1. Méthodologie: pour en finir avec …

Méthodologie: étude des méthodes (Petit Robert)

Ce chapitre résume en quelques paragraphes les principales options disponibles au modéliste des données pour faire son travail et les raisons qui nous poussent à sélectionner une approche plutôt qu'une autre. La méthode préconisée sera présentée en détail dans un autre chapitre. Si la discussion qui suit devait rebuter le lecteur ou lui paraître trop technique, qu'il passe allégrement au chapitre suivant: il n'aura rien perdu qui lui permette d'accomplir correctement son travail de modéliste de données. La présente discussion n'est destinée qu'à la personne chargée dans l'entreprise de décider du choix de la méthode et des outils utilisés pour réaliser la modélisation des données et représenter son résultat.

En fin de compte, toutes les méthodes décrites dans les cours et livres mènent à un but identique: obtenir un schéma de base de données fait de définitions de tables et d'attributs, avec les associations entre tables décrites par le mécanisme des clés primaires et étrangères. Pour y arriver, la route passe par la découverte des types d'objets (entités) et des associations.

Le choix d'une méthode prête forcément à controverse. Malheureusement, comme dans n'importe quel domaine et celui-ci n'y fait pas exception, le succès d'un gourou passe toujours par sa capacité à se différencier des autres et à imposer son point de vue: il faut faire comme ceci, ceux qui pensent différemment ont tort. Le présent chapitre soulèvera donc inévitablement des polémiques.

Notre seule doctrine: pas de doctrine!

Seuls comptent le résultat et l'efficacité du processus utilisé pour parvenir à ce résultat.

Trois niveaux de modélisation?

Les livres sur les bases de données parlent de trois niveaux de modélisation: **conceptuel, logique et physique**.

La **modélisation conceptuelle** s'effectue en prenant uniquement en compte la réalité du domaine de gestion couvert par le projet, sans tenir compte du type de SGBD (hiérarchique, relationnel, objet) qui sera utilisé par la suite.

La **modélisation logique** transforme ensuite ce modèle conceptuel en modèle logique en tenant compte des possibilités offertes et des mécanismes imposés par le **type de SGBD** utilisé. Du fait que nous nous concentrons ici sur les SGBD relationnels, seuls utilisés à notre époque, le modèle conceptuel est donc transformé en liste de tables normalisées liées par le mécanisme des clés primaires et étrangères.

La **modélisation physique,** finalement, adapte le modèle logique aux possibilités offertes et aux exigences imposées par le **SGBD spécifique** (restrictions éventuelles concernant la dénomination des tables et champs, types d'attributs offerts, etc.). Il existe effectivement des différences non négligeables à ce niveau entre les différents produits du marché. Le modèle physique doit finalement pouvoir servir, sans adaptation majeure, de script à la création automatique des tables dans le SGBD.

L'adaptation du modèle de données à un produit spécifique ne fait pas partie du sujet de ce livre et sera donc passée sous silence. C'est le pas le plus facile et qui peut en général être entièrement automatisé.

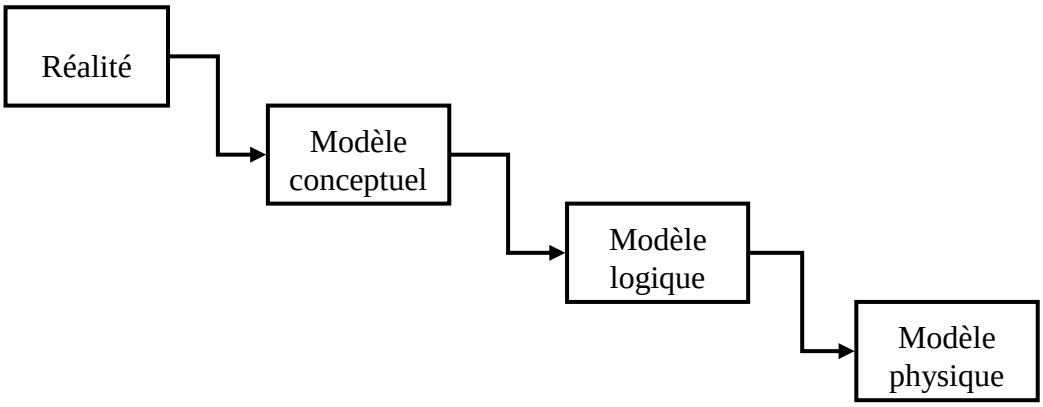

Figure 2.1. Le cheminement de la réalité à la base de données

Si la démarche du conceptuel au logique, puis au physique, telle que décrite ci-dessus, paraît évidente, encore faut-il préciser comment cette démarche est supposée être appliquée en pratique. Or c'est ici que les grands esprits divergents, que chaque professeur ou auteur présente les choses différemment et tente d'imposer ses préférences. Le malheur, c'est que l'on retrouve ces préférences dans certains outils de modélisation du marché qui nous forcent à procéder d'une certaine façon et nous empêchent de faire autrement. (À notre avis, un outil de modélisation devrait être totalement neutre au niveau méthodologique).

Des modèles séparés pour le conceptuel, le logique et le physique?

D'aucuns sont de cet avis. Certain outil de modélisation très répandu oblige même à le faire. Cet outil (nous n'en parlons que dans le but d'illustrer les différentes approches) ne distingue d'ailleurs que les couches conceptuelle et physique (le passage du conceptuel au logique/physique se faisant en un seul pas) et utilise des représentations graphiques différentes pour les deux vues (la seconde tirée des modèles hiérarchiques et réseaux du passé). Cet outil impose en outre d'autres règles (tout aussi absconses à notre avis): impossible par exemple de spécifier au niveau conceptuel un attribut décrivant une liaison (il s'agit ici d'une autre vieille querelle de méthodologue, nous y revenons ci-dessous). Cet outil crée automatiquement le **dessin** du modèle physique à partir du modèle conceptuel, mais le passage implique en même temps, c'est normal, de compléter de façon majeure la définition des données. Le problème réside dans le fait que s'il est nécessaire de retoucher le modèle conceptuel par la suite, ce qui est quasiment toujours le cas dans la réalité, le travail de définition des données devra être partiellement refait par la suite, car l'outil n'est pas vraiment itératif: impossible d'effectuer des allers-retours entre les deux modèles sans perte d'information.

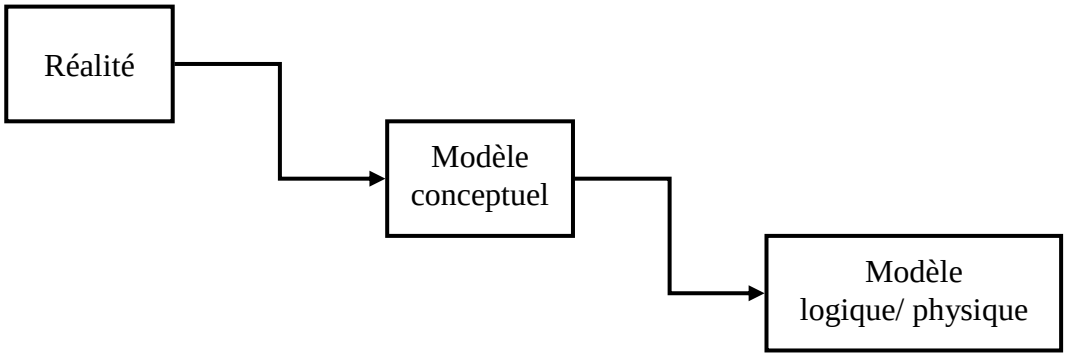

Figure 2.2. Approche conceptuelle – logique/physique de certains outils

Nous ne pensons pas qu'il soit nécessaire de réaliser des modèles conceptuel et logique séparés. Cela représente une dépense considérable sans apporter d'avantage réel. Et pose également le problème évoqué plus haut des allers-retours indispensables entre conceptuel et logique, car il faut impérativement préserver la synchronisation des modèles sur deux plans. Un des arguments majeurs avancés en faveur d'une séparation nette tient au fait que des personnes non formées à l'informatique participent à l'élaboration du modèle conceptuel. Il est alors sous-entendu que ces personnes comprendraient plus facilement un schéma conceptuel pur qu'un schéma logique/relationnel. Notre expérience de formateur et de modéliste de données dans maints projets nous a prouvé le contraire: il n'est pas plus difficile de former des non-spécialistes au schéma relationnel qu'à une autre représentation, y compris dans les cas où il est nécessaire de créer au niveau logique certaines tables pas forcément nécessaires au niveau conceptuel. Dans la pratique, les utilisateurs devront bien gérer ces tables par la suite, autant donc leur faire comprendre tout de suite à quoi elles servent. Nous ne voyons donc aucune raison de créer des modèles conceptuel et logique séparés, bien au contraire. À chacun son avis, mais, dans tous les cas, le **graphisme doit être le même** pour les deux modèles.

La séparation nette des modèles conceptuel et logique est répandue dans les milieux francophones suite à l'influence de la méthode Merise. Outre-Atlantique, l'approche est plus pragmatique. Réaliser différents modèles, c'est trop lent, trop compliqué! L'orientation objets, UML et la pléiade de spécialistes ayant gravité autour de Rational Software (avant son absorption par IBM) ont réduit l'influence des modélistes de données traditionnels des décennies précédentes et la popularité de leurs méthodes.

Dans cet ouvrage, nous proposons une approche plus directe. La démarche passe bien du conceptuel au logique/relationnel, mais se sert d'un modèle unique complété en cours de route: le modèle conceptuel se transforme en modèle logique sans changement de représentation. Ce modèle pourra par la suite être adapté si de nouveaux besoins se présentent, mais il n'y aura plus jamais de retour vers une vue purement conceptuelle. En outre, la transformation vers le modèle physique peut être automatique, **il n'y a donc en fin de compte qu'un seul modèle à dessiner**.

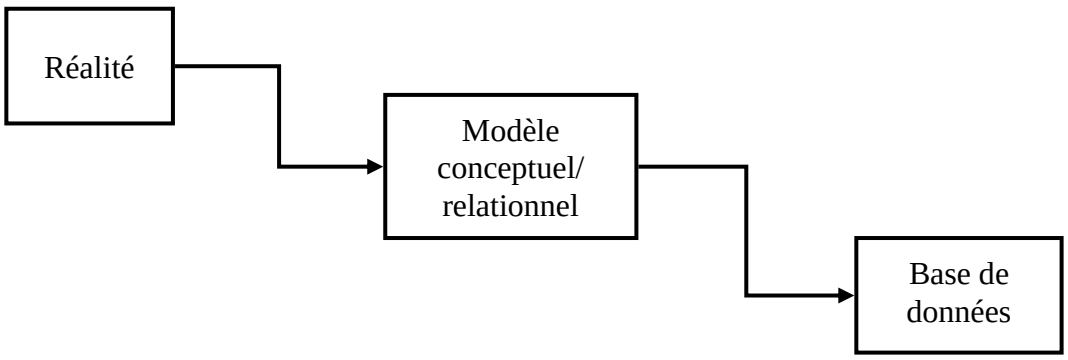

Figure 2.3. Approche conceptuelle/relationnelle – physique adoptée dans cet ouvrage

Au départ: le modèle conceptuel

Un modèle des données conceptuel comprend les éléments suivants:

Sous forme graphique:

- Les types d'objets du monde réel à décrire dans cette application
- Les associations reliant ces types d'objets

Sous forme de tableaux:

- Les tables correspondant aux types d'objets
- Pour chaque table, la liste des attributs qui décrivent les objets
- Les règles de gestion éventuelles associées aux tables et aux attributs.

Le modèle conceptuel devient modèle logique/relationnel

Le seul changement entre le modèle conceptuel et le modèle logique/relationnel consiste à **compléter** le premier en y ajoutant la spécification des **clés primaires** et celle des **clés étrangères** décrivant les associations. Aucune modification du graphique conceptuel n'est donc nécessaire. (Les concepts de clés primaires et secondaires seront définis au chapitre suivant pour qui ne les connaît pas).

Pour spécifier les clés primaires, il sera nécessaire dans certains cas d'ajouter aux définitions des tables de nouveaux attributs de nature purement technique (sans équivalent dans le système de gestion); mais dans la majorité des cas, il suffira de promouvoir des attributs (ou groupes d'attributs) existants au rôle de clé primaire ou étrangère.

Dans toutes les entreprises, les types d'objets tels que clients, articles, commandes, etc. sont souvent identifiés par des numéros ou des codes univoques qui figureront évidemment parmi les attributs des tables correspondantes (numéro de client, code article, numéro de facture, etc.). Mais ces attributs ne constituent pas forcément dans tous les cas, pour des raisons sur lesquelles nous reviendrons, de bons candidats à la clé primaire. Dans ce cas nous définirons une clé primaire "artificielle", un attribut supplémentaire géré automatiquement et que les utilisateurs ne verront peut-être même jamais.

Attributs décrivant des associations: une guerre interminable!

La liste des attributs figurant dans la définition d'un type d'objets peut-elle, oui ou non, contenir au niveau conceptuel des attributs décrivant en fait des associations entre objets? Les spécialistes se sont étripés à ce sujet et ne cessent de le faire!

Exemple: si je décris une commande d'un client, l'attribut *numéro_client* (spécifiant le client qui a passé cette commande) peut ou doit-il figurer dans la liste des attributs de la table *commande*? "En aucun cas" s'écrient de façon véhémente certains apôtres de l'approche entités-associations pure et dure. Selon eux, le lien entre la commande et le client doit exclusivement apparaître de façon **graphique** dans le modèle conceptuel. Faire le contraire consisterait à mélanger les niveaux conceptuel et logique, donc à assumer dès le départ une implémentation relationnelle. La transformation du lien entre commande et client sous la forme d'une clé étrangère ne sera alors faite que **lors du passage au modèle logique** en ajoutant à la table *commande* un attribut supplémentaire *numéro_client*.

Dans les SGBD hiérarchiques, en réseau et objets, les associations ne sont pas gérées par des attributs jouant le rôle de clé étrangère, mais au moyen de pointeurs cachés gérés par le système. Dans l'exemple cité, les commandes associées à un client forment ainsi une chaîne d'objets liés directement à l'objet qui décrit ce client et le système fournit des fonctions pour "naviguer" du client vers ses commandes et d'une commande de ce client vers une autre.

Mais presque plus personne n'utilise aujourd'hui des SGBD hiérarchiques, en réseau ou objets. Les principaux systèmes utilisés sont relationnels et, dans ce monde, les liaisons sont décrites à l'aide de clés étrangères. Dans l'exemple cité, *numéro_client* dans la table *commande* spécifie le client qui a passé cette commande. Ceci nous permet de retrouver le client à partir de la commande et, à l'inverse, de trouver toutes les commandes liées à un client. Cela nous permet également de formuler aisément des requêtes par *numéro_client* sur la table *commande* comme, d'ailleurs, par n'importe quel autre attribut. C'est la force de l'approche relationnelle.

Fidèle à notre philosophie de ne dessiner qu'un seul modèle de données évoluant progressivement du conceptuel au logique/relationnel puis au schéma de la base de données, nous n'avons donc aucune réticence à faire figurer dans la liste des attributs d'une table également des attributs qui décrivent des liaisons, attributs qui se transforment par la suite en clés étrangères. Le fait de faire figurer de tels at-

tributs dans le modèle conceptuel déjà permet également aux futurs utilisateurs de valider que des requêtes pourront être formulées sur les valeurs de ces attributs.

Il existe une autre raison importante de procéder ainsi. Lors de la modélisation conceptuelle apparaissent en effet fréquemment des attributs susceptibles de donner naissance à de nouvelles tables dont on n'aurait pas soupçonné l'existence au départ. Créer une nouvelle table permet de formaliser les valeurs qu'un attribut peut adopter. Cela peut également rendre la saisie de données plus rapide et éviter des erreurs de saisie. Quelques fois, le fait de formaliser ou non les valeurs d'un attribut au moyen d'une nouvelle table est sujet à controverse. Seule l'étude des besoins de l'application en question peut trancher un tel débat

Exemple:

Personne (no personne, nom, prénom, rue, localité, langue)

Dans cette définition du type d'objet *personne* se trouvent deux attributs susceptibles de donner naissance à des tables (types d'objets) de plein droit: *localité* et *langue*.

On pourrait donc obtenir un schéma avec trois tables, évidemment liées:

Personne (no personne, nom, prénom, rue, code localité, code langue)

Localité (code localité, nom localité)

Langue (code langue, désignation langue)

Créer des tables *Localité* et *Langue* permet de restreindre les valeurs de *localité* et de *langue* dans *Personne* à des listes de valeurs précises, donc d'empêcher, grâce à *l'intégrité référentielle*, la saisie de n'importe quelle valeur dans ces champs. Cela permet également de programmer des fonctions qui accéléreront considérablement la saisie et éviteront des erreurs en affichant les valeurs définies, par exemple la liste des localités, et en obligeant l'utilisateur à choisir une valeur dans cette liste. Mieux, une telle fonction pourrait permettre à l'utilisateur de ne saisir que les premières lettres de la localité pour lui afficher immédiatement toutes celles qui répondent à ces critères.

Faut-il créer des tables *Localité* et/ou *Langue*? Seule l'analyse des besoins nous le dira. Peut-être cette question sera-t-elle reléguée à plus tard. Si l'on interdit formellement de faire figurer dans la définition d'une entité des attributs décrivant des liaisons, la justification de la présence des attributs *localité* et *langue* dans *Personne* restera en suspens.

Ce qui précède ne signifie d'ailleurs pas qu'il faille forcément penser en termes de clés étrangères dès le départ, mais ce n'est pas interdit non plus une fois la démarche est devenue familière.

Au niveau conceptuel on pourra ainsi très bien écrire:

Commande (no commande, client, date, …)

sachant que *client* lie la commande au client et deviendra *no_client* par la suite.

Outil de modélisation

Faut-il utiliser un outil de modélisation? Évidemment! Encore faut-il en choisir un qui permette de progresser de manière efficace vers le but final: la création automatique de la base de données dans le SGBD choisi.

À notre avis, les critères de sélection pour un outil de modélisation sont les suivants:

- Permet la représentation graphique du schéma des données ainsi que sa documentation détaillée sous forme tabulaire dans un dictionnaire des données exploitable.
- N'impose pas des méthodes et principes de modélisation autres que ceux adoptés par l'équipe de modélisation, mais appuie la méthode et les principes adoptés par cette équipe.
- Permet de travailler en équipe, donc d'échanger et de partager des parties du modèle.

- Permet l'itération du processus de modélisation sur toute son étendue (réalité, conceptuel, logique, physique, schéma de la base de données) sans aucune perte d'information lors de retours en amont.
- Permet la génération automatique du schéma de la base de données **et la modification** de ce schéma lors d'adaptations ultérieures. (Il n'existe pas de raison de devoir recréer complètement la base de données avec exportation et réimportation des données du simple fait d'ajouter une nouvelle table ou de nouveaux attributs à des tables existantes).

Résumé

Les besoins de l'entreprise et du projet déterminent la structure finale du schéma de la base de données à créer.

L'efficacité du processus de développement, notamment la facilité d'arriver au schéma approprié, prime sur des questions de doctrine.

Un schéma de base de données n'est jamais terminé. Il évolue constamment, même après la mise en service de la base et de l'application. Le processus de modélisation est donc itératif. La méthode de développement et l'outil choisis doivent appuyer cette réalité.

Nous travaillons dans un monde dominé par les SGBD relationnels et il n'existe aucune raison d'adopter une démarche qui ne prenne pas en compte cette réalité dès le départ.