données.

Toutefois, plusieurs sujets avancés n'ont pas été abordés dans ce cours :

- Les notions de **normalisation avancée** (3NF, BCNF, formes normales supérieures, dénormalisation et optimisation des performances) ;
- La **gestion des transactions**, des verrous (concurrence, isolation des transactions, gestion des conflits) et des mécanismes de récupération (journalisation, reprise sur incident) ;
- Les modèles **NoSQL** (bases orientées-document, clé-valeur, graphe, colonnes) qui jouent aujourd'hui un rôle clé dans le big data et les applications web à grande échelle ;
- Les techniques de **sécurité**, de gestion fine des droits d'accès, et de chiffrement des bases de données ;
- L'automatisation de l'administration, la sauvegarde/restauration, la migration de données et la montée en charge ;
- L'intégration avec les **applications modernes** (API, ORM, frameworks web/app mobiles, business intelligence).

Pour approfondir, pratiquer ou découvrir ces thèmes, voici quelques ressources et outils complémentaires :

Plateformes de pratique SQL en ligne	SQLZoo	sqlzoo.net
	W3School SQL	w3schools.com/sql
	SQL.sh	sql.sh
Outils visuels de modélisation	DB-main (UNamur)	db-main.eu
	dbdiagram	dbdiagram.io
	Draw.io	app.diagrams.net
Découverte du NoSQL	W3School MongoDB	w3schools.com/mongodb
	Neo4j Sandbox	neo4j.com/sandbox

7 Correctif

7.1 Identification des attributs facultatifs / booléens

Enoncé	*Attribut facultatif*	*Attribut booléen*
Le touriste peut payer par carte visa ou par un autre moyen.		X
Le touriste peut avoir réservé son séjour à l'hôtel via une agence de voyage.		X
On ne sait pas toujours si le touriste a réservé via une agence de voyage ou non.	X	X
Un touriste peut venir avec son propre véhicule. Dans ce cas, un montant supplémentaire lui sera facturé pour la place de parking.		X
Un touriste peut venir avec son propre véhicule. Dans ce cas, le numéro de plaque de la voiture est mémorisé.	X	
Le touriste peut être accompagné d'un animal domestique. En cas d'incendie dans le bâtiment, le propriétaire de l'hôtel doit être capable de connaître le nombre de touristes accompagnés d'un animal.		X
Le touriste peut être accompagné d'un animal domestique. Dans ce cas, on mémorise de quelle espèce il s'agit.	X	
Le touriste peut être accompagné de plusieurs animaux domestiques. Dans ce cas, on mémorise la liste des espèces accompagnants le touriste.	X	

7.2 Exercice SQL : « Le Zoo »

1. Afficher tous les aliments dont le prix unitaire est supérieur à 3 €

```sql
SELECT nomAliment, coutUnitaire
FROM Aliment
WHERE coutUnitaire > 3;
```

2. Afficher le poids en kg et en grammes de tous les animaux

```sql
SELECT nomAnimal, poids AS poids_kg, (poids * 1000) AS poids_g
FROM Animal;
```

3. Afficher tous les animaux du plus lourd au plus léger

```sql
SELECT nomAnimal, poids
FROM Animal
ORDER BY poids DESC;
```

4. Lister toutes les spécialités de soigneur disponibles (sans doublons !)

```sql
SELECT DISTINCT specialite
FROM Soigneur;
```

5. Lister les aliments dont le nom contient le mot « frais »

```sql
SELECT nomAliment
FROM Aliment
WHERE nomAliment LIKE '%frais%';
```

6. Afficher les femelles pesant plus de 100 kg ainsi que les mâles nés après 2020

```sql
SELECT nomAnimal, sexe, poids, dateNaissance
FROM Animal
WHERE (sexe = 'F' AND poids > 100)
OR (sexe = 'M' AND dateNaissance > '2020-01-01');
```

7. Afficher les lions, tigres et éléphants nés entre 2015 et 2023

```sql
SELECT nomAnimal, nomEspece, dateNaissance
FROM Animal
WHERE nomEspece IN ('Lion', 'Tigre', 'Éléphant')
AND dateNaissance BETWEEN '2015-01-01' AND '2023-12-31';
```

8. Afficher les soins triés par coût du plus cher au moins cher puis, à coût égal, par date la plus récente

```sql
SELECT *
FROM Soin
ORDER BY cout DESC, dateSoin DESC;
```

9. Afficher tous les animaux dont on ne connaît pas la date de naissance et qui ont déjà fait un « Bilan santé »

```sql
SELECT DISTINCT Animal.nomAnimal
FROM Animal
JOIN Soin ON Animal.numPuce = Soin.numPuceAnimal
WHERE Animal.dateNaissance IS NULL
AND Soin.description = 'Bilan santé';
```

Attention : le `DISTINCT` est très important puisqu'un animal peut avoir fait plusieurs bilans.

10. Afficher le coût total des soins par soigneur

```sql
SELECT nomSoigneur, SUM(cout) AS total_soins
FROM Soin
GROUP BY nomSoigneur;
```

11. Afficher, pour chaque soin, le nom de l'animal, l'espèce, le nom du soigneur et la spécialité du soigneur

```sql
SELECT Animal.nomAnimal, Animal.nomEspece, Soin.dateSoin,
Soigneur.nomSoigneur, Soigneur.specialite, Soin.description, Soin.cout
FROM Soin
JOIN Animal ON Soin.numPuceAnimal = Animal.numPuce
JOIN Soigneur ON Soin.nomSoigneur = Soigneur.nomSoigneur
ORDER BY Soin.dateSoin DESC;
```

12. Afficher la liste des soigneurs avec le nombre total de soins qu'ils ont effectués (inclure soigneurs sans soin). Trie le tout sur base du nombre total de soins le plus élevé au moins élevé.

```sql
SELECT Soigneur.nomSoigneur, COUNT(Soin.dateSoin) AS nb_soins
FROM Soigneur
LEFT JOIN Soin ON Soigneur.nomSoigneur = Soin.nomSoigneur
GROUP BY Soigneur.nomSoigneur
ORDER BY nb_soins DESC;
```

Attention : dans ce cas-ci, `COUNT(*)` n'aurait pas été correct ! Par contre, si l'on ne prenait pas en compte les soigneurs sans soin, la requête aurait pu devenir :

```sql
SELECT nomSoigneur, COUNT(*) AS nb_soins
FROM Soin
GROUP BY nomSoigneur
ORDER BY nb_soins DESC;
```

13. Afficher tous les aliments, ainsi que pour chaque aliment, le ou les animaux qui les consomment et la quantité (même si aucun animal ne consomme cet aliment)

```sql
SELECT Aliment.nomAliment, Animal.nomAnimal, Regime.quantite
FROM Aliment
LEFT JOIN Regime ON Aliment.codeBarreAliment = Regime.aliment
LEFT JOIN Animal ON Animal.numPuce = Regime.numPuceAnimal
ORDER BY Aliment.nomAliment, Animal.nomAnimal;
```

14. Afficher les soigneurs ayant prodigué au moins 3 soins

```sql
SELECT nomSoigneur, COUNT(*) AS nb_soins
FROM Soin
GROUP BY nomSoigneur
HAVING COUNT(*) >= 3;
```

15. Afficher tous les animaux ainsi que tous les soins qui leur ont été prodigués. Assure-toi d'afficher également les animaux qui n'ont jamais reçu de soins. Trie le tout par animal puis par date de soin

```sql
SELECT Animal.nomAnimal, Soin.dateSoin, Soin.description, Soin.cout
FROM Animal
LEFT JOIN Soin ON Animal.numPuce = Soin.numPuceAnimal
ORDER BY Animal.nomAnimal, Soin.dateSoin;
```

16. Afficher les animaux qui ont reçu plus de 2 soins en 2023. Trie le tout par ordre croissant du nombre de soins

```sql
SELECT numPuceAnimal, COUNT(*)
FROM Soin
WHERE Soin.dateSoin BETWEEN '2023-01-01' AND '2023-12-31'
GROUP BY numPuceAnimal
HAVING COUNT(*) > 2
```

17. Afficher les aliments qui ne sont dans le régime d'aucun animal

```sql
SELECT Aliment.nomAliment
FROM AlimentLEFT
JOIN Regime ON Aliment.codeBarreAliment = Regime.aliment
WHERE Regime.aliment IS NULL;
```

18. Lister tous les aliments et les animaux qui les consomment (y compris ceux sans régime associé)

```sql
SELECT Aliment.nomAliment, Animal.nomAnimal, Regime.quantite
FROM Aliment
LEFT JOIN Regime ON Aliment.codeBarreAliment = Regime.aliment
LEFT JOIN Animal ON Animal.numPuce = Regime.numPuceAnimal
ORDER BY Aliment.nomAliment, Animal.nomAnimal;
```

19. Afficher les espèces dont la consommation totale d'aliments dépasse 1,000 unités

```sql
SELECT Animal.nomEspece, SUM(Regime.quantite) AS total_conso
FROM Regime
JOIN Animal ON Regime.numPuceAnimal = Animal.numPuce
GROUP BY Animal.nomEspece
HAVING SUM(Regime.quantite) > 1000;
```

20. Affiche, par espèce, la consommation totale d'aliments

```sql
SELECT Animal.nomEspece, SUM(Regime.quantite) AS total_consommation
FROM Regime
JOIN Animal ON Regime.numPuceAnimal = Animal.numPuce
GROUP BY Animal.nomEspece
ORDER BY total_consommation DESC;
```

21. Affiche, par espèce, la consommation totale de chaque aliment

```sql
SELECT Animal.nomEspece, Aliment.nomAliment, SUM(Regime.quantite) AS
total_consommation
FROM Regime
JOIN Animal ON Regime.numPuceAnimal = Animal.numPuce
JOIN Aliment ON Regime.aliment = Aliment.codeBarreAliment
GROUP BY Animal.nomEspece, Aliment.nomAliment
ORDER BY Animal.nomEspece, total_consommation DESC;
```

22. Afficher le nom des animaux ayant déjà bénéficié d'au moins un soin

Ici, plusieurs solutions possibles. D'abord, en utilisant les jointures :

```sql
SELECT DISTINCT Animal.nomAnimal
FROM Animal
JOIN Soin ON Animal.numPuce = Soin.numPuceAnimal;
```

Mais on peut également utiliser les groupements :

```sql
SELECT numPuceAnimal, COUNT(*)
FROM Soin
GROUP BY numPuceAnimal
HAVING COUNT(*) >= 1
```

23. Lister les soigneurs n'ayant jamais prodigué de soins

Ici encore, plusieurs solutions. D'abord en utilisant les sous-requêtes :

```sql
SELECT nomSoigneur
FROM Soigneur
WHERE nomSoigneur NOT IN (SELECT DISTINCT nomSoigneur FROM Soin);
```

Mais on peut aussi utiliser les jointures :

```sql
SELECT Soigneur.nomSoigneur
FROM Soigneur
LEFT JOIN Soin ON Soigneur.nomSoigneur = Soin.nomSoigneur
WHERE Soin.nomSoigneur IS NULL;
```

24. Afficher les animaux dont le poids est supérieur à tous ceux des lions

```sql
SELECT nomAnimal, poids
FROM Animal
WHERE poids > ALL (SELECT poids FROM Animal WHERE nomEspece = 'Lion');
```

25. Afficher le dernier soin reçu par chaque animal (date max)

```sql
SELECT A.nomAnimal, S.description, S.dateSoin
FROM Animal A
JOIN Soin S ON S.numPuceAnimal = A.numPuce
WHERE S.dateSoin = (
  SELECT MAX(dateSoin)
  FROM Soin
  WHERE numPuceAnimal = A.numPuce
);
```

26. Afficher le nom et le poids moyen des animaux par espèce

```sql
SELECT nomEspece, AVG(poids) AS poids_moyen
FROM Animal
GROUP BY nomEspece
```

27. Afficher pour chaque animal son poids et le nombre total d'alimentations dans son régime

```sql
SELECT numPuceAnimal, COUNT(*) AS nb_aliments
FROM Regime
GROUP BY numPuceAnimal
ORDER BY nb_aliments DESC;
```

28. Afficher pour chaque soigneur et pour chaque mois de 2024, le nombre de soins effectués. Trie les résultats d'abord par soigneur, puis par mois.

```sql
SELECT nomSoigneur, EXTRACT(MONTH FROM dateSoin) AS mois, COUNT(*) AS
nb_soins
FROM Soin
WHERE dateSoin BETWEEN '2024-01-01' AND '2024-12-31'
GROUP BY nomSoigneur, mois
ORDER BY nomSoigneur, mois;
```

Plus d'info sur **EXTRACT** ici : https://www.w3schools.com/sql/func_mysql_extract.asp

29. Afficher les animaux qui n'ont jamais reçu aucun soin

```sql
SELECT Animal.nomAnimal
FROM Animal
LEFT JOIN Soin ON Animal.numPuce = Soin.numPuceAnimal
WHERE Soin.numPuceAnimal IS NULL;
```

30. Afficher le nom des animaux qui ne mangent jamais de « Viande de boeuf »

```sql
SELECT nomAnimal
FROM Animal
WHERE numPuce NOT IN (
   SELECT numPuceAnimal
   FROM Regime
   WHERE aliment = (
      SELECT codeBarreAliment
      FROM Aliment
      WHERE nomAliment = 'Viande de boeuf')
);
```

8 Bibliographie

Bases de données et modèles de calcul, Outils et méthodes pour l'utilisateur.
HAINAUT, Jean-Luc, 2005, DUNOD.

Hainaut, Jean-Luc. *Bases de données: concepts, utilisation et développement*. 5e éd, Dunod, 2022.

Conception des bases de données I, Introduction aux bases de données relationnelles [en ligne].
CROZAT, Stéphane, 2017 [consulté le 29 janvier 2020].
Disponible sur https://stph.scenari-community.org/bdd/bdd1.pdf