Premières réflexions sur la didactique des bases de données Rapport de recherche

Emmanuel Desmontils

Premières réflexions sur la didactique des bases de données

Rapport de recherche

Emmanuel Desmontils

Laboratoire des sciences du numérique de Nantes (LS2N)

Institut de Recherche sur l'Enseignement des Mathématiques (IREM) des Pays de la Loire, **Nantes Université, emmanuel.desmontils@univ-nantes.fr**

Résumé : Dans ce rapport de recherche, nous proposons quelques réflexions autour de la didactique des bases de données. Nous montrons qu'une bonne compréhension du modèle de la base de données est important aussi bien pour l'élève que pour l'enseignant. Nous commentons le programme de NSI sur SQL. Nous proposons aussi des situations didactiques autour de l'enseignement de SQL.

Mots-Clés : Didactique, Bases de données, Modèle relationnel, SQL, Modélisation conceptuelle, NSI

Version : 1.2

1. Introduction

L'enseignement des bases de données est présent dans les curriculum universitaires depuis très longtemps. Le plus souvent, dans les modules d'initiation, les notions présentées sont : les principes des SGBD, la modélisation conceptuelle de base (souvent le modèle EAP de Merise, parfois les diagrammes de classe UML), l'algèbre relationnel, le modèle relationnel et, bien évidemment, SQL (recherche, mais aussi construction de bases de données). Pour les opérateurs relationnels, sont abordés : la sélection, la projection, le tri, les opérateurs d'agrégation, les regroupements (Group By et Having) et, assez souvent, les requêtes imbriquées (sous-requêtes, synchronisées ou pas). Il arrive aussi que soient abordés l'utilisation d'une architecture trois-tiers basée sur un serveur Web, une application (en Python par exemple) et une base de données. Ces éléments sont présents dans le module BD1 en L1 de la licence Informatique de Nantes [NU-Li-Info2021a] (page 27). Des notions plus avancées sont présentes dans le second module BD2 en L3 [NU-Li-Info2021b] (page 18).

X12I030	**Bases de données 1**
Lieu d'enseignement	Lombarderie
Niveau	Licence
Semestre	2
Responsable de l'UE	BOUDIN FLORIAN
Volume horaire total	**TOTAL :** 39.6h Répartition : **CM :** 8h **TD :** 16h **CI :** 0h **TP :** 12h **EAD :** 3.6h
Place de l'enseignement	
UE pré-requise(s)	Informatique (X11I010) Compléments Mathématiques et informatique (X11X010)
Parcours d'études comprenant l'UE	L1 MIP : Informatique,L1 MIP : Maths Informatique,L1 MIP : Mathématiques,L1 MIP : CMI Maths Informatique,L1 A2 ACCOMP-Li Informatique ,L1 A2 ACCOMP-Li Mathématiques ,L1 A2 ACCOMP-Li Maths Informatique,L1 MIP : Informatique - option santé
Evaluation	
Pondération pour chaque matière	Bases de données 1 **100%**
Obtention de l'UE	La note de contrôle continu peut contenir une ou plusieurs composantes pratiques et éventuellement une composante distancielle.
Programme	
Objectifs (résultats d'apprentissage)	A l'issue de ce module, l'étudiant saura: • Comprendre ce qu'un modèle de données • Être capable de concevoir le modèle conceptuel entité-association d'une base de données • Savoir représenter un modèle conceptuel entité-association en UML (Unified Modeling Langage) • Être capable de concevoir un schéma relationnel de base de données à partir d'un modèle entité-association • Connaître l'algèbre relationnelle • Maîtriser le langage SQL dans ses trois facettes, langage de manipulation de données, langage de définition de données et langage de contrôle de données • Comprendre une architecture trois-tiers basée sur un serveur Web, une application et une base de données
Contenu	Au cours de ce module seront présentés les points suivants: Notion de Base de Données (BD) et de Système de Gestion de BD Algèbre relationnelle Définition et manipulation de données en SQL Notion de vue Interrogation d'une base distante en PHP.
Méthodes d'enseignement	
Langue d'enseignement	Français
Bibliographie	

Les bases de données sont, depuis 2019, un chapitre à part entière du programme de NSI. Voici la présentation au BO [BO2019b] (page 5) :

> Le développement des traitements informatiques nécessite la manipulation de données de plus en plus nombreuses. Leur organisation et leur stockage constituent un enjeu essentiel de performance.
>
> Le recours aux bases de données relationnelles est aujourd'hui une solution très répandue. Ces bases de données permettent d'organiser, de stocker, de mettre à jour et d'interroger des données structurées volumineuses utilisées simultanément par différents programmes ou différents utilisateurs. Cela est impossible avec les représentations tabulaires étudiées en classe de première.
>
> Des systèmes de gestion de bases de données (SGBD) de très grande taille (de l'ordre du pétaoctet) sont au centre de nombreux dispositifs de collecte, de stockage et de production d'informations.
>
> L'accès aux données d'une base de données relationnelle s'effectue grâce à des requêtes d'interrogation et de mise à jour qui peuvent par exemple être rédigées dans le langage SQL (Structured Query Language). Les traitements peuvent conjuguer le recours au langage SQL et à un langage de programmation.
>
> Il convient de sensibiliser les élèves à un usage critique et responsable des données.

Le programme détaillé est le suivant [BO2019b] (pages 5-6) :

Contenus	**Capacités attendues**	**Commentaires**
Modèle relationnel : relation, attribut, domaine, clef primaire, clef étrangère, schéma relationnel.	Identifier les concepts définissant le modèle relationnel.	Ces concepts permettent d'exprimer les contraintes d'intégrité (domaine, relation et référence).
Base de données relationnelle.	Savoir distinguer la structure d'une base de données de son contenu. Repérer des anomalies dans le schéma d'une base de données.	La structure est un ensemble de schémas relationnels qui respecte les contraintes du modèle relationnel. Les anomalies peuvent être des redondances de données ou des anomalies d'insertion, de suppression, de mise à jour. On privilégie la manipulation de données nombreuses et réalistes.
Système de gestion de bases de données relationnelles.	Identifier les services rendus par un système de gestion de bases de données relationnelles : persistance des données, gestion des accès concurrents, efficacité de traitement des requêtes, sécurisation des accès.	Il s'agit de comprendre le rôle et les enjeux des différents services sans en détailler le fonctionnement.
Langage SQL : requêtes d'interrogation et de mise à jour d'une base de données.	Identifier les composants d'une requête. Construire des requêtes d'interrogation à l'aide des clauses du langage SQL : SELECT, FROM, WHERE, JOIN. Construire des requêtes d'insertion et de mise à jour à l'aide de : UPDATE, INSERT, DELETE.	On peut utiliser DISTINCT, ORDER BY ou les fonctions d'agrégation sans utiliser les clauses GROUP BY et HAVING.

On constate que l'enseignement s'axe sur la compréhension et l'usage des bases de données en général. Sur l'aspect SGBD, le BO propose d'amener les élèves à comprendre ce qu'est une base de données et un système de gestion de bases de données. Il propose une présentation du modèle relationnel, mais pas son fondement (algèbre relationnelle). Pour la partie SQL, les élèves abordent la modification des données : ajout, modification et suppression de données. Par contre, ils n'abordent pas la modification du modèle. Ils abordent aussi les notions de bases de la recherche (sélection et projection) et la jointure. Sur cette dernière, il n'est pas précisé le type de jointure à présenter. Est-ce seulement la θ-jointure ([inner] join on) ? Ou peut-on aborder la jointure naturelle, la semi-jointure voire la jointure externe ? De mêmes, il n'est pas fait mention de la présentation des sous-requêtes. Par contre, il est explicitement indiqué qu'il ne sera pas abordé les regroupements. Pour terminer, il n'est pas indiqué l'aspect programmation avec les bases de données. L'absence d'indication sur ce sujet est assez frustrant. En effet, il est plus agréable, dans le cadre des projets, de pouvoir faire au moins une petite application Web pour rendre plus réaliste le projet.

L'apparition de la spécialité NSI en première et terminale a relancé les réflexions sur la didactique de l'informatique déjà commencées à l'occasion des options informatique apparues il y a quelques années [Flu2019]. Les travaux en didactique sont principalement développés en algorithmique et programmation [Ars1988] [BaB2001] (et les nombreuses publications de la conférence Didapro par exemple). Pour la didactique des bases de données, assez peu de travaux sont proposés, essentiellement sur les erreurs en SQL ([MAF2021] et [Sme1995] par

exemple).

L'objectif général de ce rapport est de proposer des réflexions générales, issues d'une expérience de plus de 25 ans en enseignement des bases de données et de la modélisation de données (de la L1 au M2), sur la didactique des bases de données et de proposer quelques éléments de transposition didactique [Che85]. Dans la suite de ce travail, nous aborderons d'abord des réflexions générales sur la didactique des bases de données. Puis, nous montrerons l'importance d'un travail en profondeur sur le modèle de la base de données. Nous aborderons ensuite quelques réflexions autours du langage SQL avec, en particulier, le problème de présentation de la jointure. Nous évoquerons quelques situations didactiques caractéristiques de ce domaine en SQL et en projet de bases de données avant de faire une brève analyse des ouvrages à destination des élèves.

Avant d'entrer dans le vif du sujet, présentons l'exemple fil-rouge qui servira à illustrer une grande partie de ce document.

2. Exemple fil rouge

Dans la suite de ce document, nous nous baserons sur l'exemple de [CPTT08] (p.62) dont le contexte est le suivant :

> On souhaite créer une base de données destinée à la gestion des pays, des fleuves, des espaces maritimes (mers et océans). Chaque pays est connu par un nom, une superficie, un nombre d'habitants, le liste des pays qui ont une frontière commune avec lui et la liste des fleuves qui le traversent. Un fleuve est connu par son nom, sa longueur, l'espace maritime dans lequel il se jette, le nom du pays dans lequel il prend sa source, la liste des pays qu'il traverse et la distance parcourue dans chacun de ces pays. Un espace maritime est connu par un nom, un type (mer ou océan), la liste des pays qu'il côtoie et la liste des fleuves qui s'y jettent.

Nous avons ajouté à cela, le nom du pays par lequel le fleuve se jette dans l'espace maritime. Une analyse du contexte nous permet de proposer les dépendances fonctionnelles suivantes :

```
nom-f -> longueur
nom-f -> nom-e, nom-p-estuaire
nom-f -> nom-p-source
nom-p -> superficie, nbHab
nom-e -> type
nom-p, nom-f -> distance
nom-p-estuaire -> nom-p
nom-p-source -> nom-p

Cotoie (nom-p, nom-e)
```

La relation universelle est associée à une relation issue de l'analyse. Ce qui donne le premier modèle relationnel EM-R0 :

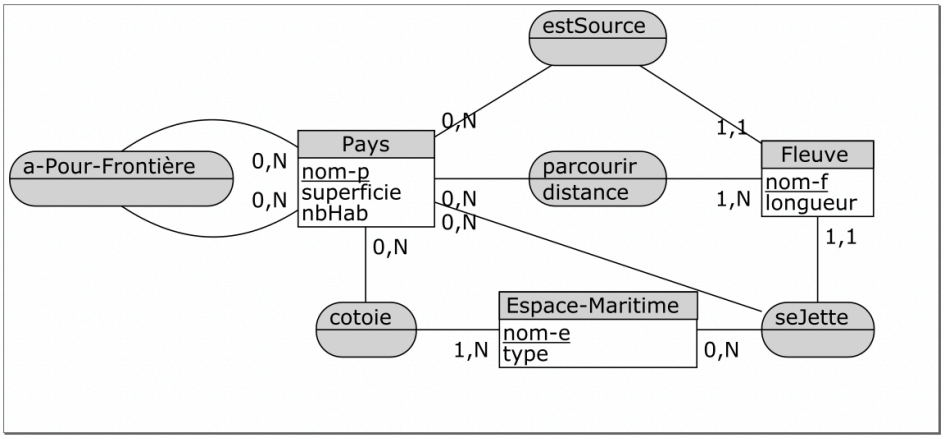
```
EM(nom-f, longueur, nom-e, type, nom-p, superficie, nbHab, nom-p-estuaire, nom-p-source, distance)
    clé primaire : nom-f

Cotoie(nom-p, nom-e)
    clé primaire : nom-p, nom-e
```

Bien évidemment, ce modèle n'a pas de bonnes propriétés (seulement en 1ere forme normale). Il convient donc de normaliser le modèle en appliquant par exemple la méthode de synthèse ([AnV2001], [CPTT2008], [Gar2005], [GUW2008]). On obtient alors le schéma relationnel EM-R1 suivant :

```
R1/Fleuve (nom-f, longueur, nom-e, nom-p-estuaire, nom-p-source)
    clé primaire : nom-f
    clé étrangère : nom-p-estuaire → R5(nom-p-estuaire)
    clé étrangère : nom-p-source → R5(nom-p-source)

R2/Pays (nom-p, superficie, nbHab)
    clé primaire : nom-p

R3/Espace-maritime (nom-e, type)
    clé primaire : nom-e

R4/parcourir (nom-p, nom-f, distance)
    clé primaire : nom-p, nom-f

R5 (nom-p-estuaire, nom-p)
    clé primaire : nom-p-estuaire
```

```
    clé étrangère : nom-p → Pays(nom-p)

R6 (nom-p-source, nom-p)
    clé primaire : nom-p-source
    clé étrangère : nom-p → Pays(nom-p)

Cotoie (nom-p, nom-e)
    clé primaire : nom-p, nom-e
    clé étrangère : nom-p → R2(nom-p)
    clé étrangère : nom-e → R3(nom-e)
```

Le modèle EM-C0 ci dessous est une solution possible qui nous servira de base à nos propos (fait avec DBConcept).



A cela, on ajoute trois contraintes :

- C1 : $\forall (p_1, p_2) \in Pays \times Pays, A-Pour-Frontière(p_1, p_2) \implies p_1 \neq p_2$
- C2 : $\forall f \in Fleuve, \forall p \in Pays, \forall e \in Espace-Maritime, seJette(f, e, p) \implies parcours(f, p) \wedge cotoie(p, e)$
- C3 : $\forall (f, p) \in Fleuve \times Pays, estSource(p, f) \implies \exists d, parcours(p, f, d)$

Le modèle relationnel induit ([AnV2001]), que nous appellerons EM-R2, est le suivant :

```
Pays (nom-p, superficie, nbHab)
    clé primaire : nom-p

Fleuve (nom-f, longueur, nom-p)
    clé primaire : nom-f
    clé étrangère : nom-p → Pays(nom-p)

Espace-Maritime (nom-e, type)
    clé primaire : nom-e

seJette (nom-f, nom-p, nom-e)
    clé primaire : nom-f
    clé étrangère : nom-f → Fleuve(nom-f)
    clé étrangère : nom-p → Pays(nom-p)
    clé étrangère : nom-e → Espace-Maritime(nom-e)

cotoie (nom-p, nom-e)
    clé primaire : nom-p, nom-e
    clé étrangère : nom-p → Pays(nom-p)
    clé étrangère : nom-e → Espace-Maritime(nom-e)

parcourir (nom-p, nom-f, distance)
    clé primaire : nom-p, nom-f
    clé étrangère : nom-p → Pays(nom-p)
    clé étrangère : nom-f → Fleuve(nom-f)

a-Pour-Frontiere (nom-p, nom-p1)
    clé primaire : nom-p, nom-p1
    clé étrangère : nom-p → Pays(nom-p)
```

```
    clé étrangère : nom-p1 → Pays(nom-p)
```

Il est possible aussi de proposer une version moins bien structurée (que nous appellerons EM-R3) :

```
Pays (nom, superficie, nbHab)
    clé primaire : nom

Fleuve (nom, longueur, #ns)
    clé primaire : nom
    clé étrangère : ns → Pays(nom)

Espace-Maritime (nom, type)
    clé primaire : nom-e

Estuaire(fl, p)
    clé primaire : fl
    clé étrangère : fl → Fleuve(nom)
    clé étrangère : p → Pays(nom)

seJette (f, e)
    clé primaire : f
    clé étrangère : f → Fleuve(nom)
    clé étrangère : e → Espace-Maritime(nom)

cotoie (pays, espace-maritime)
    clé primaire : pays, espace-maritime
    clé étrangère : pays → Pays(nom)
    clé étrangère : espace-maritime → Espace-Maritime(nom)

parcourir (nom_pays, nom_fleuve, distance)
    clé primaire : nom_pays, nom_fleuve
    clé étrangère : nom_pays → Pays(nom)
    clé étrangère : nom_fleuve → Fleuve(nom)

a-Pour-Frontiere (p2, p1)
    clé primaire : p1, p2
    clé étrangère : p1 → Pays(nom)
    clé étrangère : p2 → Pays(nom)
```

Cette dernière version, bien que correcte du point de vue de la forme normale (elle est aussi en 3e forme normale Boyce-Codd-Kent), présente certains inconvénients :

- Les attributs "nom" présents dans plusieurs tables ne représentent pas la même information (nom du fleuve, nom du pays, nom de l'espace maritime).
- Plusieurs attributs représentent la même information (par exemple nom, nom_pays, p1, p2, p, pays représentent le nom d'un pays.
- Certains attributs ne sont pas suffisamment signifiants (p pour pays, ns pour le nom du pays source par exemples).
- Le concept d'estuaire est éclaté en deux relations au lieu d'une (SeJette est devenue SeJette+Estuaire).

Cette version permettra de mettre en valeur certaines difficultés didactiques qui seront évoquées plus loin.

3. Vers une didactique des bases de données

En didactique de l'informatique, les travaux se sont, à juste titre, essentiellement focalisés sur l'algorithmique et la programmation. Nous allons donc nous intéresser ici à la thématique des bases de données en essayant de répondre à certaines questions. Comment considérer les bases de données dans le cadre de la didactique de l'informatique ? Qu'est-ce qui différencie l'enseignement des bases de données de l'enseignement mieux étudié de l'algorithmique et programmation ?

Les travaux sur la pensée informatique [Win2006, Sel2013], essentiellement dans le cadre de l'algorithmique et la programmation ont permis de proposer un certain nombre de compétences [DSS2017]. En bases de données, les compétences proposées pour la programmation sont mobilisables à différents niveaux :

- **Evaluer** : cette compétence est évidemment assez facile à mobiliser, car, nous le verrons plus loin, la compréhension et l'évaluation d'une requête fait partie des compétences élémentaires ;

- **Anticiper** : là aussi, la mise en place d'une requête SQL demande d'effectuer une composition d'opérateurs ;
- **Décomposer** : dès qu'une requête devient complexe, il est nécessaire de la décomposer ; cette décomposition peut avoir comme résultat une requête à un seul niveau (pas toujours facile à obtenir et à comprendre) ou une requête comprenant des sous-requêtes (parfois coûteuses et pas toujours faciles à appréhender et à manipuler) ;
- **Généraliser** : il n'est pas possible de mettre en place des généralisations (en tout cas par rapport au programme de NSI), cependant, il est possible d'identifier des motifs de requête pour des situations assez classiques ; on peut considérer tout de même que de construire une requête sans exemple dans l'instance relève de la généralisation.
- **Abstraire** : comme pour la généralisation, il n'y pas pas vraiment d'abstraction en bases de données, du moins pour le programme NSI en ce qui concerne la conception. Cependant, les élèves sont amenés à manipuler un modèle abstrait (modèle relationnel) avec un langage déclaratif (SQL).

Remarque : plus tard, la programmation de bases de données en PL/SQL (mise en place de fonctions, procédures, triggers) permettra de développer davantage les compétences de généralisation et d'abstraction.

Le [BO2019b] (pages 1 & 2) nous dit :

> L'enseignement de spécialité de numérique et sciences informatiques permet de développer les compétences suivantes, constitutives de la pensée informatique :
>
> - **analyser et modéliser** un problème en termes de flux et de traitement d'informations ;
> - **décomposer un problème** en sous-problèmes, **reconnaître des situations déjà analysées** et **réutiliser des solutions** ;
> - **concevoir des solutions algorithmiques** ;
> - **traduire un algorithme dans un langage de programmation**, en spécifier les interfaces et les interactions, comprendre et réutiliser des codes sources existants, développer des processus de mise au point et de validation de programmes ;
> - **mobiliser les concepts et les technologies utiles** pour assurer les fonctions d'acquisition, de mémorisation, de traitement et de diffusion des informations ;
> - **développer des capacités d'abstraction et de généralisation**.
>
> Cet enseignement se déploie en mettant en activité les élèves, sous des formes variées qui permettent de développer des compétences transversales :
>
> - faire preuve **d'autonomie**, **d'initiative** et de **créativité** ;
> - **présenter un problème ou sa solution**, **développer une argumentation** dans le cadre d'un débat ;
> - **coopérer** au sein d'une équipe dans le cadre d'un projet ;
> - **rechercher** de l'information, **partager** des ressources ;
> - faire un **usage responsable et critique** de l'informatique.

Nous retrouvons en partie les compétences de [DSS2017].

La particularité en NSI pour les bases de données, c'est que le modèle de données (modèle relationnel) est imposé. De plus, ce modèle ne comporte pas toutes les informations issues d'une analyse, ce modèle étant moins riche qu'un modèle conceptuel. C'est modèle formel strict en graphe basé sur ma théorie des ensembles (qu'ils ne voient pas en mathématiques), finalement assez différents des modèles de données qu'ils manipulent dans les autres chapitres du cours de NSI. Travailler avec les bases de données demande alors un travail particulier de compréhension du modèle (du schéma) dont la structure, du fait de la normalisation, n'est pas toujours très naturelle.

Donc, il pose des difficultés à s'y adapter. Les élèves doivent avoir des capacités à "digérer" un modèle et les outils qui sont à disposition pour le manipuler. Il doivent tout de même mettre en oeuvre des capacités d'abstraction et de généralisation pour mettre en place des requêtes sans pour autant avoir une instance à leur disposition. Comprendre le modèle relationnel demande un gros effort d'abstraction pour pouvoir l'exploiter judicieusement par rapport au besoin fonctionnel d'une requête.

Une difficulté particulière en bases de données est le langage utilisé pour exploiter et manipuler les données. C'est un nouveau langage de "programmation" bien sûr, mais en plus c'est un nouveau paradigme : langage déclaratif ! Ce paradigme est nouveau, car les paradigmes vus sont la programmation impérative majoritairement, mais aussi objet et fonctionnelle dans la partie « Langage et programmation » en terminale [BO2019b]. Par conséquent, ils n'abordent pas ce type de paradigme ailleurs. SQL n'est pas très complexe en soit, surtout sur le programme de terminale, mais ce changement de paradigme peut troubler. De plus, peu de travaux en didactique existent sur ce paradigme. Par ailleurs, la manipulation de SQL n'est pas toujours aisé, car les outils proposent peu d'outils pour analyser la trace d'exécution d'une requête.

4. Compréhension du modèle

4.1. Nécessité de proposer un modèle bien construit

Dans beaucoup d'ouvrages, cours en ligne, articles de recherche, le schéma relationnel de la base de données donnée en exemple ou en exercice est souvent négligé. Les schémas sont simplistes ou pas normalisés.

La qualité du schéma impacte la compréhension du modèle par l'élève et ses risques d'erreur. Les choix de modélisation et la qualité du modèle généré sont un facteur important dans la difficulté de l'exercice et l'accessibilité de l'exemple.

Tout d'abord, le modèle relationnel doit être au moins en 3e forme normale pour éviter les anomalies (sauf à vouloir les illustrer bien évidemment). Mais, cela ne suffit généralement pas. Il faut éviter des erreurs de nomage. Attributs comme noms de table doivent être significatifs par rapport au contexte choisi. Il ne doit pas y avoir d'attributs différents avec des noms synonymes. Il est alors préférable d'utiliser des préfixes. Par exemple, le nom du pays sera nom-p ou nom-pays alors que le nom du fleuve sera nom-f ou nom-fleuve. Ainsi, si deux attributs utilisent le même terme, alors il s'agit nécessairement de clé (primaire ou étrangères), et donc, il pourra y avoir une jointure par exemple. Il est important de bien prendre l'habitude de normaliser les bases de données pour faciliter la programmation par exemple. Il faut donc sensibiliser les élèves au bon choix des termes utilisés pour tables et attributs.

L'idéal, pour s'assurer d'une bonne architecture (par exemple en évitant les ilôts) est de passer par une modélisation conceptuelle. Commencer la construction d'une base de données par un modèle EAP (Merise) normalisé permettra d'obtenir facilement un modèle relationnel normalisé lui-aussi. Le passage du modèle conceptuel EAP vers le modèle relationnel est effectivement très simple [AnV2001]. Tous les outils proposant la construction de modèle EAP (DBConcept https://dbconcept.tuxfamily.org/, Looping-MCD https://www.looping-mcd.fr/ par exemples) permettent de générer un schéma relationnel.

Le modèle conceptuel ainsi que les dépendances fonctionnelles et les formes normales ne sont évidemment pas au programme du NSI. Il n'est donc pas évident d'expliquer aux élèves ce qui différentie un bon modèle d'un mauvais. Une première approche est de partir de la résolution des anomalies. La table universelle comporte des anomalies. L'enseignant montre alors une version à plusieurs tables (en 3e forme normale) et montre que les anomalies n'existent plus. Il montre aussi, en temps voulu, qu'il est possible de retrouver la table universelle à l'aide de jointures. Il montre ainsi que les deux modèles sont équivalents, mais que celui à plusieurs tables est préférables.

De plus, en prenant l'habitude concevoir un schéma de base de données en passant par un modèle conceptuel, il est possible de montrer aux élèves qu'il y a globalement deux types principaux de tables : les tables-entité et les tables-lien.

Définition : **les tables-entités** (issues des entités du modèle EAP) proposent des informations relativement indépendantes et dont la clé primaire est uniquement composée d'attributs propres à la table. Ces tables contiennent éventuellement des clés étrangères, qui ne font pas partie de la clé primaire. Ces clés étrangères sont un premier lien avec d'autres tables.

Définition : **les tables-liens** (issues des associations du modèle EAP) proposent d'associer entre elles des tables-entités. Leur clé primaire est uniquement composée de clés étrangères.

Dans le modèle relationnel EM-R2, les tables Pays, Fleuve, Espace-Maritime sont des tables-entité alors que les tables côtoie, a-Pour-Frontière, parcourir et seJette sont des tables-lien. Remarquons la convention de nommage qui fait commencer les nom de tables-entité par une majuscule alors que les tables-lien commencent par une minuscule. C'est une convention intéressante pour aider les élèves à appréhender le modèle.

Nous verrons dans la suite que certaines tables-entité ont une clé primaire contenant une clé étrangère : celles issues d'une entité faible ou produites par héritage dans le modèle EAP. De même, certaines tables-lien ont une clé primaire qui contient un attribut qui n'est pas clé étrangère. C'est la situation où une table a été supprimée, car calculable.

Les anomalies ne suffisent pas à montrer que l'architecture de la base de donnée est correcte ou ne l'est pas. Il faut aussi montrer qu'elle répond aux attentes, c'est-à-dire qu'elle répond aux besoins fonctionnels. Pour cela, il faut habituer l'élève à « interroger la base » à travers des scénario d'usage. Par exemple, avec notre exemple fil-rouge, est-il possible de connaître la liste des fleuves qui traversent un pays (issu du contexte). La réponse est évidemment oui. La requête SQL qui répond à ce besoin est assez facile à concevoir.

Même si la base de données est bien construite (normalisée et en forme normale), sa structure peut engendrer des difficultés pour les élèves. Il existe des "motifs" de modèle conceptuel qui vont amener parfois des erreurs de conception de la base de données, mais surtout des difficultés pour concevoir certaines requêtes SQL. On trouve par exemple : l'entité faible, l'association fonction ou la disparition de table. Nous allons évoquer ces motifs dans la suite de cette section.

4.2. Associations-fonctions

Dans le modèle EAP, une fonction est une association binaire, dont une des branches porte la cardinalité 1:1. On peut donner comme exemple l'association estSource dans le fil-rouge. Lors du passage au modèle relationnel, une telle fonction produit l'apparition du côté de la table 1:1 d'une clé étrangère, clé primaire de la table côté x:N. Nom-p dans la table Fleuve est présent pour cette raison grâce à la fonction estSource.

Ce mécanisme peut amener des erreurs lors de l'exploitation de la base de données, en particulier si elle n'est pas bien documentée. En effet, la présence de nom-p peut-être ambiguë. Aussi, dans notre exemple, il sera préférable de transformer le nom-p de Fleuve en nom-p-source par exemple. Le problème ne se pose pas en cas de fonction en boucle, car le renommage de la clé migrante est alors obligatoire (pour le pas avoir deux attributs de même nom). Ces fonctions peuvent provoquer de fausses jointures (surtout si l'utilisation de la jointure naturelle a été introduite) comme avec nom-p de Fleuve et nom-p de Parcours ou nom-p de SeJette. Cela va être troublant pour les élèves.

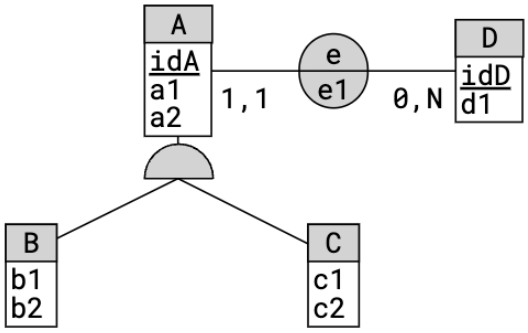
Les associations binaires en boucle posent des difficultés dans la manipulation de la base de données. En effet, ces associations engendrent des clés étrangères référençant la clé de la même table. Cette situation engendre presque inévitablement des besoins implémentés par des auto-jointures. Ce type de jointure est assez difficile à manipuler pour les élèves.

Dans le cas où il existe plusieurs fonctions entre les deux mêmes entités, il faut être attentif au nommage des clés étrangères afin de ne pas laisser d'ambiguité.

4.3. Gestion de l'héritage

Naturelle en programmation et pour certaines bases de données relationnelles-objets ou pures objets, la gestion de l'héritage est plus délicate pour le passage à un modèle relationnel pure.

Soit deux tables B et C qui héritent de A et une quatrième D liée à A par une fonction e. Comme montré ci-dessous.



Trois schémas de transformation en modèle relationnel sont possibles pour l'héritage :

1. Les tables héritent seulement de l'identifiant de la classe mère. Les informations restent associées à chacune des tables. Il y a un lien implicite entre les tables filles et la table mère. Cette solution amène beaucoup de jointures
   A(idA, a1, a2, # idD, e1) ; B(# idA, b1, b2) ; C(# idA, c1, c2)
2. Chaque table hérite de tous les attributs de la classe mère. Il n'y a pas de lien implicite entre les tables.
   A(idA, a1, a2) ; B(idA , # idD, e1, a1, a2, b1, b2) ; C(idA , # idD, e1, a1, a2, c1, c2),
   avec A qui disparaît si l'héritage possède une contrainte de totalité.
3. Une seule table est produite avec les attributs de toutes les tables. Les tuples de la table résultat risquent d'être composés de beaucoup d'attributs à NULL. A(idA , # idD, e1, a1, a2 , b1, b2, c1, c2),
   avec les attributs c1, c2 et b1 , b2 qui peuvent être à NULL.

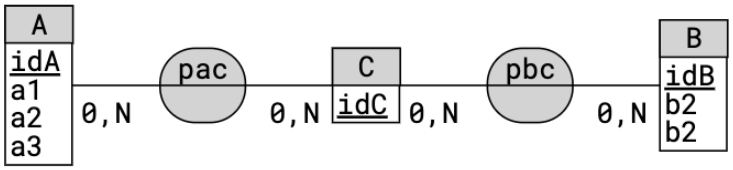
La solution 3 présente comme difficulté, la présence de beaucoup de valeur à NULL. Cela peut engendrer des difficultés dans la compréhension de l'évaluation des fonctions d'agrégation (sum, avg, count, etc.). La sémantique même de cette table n'est pas nécessairement très claire.

La solution 2 présente un problème de nommage. En effet, les attributs a1 et a2 se retrouvent dans 2 (ou 3) tables sans être ni clé primaire ni clé étrangère. De même, l'idA est aussi présent 3 fois, mais 3 fois en clé primaire. Il est préférable, même si ce n'est pas une obligation, pour ne pas induire des jointures non désirées, de renommer les attributs issus de A. On aura donc par exemple : A(idA, a1, a2) ; B(idA-B , # idD, e1, a1-B, a2-B, b1, b2) ; C(idA-C , # idD, e1, a1-C, a2-C, c1, c2) . Attention, par contre, il ne faut pas renommer les attributs clés étrangères migrés par les associations fonction (idD dans notre exemple).

La solution 1 ne pose pas de problème de nommage puisque les informations sont "réparties" entre A et B et A et C. Par contre, cette solution provoque systématiquement des jointures $A \bowtie B$ ou $A \bowtie C$. Elle engendre donc des requêtes un peu plus complexes à écrire et plus coûteuses en temps d'exécution. Notons aussi que B et C sont des tables-entité atypiques, car la clé primaire contient une clé étrangère.

4.4. Disparition de table

Dans certaines situations, les entités, qui devraient produire une table, disparaissent. C'est le cas quand une table produite s'avère calculable, c'est-à-dire qu'il existe une requête SQL qui permet de la retrouver. Du coup, des attributs se retrouvent clés étrangères sans avoir de clé primaire associée. Ils perdent donc ce statut. Cependant, ils peuvent tout de même se retrouver dans plusieurs tables. Prenons comme exemple le modèle conceptuel ci-dessous :



On suppose une contrainte de totalité sur les branches qui partent de C (imaginons par exemple une date). Nous aurons donc dans un premier temps le modèle relationnel suivant :

- A(idA, a1, a2, a3)
- B(idB, b1, b2)
- C(idC)
- pac(# idA, # idC)
- pbc(# idB, # idC)

Cependant, C peut être retrouvée en faisant :

```
Select idC From pac
Union
Select idC From pbc ;
```

Par conséquent, la table C est à supprimer. On obtient donc :

- A(idA, a1, a2, a3)
- B(idB, b1, b2)
- pac(# idA, idC)
- pbc(# idB, idC)

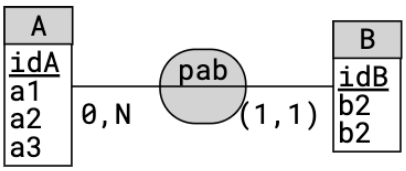
On se retrouve donc avec idC participant à deux clés primaires sans être clé étrangère. Cette situation peut troubler les élèves dans leur processus de construction d'une requête, en particulier sur les jointures pk/fk (voir plus loin). Là encore, pour ne pas induire des erreurs de jointure pk/fk, il est préférable de les renommer, par exemple en se contentant d'ajouter un suffixe. Et donc on obtient :

- A(idA, a1, a2, a3)
- B(idB, b1, b2)
- pac(# idA, idC-pac)
- pbc(# idB, idC-pbc)

Notons ici que les tables pas et pbs sont des tables-lien atypiques, car elles contiennent dans leur clé primaire des attributs qui ne sont pas clés étrangères.

4.5. Entité faible

Une entité faible est une entité pour laquelle l'identifiant n'est pas suffisant pour discriminer les différents individus. Elle s'appuie sur l'identifiant d'une autre entité à travers une association particulière. Il y a alors migration de l'identifiant de l'entité dite forte vers l'entité faible. C'est le même mécanisme qu'une association fonction sauf que la nouvelle clé étrangère participe à la clé de la table. Prenons le modèle conceptuel ci dessous :



Le modèle relationnel associé est alors :

- A (idA, a1, a2, a3)
- B (# idA, idB, b2, b2)

Comme pour la section précédente, la clé primaire composée d'un attribut propre et d'une clé étrangère peut être troublant, sortant du schéma table-entité contenant des informations (clé primaire formée d'attributs propres).

4.6. Instances ou pas ?

Lorsqu'une base de donnée est présentée aux élèves, faut-il la présenter avec des tables remplies ? Autrement dit, faut-il présenter une instance de la base de données ? Dans un premier temps, pour la bonne compréhension des opérateurs, c'est indispensable. La compétence d'évaluation des requêtes ne peut être acquise que dans ce contexte. Cependant, il faut assez vite se détacher des instances afin que les élèves puissent manipuler les requêtes de manière générale. L'instance est utile dans la partie pratique pour que les élèves puissent vérifier leur travail par eux-même. Cette étape de vérification est indispensable. Il faut montrer aux étudiants qu'une requête qui s'exécute n'est pas nécessairement correcte (même si c'est assez souvent leur comportement, car la vérification est parfois laborieuse) !

De plus, parfois, les élèves opèrent des raccourcis pour simplifier les requêtes. Par exemple, à l'intention « Donner les fleuves plus long que la Loire » certains vont aller regarder dans les instances pour s'apercevoir quelle fait 1006 km et donneront la requête simple :

```
Select nom-f
  From Fleuve
  Where longueur > 1006 ;
```

Au lieu de la requête avec une auto-jointure :

```
Select l2.nom-f
  From Fleuve as l1 Join Fleuve as l2 On l2.longueur > l1.longueur
  Where l1.nom-f = 'Loire' ;
```

Ou celle avec une sous-requête :

```
Select nom-f
  From Fleuve
  Where longueur > (Select longueur From Fleuve Where nom-f = 'Loire') ;
```

5. SQL

SQL (Structured Query Language) est un langage permettant de décrire, renseigner et interroger une base de données relationnelle. Il est apparu en 1974 (SQL 86 normalisé par l'ANSI puis par l'ISO ISO/CEI 9075:1986 en 86/87, SQL1 en 1989 ISO/CEI 9075:1989, SQ2 en 1992 ISO/CEI 9075:1992, SQL3 en 1999 ISO/CEI 9075:1999 puis à plusieurs reprises jusqu'en 2011 ISO/CEI 9075:2011). C'est le langage standard de la quasi-totalité des systèmes de gestion de bases de données relationnelles. SQL comporte quatre parties :

- Le langage de définition de données (LDD),
- Le langage de manipulation de données (LMD),
- Le langage de contrôle de données (LCD),
- Le langage de contrôle des transactions (LCT).

Il est, de fait, une partie essentielle de l'enseignement des bases de données au lycée comme l'indique le BO [BO2019b] :

> L'accès aux données d'une base de données relationnelle s'effectue grâce à des requêtes d'interrogation et de mise à jour qui peuvent par exemple être rédigées dans le langage SQL (Structured Query Language). Les traitements peuvent conjuguer le recours au langage SQL et à un langage de programmation.

Contenus	Capacités attendues	Commentaires
Langage SQL : requêtes d'interrogation et de mise à jour d'une base de données. Identifier les composants d'une requête.	Construire des requêtes d'interrogation à l'aide des clauses du langage SQL : SELECT, FROM, WHERE, JOIN. Construire des requêtes d'insertion et de mise à jour à l'aide de : UPDATE, INSERT, DELETE.	On peut utiliser DISTINCT, ORDER BY ou les fonctions d'agrégation sans utiliser les clauses GROUP BY et HAVING.

Nous constations que le programme ne porte que sur le LMD et ne précise pas quelle version est utilisée en référence. La très grande majorité des SGBD étant compatibles avec SQL2 et le programme évoquant l'opérateur *Join*, il est raisonnable de penser que nous pouvons nous baser sur la version SQL2 au moins. Les opérateurs sont assez limités et relativement flous. En particulier, derrière "Join" se trouve un ensemble d'opérateurs possibles comme la jointure naturelle, la semi-jointure ou la jointure externe. Ces variantes n'ont absolument pas la même complexité en terme de conception. Il y a d'autres opérateurs, comme Group By, mais ce n'est pas au programme ! De même, le programme n'évoque pas la possibilité d'utiliser des requêtes imbriquées.

Dans la suite de cette section, nous allons aborder quelques réflexions sur la jointure et les requêtes imbriquées.

5.1. Notion d'intention

Tout d'abord, commençons par une définition utile dans la suite de cette section.

Définition : Une *intention* est une texte en langage naturel (français, anglais...) décrivant ce que l'on cherche dans la base de données. Ce texte donne le besoin fonctionnel de la recherche (ce que l'on veut) et non une traduction en langage naturel de la requête.

Une intention est donc un élément clé lors de la construction et la manipulation la base de données. Elle participe à la construction du schéma (on doit pouvoir faire...) et, bien évidemment, elle est implémentée sous la forme d'une (voire plusieurs) requêtes SQL permettant d'extraire des informations de la base vers l'application.

5.2. La jointure

La jointure est l'opérateur relationnel le plus complexe présenté en NSI. Cependant, le programme ne dit pas grand chose sur le type de jointure à présenter. Nous allons, dans cette section, présenter ses caractéristiques et la manière de le présenter. Nous aborderons aussi les différentes versions et leur place dans le programme de NSI.

5.2.1. Comment la présenter ?

La jointure, ou θ-jointure dans son aspect général, est un des points les plus délicats de SQL (avec les requêtes imbriquées et le Group-By/Having). La jointure est un opérateur composé de l'algèbre relationnel. En effet, une jointure est définie comme la composition du produit cartésien avec une sélection sur une condition prenant au moins un attribut de chaque table :
$T_1 \bowtie_{C(T1,T2)} T_2 = \sigma_{C(T1,T2)}(T_1 \times T_2)$ avec $C(T_1, T_2)$ une expression booléenne faisant intervenir des attributs de T_1 et de T_2.

En SQL1, cet opérateur n'existait pas. Il était donc nécessaire de passer par sa définition formelle. En SQL2, a été introduit l'opérateur Join avec ses variantes : *[Inner] Join On*, *Natural Join*, *[Left|Right|Full] Outer Join* et *Cross Join*. Nous allons nous concentrer ici sur la jointure générale. Les variantes seront abordées plus loin.

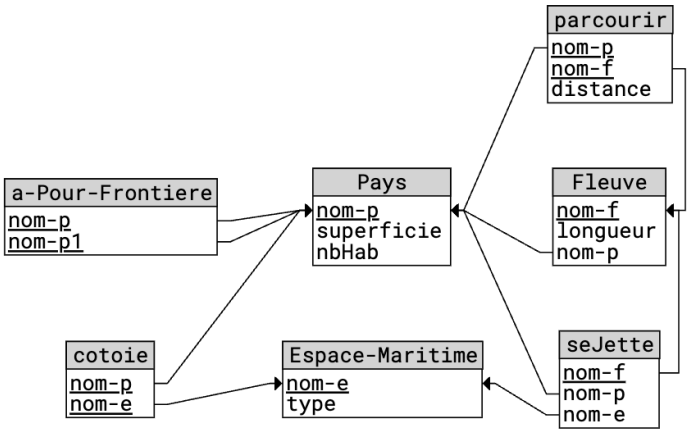
D'un point de vue formel, la jointure est assez simple. La difficulté se situe dans le bon usage de cet opérateur. Comment l'élève peut associer un besoin fonctionnel à l'utilisation de la jointure ? L'explication technique ne suffit pas pour amener à un bon usage de cet opérateur.

D'un point de vue conception de requête, nous pouvons considérer un cas particulier et très fréquent de jointure : la jointure entre une clé étrangère et la clé primaire qui lui correspond (par une contrainte d'intégrité référentielle). Ce type de jointure est fondamental dans la construction d'une grande majorité de requêtes. Nous les appellerons les jointures pk/fk.

Définition : Une **jointure pk/fk** est une équi-jointure entre deux tables telles que la condition de jointure est l'égalité entre la clef de l'une et une clé étrangère de l'autre telle que cette clé étrangère référence la clef primaire. Autrement dit, Soit A(a_1, ... a_n, b_1, ... b_n) et C(c_1,..., c_k, #(a_1, ..., a_n), d_1, ... , d_l) tel que C(a_1, ..., a_n) → A(a_1, ..., a_n) alors la jointure $A \bowtie_{A.a_1=C.a_1 \wedge ... \wedge A.a_n=C.a_n} C$ est une jointure pk/fk.

5.2.1.1. Cas des jointures pk/fk

Il faut l'aborder comme le lien entre des informations cohérentes présentes sur deux tables (pas nécessairement différentes à cause de l'auto-jointure). Une solution possible consiste en une approche par construction. Il faut amener les élèves à repérer où sont les informations dont ils ont besoin (projections ou éléments de sélection) dans la base de données. Pour cela, le modèle graphique (comme celui ci-dessous issu de EM-R2 pour notre base fil-rouge) peut aider.



Une fois les informations localisées, les tables sont associées en suivant le lien clé-primaire/clé-étrangère (les flèches sur le modèle graphique). Si le modèle est construit de manière suffisamment propre, c'est-à-dire que tout attribut de même nom décrit la même information, il suffit souvent de se baser sur les attributs identiques. Attention, nous le verrons plus loin, cette stratégie peut engendrer des erreurs.

On peut rapprocher ces liens avec les informations de la table universelle qui a été divisée pour éviter les anomalies. D'une certaine manière, la jointure pk/fk sert à la reconstruire. En effet, pour corriger les nombreuses anomalies, il a été nécessaire de diviser la table (passage du modèle EM0 au modèle EM1) [AnV2001], [CPTT2008], [Gar2005], [GUW2008]. Cette division (faite selon la méthode de synthèse ou par décomposition) est dite à jointure conservative. Par conséquent, la jointure pk/fk permet de retrouver une partie de cette relation avec les informations dont on a besoin.

Remarques :

- La méthodologie proposée suppose d'utiliser la jointure sur les paires (clé primaire, clé étrangère). Ce n'est pas le seul type de condition de jointure possible. Cependant, elle est assez facile à évaluer par un élève car $|t_1 \bowtie_{t_1.fk=t_2.pk} t_2| = |t_1|$.
- Pour ne pas compliquer la compréhension de la jointure par les élèves, il n'est pas utile de présenter sa définition formelle et, par voie de conséquence, la version SQL1 qui amène des erreurs [MAF2021].
- Afin de bien identifier les différents types de jointure, même s'il est optionnel, il est préférable d'utiliser *Inner Join* plutôt que *Join*.
- Ce type de jointure se base sur deux schémas : table-lien/table-entité ou table-entité/table-entité (sur une clé étrangère ne participant pas à une clé primaire), sauf dans certains cas particulier. Bien sûr, parfois une succession de jointures peut être simplifiée, des tables-entité pouvant être supprimées.

La méthodologie présentée ici ne concerne que des équi-jointures sur des paires (clé primaire, clé étrangère). Il existe d'autres jointures bien sûr. Cependant, elles peuvent être introduites dans un second temps, car conceptuellement plus complexes.

5.2.1.2. Auto-jointure pk/fk

En particulier, l'auto-jointure n'est pas facile à aborder dans le cas général, sauf si elle s'appuie sur une association en boucle du modèle conceptuel. La manipulation d'attributs de même nom "provenant" de la même table est assez difficile et nécessite l'utilisation des alias. Par exemple, prenons la relation "Employé(*id*, nom, prénom, #idChef)" qui décrit l'employé d'une entreprise et indique l'employé qui est son responsable. Ce type de relation induit une utilisation possible de l'auto-jointure. Par exemple, avec l'intention "Donner les employés sous la responsabilité de David Martin", on obtiendra la requête :

```
Select e2.id, e2.nom, e2.prénom
    From Employé as e1 Join Employé as e2 On e1.id = e2.idChef
    Where e1.nom = 'Martin' and e1.prénom='David' ;
```

5.2.1.3. Autres types de jointures

Les inéqui-jointures sont aussi complexes à traiter. Elles peuvent être combinées avec une autojointure, sans forcément suivre le lien clé-étrangère/clé-primaire. Les attributs concernés par ces jointures sont le plus souvent des attributs qui ne sont ni clé primaire ni clé étrangère. Par exemple, prenons l'intention suivante : "Donner les fleuves français plus long que la Seine." Cela donne la requête suivante :

```
Select f1.nom-f
    From Fleuve f1 Join Fleuve f2 On f1.longueur > f2.longueur
    Where f2.nom-f = "Seine" and f1.mon-p = 'France' ;
```

Notons que cette requête peut aussi être exprimée en utilisant une sous requête qui peut être plus lisible (voir plus loin.)

5.2.2. Attention aux fausses jointures

Avec le modèle normalisé EM-R2, faire des jointures est souvent assez simple puisqu'il suffit de proposer une condition de jointure sur une clé primaire et une clé étrangère de même nom. Les élèves peuvent assez facilement en conclure que, puisque c'est le même nom, on peut faire une jointure. Malheureusement, c'est une erreur. Par exemple, faire une jointure $Fleuve \bowtie_{Fleuve.nom-p=Pays.nom-p} Pays$ pour connaitre le nom du pays qui contient la source ne pose pas de problème. Par contre, faire une jointure $Fleuve \bowtie_{Fleuve.nom-p=Cotoie.nom-p} Côtoie$ sur nom-p n'a pas de signification évidente, sauf si on cherche les espaces maritimes qui côtoient le pays source d'un fleuve. Dans ce dernier cas, même si ce n'est pas utile d'un point de vue performance, il est préférable de faire une double jointure $(Fleuve \bowtie_{Fleuve.nom-p=Pays.nom-p} Pays) \bowtie_{Cotoie.nom-p=Pays.nom-p} Cotoie$ pour garder une certaine lisibilité de la requête et ainsi bien suivre les chemins clé-primaire/clé-étrangère. Cette lisibilité est importante à garder pour des raisons de maintenance de code (toujours utile de sensibiliser les élèves à la maintenabilité du code).

Une façon d'éviter ce type de raccourci non souhaitable est de renommer certains attributs. Ce renommage peut aussi améliorer la lisibilité de la requête. Par exemple, dans la table Fleuve, il serait préférable de renommer nom-p en pays-source par exemple. Ainsi, $(Fleuve \bowtie_{Fleuve.nom-p-source=Pays.nom-p} Pays) \bowtie_{Cotoie.nom-p=Pays.nom-p} Cotoie$ devient bien différente de $((Fleuve \bowtie_{Fleuve.nom-p=Parcours.nom-p} Parcours) \bowtie_{Parcours.nom-p=Pays.nom-p} Pays) \bowtie_{Cotoie.nom-p=Pays.nom-p} Cotoie.$

5.2.3. Naturelle, Semi ou Externe ?

L'opérateur de jointure de l'algèbre relationnel possède différentes variantes de difficulté de compréhension différentes. Nous allons les aborder dans cette section.

5.2.3.1. La jointure naturelle

La jointure naturelle (*Natural Join*) est une simplification de la jointure. Elle ne comporte pas de condition de jointure, car celle-ci est calculée. Cette condition est l'égalité des attributs de même nom. Elle est donc bien dans l'idée de "suivre" des chemins clé-primaire/clé-étrangère si le modèle est normalisé. Ainsi la requête $(Pays \bowtie Cotoie) \bowtie Espace - maritime$ est identique à la requête $(Pays \bowtie_{Cotoie.nom-p=Pays.nom-p} Cotoie) \bowtie_{Cotoie.nom-e=Espace-maritime.nom-e} Espace - maritime$.

Cette variante est cependant dangereuse, car elle provoque des jointures non désirées. Par exemple, prenons la requête dont l'intention est "donner les noms des pays parcourus par la Loire". En utilisant la jointure, un élève serait tenté d'écrire $Fleuve \bowtie Parcours \bowtie Pays$. En réalité, cette requête donne le nom du pays qui possède la source uniquement puis qu'elle est équivalente à la requête $Fleuve \bowtie_{Fleuve.nom-f=Parcours.nom-f \wedge Fleuve.nom-p=Parcours.nom-p} Parcours \bowtie_{Parcours.nom-p=Pays.nom-p \wedge Fleuve.nom-p=Pays.nom-p} Pays$ et non $Fleuve \bowtie_{Fleuve.nom-f=Parcours.nom-f} Parcours \bowtie_{Parcours.nom-p=Pays.nom-p} Pays$.

En conclusion, cette variante, bien que pratique pour le développeur, devrait être évitée pour l'initiation.

5.2.3.2. La jointure externe

La jointure externe est une variante de la jointure assez pratique pour l'exploitation d'une base de données. Elle permet d'ajouter au résultat de la jointure, les tuples d'une des deux tables ou des deux qui ne participent pas à la jointure. Les attributs de l'autre table sont alors à NULL.

D'un point de vue fonctionnel, elle permet d'effectuer des agrégations sans perdre d'information (avec par exemple des sommes à 0). Cependant, elle est surtout utile conjointement à l'utilisation des regroupements(Group-By/Having). Prenons par exemple, la requête d'intention "donner, pour chaque pays, le nombre de fleuves qui y ont leur source." Avec une jointure classique les pays n'ayant pas de source n'apparaissent pas dans le résultat alors qu'avec une jointure externe (le pays est alors la table dite directrice) les pays sans source ont un comptage à 0. Cela donne la requête suivante :

```
Select nom-p, Count(*)
From Pays Left Outer Join Fleuve
            on Pays.nom-p = Fleuve.nom-p
Group By nom-p
```

Cette variante de la jointure n'est pas facile à exprimer à partir de la jointure classique. Elle est donc très utile. Cependant, pour la spécialité NSI, les regroupements étant interdits, elle n'est pas à présenter ou, éventuellement, pour les meilleurs dans le cadre d'un projet.

5.2.3.3. La semi-Jointure

Pour terminer les variantes de la jointure, nous évoquons maintenant la semi-jointure. Cette jointure propose les t-uples d'une table qui participent à une jointure. D'un point de vue formel, elle est exprimée avec une jointure et une projection sur la table choisie. Par exemple, $Pays \ljoin Fleuve$ est équivalente à $\pi_{nom-p,superficie,nbHab}(Pays \bowtie Fleuve)$. Cette variante n'est pas vraiment utile, sa définition formelle suffit. Il n'est donc pas nécessaire de la présenter aux élèves de NSI. La requête suivante présente les pays qui ont une source :

```
Select Distinct nom-p, nbHab, superficie
From Pays Natural Join Fleuve
```

5.3. Requêtes imbriquées ?

Le programme de terminale NSI ([BO2021b]) ne mentionne pas la présentation des requêtes imbriquées. Ce type de requête est assez complexe sur certains aspects. En particulier, les sous-requêtes synchronisées (appelées aussi requêtes paramétrées) sont assez délicates à maîtriser. Elles sont parfois utiles (par exemple dans le cas de la négation ou comme solution plus simple que l'auto-jointure), mais il existe dans beaucoup de cas une version sans sous requêtes équivalente et beaucoup plus performante. Du fait de leur complexité, les sous-requêtes synchronisées ne sont pas à présenter en NSI. Par contre, il est possible d'envisager des sous-requête non synchronisées.

Ce type de construction est utile pour traiter des requêtes basées sur la négation. Pour les requêtes les plus complexes, il est possible de décomposer le problème en sous-problèmes plus simples. Il suffit alors de les traiter avec des requêtes plus simples et de les combiner.

D'un point de vue du temps d'exécution, ce type de requête est parfois plus lente, car le SGBD ne peut alors pas appliquer les algorithmes d'optimisation qu'il met en oeuvre sur des requêtes plus simples. Ceci est vrai dans le cas des requêtes paramétrées où la requête complète n'est connue qu'au moment de l'exécution.

On peut trouver ce type de requête soit au niveau du *From*, soit au niveau du *Where*. Si les élèves ont bien compris le principe de requête SQL, c'est-à-dire que le résultat d'une requête est une table, le positionnement d'une sous-requête dans le *From* est possible. Il est alors possible de combiner, non seulement des tables du modèle relationnel, mais aussi des tables issues de requêtes.

La requête ci dessous propose une sous-requête simple dans le From :

```
Select nom-p, nbHab, superficie
From Pays Natural Join
        (Select distinct nom-p
          From Fleuve
          Where longueur > 1000)
```

Dans le *Where*, c'est un peu plus délicat, car sur beaucoup d'opérateurs (comparaisons, [not] in par exemple) le résultat de la requête est considérée comme une valeur ou un ensemble de valeurs et non plus une table.

Parfois accessible dans des situations particulières de comparaisons par rapport à des agrégations tout en évitant des Group By.

Par exemple, prenons l'intention suivante : "Donner les fleuves français plus long que la Seine." Cela donne la requête suivante :

```
Select f1.nom-f
    From Fleuve f1 Join Fleuve f2 On f1.longueur > f2.longueur
    Where f2.nom-f = "Seine" and f1.mon-p = 'France' ;
```

Une version peut-être plus simple à comprendre dans ce cas serait d'utiliser une sous-requête. Si l'on reprends l'intention précédente, on obtient alors :

```
Select nom-f
    From Fleuve
    Where mon-p = 'France' and
        longueur > (Select longueur From Fleuve Where nom-f = "Seine") ;
```

Cette question peut-être précédée d'un travail sur l'intention "Donner la longueur de la Seine". Ce qui permet de développer la compétence de décomposition sur la partie BD.

```
Select longueur From Fleuve Where nom-f = "Seine"
```

Cependant, cette approche pose un problème conceptuel gênant. Elle demande à comparer une valeur numérique avec le résultat d'une requête, qui est, par essence, une table.

5.4. Vers une méthode de construction de requête

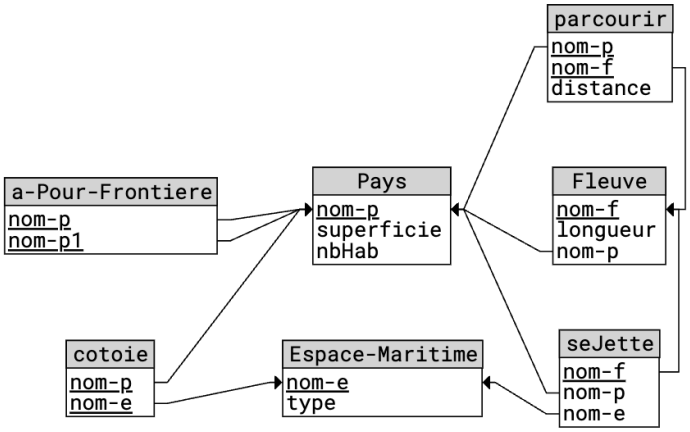
SQL est, nous l'avons déjà dit, un langage déclaratif. Construire une requête avancée (avec plusieurs jointures par exemple) est complexe. Il est donc intéressant de proposer une méthodologie. Comment peut-on construire de manière fiable et assez simple une telle requête ?

La méthode que nous proposons est la suivante :

1. à partir de l'intention, localiser dans le schéma relationnel les informations dont on a besoin en résultat (projection) ou pour faire des filtres (sélection) ;
2. proposer une chaine de jointures pk/fk (par les liens clé-primaire/clé-secondaire) pour "connecter" ces informations (et donc reconstruire partiellement la table universelle) ;
3. mettre en place l'expression logique permettant de faire la sélection (on ne garde que ce dont on a besoin) ;
4. faire la projection ou les éventuelles agrégations de table (sum, avg...) ;
5. se poser la question de la présence (ou non) du Distinct ;
6. faire un éventuel le tri.

Lorsque c'est trop complexe et qu'on trouve des requêtes pour répondre partiellement à l'intention (voire quand on sait faire le complémentaire par exemple), il est possible de prendre une approche par requête imbriquée dans le From ou dans le Where.

Par exemple, prenons par exemple l'intention suivante : "Donner les types d'espace maritime dans lequel se jettent les fleuves qui prennent leur source dans un pays de plus de 60 millions d'habitants."



La première étape est de localiser les informations utiles : "type d'espace" (dans Espace-Maritime), population (dans Pays). Ensuite, il faut trouver le chemin. Plusieurs sont possibles dans notre exemple : (1) par côtoie, (2) par seJette ou (3) par seJette-Fleuve. L'intention explicite ce chemin puisqu'il propose "les fleuves qui se jettent". Donc, c'est la solution (3). Reste pays-fleuve directement ou pays-parcours-fleuve. Là encore l'intention indique "qui prenent leur source dans" donc c'est la première solution. Par conséquent, nous avons la jointure : $Espace - Maritime \bowtie_{Q_1} seJette \bowtie_{Q_2} Fleuve \bowtie_{Q_3} Pays$. Attention cependant à la fausse jointure entre seJette.nom-p et Pays.nom-p ! Il faut bien suivre le chemin par Fleuve, et éviter les jointures naturelles. Ce qui fait la requête partielle :

```sql
Select XXXX
    From Espace-Maritime as em Join seJette as sj On em.nom-e=sj.nom-e
        join Fleuve as f On sj.nom-f = f.nom-f
        join Pays as p On p.nom-p = f.nom-p
    Where XXXX ;
```

En suite, nous définissons la sélection $nbHab > 60000000$ et la projection $type$. Il reste maintenant à déterminer si le Distinct est utile ici. "type" n'est pas une clé de la table résultant (c'est nom-f & nom-p). Donc, il peut y avoir plusieurs fois le même type. Donc un distinct est utile ici. D'où la requête :

```sql
Select Distinct type
    From Espace-Maritime as em Join seJette as sj On em.nom-e=sj.nom-e
        join Fleuve as f On sj.nom-f = f.nom-f
        join Pays as p On p.nom-p = f.nom-p
    Where nbHab > 60000000 ;
```

Remarque : cette requête à trois jointures est considérée, nous le verrons plus loin, comme une requête complexe pour les élèves.

5.5. Cas de la négation

Il est important d'être attentif aux intentions basées sur une négation. Ces intentions sont souvent assez complexes à traduire en requête SQL. Il est donc intéressant de s'intéresser à l'affirmation (complémentaire) puis à faire une requête qui "ne prend pas dedans". Souvent, on utilise l'opérateur "Not In" sur une sous requête.

Par exemple, prenons l'intention suivante : "Donner les fleuves qui passent en France et ne se jettent pas en Atlantique". Cette requête est facile à concevoir si on a à sa disposition les fleuves qui se jettent en Atlantique (attention encore à la fausse jointure entre nom-p du fleuve et nom-p de seJette !) :

```sql
# R1
Select nom-f
    From Fleuve as f Join seJette as sj On f.nom-f = sj.nom-f
    Where nom-e = "Océan Atlantique";
```

Donc, notre requête finale est :

```sql
# R2
Select nom-f
    From Pays Natural Join Parcours as pr
        Join Fleuve as f On f.nom-f = pr.nom-f
    Where nom-p = "France" and nom-f Not In (R1)
```

Remarque : il est possible ici de supprimer la jointure entre Pays et Parcours, car nous n'avons pas besoin d'autre chose que nom-p dans Pays.

6. Quelques situations didactiques en bases de données

L'enseignement des bases de données n'est pas récent. Cependant, force est de constater que les exercices proposés sont souvent du même schéma "Donnez une requête qui ...". Dans cette section, nous allons proposer une classification des exercices possibles sur SQL.

Dans la suite de cette section, nous allons proposer une classification en fonction des trois éléments en jeu :

- T : la table résultat de la requête ;
- I : l'intention visée par la requête ;
- R/r : la requête elle-même complète/partielle.

Chaque situation sera codée de la forme suivante "Données"→"Résultat attendu". La partie entre crochet est optionnelle. Par exemple, "[I]R→T" décrit une situation où l'intention (éventuellement) et la requête sont données et l'élève doit donner la table résultat. Pour certaines situations, il est possible d'ajouter une étape "⇒T" qui permet d'obtenir le résultat en exécutant la requête sur le SGBD-R.

Nous supposerons que l'élève a connaissance du modèle relationnel et, en cas d'attente de la table résultat, un exemple d'instance de la base de données. Nous évaluerons les différents cas au regard de la complexité et des compétences évaluées (au sens de [DSS2017]).

6.1. [I]R→T

Pour cette première situation, l'élève dispose d'une requête SQL bien formée. Il doit, à partir d'une instance de la base de données qui lui est proposée, donner la table résultat de cette requête. L'objectif est de l'amener à comprendre le fonctionnement des opérateurs utilisés par la requête. Le motif de cette situation est "Soit la base de données suivante, donner le résultat de la requête .... [qui recherche ...]".

Par exemple, "Donner la table résultat de la requête suivante qui recherche les fleuves de longueur supérieure à 1000 Km."

```
Select nom-f
     From Fleuve
     Where longueur > 1000
```

ou "Donner la table résultat de la requête suivante."

```
Select nom-f
     From Fleuve Natural Join Espace-Maritime
     Where type = 'mer'
```

Cette situation est aussi intéressante pour amener l'élève à comprendre l'architecture de la base de données relationnelle. Il faut proposer des requêtes simples, mais aussi des jointures clé-primaire/clé-étrangère pour l'amener à "parcourir" les tables.

Dans cette famille de situation, il est aussi possible de proposer une requête et plusieurs tables possibles. L'élève doit choisir celle qui correspond au résultat de la requête. Cette situation est plus facile que la précédente et peut être proposée en amont.

6.2. Ir→R & Tr→R

Cette situation est assez classique en programmation, moins en bases de données. L'idée est de proposer une requête à trous. L'élève doit proposer des solutions. La requête seule est une situation assez difficile, et pas nécessairement intéressante. Il faut donc l'accompagner soit de l'intention soit de la table résultat. Cette approche permet de découvrir assez facilement les différents opérateurs disponibles dans une requête SQL.

La situation suivante propose de découvrir simplement la projection en proposant l'intention de la requête "On recherche le nom des fleuves de longueur supérieure à 1000 Km." :

```
Select ????
   From Fleuve
   Where longueur > 1000
```

De même, un exemple sans intention, mais avec le résultat :

```
Select ????
     From Fleuve Natural Join Espace-Maritime
     Where ????
```

nom_f
Loire
Tajo
Elbe
Donau
Rhein
Weichsel
Dnister
Petschora
Western Dwina
Dnepr
Don
Ob
Jenissej
Chatanga
Lena
Kolyma
Amur

Ce deuxième exemple est, comme souvent, un peu délicat. Il est assez complexe de trouver la bonne requête, juste en ayant à sa disposition le résultat, car, bien évidemment, plusieurs requêtes peuvent donner le même résultat.

6.3. R[T]→I

Nous avons ici une situation un peu particulière. Une requête est proposée (éventuellement avec la table résultat) et les élèves doivent trouver l'intention sous-jacente. L'objectif est d'amener les élèves à comprendre le besoin fonctionnel d'une requête pour éventuellement la corriger. La présence de la table (ou la possibilité donnée à l'élève d'exécuter la requête) permet d'aider à trouver cette intention. Cette table peut-être fournie dans un second temps en cas de difficulté à trouver la bonne intention.

En pratique, cette situation n'est pas évidente à mettre en place, car beaucoup d'élèves vont formuler la requête en langage naturel et non énoncer son intention. Par exemple, pour la requête suivante, les élèves diront "on fait la jointure entre Fleuve et Espace-maritime, on ne garde que les mers et on projette le nom du fleuve." au lieu de "Donner le nom des fleuves qui se jettent dans une mer".

```
Select nom-f
    From Fleuve Natural Join Espace-Maritime
    Where type = 'mer'
```

nom_f
Buna
Drin
Aliakmonas
Vardar
Maas
Seine
Ebro
Oder
Elbe
Weser
Donau
Po
Etsch
Tevere
Marta
Arrone
Arno
Rhein

6.4. T→R

Cette situation est particulièrement complexe à gérer dans le cas général, pour l'élève comme pour l'enseignant. Il faut vraiment se trouver dans une situation où il n'y a pas d'ambiguïté. Une table est proposée et l'élève doit trouver la (une ?) requête permettant de l'obtenir. Souvent plusieurs solutions existent, et il n'est pas toujours simple de les anticiper. Devant machine, il est possible de permettre à l'élève de vérifier sa proposition : T1→R⇒T2, T1=T2.

6.5. I[T]→R

Cette situation est la plus courante dans la littérature, les cours en lignes, etc. L'enseignant propose une intention à l'élève et ce dernier doit trouver une requête qui permet d'y répondre. Il doit être capable de transformer un besoin fonctionnel en une requête SQL. Eventuellement, la table résultat peut être proposée. Par exemple, on peut proposer « Les fleuves qui se jettent dans une mer. ».

Attention, il faut insister sur la vérification de la requête. Ce n'est pas parce qu'une requête donne un résultat qu'elle est bonne ! De même, ce n'est pas parce qu'une requête ne donne pas de résultat qu'elle est fausse. Donc, il faut proposer des situations qui suivent le déroulement IT1→R⇒T2 avec T2=T1 quand c'est possible.

6.6. I→T

Dans cette version, plus simple que la précédente, on demande le résultat d'une intention. L'élève n'a pas à connaitre SQL, mais doit proposer ce qu'on a envie d'avoir. Cette situation permet d'amener l'élève à traduire un besoin en une requête.

6.7. Composition

Il est aussi possible de composer les situations vues précédemment pour faciliter la progression de l'élève.

Par exemple, I→T peut être un travail préalable à la forme I[T]→R, ce qui donnerait I→T→R. L'enseignant demande d'abord la table réponse à une intention avant de demander la requête elle-même. L'élève peut ainsi prendre le temps de comprendre la base de données avant de construire la requête. Cette solution peut être intéressante dans le cas de requête ne donnant aucun résultat. Les élèves peuvent avoir un blocage sur ce type de requête pourtant correcte. Cependant, pour les convaincre, il faut sans doute revenir sur le schéma précédent à savoir I→T1→R⇒T2 (T1=T2).

De même, pour amener les élèves à vérifier la validité d'une requête, il faut proposer des situations qui suivent le déroulement IT1→R⇒T2 mais aussi I→T1→R⇒T2 avec T2=T1.

6.8. Modification de la base de données

Comme pour les requêtes de recherche, il est possible de proposer des situations d'enseignement pour des modifications de la base de données. L'objectif, en plus de faire comprendre le mécanisme de ces requêtes, est de sensibiliser à la gestion de la cohérence de la base de données. Cela nécessite une bonne compréhension du modèle relationnel et des différentes contraintes, pas toujours visibles sur le modèle lui-même.

On peut reprendre beaucoup des schémas décrits précédemment (si M est la modification de type i(nsert), d(elete) ou u(pdate) par exemple et m une modification incomplète) :

- I→ M
- M→T
- T→M
- T→I
- Im→M
- etc.

Les solutions précédentes sont possibles sur un modèle normalisé. Cependant, il peut être aussi intéressant de travailler sur un modèle non normalisé (au sens des formes normales) pour mettre en évidence les soucis que peuvent engendrer les anomalies d'une base de données (disparition d'information, coût d'une modification en cas de redondances, etc.). Ces exercices sont possibles sur une base de données, mais aussi sur papier ou sur un tableur.

Un des objectifs essentiels de ces situations est de faire prendre conscience à l'élève que les modifications peuvent mettre à mal la cohérence de la base soit définitivement soit temporairement. Là, une discussion permettant d'indiquer le mécanise de transaction peut être évoqué, même s'il n'est pas au programme du BO.

C'est à ce stade que vont intervenir des contraintes qui ne sont pas nécessairement présentes dans le modèle relationnel seul. Par exemple, si on veut ajouter un fleuve à la base de données. Il faut ajouter ce fleuve, mais aussi le fait qu'il parcourt au moins un pays. Cette contrainte est une contrainte du modèle conceptuel absente du modèle relationnel. Ensuite, il faut commencer par l'ajout du fleuve avant le parcourt, car on ne peut ajouter dans parcours si le fleuve n'existe pas. Cette contrainte d'intégrité référentielle est, quand à elle, présente dans le modèle relationnel. L'ordre est donc lié à la structure du modèle.

Par exemple, nous cherchons à ajouter un nouveau fleuve : la Vilaine. Son ajout sera effectué par les deux instructions suivantes dont l'ordre est important :

```
1 Insert Into Fleuve Values ('Vilaine', 'Atlantic', 'France', 560) ;
2 Insert Into Parcourt Values ('F', 'Vilaine', 560)
```

Cette situation est aussi possible en suppression. Par exemple, supprimons maintenant la Vilaine :

```
Delete From Parcourt Where id_f = 'Vilaine' ;
Delete From Fleuve Where id_f = 'Vilaine'
```

Il faut aussi proposer des situations où la cohérence n'est pas de nature structurelle, mais sémantique. Par exemple, "Modifier la longueur de la Vilaine". Là, bien évidement, il faut modifier l'attribut *longueur* du fleuve, mais aussi ajuster la distance parcourue en France. Nous avons donc les deux instructions SQL suivantes :

```
Update Fleuve Set longueur = 218 Where id_f = 'Vilaine' ;
Update Parcourt Set distance = 218 Where id_f = 'Vilaine'
```

6.9. Synthèse sur les situations en SQL [A REFAIRE]

Tableau à retravailler

Type	Forme	Diffi-culté	Eva-luer	Anti-ciper	Décom-poser	Généra-liser	Abst-raire	Commen-taire
Select	R→T	⭐⭐	x		x			
	IR→T	⭐	x		x			
	I→T	⭐	x					
	Tr→R	⭐⭐				x	x	
	Ir→R	⭐		x		x		
	TIr→R	⭐		x		x		
	T→R	⭐⭐⭐⭐				x	x	
	I→R	⭐⭐		x	x	x	x	
	IT→R	⭐			x	x	x	
	R→I	⭐⭐⭐	x	x				Assez difficile, surtout si on refuse la paraphrase de la requête
	RT→I	⭐⭐						
Insert	Insertion : On veut ajouter ... donner l'instruction qui...	⭐/⭐⭐						Plus difficile si plusieurs instructions, et pire si nécessité d'ordre
Delete	Suppression : on veut supprimer...	⭐/⭐⭐						Plus difficile si plusieurs instructions, et pire si nécessité d'ordre
Insert/Delete / Update	Voici une modification de la BD, donner les instructions	⭐/⭐⭐						Plus difficile si plusieurs instructions, et pire si nécessité d'ordre

Bien évidemment, ces situations peuvent être proposées par l'enseignant. Cependant, il peut aussi y avoir des exercices où les situations sont proposées par les élèves eux-même. Un élève propose une situation à un autre et trouve la solution de la proposition d'un autre.

6.10. Complexité des opérateurs

La complexité des situations vues précédemment est pondérée par les opérateurs relationnels qui sont mis en oeuvre. Ils n'ont pas tous la même complexité conceptuelle pour un élève. La projection et la sélection sont assez faciles à appréhender, car ce sont des actions déjà abordées lors des chapitre d'algorithmique et programmation, en particulier sur les données en table.

Comme nous l'avons déjà évoqué plus haut, la jointure est l'opérateur du programme de NSI le plus complexe. Cette complexité est accrue lorsqu'il y a plusieurs jointures à mettre en oeuvre et si la condition de jointure n'est pas une égalité. De même, l'auto-jointure, souvent demandée, n'est pas toujours facile à mettre en oeuvre et doit être expliquée en tant que telle.

Il ne faut pas non plus négliger la manipulation de l'opérateur Distinct positionné juste après le Select. Un réflexe de débutant est d'en mettre un systématiquement. Cette approche est à proscrire quand elle n'est pas indispensable au besoin fonctionnel (présence d'opérateur d'agrégation par exemple) car :

- du point de vue conceptuel, mettre cet opérateur alors que la table résultat possède une clé est la preuve de l'incompréhension de l'essence même d'une requête qui retourne une table ;
- du pont de vue performance, car cet opérateur implique une suppression des doublons qui, d'un point de vue complexité, est assez coûteux (en O(nlog(n) au mieux).

Par exemple, la requête suivante

```
select distinct nom-p, nom-f From Pays Natural Join Pays ;
```

est plus coûteuse que la requête

```
select nom-p, nom-f From Pays Natural Join Pays ;
```

pour le même résultat.

Il faut aussi faire attention aussi aux comparaisons. On trouve assez fréquemment l'utilisation de "Like" à la place de "=", même dans des cours ou des présentations d'informaticiens. On trouve par exemple la requête

```
Select * from Fleuve where nom like 'Loire';
```

au lieu de la requête

```
Select * from Fleuve where nom = 'Loire';
```

Il ne faut pas oublier que "Like" est un opérateur qui prend une expression équivalente à une expression régulière. Cette expression est alors transformée en automate d'état fini et la valeur comparée est alors validée par cet automate. Cet opérateur est donc plus coûteux en temps d'exécution que le simple opérateur '='.

Pour conclure, il est nécessaire de considérer les situations SQL avec une complexité progressive des opérateurs. Ceci donne donc une ordre possible des opérateurs :

1. Projection ;
2. Projection + Distinct ;
3. Projection + opérateurs d'agrégation globaux (count, avg, sum, etc.) ;
4. Sélection simple ;
5. Sélection avec condition complexe ;
6. Sélection + Projection ;
7. Jointure à 2 tables (+ projection) ;
8. Jointure à 2 tables avec sélection ;
9. Sous-requête non synchronisées ;
10. Jointure à 3 ou 4 tables (+ projection).

De même, il est possible de définir une progression pour les opérations de modifications (insert, delete, update) des tables. Cette progression doit prendre en compte les contraintes du modèle qui s'appliquent. Par exemple, on a l'insertion :

1. simple ;
2. de plusieurs valeurs ;
3. avec nécessité d'ordre pour la cohérence structurelle (contrainte d'intégrité référentielle) ;
4. avec nécessité d'ordre pour la cohérence sémantique.

Pour terminer, il faut aussi aborder le tri de la table. Cet opérateur est particulier, car il n'apporte pas de modification de la table. Il modifie juste de sa présentation. En effet, dans le modèle relationnel, il n'y a pas d'ordre dans les t-uples d'une relation. Compte-tenu des connaissances des élèves sur les algorithmes de tri, ainsi que les algorithmes de k plus proches voisins, cet opérateur Order By ne pose pas de difficultés particulières. On peut même ajouter l'opérateur Limit qui permet d'obtenir les k premiers.

6.11. Travail sur le schéma d'une base de données

6.11.1. Contexte

Le BO fait une place essentielle à SQL. Cependant, mettre en place des requêtes SQL demande une bonne compréhension du modèle sous-jacent à la base de données. Cette partie est assez peut mise en valeur dans le BO. Nous allons ici proposer quelques situations permettant aux élèves d'être sensibilisés à l'exploration du modèle, voire à sa conception (même si ce n'est pas au programme).

On cherche donc à faire comprendre aux élèves l'importance du modèle de la base de données. Une question essentielle se pose alors : quel modèle proposer ? Plusieurs solutions se présentent : le modèle conceptuel, le modèle relationnel, la représentation SQL...

Le modèle conceptuel est primordial dans le processus de conception, l'architecture d'une base de données quel que soit le modèle logique choisi (relationnel, hiérarchique, graphe...). Plusieurs modèles sont disponibles : le modèle EAP de Merise, le diagramme de classe d'UML, etc. De même, de nombreux outils (payants ou gratuits) sont à disposition et proposent tous la génération du modèle relationnel issu de ce modèle conceptuel.

Cependant, cette phase essentielle de la construction de la base de données, bien que présente dans tout les curriculum d'initiation aux bases de donnée en premier cycle universitaire, n'est pas au programme de NSI. Il doit donc être utilisé par l'enseignant pour proposer des bases de données ayant de bonnes propriétés ou pour valider des bases de données proposées par des élèves (rétro-ingénierie).

Bien sûr **le modèle logique**, le modèle relationnel dans le cadre de NSI, est au programme. Plusieurs manières de le présenter sont possibles. Tout d'abord, le mode textuel est simple à faire, mais l'architecture n'est pas toujours facile à appréhender. Il est nécessaire d'indiquer les contraintes d'intégrité référentielles de la forme : Clé-étrangère → Clé-primaire.

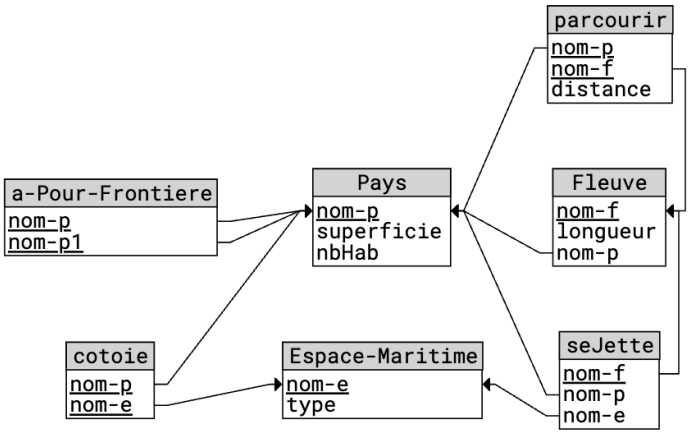
Voici une première syntaxe à laquelle on peut ajouter les domaines sur notre fil-rouge :

- Pays (nom-p, superficie, nbHab) clé primaire : nom-p
- Fleuve (nom-f, longueur, nom-p) clé primaire : nom-f clé étrangère : nom-p → Pays(nom-p)
- Espace-Maritime (nom-e, type) clé primaire : nom-e
- seJette (nom-f, nom-p, nom-e) clé primaire : nom-f clé étrangère : nom-f → Fleuve(nom-f) clé étrangère : nom-p → Pays(nom-p) clé étrangère : nom-e → Espace-Maritime(nom-e)
- cotoie (nom-p, nom-e) clé primaire : nom-p, nom-e clé étrangère : nom-p → Pays(nom-p) clé étrangère : nom-e → Espace-Maritime(nom-e)
- parcourir (nom-p, nom-f, distance) clé primaire : nom-p, nom-f clé étrangère : nom-p → Pays(nom-p) clé étrangère : nom-f → Fleuve(nom-f)
- a-Pour-Frontiere (nom-p, nom-p1) clé primaire : nom-p, nom-p1 clé étrangère : nom-p → Pays(nom-p) clé étrangère : nom-p1 → Pays(nom-p)

On peut aussi le représenter de la manière suivante, qui est un peu moins verbeuse, mais plus codifiée (clés primaires soulignées et clé étrangères précédées d'un #) :

- Pays (*nom-p*, superficie, nbHab)
- Fleuve (*nom-f*, longueur, #nom-p) nom-p → Pays(nom-p)
- Espace-Maritime (*nom-e*, type)
- seJette (#*nom-f*, #nom-p, #nom-e) nom-f → Fleuve(nom-f) nom-p → Pays(nom-p) nom-e → Espace-Maritime(nom-e)
- cotoie (#*nom-p*, #*nom-e*) nom-p → Pays(nom-p) nom-e → Espace-Maritime(nom-e)
- parcourir (#*nom-p*, #*nom-f*, distance) nom-p → Pays(nom-p) nom-f → Fleuve(nom-f)
- a-Pour-Frontiere (#*nom-u*, #*nom-p1*) nom-p → Pays(nom-p) nom-p1 → Pays(nom-p)

Le mode graphique est souvent plus clair (quand il n'y a pas beaucoup de tables) et permet de mieux construire les requêtes. Quand le schéma de la base de données est important, la représentation graphique devient difficile à lire. La représentation graphique n'est pas au programme du BO, mais est fondamentale pour la compréhension de l'architecture de la base. La encore, plusieurs représentations graphiques existent. Le style le plus intéressant est celui qui propose un rectangle pour une table et un flèche de la clé étrangère vers la clé primaire comme ci-dessous pour notre exemple.



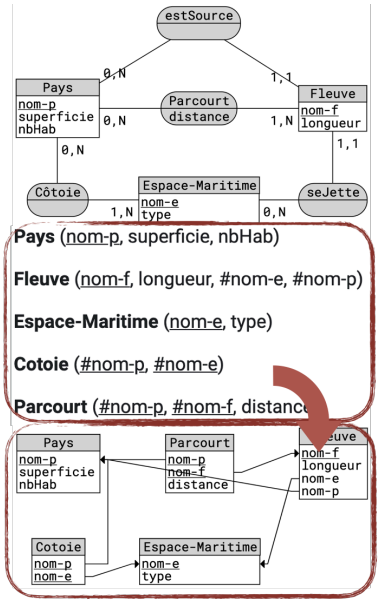
Le schéma relationnel comprend les tables et les clés primaires et étrangères, mais aussi les dépendances fonctionnelles. Cependant, les dépendances fonctionnelles (et les formes normales) ne sont pas au programme. Elle peuvent peut-être servir de manière intuitive à présenter les anomalies d'une table (au programme du BO).

Finalement, il est aussi possible de présenter **le schéma de la base de données SQL** en donnant les instructions LDD de création de la base (Create Table...). Cette solution, plutôt lourde, n'apporte pas grand chose en plus de ne pas être au programme.

Nous allons maintenant présenter quelques situations pour sensibiliser les élèves à la compréhension du modèle logique.

6.11.2. Construire le modèle graphique

Une première situation est de partir d'un modèle textuel pour construire un modèle graphique. Cela permet de réfléchir sur la structure et de prendre conscience des « liens » entre les tables.



Il est possible aussi de demander aux élèves de différentier les tables-entité des tables-lien par des couleurs différentes. On peut aussi demander à l'élève d'énumérer toutes les jointures possibles (en dehors des auto-jointure).

6.11.3. Vérification et correction des anomalies

Evidemment, les réflexions sur le schéma de la base de données impliquent des situations de détection d'anomalies. L'idée est de faire réfléchir aux enjeux d'une "bonne" structure de schéma. Cette situation peut être en deux étapes : (1) tout d'abord analyser les anomalies sur une base de données avec des instances de table puis (2) sur une autre base de données sans les instances. Par exemple, on peut proposer le modèle suivant :

```
Fleuve(nom-f, longueur, nom-pays-source, superficie, nbHab,
            liste-noms-pays-parcourus)
seJette(#nom-f, #nom-e)
Espace-Maritime(nom-e, type, liste_pays_cotoyés)
```

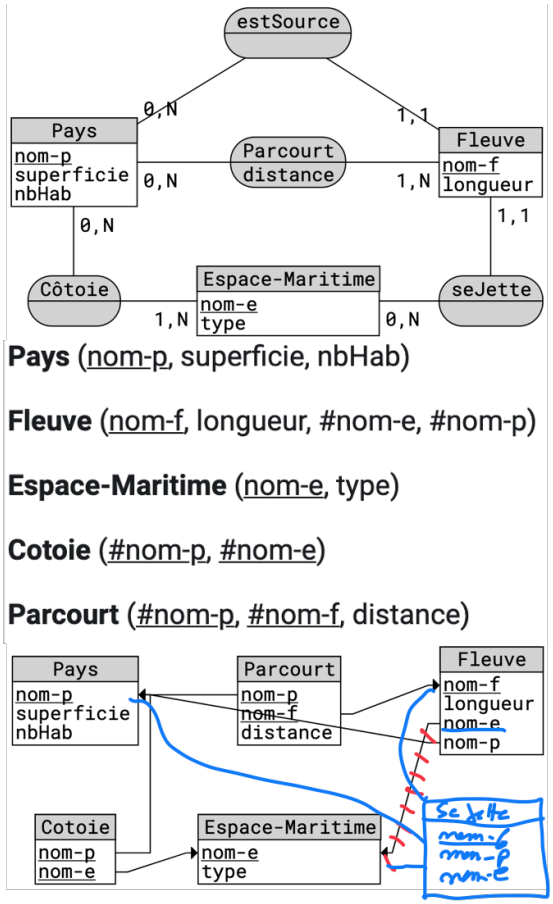
6.11.4. Vérifications face aux besoins

Il est important que les élèves aient conscience que la structure de la base dépend de l'expression de besoins fonctionnels. Dans ce type de situation, les besoins peuvent être exprimés sous forme de requêtes SQL ou en langage naturel.

Il est intéressant aussi de sensibiliser les élèves au fait que le besoin dépend du profil de l'utilisateur de la base de données. Les élèves disposent donc d'une base de donnée (schéma et, éventuellement, une instance des tables), du contexte et de l'expressions de besoins fonctionnels. Ils vérifient alors que la base est en mesure de répondre à ces besoins (il existe une requête qui...). Ils vont être amenés à découvrir que les données ne sont pas correctes, que les liens entre les tables ne sont pas satisfaisant (voir inexistant avec la présence d'îlots), etc. Ils peuvent identifier les difficultés et proposer des solutions. Par exemple, avec notre base de données fil-rouge, on peut avoir le besoin :

```
On veut la liste des estuaires français (par le nom du pays et du fleuve).
```

Seulement la base de données proposée ne propose pas le concept d'estuaire. L'élève va alors proposer une modification adaptée.



Pour aller plus loin, l'enseignant peut seulement donner la base de données et le contexte, les élèves proposent alors des besoins.

6.11.5. Travail sur un schéma incomplet ou dégradé

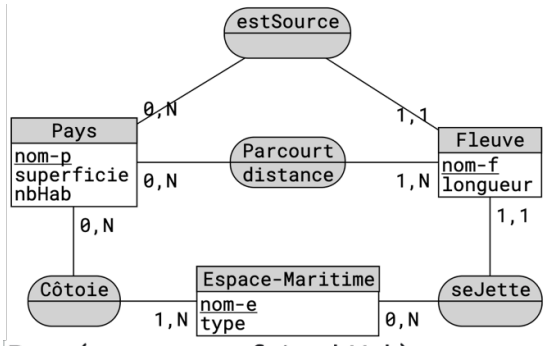
Pour aller plus loin dans la compréhension du modèle et de l'impact de l'architecture sur la capacité à répondre aux besoins, il est possible de dégrader le modèle, c'est-à-dire de proposer un modèle bien structuré, mais incomplet.

Dans un premier temps, il suffit juste d'enlever les contraintes d'intégrités référentielles et le signalement des clés primaires par exemple. Pour ne pas trop compliquer la situation, il est préférable de rester dans un premier temps sur un modèle normalisé avec, en particulier, toujours le même terme pour la même information (pas de synonymes, etc.). Pour aller plus loin, on peut proposer une base non normalisée, voir pour laquelle il manque les domaines ou les attributs de certaines (ou toutes) les tables.

Par exemple, il est possible de proposer le modèle suivant qui ne comporte aucune mention de clé primaire ou étrangère :

```
Pays (nom-p, superficie, nbHab)
Fleuve (nom-f, longueur, nom-e, nom-p)
Espace-Maritime (nom-e, type)
cotoie (nom-e, nom-p)
parcourt (nom-p, nom-f, distance)
```

Et même un modèle à « à trous » comme ci-dessous :



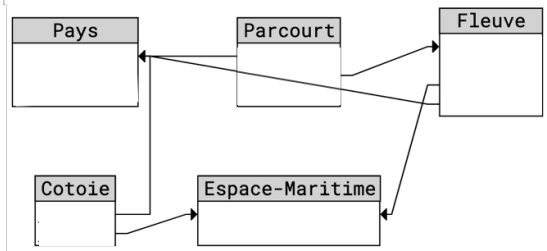
Pays (<u>nom-p</u>, superficie, nbHab)

Fleuve (<u>nom-f</u>, longueur, #nom-e, #nom-p)

Espace-Maritime (<u>nom-e</u>, type)

Cotoie (<u>#nom-p</u>, <u>#nom-e</u>)

Parcourt (<u>#nom-p</u>, <u>#nom-f</u>, distance)



6.11.6. Abstraire un modèle

Pour terminer, nous pouvons ajouter une situation où les élèves travaillent avec une base de données abstraite et non-normalisée. D'abord, on présente la BD avec des noms abstraits et on demande quelques requêtes. Puis on nomme correctement et on repose les mêmes requêtes (elles aussi remises en contexte). Cette situation montre l'importance de la sémantique et de la normalisation.

Version abstraite	Version nommée	Version normalisée
A(<u>a</u>, #b, c) - b → B(a) B(<u>a</u>, c, e, f)	Salarié(<u>id</u>, #service, nom) - service → Service(id) Service(<u>id</u>, nom, adresse, secteur)	Salarié(<u>idSal</u>, #idServ, nomSal) - idServ → Service(idServ) Service(<u>idServ</u>, nomService, adresse, secteur)

Les trois bases peuvent être différentes ou ce peut être la même base « dévoilée » progressivement. Les élèves vont alors ajuster leurs requêtes en fonction de l'évolution du modèle.

7. Typologie des projets en BD

Une partie motivante pour les élèves en bases de données est la partie projet. Dans cette section, nous nous poserons la question du type de projet que l'on peut poser. Pour l'enseignant, la partie projet est parfois un peu complexe à gérer. Les élèves peuvent proposer des contextes motivant pour eux et l'enseignant doit être en mesure de les accompagner. Il doit mobiliser des compétences de modélisation pour « sécuriser » les bases qu'il propose ou vérifier celles proposées pas les élèves. Pour cela, il est important de maîtriser la notion de modèle conceptuel, surtout quand il y a des objectifs de modifications de la base et de programmation. En effet, la perte d'informations (de contraintes en particulier) quand on passe au modèle relationnel reste un problème. Par ailleurs, lorsqu'un modèle relationnel douteux est proposé par les élèves (ou plus généralement pour le vérifier), une méthode efficace est de passer par une étape de rétro-ingénierie pour déterminer le modèle conceptuel sous jacent et, le cas échéant, de modifier l'approche. De même, et en complément, il est fondamental que l'enseignant maitrise les bases des dépendances fonctionnelles, des formes normales et de la normalisation des bases de données. Ceci permet de vérifier les bases de données proposées, mais aussi de s'assurer que les bases proposées par lui/elle ont les bonnes propriétés (et donc pas les anomalies).

7.1. La base complète existe

Le premier type de projet est le plus facile pour l'enseignant. Il propose une base de données complète (schéma et instances). Les données peuvent être issues de l'Open Data par exemple. Le contexte, les besoins, le modèle relationnel et les scripts SQL/LDD et SQL/LMD de création de la base sont donnés. Les élèves doivent proposer : les requêtes permettant de répondre aux besoins fonctionnels, un programme (Python) qui exploite les données et, éventuellement, proposer et implémenter une extension de la base de données.

La partie programmation n'est pas indispensable. Les élèvent, peuvent très bien proposer les requêtes et les scénarios nécessaires à la satisfaction des différents besoin et décrire les scénario d'utilisation de ces requêtes. Cependant, c'est assez frustrant. Il est préférable d'avoir une petite application Web (par exemple) permettant de donner vie au projet. Ceci dit, dans un usage basique, l'accès à une base de donnée par un programme n'est pas réellement un problème pour un débutant. C'est une approche pratiquée en TP en initiation aux bases de données en première année de licence. Les motifs de programmation sont peu nombreux et assez faciles à présenter sur un exemple ou deux.

7.2. La base physique n’existe pas

En fonction du temps, il est possible de ne proposer que le contexte, les besoins, le modèle relationnel et les scripts de création SQL/LDD. Les élèves peuvent alors renseigner les tables et manipuler le SQL/LMD avec insert, update et delete. Cette phase de "remplissage" de la base est intéressant, car elle permet aux élèves de découvrir les tables et l'architecture de la base de données.

7.3. Le modèles logique n’existe pas

En fonction de la confiance de l'enseignant et du niveau des étudiants, il est envisageable de ne donner que le contexte et les besoins. Les élèves doivent alors aussi proposer un modèle relationnel possible. La difficulté pour l'enseignant est de valider les modèles proposés. Même s'il est en a en réserve (peut-être pour les élèves plus faibles), il doit pouvoir accepter d'autres modèles équivalents ou basés sur les implicites du contextes. C'est là que les capacités à déterminer le modèle conceptuel sous jacent peut aider l'enseignant à critiquer les solutions proposées, voir à débloquer les élèves.

Il est même possible d'aller encore plus loin dans le processus. Le projet est alors décomposé en deux parties : d'abord les élèves proposent un contexte et des besoins puis l'enseignant valide/corrige les propositions et enfin les élèves développe le projet. Il est même possible de faire développer le projet par un autre groupe. Les élèves qui ont fait la proposition jouent en rôle de **maitre d’ouvrage** et ceux qui implémentent de **maitre d’oeuvre**.

Bien évidemment, sur cette section, nous sommes en dehors du programme puisque le BO ne demande pas de phase de modélisation. Il faut donc proposer ces solutions que pour des classes motivées, sans doute en petit effectif.

8. Une brève analyse des manuels scolaires

8.1. Contexte

Au regard de ce que nous avons abordé dans ce document, nous avons parcouru différents ouvrages de NSI du commerce. Pour la sélection, nous avons pris les ouvrages mis à disposition lors des épreuves de Capes 2021 (sélection présente dans CAPESOS 2021) : [Connan et al. 2020], [Bonnefoy-Petit 2020], [Balabonski et al. 2020] et [Bays 2020]. L'étude a été découpée selon le programme du BO : le modèle relationnel, les SGBD et SQL. Nous allons d'abord présenter les parties cours puis les exercices.

8.2. Partie cours

8.2.1. Modèle relationnel

Nous allons d'abord observer ce qui est présenté dans les parties cours sur le modèle relationnel. Bien évidemment, tous les ouvrages abordent le modèle relationnel et les contraintes d'intégrité. [Bays 2020] va beaucoup plus loins en proposant des parties hors programme : l'algèbre relationnel, les dépendances fonctionnelles et les formes normales.

	Modèle relationnel de base	Contraintes d’intégrité	Algèbre relationnel	Dépendances fonctionnelles	Formes normales & normalisation
[Connan et al. 2020]	x	x			
[Bonnefoy-Petit 2020]	x	x			
[Balabonski et al. 2020]	x	x			
[Bays 2020]	x	assez diffus	x	x	FN

8.2.2. SGBD

Pour la partie SGBD, deux des quatre ouvrages abordent trop partiellement la modélisation conceptuelle et la passage vers le modèle relationnel. C'est sans doute trop (par rapport au programme) ou trop peu (par rapport à l'intérêt de cette approche). La partie historique des bases de données, objectif transversal dans le BO, est tout de même rapidement abordée par trois ouvrages. Les fonctions d'un SGBD, au programme, ne sont présentés que par deux ouvrages. Les vues et index, hors programme, ne sont évidemment pas abordées. Il en est de même de la partie programmation sur les SGBD (PL/SQL & triggers). Par contre, bien qu'aussi en dehors du programme, deux ouvrages abordent les transactions.

	MCD EAP	MCD → MLD	Historique	Fonctions	Transactions	Vues	Index
[Connan et al. 2020]	x (sans la représentation graphique)	x			très rapidement par l'usage		
[Bonnefoy-Petit 2020]	x (que des exemples avec asso binaires)	x	x	x			
[Balabonski et al. 2020]			x		x		
[Bays 2020]			rapide	x			

8.2.3. SQL

Pour la partie SQL, bien évidemment, la projection, la sélection et la jointure sont abordées par tous les ouvrages. Par contre, deux n'abordent pas le Distinct et aucun ne présente la jointure naturelle, la semi-jointure et la jointure externe. Les fonctions d'agrégation sont présentées par tous.

	Select (projection)	Distinct	Where (sélection)	(inter) Join..on	Natural join	Autre jointure	Fonctions de calcul
[Connan et al. 2020]	x		x	x			x
[Bonnefoy-Petit 2020]	x	x	x	x			x
[Balabonski et al. 2020]	x	x	x	x			x
[Bays 2020]	x		x	x			x

Deux ouvrages présentent les sous-requêtes (dans le From et dans le Where). Les regroupements, pas dans le programme, sont présentés dans deux ouvrages dont un seulement en mémo (ce qui ne sert pas à grand chose vu la complexité de cet opérateur). Le tri de la table résultat est systématiquement présenté.

	Sous-requête From	Sous-requête Where	Group by	Having	Tri
[Connan et al. 2020]			dans mémo + un exercice	dans mémo	x
[Bonnefoy-Petit 2020]					x
[Balabonski et al. 2020]	x	x	x (mais indique hors programme)		x
[Bays 2020]	pas vraiment	x			x

Pour la partie création et modification de la base de données, un seul livre aborde le LDD sur la base, mais tous présentent la création des tables et (presque tous) la suppression et modification de table, bien que ce ne soit pas au programme. L'ajout, la modification et la suppression de données sont présentés dans tous les ouvrages. Aucun évoque le LCD (Grant). Pour ce qui est de la programmation avec les bases de données, deux ouvrages abordent le sujet (en Python).

	Create, Drop Database	Create Table	Drop Table	Alter	INSERT Delete UPDATE	Grant	Python	Python requête paramétrées
[Connan et al. 2020]		x	x	x	x			
[Bonnefoy-Petit 2020]	x	x	x	x	x			
[Balabonski et al. 2020]		Assez développé, types, clé primaire, clé étrangère, null, contraintes utilisateur.	x		x		x	x
[Bays 2020]		très superficiel			x		x	

8.3. Partie exercices

8.3.1. SQL

Pour les exercices en SQL, on constate, comme beaucoup de site sur le Web, que la très grande majorité des questions (90% à 100%) est de la forme I→R. Ce n'est, hélas, pas une surprise. [Balabonski et al. 2020] est un peu plus original en proposant d'autres formes (30%) : R→T et R→I.

	IR-T	R-T	r-R	Ir-R	IrT-R	R-I	RT-I	T-R	IT-R	I-R
[Connan et al. 2020]						1				9
[Bonnefoy-Petit 2020]									1	10
[Balabonski et al. 2020]		5				4				20
[Bays 2020]										68

On trouve aussi des exercices sur la modification de bases de données, mais pas vraiment complexes. Deux ouvrages proposent aussi des exercices de programmation.

	Insert, Delete, Update : R-T	Python	Manipulation de PHPMyAdmin			Transaction : propriété	Besoin → transaction
[Connan et al. 2020]	3	x					
[Bonnefoy-Petit 2020]	1						
[Balabonski et al. 2020]	x					2	5
[Bays 2020]		x	x				

8.3.2. Modèle relationnel

Tous les ouvrages proposent quelques exercices (2 à 4) sur la construction d'un modèle à partir d'un contexte.

	Contexte → modèle	tuples compatibles avec un modèle	Propriétés d'un modèle (attributs, tuples)	Normalisation
[Connan et al. 2020]	2	1	2	1 de manière intuitive
[Bonnefoy-Petit 2020]	2		2	
[Balabonski et al. 2020]	4	2	1	
[Bays 2020]	3		1	

8.3.3. Gestion de la BDD

En complément, on trouve des exercices (en marge tout de même) sur la création de tables en fonction du besoin, sur la manipulation des MCD (simpliste) et sur la compréhension des contraintes.

	Creation de table par besoin	Compréhention SQL create	Compréhension contraintes	MCD à analyser	MCD→ MLD	QCM
[Connan et al. 2020]	1					9+9
[Bonnefoy-Petit 2020]				2	2	
[Balabonski et al. 2020]	2	2	1			
[Bays 2020]						10

9. Conclusion

Dans ce rapport de recherche, nous avons commencé à poser les premiers éléments en didactique des bases de données et proposé quelques démarches de transposition didactique. Nous avons abordé aussi bien le problème de construction du modèle relationnel que les difficultés de l'enseignement de SQL. Nous avons aussi proposé des idées de projets possibles. La didactique en informatique et, plus particulièrement la didactique en bases de données doit être développée, expérimentée et approfondie.

10. Bibiliographie

Articles de recherche :

- [Ars1988] Arsac, J. (1988). La didactique de l'informatique : Un problème ouvert ? Dans *Actes du Colloque francophone sur la didactique de l'informatique* (p. 9-18).
- [BaB2001] Baron, G.-L. et Bruillard, E. (2001). Une didactique de l'informatique ? *Revue Française de Pédagogie*, *135*, 163-172.
- [Bro1999] Brousseau G., Théorie des situations didactiques, La Pensée Sauvage, 1999.
- [Bueno08] "Conception Méthodique des Bases de Données un Guide de Bonne Pratique", de G. Bueno, Ellipses Marketing (2008), Collection Technosup, ISBN:2729838708
- [Che1985] Chevallard Y., La transposition didactique, La Pensée Sauvage, 1985.
- [Dec2020] Christophe DECLERCQ, « Cours didactique MEEF »,
- [Dec2021] Christophe DECLERCQ "Didactique de l'informatique : une formation nécessaire", *Sticef*, vol. 28, numéro 3, 2021, DOI :10.23709/sticef.28.3.8
- [Dro2013] Drot-Delange, B. (2013). Enseigner l'informatique débranchée: Analyse didactique d'activités. *AREF*, 1-13.
- [DSS2017] Valentina Dagiene, Sue Sentance, Gabrielé Stupuriené, « Developing a Two-Dimensional Categorization System for

Educational Tasks in Informatics », Informatica, Vol 28, N°1, pp 23-44, January 2017.
- [ElS04] «Conception et architecture des bases de données», R. Elmasri, S. Navathe, Pearson Education, 2004, ISBN 2-7440-7055-6
- [Flu2019] Fluckiger. C. (2019). *Une approche didactique de l'informatique scolaire*. Presses universitaires de Rennes.
- [MAF2001] Daphne Miedema, Efthimia Aivaloglou, and George Fletcher. 2021. Identifying SQL Misconceptions of Novices: Findings from a Think-Aloud Study. In *Proceedings of the 17th ACM Conference on International Computing Education Research (ICER 2021), August 16–19, 2021, Virtual Event, USA.* ACM, New York, NY, USA 13 Pages. https://doi.org/10.1145/3446871.3469759
- [Sel2013] Cynthia C. Selby, « Computational Thinking: The Developing Definition », ITiCSE 2013, Canterbury, England, July 1-3, 2013
- [Sme1995] John B. Smelcer. 1995. User errors in database query composition. International Journal of Human-Computer Studies 42, 4 (1995), 353–381. 29 pages https://dl.acm.org/doi/10.1006/ijhc.1995.1017 https://www-sciencedirect-com.budistant.univ-nantes.fr/science/article/pii/S1071581985710178?via%3Dihub
- [Win2006], Jeannette M. Wing, « Computational Thinking », Communications of th ACM, Vol. 49, N°3, March 2006, pp. 33-35

Ouvrages de référence en bases de données

- [AnV2001] Pascal André, Alain Vailly, "Conception des systèmes d'information : panorama des méthodes et des techniques", Ellipses, ISBN:2-7298-0479-X, 2001
- [AHV94] S. Abiteboul, R. Hull, V. Vianu, "Foundations of Databases", Addison Wesley ed., 1994 http://webdam.inria.fr/Alice/
- [CPTT2008] "Bases de données relationnelles : Concepts, mise en oeuvre et exercices", de C. Chrisment, K. Pinel-Sauvagnat, O. Teste, M. Tuffery, Hermes Science Publications (10 juin 2008), Collection Informatique, ISBN 978-2-7462-2086-7
- [Gar2005] «Bases de données», G. Gardarin, Eyrolles, 2005, ISBN 2-212-11281-5, http://georges.gardarin.free.fr/Livre_BD_Contenu/XX-TotalBD.pdf ; https://izibook.eyrolles.com/produit/2385/9782212175035/Bases%20de%20donnees
- [GUW2008] Database Systems - The Complete Book, Second Edition, Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom, Pearson, Année 2008

Ouvrages destinés aux élèves :

- [Connan et al. 2020] Prépa bac, Spécialité NSI, Tle générale, G. Connan, V. Petrov, G. Rozsavolgyi, L. Signac, Hatier, 2020, 978-2-401-06461-4, 352 pages
- [Bonnefoy-Petit 2020] Numérique et sciences informatiques Tle, J.-C. Bonnefoy, B. Petit, Ellipses, 2020, 9782340-038158, 332 pages.
- [Balabonski et al. 2020] Numérique et sciences informatiques : 24 leçons avec exercices corrigés, T. Balabonski, S. Conchon, J.-C. Filliâtre, K. Nguyen, Ellipses, 2020, 515 pages.
- [Bays 2020] Spécialité Numérique et sciences informatiques, S. Bays, Ellipses, 2020, 9782340-038448, 424 pages

Sites Web institutionnels :

- [BO2019a] *Programme d'enseignement de spécialité de numérique et sciences informatiques de la classe de première de la voie générale*, Bulletin officiel spécial n°1 du 22 janvier 2019, https://www.education.gouv.fr/bo/19/Special1/MENE1901633A.htm . https://cache.media.education.gouv.fr/file/SP1-MEN-22-1-2019/26/8/spe633_annexe_1063268.pdf
- [BO2019b] *Programme de l'enseignement de spécialité de numérique et sciences informatiques de la classe terminale de la voie générale*, Bulletin officiel spécial n°8 du 25 juillet 2019, https://www.education.gouv.fr/bo/19/Special8/MENE1921247A.htm . https://cache.media.education.gouv.fr/file/SPE8_MENJ_25_7_2019/93/3/spe247_annexe_1158933.pdf
- [NU-Li-Info2021b] Licence Informatique, Faculté des sciences et techniques, Nantes Université https://sciences-techniques.univ-nantes.fr/formations/licences-generales/licence-informatique ; https://sciences-techniques.univ-nantes.fr/programme-l1-mip-parcours-informatique
- [NU-Li-Info2021b] Licence Informatique, Faculté des sciences et techniques, Nantes Université https://sciences-techniques.univ-nantes.fr/formations/licences-generales/licence-informatique ; https://sciences-techniques.univ-nantes.fr/programme-l3-informatique-parcours-informatique