

Migration de données fonctorielle

David I. Spivak

Résumé

Dans cet article, nous présentons un langage de définition de bases de données simple : celui des catégories et des foncteurs. Un schéma de base de données est une petite catégorie et une instance est un foncteur à valeurs dans les ensembles. Nous montrons que les morphismes de schémas induisent trois “foncteurs de migration de données”, qui traduisent les instances d’un schéma à l’autre de manière canonique. Ces foncteurs paramétrisent les projections, les unions et les jointures sur toutes les tables simultanément et peuvent être utilisés à la place des requêtes conjonctives et disjonctives. Nous montrons également comment connecter une base de données et un langage de programmation fonctionnel en introduisant une connexion fonctorielle entre le schéma et la catégorie des types de ce langage. Nous commençons l’article par une multitude d’exemples pour motiver les définitions, et vers la fin, nous fournissons un dictionnaire permettant de traduire les concepts de bases de données en concepts catégoriques et vice-versa.

1. Introduction

Cet article a deux objectifs principaux. Le premier est de présenter un modèle catégorique simple des bases de données selon lequel tout théorème sur les petites catégories devient un théorème sur les bases de données. Pour ce faire, nous présenterons une catégorie Sch des schémas de bases de données, qui possède trois caractéristiques importantes :

- la catégorie Sch est équivalente à Cat , la catégorie des petites catégories,
- la catégorie Sch est un modèle fidèle pour les schémas de bases de données réels, et
- la catégorie Sch sert de fondation sur laquelle les concepts de bases de données de haut niveau reposent facilement et harmonieusement.

Le second objectif est d’appliquer cette formulation catégorique pour fournir de nouveaux foncteurs de migration de données, afin que pour toute traduction de schémas $F : \mathcal{C} \rightarrow \mathcal{D}$, on puisse transporter les instances du schéma source $\mathcal{C}$ vers les instances du schéma cible $\mathcal{D}$ et vice-versa, avec des propriétés d’“aller-retour” prouvables. Par exemple, les homomorphismes d’instances sont préservés par tous les foncteurs de migration. Bien que ces foncteurs de migration ne semblent pas avoir été discutés dans la littérature sur les bases de données, leurs analogues sont bien connus dans la théorie moderne des langages de programmation, par exemple la théorie des types dépendants ([Mar]), et des types de données polynomiaux ([Kan]). Cela fait partie d’une connexion plus profonde entre les schémas de bases de données et les *genres* (collections structurées de types, voir [SH], [Ham]) dans les langages de programmation. Voir aussi la section 3.6.

Un nombre croissant de chercheurs dans une variété croissante de disciplines constatent que les catégories et les foncteurs offrent des modèles de haute qualité, qui simplifient et unifient leurs domaines respectifs. La qualité d’un modèle doit être jugée par son efficacité en tant que proxy ou interface - c’est-à-dire par la facilité avec laquelle un expert peut travailler avec la seule compréhension du modèle, et ce faisant, opérer avec succès sur la chose elle-même. Notre objectif dans cet

Référence : <https://arxiv.org/pdf/1009.1166>.

Traduction : Denise Vella-Chemla, deepseek, octobre 2026.

article est de fournir un modèle de haute qualité des bases de données.

D’autres modèles catégoriques de bases de données ont été présentés dans le passé ([JoM], [BCW], [JRW], [IP], [PS], [DC], [Tui]).

Presque tous utilisaient la notion plus expressive d’esquisse (sketch) là où nous avons utilisé des catégories. L’expressivité supplémentaire a eu un coût qui peut être formulé en termes de nos deux objectifs pour cet article. Premièrement, les modèles précédents étaient plus complexes et cela a pu créer une barrière à la compréhension et à l’adoption à grande échelle. Deuxièmement, les morphismes d’esquisses n’induisent généralement pas les types de foncteurs de migration de données que les morphismes de catégories induisent.

Nous espérons que le présent modèle est suffisamment simple pour que quiconque ayant une compréhension élémentaire des catégories (c’est-à-dire qui connaît la définition de catégorie, foncteur et transformation naturelle) puisse, sans trop de difficulté, comprendre l’idée de base de notre formulation : les schémas de bases de données en tant que catégories, les instances de bases de données en tant que foncteurs. De plus, nous fournirons un dictionnaire (voir section 3.7, Tableau 1) grâce auquel les principaux résultats et définitions de cet article correspondront simplement à des résultats ou définitions de la théorie des catégories standard ; ainsi, le lecteur pourra s’appuyer sur des sources éprouvées pour expliquer les idées plus techniques présentées ici. De plus, on peut espérer tirer parti de la théorie mathématique existante pour ses propres problèmes de bases de données grâce à cette connexion.

Avant de décrire le plan de l’article dans la section 1.2, nous donnerons une brève introduction à l’idée fondamentale sur laquelle repose l’article, et fournirons une “forme normale catégorique” correspondante pour les bases de données, dans la section 1.1.

1.1. Forme normale catégorique

Un schéma de base de données peut contenir des centaines de tables et de clés étrangères. Chaque clé étrangère relie une table à une autre, et chaque séquence de clés étrangères $T_1 \rightarrow T_2 \rightarrow \dots \rightarrow T_n$ résulte en une fonction f de l’ensemble des enregistrements de T_1 vers l’ensemble des enregistrements de T_n . Il est courant que deux chemins de clés étrangères différents, reliant tous deux la table T_1 à la table T_n , existent ; et ils peuvent ou non définir la même application au niveau des enregistrements. Par exemple, un téléphone “fixe” opérationnel se voit attribuer un numéro de téléphone dont l’indicatif régional correspond à la région où se trouve le téléphone physique. Ainsi, nous avons deux chemins de OLP (LigneTelOperationnelle) vers R (Region) $OLP \rightarrow R$:

$$\begin{array}{ccccc}
 & & \text{NumTel} & \xrightarrow{\text{a}} & \text{CodeDeRegion} \\
 & \swarrow \text{Assigne} & & & \downarrow \text{Corresp} \\
 \text{LigneTelOperationnelle} & \xrightarrow{\text{est}} & \text{TelephonePhysique} & \xrightarrow{\text{SitueDans}} & \text{Region}
 \end{array} \tag{1}$$

et l’architecte de données pour ce schéma sait si ces deux chemins doivent toujours produire la même application. Dans (1), les deux chemins $OLP \rightarrow R$ produisent la même application, et le signe $\simeq$ est destiné à enregistrer ce fait. Par contraste, si nous remplaçons `operationalLandlinePhone`

par `operationalMobilePhone` (OMP), les deux chemins $OMP \rightarrow R$ ne produiraient pas la même application, car un téléphone portable n'a pas besoin d'être actuellement situé dans la région indiquée par son indicatif régional. Ainsi, nous obtiendrions un diagramme similaire mais différent,

$$\begin{array}{ccccc}
 & & \text{NumTel} & \xrightarrow{a} & \text{CodeRegion} \\
 & \nearrow \text{assigne} & & & \downarrow \text{corresp} \\
 \text{TelephoneMobileOperationnel} & \xrightarrow{\text{est}} & \text{TelephonePhysique} & \xrightarrow{\text{SitueDans}} & \text{Region}
 \end{array} \quad (2)$$

Nous soulignons ici que la notion d'équivalence de chemin est une contrainte d'intégrité importante et naturelle, qui fournit un indice crucial sur la sémantique prévue du schéma. Son application est souvent laissée à la couche applicative, mais elle devrait en fait être incluse dans le schéma ([JD]). L'inclusion d'informations d'équivalence de chemin dans le schéma de base de données a trois avantages principaux :

- elle permet l'inclusion de colonnes de requête “chaudes” sans redondance,
- elle fournit une vérification importante pour la création de mappages de schémas, et
- elle favorise une évolution saine du schéma.

Une catégorie (au sens mathématique) est à peu près un graphe avec un petit supplément de pouvoir expressif : la capacité de déclarer deux chemins équivalents. Nous avons maintenant la connexion souhaitée entre les schémas de bases de données et les catégories : les tables d'un schéma sont spécifiées par des sommets (ou comme nous les avons dessinés dans le diagramme (1), par des boîtes) ; les colonnes sont spécifiées par des flèches ; et l'équivalence fonctionnelle des chemins de clés étrangères est spécifiée par la notion catégorique d'équivalence de chemin (indiquée par le symbole $\simeq$). La définition catégorique du schéma sera présentée rigoureusement dans la section 3.

La collection d'idées ci-dessus nous conduit à la forme normale suivante pour les bases de données.

Définition 1. *Une base de données est sous forme normale catégorique si*

- chaque table t a une seule colonne de clé primaire ID_t , choisie au départ. Les cellules de cette colonne sont appelées les *id-de-ligne* de t ;
- pour chaque colonne c d'une table t , il existe une table cible t' telle que la valeur dans chaque cellule de la colonne c se réfère à un *id-de-ligne* de t' . Nous dénotons cette relation par

$$c : t \rightarrow t';$$

- en particulier, si une colonne d de t consiste en “données pures” (comme des chaînes de caractères ou des entiers), alors sa table cible t' est simplement une table à 1 colonne (c'est-à-dire un vocabulaire contrôlé, contenant au moins le domaine actif de la colonne d), et nous écrivons toujours $d : t \rightarrow t'$; et
- lorsqu'il y a deux chemins p, q à travers la base de données de la table t à la table u (notés $p : t \rightarrow u$, $q : t \rightarrow u$) et qu'il est connu du concepteur du schéma que p et q doivent correspondre à la même application des *id-de-lignes*, alors cette équivalence de chemin doit être déclarée comme faisant partie du schéma. Nous dénotons cette équivalence de chemin par

$$p \simeq q.$$

1.2. Plan de l'article

Nous commençons à la section 2 par une variété d'exemples, qui illustrent les vertus de l'approche catégorique. Ceux-ci incluent la clarté conceptuelle, la migration simplifiée des données, les vues mises à jour, l'interopérabilité avec les données RDF, et une connexion étroite entre les données et le programme. Dans la section 3, nous donnerons les définitions précises des schémas catégoriques et des traductions, et montrerons que la catégorie de ceux-ci, notée *Sch*, est équivalente à *Cat*, la catégorie des petites catégories. Dans cette section, nous définirons également la catégorie des instances de base de données sur un schéma donné. Dans la section 4, nous définirons les foncteurs de migration de données associés à une traduction et commencerons à conclure notre tour en revenant aux exemples de la section 2 pour un traitement plus approfondi. Nous terminons ce travail dans la section 5, où nous discutons des types de données et du filtrage.

1.3. Terminologie et notation

Un obstacle à l'écriture de cet article est un certain chevauchement de terminologie entre les bases de données et les catégories : le mot “objet” est couramment utilisé dans les deux contextes. Bien que les bases de données orientées objet soient intéressantes et peut-être pertinentes pour certaines des idées présentées ici, nous n'en discuterons pas du tout, gardant ainsi l'espace de noms clair pour la terminologie catégorique. Sauf indication contraire, le mot objet sera toujours entendu au sens catégorique.

Disons que nous avons des applications $A \xrightarrow{f} B \xrightarrow{g} C$; nous pouvons dénoter leur composé $A \rightarrow C$ de l'une des deux manières suivantes, selon ce qui semble le plus lisible. La première est appelée “ordre diagrammatique” et s'écrit $f;g$. La seconde est appelée “ordre classique” et s'écrit $g \circ f$. Nous pouvons parfois choisir de ne pas écrire de symbole entre f et g , et dans ce cas nous utilisons l'ordre diagrammatique $fg : A \rightarrow C$.

1.4. Remerciements

Le présent article est une refonte totale de [Sp1], qui, dans une tentative d'accommoder deux communautés disjointes (mathématiciens et experts en bases de données), a fini par être un document tentaculaire et quelque peu incohérent. Mes remerciements vont aux arbitres et à Bob Harper pour m'avoir orienté vers une présentation épurée. Je voudrais également remercier Peter Gates, Dave Balaban, John Launchbury et Greg Morrisett pour de nombreuses conversations utiles. J'apprécie beaucoup les encouragements de Jack Morava, en particulier sous la forme de sa vision ([Mor]) de la façon dont les idées de cet article peuvent être utiles pour exposer des motifs en mathématiques pures. Des remerciements spéciaux vont également à Scott Morrison pour avoir codé beaucoup des idées de cet article sous forme fonctionnelle, disponible en ligne pour démonstration ou participation open-source à <http://categoricaldata.net/>.

2. Vertus par l'exemple

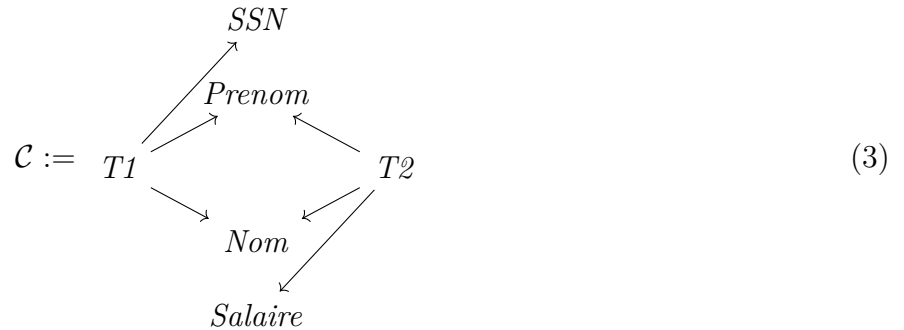
Dans ce qui suit, nous illustrons les mérites de l'approche catégorique au moyen de plusieurs exemples. Une chose à noter est que chacune de ces caractéristiques découle naturellement de

nos définitions mathématiques compactes des schémas et des traductions. Ces définitions seront données dans la section 3.

2.1. Clarté conceptuelle

Dans un schéma catégorique (Définition 3.2.6), chaque table est un sommet et chaque colonne est une flèche. Une flèche $T \xrightarrow{c} U$ représente une colonne de la table T, avec la table cible U, c’est-à-dire une contrainte de clé étrangère déclarant que chaque cellule de la colonne c se réfère à un id-de-ligne de la table U. Nous dessinons une boîte autour de notre système de sommets et de flèches, et le résultat est une représentation catégorique du schéma.

Exemple 2. L’image suivante représente un schéma S avec six tables, dont deux sont des “tables de faits” multi-colonnes et quatre sont des tables “feuilles” à 1 colonne.



La table de faits $T1$ a trois colonnes (pointant vers SSN , $Prenom$, Nom), en plus de sa colonne ID ; la table de faits $T2$ a également trois colonnes non- ID (pointant vers $Prenom$, Nom , $Salaire$). Les tables feuilles, SSN , $Prenom$, Nom et $Salaire$ n’ont que des colonnes ID , comme on le voit par le fait qu’aucune flèche n’en émane.

En tant qu’ensemble de tables, une instance sur le schéma C peut ressembler à ceci :

$T1$				$T2$			
ID	SSN	$Prenom$	Nom	ID	$Prenom$	Nom	$Salaire$
$T1-001$	115-234	Bob	Smith	$T2-A101$	Alice	Jones	\$100
$T1-002$	122-988	Sue	Smith	$T2-A102$	Sam	Miller	\$150
$T1-003$	198-877	Alice	Jones	$T2-A104$	Sue	Smith	\$300
				$T2-A110$	Carl	Pratt	\$200

(4)

SSN	$Prenom$	Nom	$Salaire$
ID	ID	ID	ID
115-234	Adam	Jones	\$100
118-334	Alice	Miller	\$150
122-988	Bob	Pratt	\$200
198-877	Carl	Richards	\$250
342-164	Sam	Smith	\$300
	Sue		

(5)

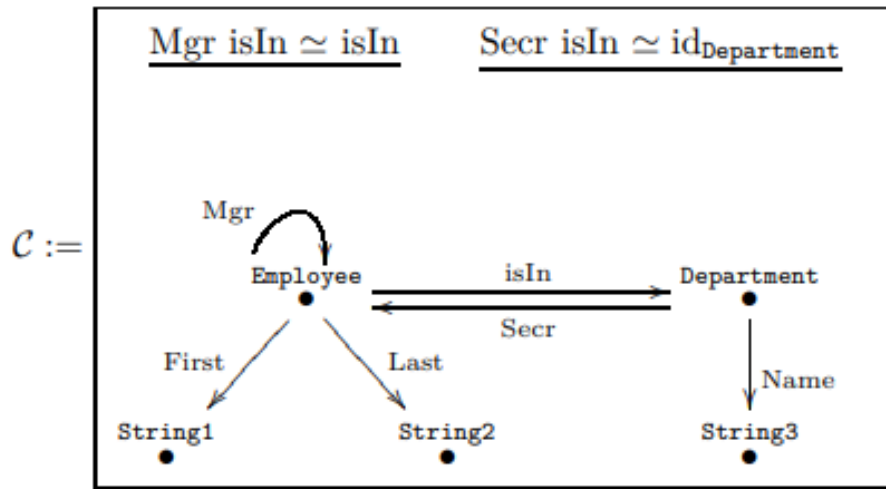
La chose à reconnaître ici est que chaque en-tête de colonne c de $T1$ (respectivement, de $T2$) pointe vers une table cible, de telle sorte que chaque cellule de la colonne c se réfère à un id-de-ligne de cette table cible. Les tables feuilles servent de vocabulaires contrôlés pour les tables de faits.

Notation 3. Dans l'exemple 2, nous avons écrit les quatre tables feuilles (Affichage 5). À l'avenir, nous n'écrirons généralement pas les tables feuilles pour des raisons d'espace. En fait, toute table qui n'ajoute pas de pouvoir explicatif à un exemple donné peut être omise de nos affichages.

Exemple 4. Dans cet exemple, nous présentons un schéma $\mathcal{C}$ qui inclut des équivalences de chemin, et tire donc parti de toute l'expressivité des catégories. Nous imaginons une entreprise avec des employés et des départements; chaque employé est dans un département, chaque employé a un manager employé, et chaque département a un employé secrétaire. En utilisant les équivalences de chemin, nous imposons les faits suivants :

- le manager d'un employé est dans le même département que cet employé, et
- le secrétaire d'un département est dans ce département.

Ces faits sont enregistrés comme des équations au sommet du diagramme suivant :



(6)

En tant qu'ensemble de tables, une instance sur $\mathcal{C}$ peut ressembler à ceci :

Employee					Department		
ID	Prenom	Nom	Mgr	isIn	ID	Name	Secr
101	David	Hilbert	103	q10	q10	Sales	101
102	Bertrand	Russell	102	x02	x02	Production	102
103	Alan	Turing	103	q10			

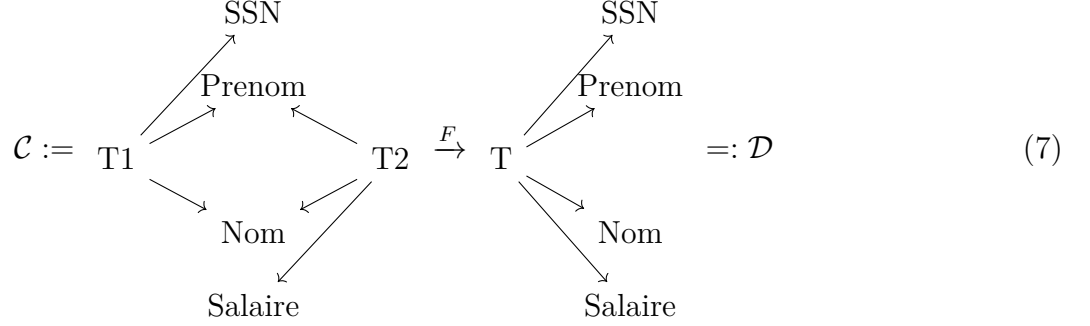
Ce n'est pas une coïncidence s'il y a un total de six colonnes non-ID dans les deux tables et un total de six flèches dans le schéma $\mathcal{C}$. Les équations peuvent être vérifiées sur ces cellules; par exemple, nous pouvons vérifier la première équation sur le id-de-ligne 101 :

$$101.\text{Manager.isIn} = 103.\text{isIn} = q10 \quad \text{et} \quad 101.\text{isIn} = q10.$$

Une instance I sur $\mathcal{C}$ n'est valide que si les deux équations tiennent pour chaque ligne de I .

2.2. Migration simplifiée des données

Une traduction $F : \mathcal{C} \rightarrow \mathcal{D}$ de schémas (Définition 3.3.1) est une application qui envoie les sommets de $\mathcal{C}$ vers les sommets de $\mathcal{D}$ et les flèches de $\mathcal{C}$ vers des *chemins* de $\mathcal{D}$; ce faisant, elle doit respecter les sources des flèches, les cibles des flèches et les équivalences de chemin. Nous utiliserons la traduction suivante $F : \mathcal{C} \rightarrow \mathcal{D}$ tout au long de la section 2.2.



L'application F est dessinée de manière aussi suggestive que possible. À l'avenir, nous nous fierons à ce “pouvoir de suggestion” pour indiquer les traductions, mais cette fois nous serons explicites. Chacun des quatre sommets feuilles, SSN, Prenom, Nom et Salaire dans $\mathcal{C}$ est envoyé vers le sommet de $\mathcal{D}$ de même étiquette. Les deux autres sommets de $\mathcal{C}$, à savoir T1 et T2, sont envoyés vers le sommet T de $\mathcal{D}$. Comme les traductions doivent respecter les sources et les cibles des flèches, il n'y a pas de choix supplémentaire quant à l'endroit où les flèches de $\mathcal{C}$ sont envoyées ; par exemple F envoie la flèche $T1 \rightarrow \text{Prenom}$ de $\mathcal{C}$ vers la flèche $T \rightarrow \text{Prenom}$ de $\mathcal{D}$.

Nous avons maintenant spécifié une traduction F du schéma $\mathcal{C}$ vers le schéma $\mathcal{D}$, illustrée dans le diagramme 7. De cette traduction jaillissent trois foncteurs de migration de données, que nous discuterons tour à tour dans les exemples 2.2.1, 2.2.3 et 2.2.5. Ils migrent les données d'instance sur $\mathcal{D}$ vers des données d'instance sur $\mathcal{C}$ et vice-versa. Les instances sur $\mathcal{C}$ (respectivement sur $\mathcal{D}$) forment une catégorie, que nous dénotons par $\mathcal{C}\text{-Inst}$ (respectivement par $\mathcal{D}\text{-Inst}$) ; elles sont définies dans la définition 3.5.1. Nous avons le tableau suivant, dont le jargon sera introduit sous peu :

Foncteurs de migration de données induits par une traduction $F : \mathcal{C} \rightarrow \mathcal{D}$			
Nom	Symbole	Symbole long	Idée de la définition
Pullback	Δ_F	$\Delta_F : \mathcal{D}\text{-Inst} \rightarrow \mathcal{C}\text{-Inst}$	Composition avec F
Pushforward droit	Π_F	$\Pi_F : \mathcal{C}\text{-Inst} \rightarrow \mathcal{D}\text{-Inst}$	Adjoint à droite de Δ_F
Pushforward gauche	Σ_F	$\Sigma_F : \mathcal{C}\text{-Inst} \rightarrow \mathcal{D}\text{-Inst}$	Adjoint à gauche de Δ_F

Tout dans le tableau sera défini dans la section 4. Pour l'instant, nous donnons trois exemples. Dans chacun, nous partons de la traduction $F : \mathcal{C} \rightarrow \mathcal{D}$, donnée ci-dessus dans le diagramme (7).

Exemple 5 (Pullback). Soit $F : \mathcal{C} \rightarrow \mathcal{D}$ la traduction donnée dans le diagramme (7). Dans cet exemple, nous explorons le foncteur de migration de données $\Delta_F : \mathcal{D}\text{-Inst} \rightarrow \mathcal{C}\text{-Inst}$ en l'appliquant à une instance $\mathcal{D}$ -instance J . Notez que même si notre traduction F pointe vers l'avant (de $\mathcal{C}$ vers $\mathcal{D}$), notre foncteur de migration Δ_F pointe “vers l'arrière” (des instances $\mathcal{D}$ vers les instances $\mathcal{C}$). Nous verrons pourquoi cela fonctionne ainsi, mais d'abord, ramenons la discussion sur terre en travaillant avec une instance $\mathcal{D}$ particulière.

Considérons l'instance J , sur le schéma $\mathcal{D}$, définie par la table

$$J := \begin{array}{c|c|c|c|c} & \text{ID} & \text{SSN} & \text{Prenom} & \text{Nom} & \text{Salaire} \\ \hline & XF667 & 115-234 & Bob & Smith & \$250 \\ & XF891 & 122-988 & Sue & Smith & \$300 \\ & XF221 & 198-877 & Alice & Jones & \$100 \end{array} \quad (8)$$

et ayant les quatre tables feuilles de l'exemple 2, Affichage (5). En tirant en arrière le long de la traduction F , nous sommes censés obtenir une instance $\Delta_F(J)$ sur le schéma $\mathcal{C}$, que nous devons décrire. Mais la description est facile : $\Delta_F(J)$ divise les colonnes de la table T selon la traduction F . Les quatre tables feuilles seront exactement les mêmes que ci-dessus (c'est-à-dire les mêmes que dans l'exemple 2 (5)), et les deux tables de faits seront quelque chose comme

$$\begin{array}{c|c|c|c} & \text{ID} & \text{SSN} & \text{Prenom} & \text{Nom} \\ \hline & XF667T1 & 115-234 & Bob & Smith \\ & XF891T1 & 122-988 & Sue & Smith \\ & XF221T1 & 198-877 & Alice & Jones \end{array} \quad \begin{array}{c|c|c|c} & \text{ID} & \text{Prenom} & \text{Nom} & \text{Salaire} \\ \hline & A21 & Alice & Jones & \$100 \\ & A67 & Bob & Smith & \$250 \\ & A91 & Sue & Smith & \$300 \end{array} \quad (9)$$

Le fait que $T1$ et $T2$ soient simplement des projections de T est un résultat de notre choix de traduction F .

Remarque 6. Nous avons vu que **le foncteur de pullback** Δ_F , qui surgit naturellement pour toute traduction F entre schémas, **produit automatiquement des projections**.

Dans les deux exemples suivants, nous explorerons les foncteurs de migration pushforward droit et gauche induits par la traduction $F : \mathcal{C} \rightarrow \mathcal{D}$ donnée dans le diagramme (7). Ces foncteurs, notés Π_F et Σ_F , envoient les instances $\mathcal{C}$ vers les instances $\mathcal{D}$. Ainsi, nous commençons avec l'instance I (qui a été présentée dans l'exemple 2) et expliquons ses pushforwards $\Pi_F(I)$ et $\Sigma_F(I)$ ci-dessous dans les exemples 7 et 9, respectivement.

Exemple 7 (Pushforward droit). Soit $F : \mathcal{C} \rightarrow \mathcal{D}$ la traduction donnée dans le diagramme (7). Dans cet exemple, nous explorons le foncteur de migration de données $\Pi_F : \mathcal{C}\text{-Inst} \rightarrow \mathcal{D}\text{-Inst}$ en l'appliquant à l'instance $\mathcal{C}$ -instance I montrée dans les affichages (4) et (5). Notez que notre foncteur de migration Π_F pointe dans la même direction que F : il prend les instances $\mathcal{C}$ vers les instances $\mathcal{D}$. Nous décrivons maintenant l'instance $\mathcal{D}$ -instance $\Pi_F(I)$, qui a quatre tables feuilles $\Pi_F(I)(SSN)$, etc., et une table de faits $\Pi_F(I)(T)$.

Les quatre tables feuilles de $\Pi_F(I)$ seront comme dans l'affichage (5). La table de faits de $\Pi_F(I)$ sera la jointure de $T1$ et $T2$:

T				
ID	SSN	$Prenom$	Nom	$Salaire$
$T1-002T2-A104$	122-988	Sue	Smith	\$300
$T1-003T2-A101$	198-877	Alice	Jones	\$100

Remarque 8. Nous avons vu que **le foncteur de pushforward droit** Π_F , qui surgit naturellement pour toute traduction F entre schémas, **produit automatiquement des jointures**.

Exemple 9 (Pushforward gauche). Soit $F : \mathcal{C} \rightarrow \mathcal{D}$ la traduction donnée dans le diagramme (7). Dans cet exemple, nous explorons le foncteur de migration de données $\Sigma_F : \mathcal{C}\text{-Inst} \rightarrow \mathcal{D}\text{-Inst}$ en l’appliquant à l’instance $\mathcal{C}$ -instance I montrée dans les affichages (4) et (5). Notez que notre foncteur de migration Σ_F pointe dans la même direction que F : il prend les instances $\mathcal{C}$ vers les instances $\mathcal{D}$. Nous décrivons maintenant l’instance $\mathcal{D}$ -instance $\Sigma_F(I)$, qui a quatre tables feuilles $\Sigma_F(I)(SSN)$, etc., et une table de faits $\Sigma_F(I)(T)$.

Au lieu d’être une jointure, comme dans le cas de $\Pi_F(I)$ ci-dessus, la table de faits T dans l’instance $\Sigma_F(I)$ sera l’union de $T1$ et $T2$. On peut se demander alors comment la construction catégorique traitera le fait que les enregistrements de $T1$ n’ont pas d’information de salaire et que les enregistrements de $T2$ n’ont pas d’information de SSN. La réponse est que les cellules respectives sont skolémisées. En d’autres termes, la réponse universelle est de simplement ajouter une toute nouvelle “variable” partout où cela est nécessaire dans et en aval de T . Ainsi, dans l’instance $\Sigma_F(I)$, la table T ressemble à ceci :

T				
<i>ID</i>	<i>SSN</i>	<i>Prenom</i>	<i>Nom</i>	<i>Salaire</i>
$T1-001$	$115-234$	Bob	$Smith$	$T1-001.Salaire$
$T1-002$	$122-988$	Sue	$Smith$	$T1-002.Salaire$
$T1-003$	$198-877$	$Alice$	$Jones$	$T1-003.Salaire$
$T2-A101$	$T2-A101.SSN$	$Alice$	$Jones$	$\$100$
$T2-A102$	$T2-A102.SSN$	Sam	$Miller$	$\$150$
$T2-A104$	$T2-A104.SSN$	Sue	$Smith$	$\$300$
$T2-A110$	$T2-A110.SSN$	$Carl$	$Pratt$	$\$200$

Les variables de Skolem (telles que $T1-001.Salaire$) peuvent être assimilées à des valeurs réelles plus tard. Elles peuvent également être assimilées les unes aux autres ; par exemple, nous pouvons savoir que $T1-001.Salaire = T2-002.Salaire$, sans connaître la valeur de ces salaires.

Remarque 10. Nous avons vu que **le foncteur de pushforward gauche** Σ_F , qui surgit naturellement pour toute traduction F entre schémas, **produit automatiquement des unions et skolémise automatiquement les valeurs inconnues.**

2.3. Vues mises à jour et multi-vues liées

Supposons que nous ayons une traduction $F : \mathcal{C} \rightarrow \mathcal{D}$. Dans ce cas, nous pouvons considérer $\mathcal{D}$ comme une vue sur $\mathcal{C}$ et considérer $\mathcal{C}$ comme une vue sur $\mathcal{D}$. Contrairement à la version classique des vues, notre définition permet un nombre arbitrairement grand de clés étrangères entre les tables de vue ; en effet, $\mathcal{C}$ et $\mathcal{D}$ peuvent être des schémas arbitraires. Les systèmes de gestion de bases de données relationnelles typiques tels que SQL ne supportent pas les “multi-vues liées”, c’est-à-dire plusieurs tables de vue avec des clés étrangères entre elles. Pour nos foncteurs de migration de données Π_F , Σ_F et Δ_F , ce n’est pas un problème.

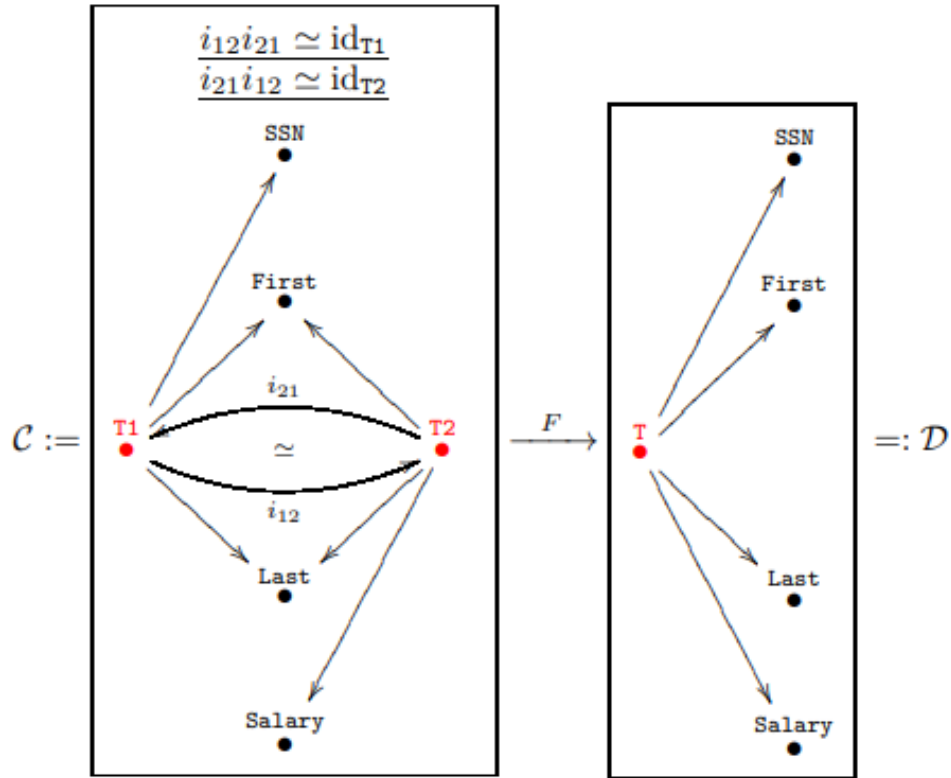
En fait, de par la nature même de ces trois foncteurs de migration (c’est-à-dire par définition du fait qu’ils sont des foncteurs), nous avons accès à de puissants théorèmes reliant les mises à jour des instances $\mathcal{C}$ aux mises à jour des instances $\mathcal{D}$. Par exemple, étant donné une instance I sur $\mathcal{D}$

dont la vue Δ_F est l'instance $J = \Delta_F(I)$ sur $\mathcal{C}$, et étant donné une mise à jour $J \rightarrow J'$ sur $\mathcal{C}$, il existe une unique mise à jour $I \rightarrow \Pi_F J'$, d'instances sur $\mathcal{D}$. Un résultat similaire tient pour Σ_F à la place de Π_F : ces faits découlent du fait que Π (respectivement Σ_F) est “adjoint” à Δ_F . Voir la section 4.

Le problème de la mise à jour des vues est souvent formulé comme demandant que “les allers-retours soient des équivalences” ([BCFGP]), ce qui pour nous revient à dire que les composés $\Delta_F \Pi_F$ et $\Pi_F \Delta_F$ (respectivement $\Sigma_F \Delta_F$ et $\Delta_F \Sigma_F$) sont des isomorphismes. Cela ne se produira que si F est une équivalence de catégories. Cependant, nos adjonctions de migration de données fournissent des mises à jour de vues dans des circonstances plus générales, et celles-ci ont des propriétés formelles prouvables (par exemple, Σ_F et Δ_F commutent avec les insertions et Π_F et Δ_F commutent avec les suppressions). Mais bien sûr, les meilleures propriétés formelles se produisent lorsque F est une équivalence.

L'exemple suivant montre deux choses. Premièrement, il donne un exemple de multi-vue liée (clés étrangères entre vues). Deuxièmement, la traduction F est une équivalence de catégories (un fait qui repose essentiellement sur le fait que $\mathcal{C}$ a des équivalences de chemin déclarées), et donc les foncteurs de migration de données Δ, Π et Σ sont aussi des équivalences - ils ne présentent aucune perte d'information.

Exemple 11. *Considérons les deux schémas dessinés ici :*



(10)

Les flèches i_{12} et i_{21} , qui sont déclarées mutuellement inverses, assurent que les données qui peuvent

être capturées par le schéma $\mathcal{C}$ sont équivalentes à celles qui peuvent être capturées par le schéma $\mathcal{D}$. La traduction F envoie i_{12} et i_{21} vers le chemin trivial id_T sur T (voir Définitions 3.2.1 et 3.3.1, et Exemple 3.3.3).

Si la table T est comme dans l'exemple 5 (8), alors son pullback sous Δ_F vers une instance sur $\mathcal{C}$ est similaire à (9), mais avec des colonnes supplémentaires i_{12} et i_{21} (parce que notre schéma a des flèches supplémentaires i_{12} et i_{21}) :

$T1$					$T2$				
ID	SSN	Prenom	Nom	i_{12}	ID	Prenom	Nom	Salaire	i_{21}
XF667T1	115-234	Bob	Smith	A67	A21	Alice	Jones	\$100	XF221T1
XF891T1	122-988	Sue	Smith	A91	A67	Bob	Smith	\$250	XF667T1
XF221T1	198-877	Alice	Jones	A21	A91	Sue	Smith	\$300	XF891T1

(11)

Les colonnes de clés étrangères i_{12} et i_{21} sur la vue $\mathcal{C}$ gardent la trace des données nécessaires pour un aller-retour réussi. Une mise à jour d'une instance $\mathcal{D}$ produira une mise à jour correspondante d'une instance $\mathcal{C}$ et vice-versa. Le fait que F soit une équivalence de catégories implique que Σ_F, Π_F et Δ_F sont aussi des équivalences de catégories, et les isomorphismes d'aller-retour tiendront pour toutes les mises à jour possibles.

2.4. Interopérabilité avec les données RDF

Le *Resource Descriptive Framework* (RDF) est le format de données standard du web sémantique [KC]. L'idée de base est d'encoder tous les faits en termes de triples de base

(Subject Predicate Object)

tels que (Bob hasMother Sue). Il y a des articles consacrés à la compréhension de la transformation des bases de données relationnelles vers les triple stores RDF, et vice-versa ([ASX],[KT]). Dans cette section, nous supposons une familiarité de base avec le jargon de ce domaine, tel que *URI* (uniform resource identifier).

Catégoriquement, la formulation des triple stores RDF est assez simple. Étant donné un schéma $\mathcal{C}$, un triple store sur $\mathcal{C}$ est une catégorie $\mathcal{S}$ (représentant les triples) et un foncteur $\pi : \mathcal{S} \rightarrow \mathcal{C}$ (représentant leurs types). Les objets de $\mathcal{S}$ sont des URI ; les flèches de $\mathcal{S}$ sont des triplets

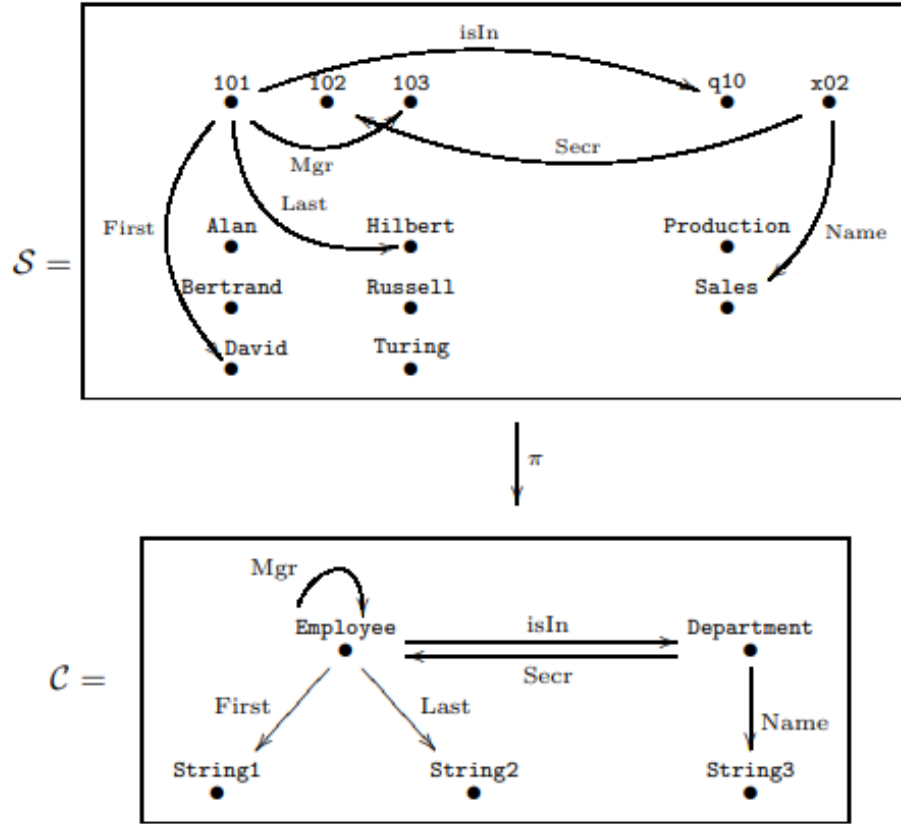
$$\underbrace{\text{subject}}_{\bullet} \xrightarrow{\text{Predicate}} \underbrace{\text{object}}_{\bullet}$$

Étant donné un objet $c \in \mathcal{C}$ dans le schéma, l'image inverse $\pi^{-1}(c) \subseteq \mathcal{S}$ consiste en toutes les URI de type c . Étant donné une flèche $f : c \rightarrow c'$ dans $\mathcal{C}$, l'image inverse est la f -relation entre $\pi^{-1}(c)$ et $\pi^{-1}(c')$.

Il existe une opération catégorique de base qui convertit une instance de base de données relationnelle en un triple store RDF (et un inverse simple également, convertissant un triple store RDF en une instance de base de données relationnelle). Elle est appelée la *construction de Grothendieck*. Considérons par exemple l'instance I de l'exemple 4, Affichage (6) :

Employee					Department		
ID	Prenom	Nom	Mgr	isIn	ID	Name	Secr
101	David	Hilbert	103	q10	q10	Sales	101
102	Bertrand	Russell	102	x02	x02	Production	102
103	Alan	Turing	103	q10			

En prenant la construction de Grothendieck, on obtient le triple store suivant $\mathcal{S} = Gr(I)$ où chaque flèche désigne un triple RDF, comme ci-dessus :



(12)

Dans l’affichage (12), dix flèches ont été omises de l’image de $\mathcal{S}$ (par exemple, la flèche $102 \xrightarrow{\text{Nom}} Russell$ n’est pas représentée) pour des raisons de lisibilité. Le point est que le triple store RDF associé à l’instance I est bien représenté en utilisant la construction standard de Grothendieck.

2.5. Connexion étroite entre les données et le programme

Actuellement, il y a un “décalage d’impédance” entre les bases de données et les langages de programmation ; leurs formulations respectives et leurs modèles sous-jacents ne cohèrent pas aussi bien qu’ils le devraient ([CI]). Alors que la communauté des langages de programmation (PL) a adopté la théorie des catégories pour la clarté conceptuelle et la puissance expressive qu’elle apporte, la plupart des théoriciens des bases de données ont tendance à se concentrer sur des considérations pratiques, telles que la vitesse, la fiabilité et l’évolutivité. L’importance des bases de données dans le monde moderne ne peut être surestimée, et pourtant, pour que les bases de données atteignent

leur plein potentiel, une meilleure intégration théorique avec les applications doit être développée.

Comme indiqué dans l'introduction (section 1), le premier objectif de cet article est de présenter un modèle simple de bases de données selon lequel tout théorème sur les petites catégories devient un théorème sur les bases de données. Ainsi, la catégorie préférée des théoriciens des PL, à savoir la catégorie *Type* des types et des termes (pour un λ -calcul fixé, voir [Awo, Section 6.5]), est une sorte de schéma de base de données infini : ses tables correspondent à des types et ses colonnes de clés étrangères correspondent à des termes. Bien sûr, contrairement aux bases de données réelles dans lesquelles les tables modélisent des entités du monde réel et leurs relations (comme les personnes et leurs tailles), le schéma *Type* modélise des entités mathématiques et leurs relations (comme les entiers et leurs factorielles). Cependant, ces idées vivent clairement dans le même royaume platonicien, pour ainsi dire, et cette notion est exprimée en disant que les schémas de bases de données et *Type* peuvent être considérés comme des catégories et reliés par des foncteurs.

Cela conduit à une belle intégration entre les données et le programme. Par exemple, de nombreuses capacités de tableur, comme la somme des valeurs de deux colonnes pour obtenir la valeur d'une troisième, peuvent être incluses au niveau du schéma. À ce stade de l'article, nous n'avons pas encore la machinerie nécessaire pour montrer exactement comment cela devrait fonctionner (voir section 5.1), mais dans le diagramme suivant, on peut voir la présentation schématique de la sous-catégorie pertinente de *Type* :

$$P := (\text{Int}, \text{Int}) \begin{array}{c} \xrightarrow{\text{outl}} \\ \xrightarrow{+} \\ \xrightarrow{\text{outr}} \end{array} \text{Int} \quad (13)$$

Le point est de voir simultanément deux choses différentes dans ce seul diagramme (comme une illusion d'optique). La première chose à voir dans le diagramme (13) est un schéma de base de données. Dans le schéma P , nous avons deux tables :

(Int,Int)					
ID	outl	outr	+		
P0c0	0	0	0	Int	ID
P1c0	1	0	1		
P1c1	1	1	2		
P0c1	0	1	1		
P2c0	2	0	2		
P2c1	2	1	3		
P2c2	2	2	4		
P1c2	1	2	3		
P0c2	0	2	2		
⋮	⋮	⋮	⋮		

La seconde chose à voir dans le diagramme (13) est une sous-catégorie $P \subseteq \text{Type}$, c'est-à-dire une connexion étroite avec la théorie standard des PL. La même catégorie P est vue extensionnellement dans le contexte des bases de données et intentionnellement dans le contexte des programmes. Cette

double citoyenneté des catégories fait de la théorie des catégories un bon candidat pour résoudre le décalage d'impédance entre les bases de données et les langages de programmation.

3. Définitions

Dans cette section, notre objectif principal est de définir une catégorie de schémas et de traductions, et de montrer qu'elle est équivalente à *Cat*, la catégorie des petites catégories. En cours de route, nous définirons la catégorie des instances sur un schéma donné. Enfin, nous donnerons un dictionnaire que l'on peut utiliser pour traduire entre les concepts de bases de données (par exemple, trouvés dans [EN]) et les concepts catégoriques (par exemple, trouvés dans [Mac]).

3.1. Quelques références

Tout au long de cette section, nous supposons que le lecteur est familier avec les notions fondamentales de la théorie des catégories : objets, morphismes et diagrammes commutatifs dans une catégorie ; ainsi que les catégories, foncteurs et transformations naturelles. Il existe de nombreuses bonnes références sur la théorie des catégories, notamment [LS], [Sic], [Pie], [BW1], [Awo] et [Mac] ; les première et deuxième sont adaptées au grand public, les troisième et quatrième aux informaticiens, et les cinquième et sixième aux mathématiciens (dans chaque classe, la première référence est plus facile que la seconde). On peut également consulter [SK] pour une perspective différente.

3.2. Graphes, chemins, schémas et instances

Un graphe (parfois appelé multi-graphe dirigé) est une collection de sommets et de flèches, ressemblant à ceci :

$$\begin{array}{ccc}
 A & \xrightarrow{f} & B \begin{array}{c} \xrightarrow{g} \\ \xleftarrow{h} \end{array} C \\
 & & \\
 i \hookrightarrow D & \begin{array}{c} \xrightarrow{j} \\ \xleftarrow{k} \end{array} & E
 \end{array} \tag{14}$$

C'est un graphe avec deux composantes connexes ; il a cinq sommets et six flèches.

Définition 12. *Un graphe G est une séquence $G = (A, V, \text{src}, \text{tgt})$, où A et V sont des ensembles (respectivement appelés l'ensemble des flèches et l'ensemble des sommets de G), et $\text{src} : A \rightarrow V$ et $\text{tgt} : A \rightarrow V$ sont des fonctions (respectivement appelées la fonction source et la fonction cible pour G). Si $a \in A$ est une flèche avec source $\text{src}(a) = v$ et cible $\text{tgt}(a) = w$, nous la dessinons comme*

$$v \xrightarrow{a} w.$$

Définition 13. *Soit $G = (A, V, \text{src}, \text{tgt})$ un graphe. Un chemin de longueur n dans G , noté $p \in \text{Path}_G^{(n)}$, est une séquence tête-à-queue*

$$p = (v_0 \xrightarrow{a_1} v_1 \xrightarrow{a_2} v_2 \xrightarrow{a_3} \dots \xrightarrow{a_n} v_n) \tag{15}$$

de flèches dans G . En particulier, $\text{Path}_G^{(1)} = A$ et $\text{Path}_G^{(0)} = V$; nous appelons le chemin de longueur 0 sur le sommet v le chemin trivial sur v et le notons id_v . Nous notons Path_G l'ensemble de tous les chemins sur G ,

$$\text{Path}_G := \bigcup_{n \in \mathbb{N}} \text{Path}_G^{(n)}.$$

Chaque chemin $p \in \text{Path}_G$ a un sommet source et un sommet cible, et nous pouvons abuser de la notation et écrire $\text{src}, \text{tgt} : \text{Path}_G \rightarrow V$. Si p est un chemin avec $\text{src}(p) = v$ et $\text{tgt}(p) = w$, nous pouvons le noter $p : v \rightarrow w$. Étant donné deux sommets $v, w \in V$, nous écrivons $\text{Path}_G(v, w)$ pour désigner l'ensemble de tous les chemins $p : v \rightarrow w$.

Il y a une opération de composition sur les chemins. Étant donné un chemin $p : v \rightarrow w$ et $q : w \rightarrow x$, nous définissons la composition, notée $pq : v \rightarrow x$, de manière évidente. En particulier, si p (resp. r) est le chemin trivial sur le sommet v (resp. le sommet w), alors pour tout chemin $q : v \rightarrow w$, nous avons $pq = q$ (resp. $qr = q$). Ainsi, pour plus de clarté, nous pouvons toujours noter un chemin comme commençant par un chemin trivial sur son sommet source ; par exemple, le chemin p du diagramme (15) peut être noté $p = \text{id}_{v_0} a_1 a_2 \cdots a_n$.

Exemple 14. Dans le diagramme (14), il n'y a pas de chemin de A à D , un chemin (f) de A à B , deux chemins (fg et fh) de A à C , et une infinité de chemins $\{i^{p_1}(jk)^{q_1} \cdots i^{p_n}(jk)^{q_n} \mid n, p_1, q_1, \dots, p_n, q_n \in \mathbb{N}\}$ de D à D .

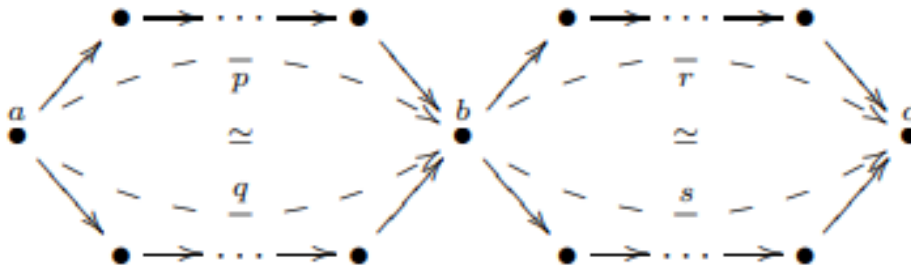
Nous définissons maintenant la notion de relation d'équivalence catégorique sur l'ensemble des chemins d'un graphe. Une telle relation d'équivalence (en plus d'être réflexive, symétrique et transitive) a deux sortes de propriétés supplémentaires : les chemins équivalents ont la même source et la même cible, et la composition de chemins équivalents avec d'autres chemins équivalents doit produire des chemins équivalents. Formellement, nous avons la définition 15.

Définition 15. Soit $G = (A, V, \text{src}, \text{tgt})$ un graphe. Une relation d'équivalence de chemin catégorique (ou CPER) sur G est une relation d'équivalence $\simeq$ sur Path_G qui a les propriétés suivantes :

1. Si $p \simeq q$ alors $\text{src}(p) = \text{src}(q)$.
2. Si $p \simeq q$ alors $\text{tgt}(p) = \text{tgt}(q)$.
3. Supposons que $p, q : b \rightarrow c$ soient des chemins, et $m : a \rightarrow b$ une flèche. Si $p \simeq q$ alors $mp \simeq mq$.
4. Supposons que $p, q : a \rightarrow b$ soient des chemins, et $n : b \rightarrow c$ une flèche. Si $p \simeq q$ alors $pn \simeq qn$.

Lemme 16. Supposons que G soit un graphe et $\simeq$ une CPER sur G . Supposons $p \simeq q : a \rightarrow b$ et $r \simeq s : b \rightarrow c$. Alors $pr \simeq qs$.

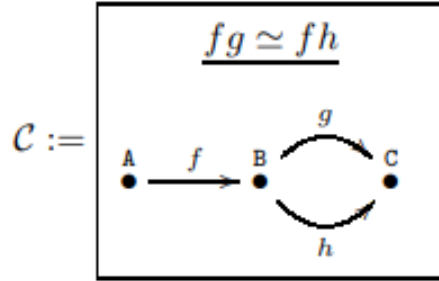
Démonstration. L'image à avoir en tête est celle-ci :



En appliquant la condition (3) de la définition 15 à chaque flèche du chemin p , il s'ensuit par induction que $pr \simeq ps$. En appliquant la condition (4) à chaque flèche du chemin s , il s'ensuit de même que $ps \simeq qs$. Parce que $\simeq$ est une relation d'équivalence, il s'ensuit que $pr \simeq qs$. $\square$

Définition 17. Un schéma catégorique $\mathcal{C}$ consiste en une paire $\mathcal{C} := (G, \simeq)$ où G est un graphe et $\simeq$ est une relation d'équivalence de chemin catégorique sur G . Nous appelons parfois un schéma catégorique simplement un schéma.

Exemple 18. Considérons le schéma, c'est-à-dire le graphe avec l'équivalence indiquée, représenté dans la boîte ci-dessous :



Ce schéma modélise, par exemple, le phénomène d'envoi d'un email à soi-même. Supposons que nous peuplions B avec des emails, C avec des personnes, g et h avec les champs expéditeur et destinataire, respectivement. Alors pour que fg soit égal à fh , nous devons avoir que les expéditeurs soient égaux aux destinataires sur l'image de f , et ainsi le sous-ensemble des auto-emails est un ajustement parfait pour A . Voir l'exemple 20.

Plus d'informations sur le sujet des schémas catégoriques, y compris une image d'un schéma et d'un ensemble de tables associé, peuvent être trouvées dans l'exemple 4.

Dans ce qui suit, nous définirons ce que signifie être une instance d'un schéma catégorique $\mathcal{C}$. Nous considérons le cas où nos instances sont des modèles ensemblistes de $\mathcal{C}$, mais la même idée fonctionne de manière beaucoup plus générale (voir définition 28).

Définition 19. Soit $\mathcal{C} := (G, \simeq)$ un schéma catégorique, où $G = (A, V, \text{src}, \text{tgt})$. Une instance sur $\mathcal{C}$, notée I , consiste en ce qui suit :

1. Pour chaque sommet $v \in V$, un ensemble $I(v)$.
2. Pour chaque flèche $a : v \rightarrow v'$ dans A , une fonction $I(a) : I(v) \rightarrow I(v')$.
3. Pour chaque équivalence de chemin $p \simeq q$, une garantie que l'équation $I(p) = I(q)$ tient.

Exemple 20. Nous revenons maintenant à l'exemple 18, et écrivons une instance exemple I pour le schéma $\mathcal{C} = (G, \simeq)$.

A		B			C	
ID	f	ID	g	h	ID	
SEm1207	Em1207	Em1206	Bob	Sue	Bob	
SEm1210	Em1210	Em1207	Carl	Carl	Carl	
SEm1211	Em1211	Em1208	Sue	Martha	Chris	
		Em1209	Chris	Bob	Julia	
		Em1210	Chris	Chris	Martha	
		Em1211	Julia	Julia	Sue	
		Em1212	Martha	Chris		

Pour chaque sommet v de G , l'ensemble $I(v)$ est donné par l'ensemble des lignes de la table correspondante (par exemple, $I(A) = \{\text{SEm1207}, \text{SEm1210}, \text{SEm1211}\}$). Pour chaque flèche $a : v \rightarrow w$ de G , la fonction $I(a) : I(v) \rightarrow I(w)$ est également évidente en tant que colonne dans la table. Par exemple, $I(g) : I(\text{textit{B}}) \rightarrow I(C)$ envoie Em1206 vers Bob , etc. Enfin, l'équivalence de chemin $fg = fh$ est confirmée par le fait que pour chaque id-de-ligne de la table A , suivre f puis g donne le même résultat que suivre f puis h .

3.3. Traductions

Une traduction est une application d'un schéma catégorique à un autre. Les sommets sont envoyés vers des sommets, les flèches sont envoyées vers des chemins, et toutes les équivalences de chemin sont préservées. Plus précisément, nous avons la définition suivante.

Définition 21. Soient $G = (A_G, V_G, \text{src}_G, \text{tgt}_G)$ et $H = (A_H, V_H, \text{src}_H, \text{tgt}_H)$ des graphes (voir définition 12), et soient $\mathcal{C} = (G, \simeq_{\mathcal{C}})$ et $\mathcal{D} = (H, \simeq_{\mathcal{D}})$ des schémas catégoriques. Une traduction F de $\mathcal{C}$ vers $\mathcal{D}$, notée $F : \mathcal{C} \rightarrow \mathcal{D}$, consiste en les constituants suivants :

1. une fonction $V_F : V_G \rightarrow V_H$, et
2. une fonction $A_F : A_G \rightarrow \text{Path}_H$

subjettées aux conditions suivantes :

1. la fonction A_F préserve les sources et les cibles ; en d'autres termes, les diagrammes d'ensembles suivants commutent :

$$\begin{array}{ccc}
 A_G & \xrightarrow{A_F} & \text{Path}_H \\
 \text{src}_G \downarrow & & \downarrow \text{src}_H \\
 V_G & \xrightarrow{V_F} & V_H
 \end{array}
 \qquad
 \begin{array}{ccc}
 A_G & \xrightarrow{A_F} & \text{Path}_H \\
 \text{tgt}_G \downarrow & & \downarrow \text{tgt}_H \\
 V_G & \xrightarrow{V_F} & V_H
 \end{array}$$

2. la fonction A_F préserve les équivalences de chemin. Précisément, supposons que l'on nous donne des longueurs $m, n \in \mathbb{N}$ et des chemins $p = \text{id}_{v_0} f_1 f_2 \cdots f_m$ et $q = \text{id}_{v_0} g_1 g_2 \cdots g_n$ dans G . Soit $v'_0 = V_F(v_0)$ et pour chaque $i \leq m$ (resp. $j \leq n$), soit $f'_i = A_F(f_i)$ (resp. $g'_j = A_F(g_j)$), et soit $p' = \text{id}_{v'_0} f'_1 f'_2 \cdots f'_m$ (resp. $q' = \text{id}_{v'_0} g'_1 g'_2 \cdots g'_n$). Si $p \simeq_{\mathcal{C}} q$ alors $p' \simeq_{\mathcal{D}} q'$.

Deux traductions $F, F' : \mathcal{C} \rightarrow \mathcal{D}$ sont considérées identiques si elles coïncident sur les sommets (c'est-à-dire $V_F = V_{F'}$) et si, pour chaque flèche f de $\mathcal{C}$, il y a une équivalence de chemin

$$A_F(f) \simeq_{\mathcal{D}} A_{F'}(f).$$

Dans les deux exemples suivants, nous reconsidérerons les traductions discutées dans la section 2.

Exemple 22. Rappelons l'application F donnée dans le diagramme (7). Les schémas $\mathcal{C}$ et $\mathcal{D}$ sont juste des graphes dans le sens où il n'y a pas d'équivalences de chemin déclarées dans l'un ou l'autre. L'application F envoie les sommets de $\mathcal{C}$ vers les sommets de $\mathcal{D}$ et les flèches de $\mathcal{C}$ vers les flèches de $\mathcal{D}$. Puisqu'une flèche est un type particulier de chemin, et puisqu'il n'y a pas d'équivalences de chemin à préserver, $F : \mathcal{C} \rightarrow \mathcal{D}$ est bien une traduction.

Exemple 23. Rappelons l'application F donnée dans l'exemple 11, diagramme (10). Dans cette configuration, $\mathcal{C}$ a des équivalences de chemin déclarées et $\mathcal{D}$ n'en a pas ; cependant $\mathcal{D}$ a encore une relation d'équivalence de chemin catégorique $\simeq_{\mathcal{D}}$ sur lui, la relation réflexive minimale. L'application F sur les sommets (V_F) est explicite ; les seules flèches sur lesquelles A_F n'est pas explicite sont i_{12} et i_{21} , toutes deux envoyées vers le chemin trivial id_T sur le sommet T .

Parce que $V_F(T1) = V_F(T2) = T$, il est clair que A_F préserve les sources et les cibles. Les équivalences de chemin $i_{12}i_{21} = \text{id}_{T1}$ et $i_{21}i_{12} = \text{id}_{T2}$ sont préservées parce que $A_F(i_{12}) = A_F(i_{21}) = \text{id}_T$, et la concaténation d'un chemin trivial avec n'importe quel chemin p donne p .

3.4. L'équivalence $Sch \simeq Cat$

Nous supposons une familiarité avec les catégories et les foncteurs, et en particulier la catégorie Cat des petites catégories et foncteurs (une liste de références est donnée dans la section References. Dans cette section, nous définirons la catégorie Sch et montrerons qu'elle est équivalente à Cat . C'est ce résultat qui justifie notre annonce dans l'introduction que "tout théorème sur les petites catégories devient un théorème sur les bases de données".

Définition 24. Rappelons les notions de schémas catégoriques et de traductions des définitions 17 et 21. La catégorie des schémas catégoriques, notée Sch , est la catégorie dont les objets sont des schémas catégoriques et dont les morphismes sont des traductions.

Construction 25 (Du schéma à la catégorie). Nous définirons un foncteur $L : Sch \rightarrow Cat$. Soit $\mathcal{C} = (G, \simeq_{\mathcal{C}})$ un schéma catégorique, où $G = (A, V, \text{src}, \text{tgt})$. Définissons $\mathcal{C}'$ comme la catégorie libre avec objets V générée par les flèches A . Définissons $L(\mathcal{C}) \in Cat$ comme la catégorie définie comme le quotient de $\mathcal{C}'$ par la relation d'équivalence $\simeq_{\mathcal{C}}$ (voir [Mac, Section 2.8]). Ceci définit L sur les objets de Sch .

Étant donné une traduction $F : \mathcal{C} \rightarrow \mathcal{D}$, il y a un foncteur induit sur les catégories libres $F' : \mathcal{C}' \rightarrow \mathcal{D}'$, envoyant chaque générateur $f \in A$ vers le morphisme de $\mathcal{D}'$ défini comme le composé du chemin $A_F(f)$. La préservation de l'équivalence de chemin assure que F' descend vers un foncteur $L(F) : L(\mathcal{C}) \rightarrow L(\mathcal{D})$ sur les catégories quotients. Ceci définit L sur les morphismes de Sch . Il est clair que L préserve la composition, donc c'est un foncteur.

Construction 26 (De la catégorie au schéma). Nous définirons un foncteur $R : Cat \rightarrow Sch$. Soit $\mathcal{C}$ une petite catégorie avec ensemble d'objets $\text{Ob}(\mathcal{C})$, ensemble de morphismes $\text{Mor}(\mathcal{C})$, fonctions source et cible $s, t : \text{Mor}(\mathcal{C}) \rightarrow \text{Ob}(\mathcal{C})$, et loi de composition $\circ : \text{Mor}(\mathcal{C}) \times_{\text{Ob}(\mathcal{C})} \text{Mor}(\mathcal{C}) \rightarrow \text{Mor}(\mathcal{C})$. Soit $R(\mathcal{C}) = (G, \simeq)$ où G est le graphe

$$G = (\text{Mor}(\mathcal{C}), \text{Ob}(\mathcal{C}), s, t)$$

et avec $\simeq$ défini comme suit : pour tous $f, g \in \text{Mor}(\mathcal{C})$ avec $t(f) = s(g)$ nous posons

$$fg \simeq (g \circ f). \quad (16)$$

Ceci définit R sur les objets de Cat .

Un foncteur $F : \mathcal{C} \rightarrow \mathcal{D}$ induit une traduction $R(F) : R(\mathcal{C}) \rightarrow R(\mathcal{D})$, parce que les sommets sont envoyés vers des sommets, les flèches sont envoyées vers des flèches, et l'équivalence de chemin est préservée par (16) et le fait que F préserve la loi de composition. Ceci définit R sur les morphismes de Cat . Il est clair que R préserve les compositions, donc c'est un foncteur.

Théorème 27. *Les foncteurs*

$$L : \text{Sch} \rightleftarrows \text{Cat} : R$$

sont des équivalences de catégories mutuellement inverses.

Esquisse de preuve. Il est clair qu'il y a un isomorphisme naturel $\epsilon : \text{id}_{\text{Cat}} \rightarrow L \circ R$; c'est-à-dire que pour toute catégorie $\mathcal{C}$, il y a un isomorphisme $\mathcal{C} \cong L(R(\mathcal{C}))$. Ainsi le foncteur L est essentiellement surjectif. Nous montrons d'abord que L est pleinement fidèle.

Choisissons des schémas X et Y , et supposons $X = (A_X, V_X, \text{src}_X, \text{tgt}_X)$; nous devons montrer que la fonction $L_1 : \text{Hom}_{\text{Sch}}(X, Y) \rightarrow \text{Hom}_{\text{Cat}}(LX, LY)$ est une bijection. Elle est clairement injective. Pour montrer qu'elle est surjective, choisissons un foncteur $G : LX \rightarrow LY$; nous définirons une traduction $F : X \rightarrow Y$ avec $L_1(F) = G$. Définissons F sur les sommets de X comme G est défini sur les objets de LX . Définissons F sur les flèches de X via la fonction $A_X \rightarrow \text{Path}_X \rightarrow \text{Mor}(LX) \xrightarrow{G} \text{Mor}(LY)$, et choisissons un représentant pour sa classe d'équivalence dans Path_{LY} (notez que deux choix quelconques résultent en la même traduction : voir définition 21). Deux chemins équivalents dans X se composent vers le même élément de $\text{Mor}(LX)$, donc F préserve l'équivalence de chemin. Ceci définit F , complétant la preuve que L est une équivalence de catégories.

Par un raisonnement similaire, on prouve que R est pleinement fidèle, et on conclut qu'il est inverse de L . $\square$

3.5. La catégorie des instances sur un schéma

Étant donné le théorème 27, la notion composée de schémas catégoriques et de traductions est équivalente à celle de catégories et foncteurs. Dans le reste de l'article, nous élidons la différence entre Sch et Cat , en utilisant la nomenclature de chacun de manière interchangeable.

On voit facilement (cf. définition 19) qu'une instance I sur un schéma $\mathcal{C}$ est la même chose qu'un foncteur $\mathcal{C} \rightarrow \text{Set}$, où Set est la catégorie des ensembles. Ainsi, nous avons un concept tout prêt de morphismes entre instances : les transformations naturelles de foncteurs. C'est une notion établie dans la littérature sur les bases de données, souvent appelée *homomorphisme* d'instances (voir par exemple [DNR]).

Définition 28. *Soit $\mathcal{C}$ un schéma et soient $I, J : \mathcal{C} \rightarrow \text{Set}$ des instances sur $\mathcal{C}$. Un morphisme m de I vers J , noté $m : I \rightarrow J$, est simplement une transformation naturelle entre ces foncteurs. Nous définissons la catégorie des instances sur $\mathcal{C}$, notée $\mathcal{C}\text{-Inst}$, comme la catégorie des instances et des*

morphismes, comme ci-dessus.

Plus généralement, soit $\mathbb{S}$ une catégorie quelconque ; nous définissons la catégorie des instances à valeurs dans $\mathbb{S}$ sur $\mathcal{C}$, notée $\mathcal{C}\text{-Inst}_{\mathbb{S}} := \mathbb{S}^{\mathcal{C}}$, comme la catégorie dont les objets sont les foncteurs $\mathcal{C} \rightarrow \mathbb{S}$ et dont les morphismes sont les transformations naturelles. Nous appelons $\mathbb{S}$ la catégorie de valeurs dans cette configuration.

Remarque 29. Il semble que les théoriciens des langages de programmation n'incluent pas les homomorphismes entre instances dans leur conception des instances de base de données en tant qu'éléments d'un type, préférant plutôt travailler avec juste l'ensemble $\text{Ob}(\mathcal{C}\text{-Inst})$ des instances sur un schéma $\mathcal{C}$. Cela facilite la définition de fonctions d'agrégation, telles que les sommes et les comptages ; voir par exemple [LT].

Dans le reste de l'article, nous travaillerons généralement avec $\mathcal{C}\text{-Inst}$, la catégorie des instances à valeurs dans Set . Cependant, la plupart des résultats tiennent plus généralement pour $\mathcal{C}\text{-Inst}_{\mathbb{S}}$, à condition que $\mathbb{S}$ soit complète et cocomplète (c'est-à-dire qu'elle ait toutes les petites limites et tous les petits colimites). Évidemment, étant donné un foncteur $\mathbb{S} \rightarrow \mathbb{S}'$, il y a un foncteur induit $\mathcal{C}\text{-Inst}_{\mathbb{S}} \rightarrow \mathcal{C}\text{-Inst}_{\mathbb{S}'}$, donc le choix de la catégorie de valeurs peut être changé sans trop de coût.

Exemple 30. Étant donné un schéma $\mathcal{C}$, il y a de nombreuses catégories $\mathbb{S}$, autres que $\mathbb{S} = \text{Set}$, pour lesquelles on pourrait être intéressé par $\mathcal{C}\text{-Inst}_{\mathbb{S}}$. Par exemple, étant donné un lambda-calcul, la catégorie associée $\mathbb{S} = \text{Type}$ des types et des termes est un bon choix ([Awo, Section 6.5]). On peut aussi utiliser la catégorie Fin des ensembles finis, cpo des ordres partiels complets, Cat des petites catégories, ou Top des espaces topologiques.

Le choix de la catégorie de valeurs est basé sur la façon dont on choisit de voir la collection de lignes dans chaque table. Nous considérons généralement cette collection comme un ensemble, mais par exemple on peut imaginer à la place un espace topologique de lignes, et dans ce cas chaque colonne consisterait en une application continue d'un espace à un autre.

Les topos, inventés par Grothendieck et Verdier [GV] et étendus par Lawvere [Law], sont des catégories qui imitent la catégorie des ensembles de plusieurs manières importantes (voir [MM]). Dans la proposition 31, nous montrons que les catégories d'instances sont souvent des topos.

Proposition 31. Si $\mathbb{S} = \text{Set}$ alors pour tout schéma $\mathcal{C} \in \text{Sch}$, la catégorie $\mathcal{C}\text{-Inst} = \mathcal{C}\text{-Inst}_{\text{Set}}$ est un topos. Si $\mathcal{C}$ est une catégorie finie alors pour tout topos $\mathbb{S}$ (par exemple $\mathbb{S} = \text{Fin}$, la catégorie des ensembles finis), la catégorie $\mathcal{C}\text{-Inst}_{\mathbb{S}}$ est un topos. Si $\mathbb{S}$ est complète (resp. cocomplète) alors $\mathcal{C}\text{-Inst}_{\mathbb{S}}$ est aussi complète (resp. cocomplète).

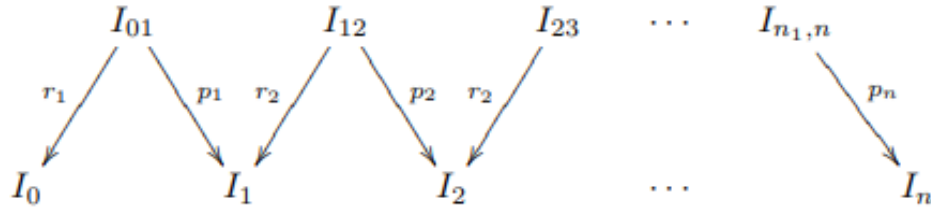
Démonstration. La première paire d'assertions se trouve dans [JoP, A.2.1.3]. La seconde paire d'assertions se trouve dans [Bor1, Théorème 2.15.2] et [Awo, Proposition 8.8], respectivement. $\square$

Étant donné une instance de base de données I , les mises à jour sur I incluent la suppression de lignes, l'insertion de lignes, la division (une ligne devenant deux), et la fusion (deux lignes devenant une). En fait, nous classons les insertions et les fusions ensemble comme *mises à jour progressives* et nous classons les suppressions et les divisions ensemble comme *mises à jour régressives*. Alors chaque mise à jour peut être considérée comme une mise à jour régressive suivie d'une mise à jour progressive.

Définition 32. Soit $\mathcal{C}$ un schéma et $I \in \mathcal{C}\text{-Inst}$ une instance. Une mise à jour progressive sur I consiste en une instance J et une transformation naturelle $p : I \rightarrow J$. Une mise à jour régressive sur I consiste en une instance J et une transformation naturelle $r : J \rightarrow I$. C'est-à-dire qu'une mise à jour régressive est juste une mise à jour progressive en sens inverse. Une mise à jour est une séquence finie de mises à jour progressives et régressives.

Proposition 33. Soit $\mathcal{C}$ un schéma et I une instance sur $\mathcal{C}$. Toute mise à jour sur I peut être obtenue comme une seule mise à jour régressive suivie d'une seule mise à jour progressive.

Démonstration. La composition de deux mises à jour progressives (resp. régressives) est une mise à jour progressive (resp. régressive). Par conséquent, toute mise à jour sur $I_0 := I$ peut être écrite comme un diagramme D dans $\mathcal{C}\text{-Inst}$:



Mais la limite de ce diagramme (qui peut être prise, si l'on veut, en prenant des produits fibrés tels que $I_{j-1,j} \rightarrow I_j \leftarrow I_{j,j+1}$, et en répétant $\frac{n^2-n}{2}$ fois), est ce dont nous avons besoin :

$$I_0 \leftarrow \lim D \rightarrow I_n.$$

□

Remarque 34. Une autre façon de comprendre les suppressions est via le filtrage - on filtre toutes les lignes d'une certaine forme. Le filtrage sera discuté dans la section Filtrage.

3.6. Construction de Grothendieck

Dans la section RDF, nous avons montré comment convertir une instance $I : \mathcal{C} \rightarrow \text{Set}$ en une nouvelle catégorie $Gr(I)$, appelée la *construction de Grothendieck* ou *catégorie des éléments* de I . Cette construction modélise la conversion des formes relationnelles vers les formes RDF des données. Il existe une construction inverse qui est décrite dans [Sp4, Proposition 2.3.9].

Nous notons ici qu'il existe une construction de Grothendieck plus générale qui peut être utile dans le contexte des bases de données fédérées. Dans la théorie des langages de programmation, on peut entendre parler d'une catégorie de *genres*, dont chaque objet est lui-même une catégorie de types. Ici, chaque genre est analogue à un schéma, et chaque type dans ce genre est analogue à une table dans ce schéma. Étant donné une catégorie de genres, nous pouvons "jeter tous leurs types ensemble" en appliquant la construction de Grothendieck généralisée. Ceci est analogue à prendre une base de données fédérée (c'est-à-dire un schéma de schémas liés) et les fusionner tous en un seul grand schéma. On peut appliquer cette construction au niveau des données également, en fusionnant toutes les instances en une instance sur le grand schéma unique.

La version de la construction de Grothendieck donnée dans la section RDF est pour les foncteurs $I : \mathcal{C} \rightarrow \mathbf{Set}$. Chaque objet $c \in \mathbf{Ob}(\mathcal{C})$ correspond à une table dont l'ensemble des lignes est $I(c)$. On peut trouver une description de cette construction dans [MM, Section 1.5]. La version de la construction de Grothendieck dans laquelle les schémas fédérés sont combinés en un seul grand schéma est pour les foncteurs $D : \mathcal{C} \rightarrow \mathbf{Cat}$. Chaque objet $c \in \mathbf{Ob}(\mathcal{C})$ correspond à une base de données dont la catégorie de tables est $D(c)$. On peut trouver une description de cette construction dans [JoP, B.1.3.1].

3.7. Dictionnaire

Notre espoir est que cet article serve de dictionnaire, grâce auquel les résultats de la littérature catégorique peuvent être importés directement dans la théorie des bases de données. Dans la section Migration des données, nous verrons un tel résultat : les traductions entre schémas fournissent des foncteurs de migration de données qui ont des propriétés utiles et prouvables. Dans le tableau 3.7., nous rassemblons certains des liens fondamentaux entre les bases de données et les catégories, tels que présentés tout au long de l'article.

Dictionnaire entre la terminologie des bases de données et la terminologie de la théorie des catégories.

Concept de base de données	Concept de théorie des catégories
Schéma de base de données, $\mathcal{C}$	Catégorie, $\mathcal{C}$
Table $T \in \mathcal{C}$	Objet $T \in \mathbf{Ob}(\mathcal{C})$
Colonne f de T	Morphisme sortant, $f : T \rightarrow ?$
Colonne de clé étrangère f de T pointant vers U	Morphisme $f : T \rightarrow U$
Séquence de clés étrangères	Composition de morphismes
Colonne de clé primaire ID de T	Flèche identité, $\mathrm{id}_T : T \rightarrow T$
Vocabulaire contrôlé (c'est-à-dire table à une colonne D)	Objet D sans morphismes sortants (sauf $\mathrm{id}_D : D \rightarrow D$)
Équivalence de chemin de clé étrangère dans $\mathcal{C}$	Diagramme commutatif dans $\mathcal{C}$
Instance I sur le schéma $\mathcal{C}$	Foncteur $I : \mathcal{C} \rightarrow \mathbf{Set}$ (ou $I : \mathcal{C} \rightarrow \mathbb{S}$ pour une autre catégorie "gentille" $\mathbb{S}$)
Conversion de Relationnel en RDF	Construction de Grothendieck
Mise à jour d'insertion $u : I_0 \rightarrow I_1$	Transformation naturelle $u : I_0 \rightarrow I_1$
Mise à jour de suppression $u : I_0 \rightarrow I_1$	Transformation naturelle $u : I_1 \rightarrow I_0$
Mappage de schéma $\mathcal{C} \rightarrow \mathcal{D}$	Foncteur $F : \mathcal{D} \rightarrow \mathcal{C}$
Processus ETL de base $\mathcal{C}\text{-Inst} \rightarrow \mathcal{D}\text{-Inst}$	Foncteur de pullback, souvent noté Δ_F ou $F^* : \mathcal{C}\text{-Set} \rightarrow \mathcal{D}\text{-Set}$

4. Foncteurs de migration de données

Étant donné des schémas $\mathcal{C}$ et $\mathcal{D}$, un foncteur de migration de données est chargé de transformer toute instance $\mathcal{C}$ -instance I en une instance $\mathcal{D}$ -instance J (ou vice-versa). De plus, il doit le faire de manière *naturelle*, ce qui signifie que les mises à jour progressives (resp. régressives) sur I doivent résulter en des mises à jour progressives (resp. régressives) sur J . Les foncteurs de migration de données ont été concrètement exemplifiés dans la section Migration des données simplifiée.

Dans cette section, nous commencerons par une traduction entre schémas $F : \mathcal{C} \rightarrow \mathcal{D}$. Rappelons de la définition 21 que ce n'est qu'une application des sommets de $\mathcal{C}$ vers les sommets de $\mathcal{D}$ et des flèches de $\mathcal{C}$ vers les chemins de $\mathcal{D}$, respectant l'équivalence de chemin. Toute traduction F génère trois foncteurs de migration de données. Ceux-ci seront notés

$$\Sigma_F : \mathcal{C}\text{-Inst} \rightarrow \mathcal{D}\text{-Inst} \quad \Delta_F : \mathcal{D}\text{-Inst} \rightarrow \mathcal{C}\text{-Inst} \quad \Pi_F : \mathcal{C}\text{-Inst} \rightarrow \mathcal{D}\text{-Inst}.$$

On peut remarquer que Δ_F semble aller à l'envers - la direction opposée à celle de F . Bien que cela puisse sembler contre-intuitif, en fait Δ_F est le plus simple des trois foncteurs de migration de données et le plus simple à décrire.

Avant de ce faire, discutons rapidement des catégories de valeurs. Rappelons de la définition 28 que pour toute catégorie $\mathbb{S}$, nous avons une catégorie $\mathcal{C}\text{-Inst}_{\mathbb{S}}$ des instances $\mathcal{C}$ à valeurs dans $\mathbb{S}$, c'est-à-dire la catégorie des foncteurs $\mathcal{C} \rightarrow \mathbb{S}$. Le foncteur de migration $\Delta_F : \mathcal{D}\text{-Inst}_{\mathbb{S}} \rightarrow \mathcal{C}\text{-Inst}_{\mathbb{S}}$ existe indépendamment du choix de $\mathbb{S}$. Pour que Σ_F existe, $\mathbb{S}$ doit être cocomplète, et pour que $\Pi_{\mathbb{S}}$ existe, $\mathbb{S}$ doit être complète. Pour fixer les idées, la plupart des lecteurs devraient simplement prendre $\mathbb{S} = \text{Set}$, à moins d'être contraints de faire autrement. C'est le cas où les lignes de chaque table forment un ensemble.

4.1. Le foncteur de migration de données pullback Δ

Supposons que nous ayons une traduction $F : \mathcal{C} \rightarrow \mathcal{D}$. Étant donné une instance $\mathcal{D}$ -instance, $I \in \mathcal{D}\text{-Inst}_{\mathbb{S}}$, nous devons la transformer en une instance $\mathcal{C}$ de manière naturelle. Mais c'est simple, parce que $I : \mathcal{D} \rightarrow \mathbb{S}$ est un foncteur et la composition de foncteurs est un foncteur, donc le composé

$$\mathcal{C} \xrightarrow{F} \mathcal{D} \xrightarrow{I} \mathbb{S},$$

est un objet de $\mathcal{C}\text{-Inst}_{\mathbb{S}}$, comme souhaité. De même, une transformation naturelle $m : I \rightarrow J$ est whiskerisée avec F pour donner une transformation naturelle $(m \circ F) : (I \circ F) \rightarrow (J \circ F)$. Ainsi, nous avons défini un foncteur

$$\Delta_F : \mathcal{D}\text{-Inst}_{\mathbb{S}} \rightarrow \mathcal{C}\text{-Inst}_{\mathbb{S}}, \quad \Delta_F(-) := (- \circ F)$$

Le slogan est “ Δ_F est donné par composition avec F ”.

Nous avons maintenant défini le foncteur de pullback $\Delta_F : \mathcal{D}\text{-Inst}_{\mathbb{S}} \rightarrow \mathcal{C}\text{-Inst}_{\mathbb{S}}$. Il a été explicitement discuté dans l'exemple 5. En gros, il peut accommoder : le renommage de tables, le renommage de colonnes, la suppression de tables, la projection de colonnes, la duplication de tables et la duplication de colonnes.

4.2. Le foncteur de migration de données pushforward droit Π

Supposons que nous ayons une traduction $F : \mathcal{C} \rightarrow \mathcal{D}$. Étant donné une instance $\mathcal{C}$ -instance, $I \in \mathcal{C}\text{-Inst}_{\mathbb{S}}$, nous devons la transformer en une instance $\mathcal{D}$ de manière naturelle. Nous le ferons en utilisant l'adjoint à droite de Δ_F ; cependant, pour ce faire, nous devons supposer que $\mathbb{S}$ est complète (c'est-à-dire que $\mathbb{S}$ a des petites limites). Notez que $\mathbb{S} = \text{Set}$ est complète.

Proposition 35. *Soit $F : \mathcal{C} \rightarrow \mathcal{D}$ un foncteur, et soit $\mathbb{S}$ une catégorie complète. Alors le foncteur $\Delta_F : \mathcal{D}\text{-Inst}_{\mathbb{S}} \rightarrow \mathcal{C}\text{-Inst}_{\mathbb{S}}$ a un adjoint à droite, que nous notons*

$$\Pi_F : \mathcal{C}\text{-Inst}_{\mathbb{S}} \rightarrow \mathcal{D}\text{-Inst}_{\mathbb{S}}.$$

Démonstration. C'est [Mac, Corollaire X.3.2]. □

Nous avons maintenant défini le foncteur pushforward droit $\Pi_F : \mathcal{C}\text{-Inst}_{\mathbb{S}} \rightarrow \mathcal{D}\text{-Inst}_{\mathbb{S}}$. Il a été explicitement discuté dans l'exemple 7. En gros, il peut accommoder : le renommage de tables, le renommage de colonnes et les jointures de tables. Pour voir cela, on applique la “formule point par point” pour les extensions de Kan droites, par exemple comme donné dans [Mac, Théorème X.3.1]; une formulation plus explicite est donnée dans [Sp3].

4.3. Le foncteur de migration de données pushforward gauche Σ

Supposons que nous ayons une traduction $F : \mathcal{C} \rightarrow \mathcal{D}$. Étant donné une instance $\mathcal{C}$ -instance, $I \in \mathcal{C}\text{-Inst}_{\mathbb{S}}$, nous devons la transformer en une instance $\mathcal{D}$ de manière naturelle. Nous le ferons en utilisant l'adjoint à gauche de Δ_F ; cependant, pour ce faire, nous devons supposer que $\mathbb{S}$ est cocomplète (c'est-à-dire que $\mathbb{S}$ a des petits colimites). Notez que $\mathbb{S} = \text{Set}$ est cocomplète.

Proposition 36. *Soit $F : \mathcal{C} \rightarrow \mathcal{D}$ un foncteur, et soit $\mathbb{S}$ une catégorie cocomplète. Alors le foncteur $\Delta_F : \mathcal{D}\text{-Inst}_{\mathbb{S}} \rightarrow \mathcal{C}\text{-Inst}_{\mathbb{S}}$ a un adjoint à gauche, que nous notons*

$$\Sigma_F : \mathcal{C}\text{-Inst}_{\mathbb{S}} \rightarrow \mathcal{D}\text{-Inst}_{\mathbb{S}}.$$

Démonstration. Cela découle de [Bor1, Théorème 3.7.2] et [BW2, Proposition 9.11]. □

Nous avons maintenant défini le foncteur pushforward gauche $\Sigma_F : \mathcal{C}\text{-Inst}_{\mathbb{S}} \rightarrow \mathcal{D}\text{-Inst}_{\mathbb{S}}$. Il a été explicitement discuté dans l'exemple 9. En gros, il peut accommoder : le renommage de tables, le renommage de colonnes, l'union de tables et la création de variables de Skolem. Pour voir cela, on applique la “formule point par point” pour les extensions de Kan gauches, par exemple comme donné dans [Bor1, Théorème 3.7.2]; une formulation plus explicite est donnée dans [Sp3].

5. Types de données et filtrage

Dans cette section, nous formulerons la relation de typage qui existe entre les données abstraites et leur représentation. Jusqu'à présent, nous avons considéré les données comme simplement une collection d'éléments interconnectés - une instance au sens de la définition 19 garde la trace de divers ensembles d'éléments abstraits, ségrégés en tables et connectés ensemble de manière précise. Cependant, en réalité, chacun de ces éléments est représenté dans sa table par un type de données,

tel que des chaînes de caractères ou des entiers. Sémantiquement, chaque cellule d’une colonne donnée $c : t \rightarrow t'$ d’une table t devrait avoir le même type de données, à savoir qu’elles devraient toutes avoir le type de données de la table cible, t' .

Cependant, les types de données ne fournissent pas seulement une méthode uniforme pour afficher chaque élément de données, mais ils peuvent aussi porter une notion de valeur. Par exemple, les salaires sont des nombres qui peuvent être additionnés pour donner des invariants significatifs. C’est pour cette raison que nous devons trouver une connexion entre le formalisme des bases de données et le formalisme des langages de programmation, comme discuté dans la section Données et programme. La théorie des catégories fournit une telle connexion : les schémas de bases de données et les langages de programmation forment des catégories, et ces catégories peuvent être reliées par des foncteurs.

Ci-dessous, nous expliquerons ces concepts, en particulier comment attacher des types de données d’un langage de programmation aux tables d’un schéma catégorique. Pour ce faire, nous utiliserons les concepts de la section Migration des données. Après avoir défini les signatures de type sur les schémas, nous procéderons à la définition des morphismes de signatures de type, ce qui nous permettra de filtrer les données. Par exemple, pour filtrer tous les noms qui commencent par la lettre R, nous pourrions tirer en arrière le long d’une inclusion $\{R\} \rightarrow \text{Str}$, où Str est l’ensemble des chaînes de caractères.

5.1. Assigner des types de données via une transformation naturelle

Nous commençons par deux exemples pour motiver la définition.

Exemple 37. *Que signifie mathématiquement l’expression “un ensemble d’entiers” ? Considérez qu’il s’agit d’un ensemble étiqueté X , avec une fonction $f : X \rightarrow \mathbb{Z}$. En permettant à notre ensemble d’entiers de changer, nous obtenons différents ensembles étiquetés X et différentes fonctions étiquetées f , mais l’ensemble $\mathbb{Z}$ des entiers est inchangé. Du point de vue catégorique, nous pouvons comprendre “un ensemble X d’entiers” de plusieurs façons différentes :*

1. *comme une instance de base de données $I : \mathcal{C} \rightarrow \text{Set}$ sur le schéma*

$$\mathcal{C} := X \xrightarrow{f} \mathbb{Z}$$

tel que l’image sur $\mathbb{Z}$ soit fixée comme $I(\mathbb{Z}) = \mathbb{Z}$; ou

2. *comme une instance de base de données $J : \mathcal{D} \rightarrow \text{Set}$ sur le schéma*

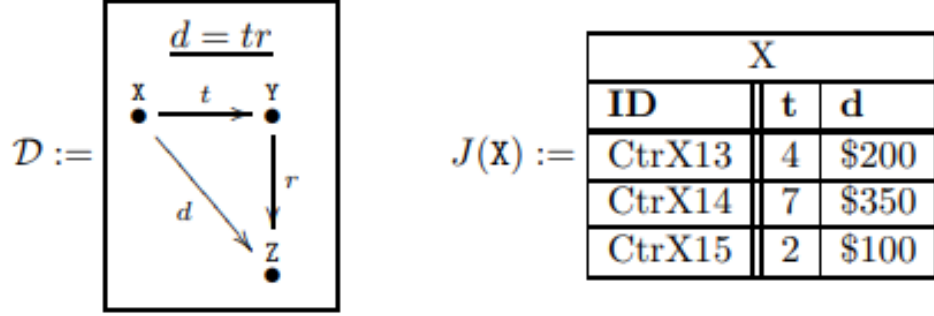
$$\mathcal{D} := X$$

équipé d’une transformation naturelle $f : J \rightarrow \{\mathbb{Z}\}$ (où $\{\mathbb{Z}\}$ est une abréviation pour le foncteur $\mathcal{D} \rightarrow \text{Set}$ donné par $D(X) = \mathbb{Z}$).

Exemple 38. *Supposons que nous ayons une base de données et que nous souhaitions imposer une relation mathématique entre deux colonnes, disons t et d , dans une certaine table X . Par exemple, il se pourrait que la colonne t , disons “temps passé” (en un nombre entier d’heures) soit liée à la colonne d , disons “dette due” (en dollars) par une fonction mathématique $d = r(t)$, disons*

$$d(x) = r(t(x)) := \$50 * t(x)$$

pour chaque $x \in X$. Tout comme dans l'exemple 37, cette situation pourrait être représentée catégoriquement de plusieurs façons, mais nous nous concentrerons sur une seule. À savoir, nous la comprenons comme la collection d'instances de base de données $J : \mathcal{D} \rightarrow \text{Set}$ sur le schéma $\mathcal{D}$ comme exemplifié ci-dessous



(17)

Le fait que d ait le type de données dollar et que $d = \$50 * t$ sont imposés par une certaine transformation naturelle, $f : J \rightarrow P$, où P est une *instance de typage*. Nous donnerons plus de détails dans l'exemple 43.

Définition 39. Soit $\mathcal{C}$ un schéma et soit $P \in \text{Ob}(\mathcal{C}\text{-Inst})$ une instance. La catégorie des instances typées P sur $\mathcal{C}$, notée $\mathcal{C}\text{-Inst}_P$, est définie comme la catégorie “tranche” des instances au-dessus de P (voir, par exemple, [MM, Préliminaires catégoriques]). En d’autres termes, une instance typée P sur $\mathcal{C}$ est une paire (I, τ) où I est une instance et $\tau : I \rightarrow P$ est une transformation naturelle ; et un morphisme d’instances typées P est un triangle commutatif.

Remarque 40. Étant donné un schéma $\mathcal{C}$, nous pouvons appeler toute instance $P \in \text{Ob}(\mathcal{C}\text{-Inst})$ une instance de typage si notre plan est de considérer des instances typées P , c’est-à-dire la catégorie $\mathcal{C}\text{-Inst}_P$.

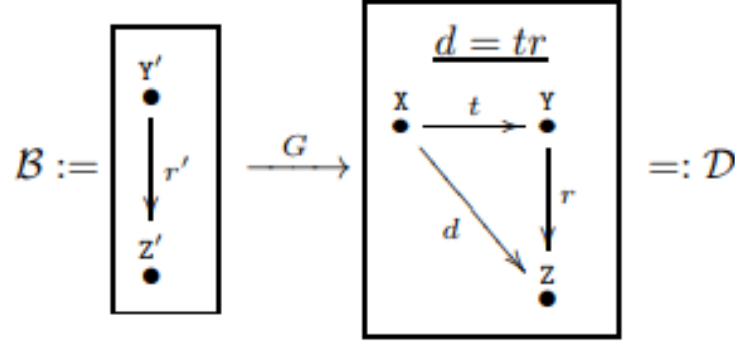
Remarque 41. Fixons un schéma $\mathcal{C}$ et une catégorie $\mathbb{S}$ et soit $\mathcal{E} := \mathcal{C}\text{-Inst}_{\mathbb{S}}$ la catégorie des instances à valeurs dans $\mathbb{S}$ sur $\mathcal{C}$. En pratique, $\mathcal{E}$ est souvent un topos (voir proposition 31). Dans ce cas, pour toute instance $P \in \text{Ob}(\mathcal{E})$, la catégorie $\mathcal{E}_P$ des instances typées P à valeurs dans $\mathbb{S}$ sur $\mathcal{C}$ est encore un topos (voir [MM, Théorème IV.7.1]).

Construction 42. Supposons donné une catégorie Type de types pour un certain langage de programmation et un foncteur $V : \text{Type} \rightarrow \mathbb{S}$ à valeurs dans $\mathbb{S}$ qui envoie chaque type vers son ensemble (ou objet de $\mathbb{S}$) de valeurs. Nous souhaitons souvent utiliser un fragment de Type pour ajouter des informations de typage à notre schéma de base de données $\mathcal{C}$. Si le fragment est donné par le foncteur $\mathcal{B} \xrightarrow{F} \text{Type}$ et que $\mathcal{B}$ est associé au schéma via un foncteur $\mathcal{B} \xrightarrow{G} \mathcal{C}$,

$$\mathbb{S} \xleftarrow{V} \text{Type} \xleftarrow{F} \mathcal{B} \xrightarrow{G} \mathcal{C},$$

alors $P := \Pi_G \circ \Delta_F(V)$ est l’instance de typage implicite. Nous appelons la séquence $(\mathcal{B}, F, G)$ l’auxiliaire de typage dans cette configuration.

Exemple 43. Nous revenons à l'exemple 38 avec le langage de la construction 42. Nous allons maintenant décrire un auxiliaire de typage. Soit $\mathcal{B}$ la catégorie à une flèche dessinée ci-dessous, et soit $G : \mathcal{B} \rightarrow \mathcal{D}$ le foncteur suggéré



Considérons aussi le foncteur $F : \text{Type} \rightarrow \text{Type}$ envoyant Y' vers Int , le type des entiers, Z' vers Dollar le type des valeurs en dollars, et r' vers la fonction qui multiplie un entier par 50.

Avec $V : \text{Type} \rightarrow \mathbb{S}$ comme dans la construction 42, l'instance de typage implicite $P := \Pi_G \circ \Delta_F(V) : \mathcal{D} \rightarrow \mathbb{S}$ a

$$P(X) = \text{Int} \times \text{Dol}; \quad P(Y) = \text{Int}; \quad P(Z) = \text{Dollar},$$

et $P(r) : P(Y) \rightarrow P(Z)$ est bien l'application de multiplication par 50.

Maintenant, une instance typée $P \tau : I \rightarrow P$ est exactement ce que nous voulons. Pour chacun de X, Y, Z , c'est un ensemble avec une application vers le type de données donné, et la naturalité de τ assure les propriétés décrites dans l'exemple 38 (c'est-à-dire que $d = \$50 * t$ dans la table $J(X)$ de l'affichage (38)). En d'autres termes, elle assure que pour toute ligne de $J(X)$, la valeur de la cellule dans la colonne d sera 50 fois la valeur de la cellule dans la colonne t .

5.2. Morphismes de signatures de type

Chaque foncteur de migration de données discuté dans la section Migration des données est une sorte d'outil pour l'évolution du schéma. Au fur et à mesure que de nouvelles tables et colonnes sont créées et que d'autres sont supprimées, la traduction entre l'ancien et le nouveau schéma induira des foncteurs de migration de données qui convertissent de manière transparente les anciennes données en nouvelles (et vice-versa) et les requêtes sur l'ancien schéma en requêtes sur le nouveau (voir aussi [Sp4]).

Il existe un type légèrement différent d'évolution de schéma qui revient souvent, à savoir le changement de types de données. Par exemple, si une entreprise dépasse environ 32 000 employés, elle devra peut-être changer le type de données de sa table *Employe* d'un *smallint* à un *bigint*. Des changements plus complexes incluent la réduction de moitié du prix de chaque action, ou la concaténation d'une paire prénom-nom pour former un nouveau champ. Il est important de pouvoir raisonner sur la façon dont les requêtes contre le schéma d'aujourd'hui différeront de celles contre

celui d'hier. Les foncteurs de changement de type sont comme les foncteurs de migration de données, et la nature formelle de leur description permet de raisonner sur leur comportement.

Soit $\mathcal{C}$ un schéma et soit $\mathcal{E}$ le topos $\mathcal{C}\text{-Inst}$. Étant donné un morphisme d'instances de typage $k : P \rightarrow Q$, il y a des adjonctions induites

$$\begin{array}{ccc} \mathcal{E}_{/P} & \xrightarrow{\hat{\Sigma}_k} & \mathcal{E}_{/Q} \\ \mathcal{E}_{/Q} & \xrightarrow{\hat{\Delta}_k} & \mathcal{E}_{/P} \end{array} \quad \begin{array}{ccc} \mathcal{E}_{/Q} & \xrightarrow{\hat{\Delta}_k} & \mathcal{E}_{/P} \\ \mathcal{E}_{/P} & \xrightarrow{\hat{\Pi}_k} & \mathcal{E}_{/Q} \end{array}$$

En d'autres termes, k induit un morphisme géométrique essentiel de topos ([MM, Théorème IV.7.2]).

Définition 44. Soit $\mathcal{C}$ un schéma et $k : P \rightarrow Q$ un morphisme d'instances de typage. Nous appelons les foncteurs induits

$$\hat{\Sigma}_k, \hat{\Pi}_k : \mathcal{C}\text{-Inst}_{/P} \rightarrow \mathcal{C}\text{-Inst}_{/Q} \quad \hat{\Delta}_k : \mathcal{C}\text{-Inst}_{/Q} \rightarrow \mathcal{C}\text{-Inst}_{/P}$$

des foncteurs de changement de type. Pour être plus spécifique, $\hat{\Sigma}_k$ sera appelé le foncteur de changement de type pushforward gauche, $\hat{\Pi}_k$ sera appelé le foncteur de changement de type pushforward droit, et $\hat{\Delta}_k$ sera appelé le foncteur de changement de type pullback.

Exemple 45. Nous revenons à l'exemple 38 avec $\mathcal{D}$ et P comme définis là-bas. Considérons le foncteur de changement de type pushforward gauche $\hat{\Sigma}_k$ dans le cas où $k : P \rightarrow Q$ envoie une valeur en dollars x vers *True* si $x \geq \$200$ et *False* si $x < \$200$. Le foncteur $\hat{\Sigma}_k$ convertit la table de gauche en la table de droite ci-dessous :

$\tau :=$	X			$\hat{\Sigma}_k(\tau) =$	X		
	ID	t	d		ID	t	d
	<i>CtrX13</i>	4	\$200		<i>CtrX13</i>	4	<i>True</i>
	<i>CtrX14</i>	7	\$350		<i>CtrX14</i>	7	<i>True</i>
	<i>CtrX15</i>	2	\$100		<i>CtrX15</i>	2	<i>False</i>

Les autres foncteurs de changement de type, $\hat{\Pi}_k$ et $\hat{\Delta}_k$ n'ont pas de résultats utiles dans le contexte de cet exemple particulier, mais voir les exemples 46 et 47.

Le foncteur de changement de type droit $\hat{\Pi}$ gère ce qu'on pourrait appeler la "satisfaction de groupe". Supposons que nous ayons un groupe (P) de personnes et un ensemble (I) d'objets distribués parmi elles ($\tau : I \rightarrow P$). Si les personnes sont ensuite subdivisées en groupes ($k : P \rightarrow Q$), alors nous pouvons demander à chaque groupe $q \in Q$: "de combien de façons chacun de vos membres peut-il offrir l'un de ses objets?" (cardinalité de $\hat{\Pi}_k(\tau)^{-1}(q) \subseteq \hat{\Pi}_k(I)$). Par exemple, si l'une des personnes s'est vu remettre un ensemble vide d'objets, alors son groupe n'aura pas une telle offre conjointe ($\hat{\Pi}_k(\tau)^{-1}(q) = \emptyset$). Nous expliquons maintenant cela par exemple.

Exemple 46. Soit $\mathbb{S}$ une catégorie complète et écrivons $\mathcal{C}\text{-Inst}$ au lieu de $\mathcal{C}\text{-Inst}_{\mathbb{S}}$. Dans cet exemple, nous expliquons comment, étant donné un morphisme d'instances de typage $k : P \rightarrow Q$ sur un schéma $\mathcal{C}$, le foncteur de changement de type $\hat{\Pi}_k : \mathcal{C}\text{-Inst}_{/P} \rightarrow \mathcal{C}\text{-Inst}_{/Q}$ opère sur une instance

typée P pour retourner une instance typée Q par ce que nous avons appelé la satisfaction de groupe ci-dessus. Supposons que nous ayons $\mathcal{C}$ et $\mathcal{B}$ comme dessinés,

$$\mathcal{B} = L' \quad \mathcal{C} = L \xrightarrow{f} M$$

avec le foncteur $G : \mathcal{B} \rightarrow \mathcal{C}$ donné par $L' \mapsto L$ et le foncteur $P' : \mathcal{B} \rightarrow \mathbb{S}$ donné par $P'(L') = \{1, 2, 3, 4\}$. Soit $Q' : \mathcal{B} \rightarrow \mathbb{S}$ donné par $Q'(L') = \{x, y\}$ et soit $k' : P' \rightarrow Q'$ l'application envoyant $1, 2 \mapsto x$; $3, 4 \mapsto y$. Enfin, soit

$$P := \Pi_G(P'), \quad Q = \Pi_G(Q'), \quad \text{et} \quad k := \Pi_G(k') : P \rightarrow Q.$$

Nous sommes prêts à calculer le foncteur de changement de type droit le long de k sur toute instance typée P $I \rightarrow P$; nous avons juste besoin d'écrire une telle instance. Donc, si $I \in \mathcal{C}\text{-Inst}$ est la table de gauche, alors $\hat{\Pi}_k(I)$ est la table du milieu :

$$I := \begin{array}{c|c} L & \\ \hline \textbf{ID} & \textbf{f} \\ \hline a & 1 \\ b & 2 \\ c & 1 \\ d & 3 \\ e & 2 \\ f & 4 \\ g & 2 \end{array} \quad \hat{\Pi}_k(I) = \begin{array}{c|c} L & \\ \hline \textbf{ID} & \textbf{f} \\ \hline (a,b) & x \\ (a,e) & x \\ (a,g) & x \\ (c,b) & x \\ (c,e) & x \\ (c,g) & x \\ (d,f) & y \end{array}$$

et un croquis du raisonnement est

$$\begin{array}{cccc} & & \text{b} & \\ & \text{a} & \text{e} & \\ \text{c} & \text{g} & \text{d} & \text{f} \\ \hline 1 & 2 & 3 & 4 \\ \hline \underbrace{1 \ 2} & & \underbrace{3 \ 4} & \\ x & & y & \end{array}$$

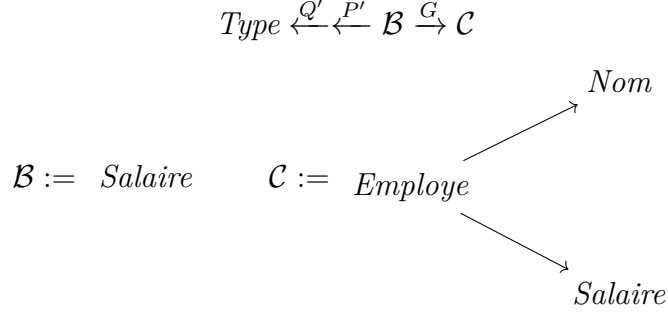
5.3. Filtrage des données

Dans cette section, nous montrons comment utiliser le foncteur de changement de type pullback $\hat{\Delta}$ pour filtrer les données (par exemple, répondre à des requêtes comme “retourner l'ensemble des employés dont le salaire est inférieur à 100 ”).

En fait, étant donné un morphisme d'instances $k : P \rightarrow Q$, le foncteur de changement de type pullback associé $\hat{\Delta}_k$ peut soit filtrer, soit multiplier les données (ou les deux) selon l'injectivité ou

la surjectivité de k . Dans cette courte section, nous nous concentrons uniquement sur le filtrage, car il semble plus utile en pratique. Nous travaillons entièrement par exemple ; la définition de $\hat{\Delta}$ est donnée dans la section Morphismes de types ou dans la littérature ([MM, Section IV.7]).

Exemple 47. *Comme annoncé ci-dessus, nous montrons comment filtrer les employés par leur salaire, en particulier en montrant seulement ceux avec des salaires inférieurs à 100 . Supposons que nous ayons un schéma $\mathcal{C}$ et deux auxiliaires de typage, $(\mathcal{B}, P, G)$ et $(\mathcal{B}, P', G)$, montrés ci-dessous :*



Ici, nous voulons que $Q'(\text{Salaire})$ soit le type de données en dollars, nous voulons que $P'(\text{Salaire})$ soit le sous-type donné en exigeant qu'une valeur en dollars x soit inférieure à 100, et nous voulons que $k' : P' \rightarrow Q'$ soit l'inclusion. Ces auxiliaires de typage induisent un morphisme d'instances de typage

$$k := \Pi_G(k') : P \rightarrow Q, \quad \text{où } P := \Pi_G(P') \quad \text{et} \quad Q := \Pi_G(Q').$$

Supposons maintenant que $I \in \mathcal{C}\text{-Inst}$ soit l'instance typée Q sur $\mathcal{C}$ montrée à gauche ci-dessous. Alors $\hat{\Pi}_k(I)$ est l'instance typée P à droite ci-dessous.

<i>Employe</i>		
ID	Name	Salaire
<i>Em101</i>	<i>Smith</i>	<i>\$65</i>
<i>Em102</i>	<i>Juarez</i>	<i>\$120</i>
<i>Em103</i>	<i>Jones</i>	<i>\$105</i>
<i>Em104</i>	<i>Lee</i>	<i>\$90</i>
<i>Em105</i>	<i>Carlsson</i>	<i>\$80</i>

$\hat{\Pi}_k(I) =$

<i>Employe</i>		
ID	Name	Salaire
<i>Em101</i>	<i>Smith</i>	<i>\$65</i>
<i>Em104</i>	<i>Lee</i>	<i>\$90</i>
<i>Em105</i>	<i>Carlsson</i>	<i>\$80</i>

Ainsi, nous voyons que le filtrage est simplement une application de la même histoire de foncteur de migration de données.

5.4. Une forme normale pour la migration de données

Nous avons vu plusieurs formes différentes de foncteurs de migration de données tout au long de cet article. On peut effectuer une séquence de foncteurs de migration de données, par exemple déplacer des données d'un schéma à un autre, puis changer les types de données, et enfin filtrer le résultat. Dans de nombreux cas, une telle combinaison de foncteurs de migration de données peut être réécrite comme une séquence de trois : un pullback, un pushforward droit et un pushforward gauche ; voir [SW]. Cela signifie qu'il existe une forme normale pour une classe assez générale de requêtes. Toute combinaison de telles requêtes peut être écrite sous la forme $\Delta_F \Pi_G \Sigma_H$ pour certains F, G, H . Cette forme peut ne pas être optimale en termes de vitesse, mais elle peut servir de format d'entrée unique pour les optimiseurs de requêtes.

Références

- [Awo] Awodey, S. *Category Theory* 2nd edition. Oxford Logic Guides, 52. Oxford University Press, Oxford 2010.
- [AW] Awodey, S., Warren M. A. (2009) "Homotopy theoretic models of identity types." *Math. Proc. Cambridge Philos. Soc.* 146 (1), pp. 45-55.
- [AF] Awodey, S. and Forssell, H. *First-order logical duality*. Available online : <http://front.math.ucdavis.edu/1008.3145>.
- [ASX] Agrawal, R. ; Somani, A. ; Xu, Y. "Storage and querying of e-commerce data". Proceedings of the 27th VLDB Conference, Roma, Italy, 2001.
- [BS] Baez, J. ; Stay, M. "Physics, topology, logic and computation : a Rosetta Stone." *New structures for physics*, 95–172. Lecture Notes in Phys., 813, Springer, Heidelberg, 2011.
- [BW1] Barr, M. ; Wells, C. *Category theory for computing science*. Prentice Hall International Series in Computer Science. New York 1990.
- [BW2] Barr, M. ; Wells, C. *Toposes, Triples, and Theories*. Reprints in Theory and Application of Categories, No. 1. 2005.
- [Bor1] Borceux, F. *Handbook of categorical algebra 1*. Encyclopedia of Mathematics and its Applications 50, 51, 52. Cambridge University Press, Cambridge, 1994.
- [BCW] Baclawski, K., Simovici, D., White, W. "A categorical approach to database semantics" *Mathematical structures in Computer Science* (1994) vol 4., pp. 147–183.
- [Ber] Bernstein, P.A. *Generic model management : A database infrastructure for schema manipulation*, pp. 1–6, Springer Berlin/Heidelberg, 2001.
- [BVP] Bohannon, A. ; Vaughan, J.A. ; Pierce, B.C. *Relational Lenses : A Language for Updateable Views*. In Principles of Database Systems (PODS), 2006.
- [CI] Cook, W.R. ; Ibrahim, A.H. "Integrating Programming Languages & Databases : What's the Problem ?" ODBMS Expert Article. 2005.
- [BCFGP] Barbosa, D.M.J. ; Cretin, J. ; Foster, N. ; Greenberg, M. ; Pierce, B. "Matching Lenses : Alignment and View update". Department of Computer and Information science, Technical reports. 2010.
- [Dis] Diskin, Z. *Databases as diagram algebras : Specifying queries and views via the graph-based logic of sketches*, Tech. report, Frame Inform Systems, 1996.
- [DI] Döring, A. ; Isham, C. J. "A topos foundation for theories of physics. I. Formal languages for physics." *J. Math. Phys.* 49 (2008), no. 5, 053515.
- [DK] Diskin, Z. ; Kadish, B. *Algebraic graph-oriented=category-theory-based manifesto of categorizing data base theory*, Tech. report, Frame Inform Systems, 1994.
- [DNR] Deutsch, A. ; Nash, A. ; Remmel, J. "The chase revisited." PODS 2008, pp. 149–158.
- [DC] Diskin, Z. ; Cadish, B. "Databases as Graphical Algebras : Algebraic Graph-Based Approach to Data Modeling and Database Design". Available online : <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.3655>.
- [EN] Emasri, R. ; Navathe, S. *Database systems : models, languages, design, and application programming*, Edition 6. Pearson, 2011.
- [EV] Ehresmann, A.C. ; Vanbremeersch, J.P., *Memory Evolutive Systems ; hierarchy, emergence, cognition* Elsevier Science, 2007.
- [F] Fowler, M. *Refactoring : improving the design of existing code*. Addison-Wesley, 1999.
- [G] Goguen, J. *Sheaf Semantics for Concurrent Interacting Objects*, Mathematical Structures in Computer Science, 1992, pp. 159–191.
- [GB] Goguen, J.A. ; Burstall, R.M. *Institutions : abstract model theory for specification and programming*, J. Assoc. Comput. Mach. **39** (1992), no. 1, 95–146.
- [GK] N. Gambino and J. Kock, *Polynomial functors and polynomial monads*. ePrint available : <http://arxiv.org/pdf/0906.4931v2>

- [GSB] Giesa, T. ; Spivak, D.I. ; Buehler, M.J. "Reoccurring Patterns in Hierarchical Protein Materials and Music : The Power of Analogies" (2011) *BioNanoScience*. Volume 1, Number 4, 153-161.
- [GV] Grothendieck ; A. Verdier, J-L. *Théorie des topos et cohomologie étale des schémas* (known as *SGA4*). New York/Berlin : Springer. (Lecture notes in mathematics, 269 – 270)
- [Ham] Hamana, M. "Polymorphic abstract syntax via Grothendieck construction." Lecture notes in computer science, 2011, Vol 6604, 381-395.
- [IP] Islam, A., Phoa, W. "Categorical models of relational databases I : fibrational formulation, schema integration". *Lecture Notes in Comput. Sci.*, 789, Springer, Berlin, 1994.
- [JD] Johnson, M. ; Dampney, C.N.G. "On the value of commutative diagrams in information modeling." AMAST '93 Proceedings of the Third International Conference on Methodology and Software Technology : Algebraic Methodology and Software Technology. Springer-Verlag, 1993.
- [JoM] Johnson, M. "On Category Theory as a (meta) Ontology for Information Systems Research" *Proceedings of the international conference on Formal Ontology in Information Systems*, 2001.
- [JoP] Johnstone, P. *Sketches of an elephant, Volume 1,2*. Oxford logic guides 43, 44. The Clarendon Press, Oxford University Press, Oxford, 2002.
- [Joy] Joyal, A. *Catlab*, available online : <http://ncatlab.org/joyalcatlab/show/Factorisation+systems>
- [JRW] Johnson, M. ; Rosebrugh, R. ; Wood, R.J. *Entity-relationship-attribute designs and sketches*, Theory Appl. Categ. **10** (2002), 94–112 (electronic).
- [Kan] Kanda, Akira. "Data types as initial algebras : A unification of Scottery and ADJery," Foundations of Computer Science, IEEE Annual Symposium on, pp. 221-230, 19th Annual Symposium on Foundations of Computer Science (FOCS 1978), 1978.
- [KC] Klyne G. ; Carroll J.J. (Editors). "Resource Description Framework (RDF) : Concepts and Abstract Syntax" *W3C Recommendation* 2004/02/10. <http://www.w3.org/TR/rdf-concepts/>.
- [KT] Korotkiy, M. ; Top, J. *From relational data to RDFS models*. Lecture Notes in Computer Science 2004, Vol 3140.
- [LS] Lawvere, F.W. ; Schanuel, S.H. *Conceptual Mathematics. A First Introduction to Categories*. Second edition. Cambridge University Press, Cambridge (2009).
- [LT] Lellahi, K. ; Tannen, V. "A calculus for collections and aggregates." Proceeding CTCS '97 Proceedings of the 7th International Conference on Category Theory and Computer Science. Springer-Verlag, 1997.
- [Mac] Mac Lane, S. *Categories for the working mathematician* 2nd edition. Graduate texts in mathematics 5, Springer Verlag, New York, 1998.
- [MM] Mac Lane, S. ; Moerdijk, I. *Sheaves in Geometry and Logic : a first introduction to topos theory*, Universitext. Springer-Verlag, New York, 1994.
- [Mak] Makkai, M. *Generalized sketches as a framework for completeness theorems I.* J. Pure Appl. Algebra 115 (1997), no. 1, 49–79.
- [Mar] Martin-Löf, P. *Intuitionistic type theory*. 1984.
- [Mog] Moggi, E. "A category-theoretic account of program modules." *Category theory and computer science (Manchester, 1989)*, 101–117, Lecture Notes in Comput. Sci., 389, Springer, Berlin, 1989.
- [Mor] Morava, J. "Theories of anything", ePrint available : <http://arxiv.org/abs/1202.0684v1>
- [Pie] B. Pierce, Basic category theory for computer scientists. Foundations of Computing Series. MIT Press, Cambridge, MA, 1991, pp. 391–407.
- [PS] Piessens, F. ; Steegmans, E. *Categorical data-specifications*, Theory Appl. Categ. **1** (1995), No. 8, 156–173 (electronic).
- [RW] Rosebrugh, R. ; Wood, R.J. *Relational databases and indexed categories*, Category theory 1991 (Montreal, PQ, 1991), CMS Conf. Proc., vol. 13, Amer. Math. Soc., Providence, RI, 1992.
- [Ros] Rosen, R. "The representation of biological systems for the stand-point of the theory of categories." (1958) *Bull. Math. Biophys.* 20, pp. 317 – 342.
- [Sic] Sica, G. (ed.) "What is category theory ?". Polimetrica S.a.s. Milan, Italy (2006).

- [Sp1] Spivak, D.I. *Simplicial databases*. 2009. ePrint available : <http://arxiv.org/abs/0904.2012>
- [Sp2] Spivak, D.I. *Table manipulation in simplicial databases*. 2010. Available online : <http://arxiv.org/abs/1003.2682>
- [Sp3] Spivak, D.I. "Functorial data migration". (2010) ePrint available : <http://arxiv.org/abs/1009.1166>
- [Sp4] Spivak, D.I. "Queries and constraints via lifting problems". (2012) Available online : <http://arxiv.org/abs/1202.2591>
- [SK] Spivak, D.I., Kent, R. "Ologs : a category-theoretic foundation for knowledge representation" (2012) *PLoS ONE* 7(1) : e24274. doi :10.1371/journal.pone.0024274.
- [SGWB] Spivak, D.I.; Giesa, T.; Wood, E.; Buehler, M.J. (2011) "Category Theoretic Analysis of Hierarchical Protein Materials and Social Networks." *PLoS ONE* 6(9) : e23911. doi :10.1371/journal.pone.0023911.
- [SH] Stone, C.; Harper, R. "Decidable type equivalence in a language with singleton kinds". Principles of programming languages conference, 1999.
- [SW] Spivak, D.I., Wisnesky, R. "On the relational foundations of functorial data migration". Submitted to *PODS 2013*. ePrint available : <http://arxiv.org/abs/1212.5303>
- [Tui] Tuijn, C. "Data modeling from a categorical perspective" PhD. thesis. University of Antwerpen, 1994.