3. Le vocabulaire de la modélisation

Ce chapitre passe en revue les concepts que nous allons utiliser dans le cadre de la modélisation des données. Ces concepts sont dans leur majorité archi-connus et expliqués en long et en large dans tous les livres et cours, bien qu'il nous ait paru utile de préciser les choses vu que certains auteurs ne font que recopier ce qu'ils ont lu à gauche et à droite sans réfléchir une minute à ce qu'ils écrivent. Nous ne ferons par exemple pas l'erreur de confondre entité et type d'entité, souvent commise. Un rappel de ces définitions figure aussi dans l'annexe X.

Pour les représentations graphiques nous utilisons le formalisme UML, de plus en plus répandu. Un rappel des éléments graphiques utilisés dans cet ouvrage est fourni dans l'annexe Y.

Objet, entité

Le terme **entité** désigne chaque **objet** distinct (objet au sens large du terme) au sujet duquel des informations vont être enregistrées dans une base de données. Exemples: le client John Détraz, l'employée Marie Dutronc, la commande 2009-327 du 23.4.2009 du client John Détraz. Les informations concernant ces objets seront gérées dans le cadre du système projeté. À l'objet physique (la personne, la facture) correspond un objet dans le système informatique sous la forme d'un enregistrement de données. Nous préférons le terme **objet** au terme **entité** qui a souvent été galvaudé et utilisé dans des sens différents, mais les deux sont synonymes ici.

Type d'objet, type d'entité

Dans toute application informatique, toute base de données, on gère des **ensembles** d'objets d'un même **type**. Exemples: les employés, les clients, les articles, les factures.

Les termes **type d'objet** et **type d'entité** sont synonymes. Nous utilisons le premier.

Le nom d'un type d'objet est toujours singulier: client, article, etc.

Les informations enregistrées au sujet de l'ensemble des objets d'un même type sont stockées dans une base de données relationnelle sous forme d'une table. Chaque type d'objet dans le modèle des données correspond à une table dans la base de données et inversement.

Représentation de types d'objet dans le modèle conceptuel:

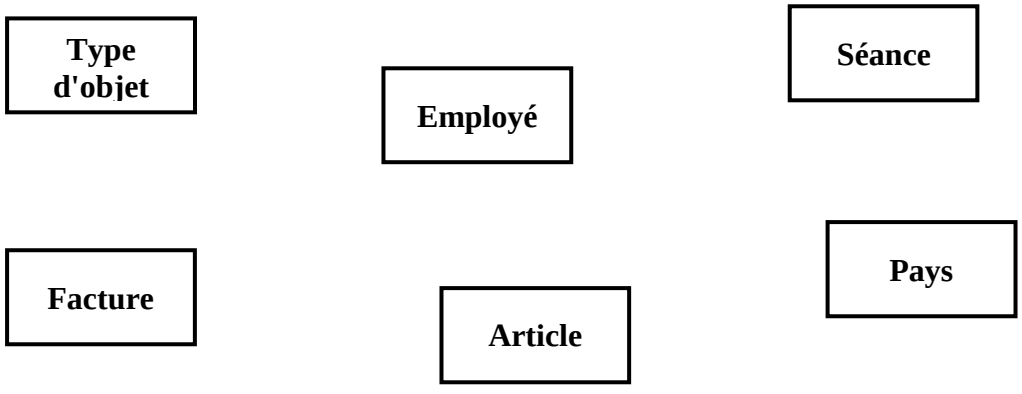

Figure 3.1 Exemples de types d'objets

Chaque élément de la figure ci-dessus représente *un* objet (quelconque) d'un type particulier: *un* employé, *une* séance, *une* facture, *un* article.

Attribut

Un **attribut** est un élément d'information stocké et géré dans le contexte d'un type d'objet spécifique.

La définition d'un type d'objet se fait sous forme d'une **liste d'attributs**.

Exemple: si l'on gère le type d'objet *personne*, la liste des attributs pourrait être la suivante:

Personne (nom, prénom, date de naissance, sexe, adresse, langue maternelle, langues parlées)

Nous reviendrons plus tard (chapitre 5) sur la théorie de la normalisation de Codd et Date. Il suffit de remarquer ici qu'elle prescrit, dans le modèle relationnel, certaines règles concernant la nature et la structure des attributs d'un type d'objet. Ces lois, appelées formes normales, sont aussi résumées dans l'annexe Z. Remarquons qu'elles ont aujourd'hui perdu une grande partie de leur justification théorique et pratique pour des raisons énoncées au chapitre 5.

Néanmoins, il est bon, dans un premier temps de respecter ces règles. En ce qui concerne la définition des attributs, elles stipulent que:

a) Chaque attribut doit être "atomique", c'est-à-dire non décomposable. Ce qui n'est clairement pas le cas de l'attribut *adresse* dans l'exemple ci-dessus.
A fortiori, un attribut ne devrait pas posséder une structure à plusieurs niveaux, tel que:
adresse (rue, no, localité (code postal, nom de ville))

b) Aucun attribut ne doit constituer une table de valeurs, ce qui est le cas pour l'attribut *langues parlées* dans l'exemple ci-dessus.

Une définition de type d'objet qui correspond à ces deux critères est dite **plate** et obéit à la formulation traditionnelle de la **première forme normale de Codd – Date**. Cette règle est aujourd'hui formellement abolie (voir chapitre 5 et annexe Z), mais garde en pratique sa signification du fait que les définitions plates sont considérablement plus faciles à gérer et que certains SGBD du marché nous les imposent de toute manière. Nous conseillons donc de ne spécifier, sauf exception, que des types d'objets à structure plate.

Nous verrons au chapitre 5 comment procéder pour éliminer des violations de ces règles.

Terminons provisoirement cette digression en remarquant que, dans les premières étapes de l'analyse des données, on ne se soucie en général pas de tels problèmes.

Type d'attribut

Le **type** de l'attribut désigne le genre d'information enregistré dans un attribut, définit l'ensemble des valeurs qu'il peut contenir.

On connaît les types de base traditionnels tels que: alphabétique, numérique, alphanumérique, booléen, date, heure, etc. Tous les SGBD modernes connaissent et gèrent ces types, même si les dénominations et définitions de ces types diffèrent quelque peu de produit à produit.

Ainsi, un SGBD refusera de stocker un texte dans un attribut spécifié *numérique*, autre chose qu'une date valable dans un attribut de type *date* ou encore autre chose que *vrai* ou *faux* dans un champ de type *booléen*.

La plupart des SGBD modernes offrent également le type **blob** (binary large object) pour stocker des documents, images, séquences sonores ou audiovisuelles. Si l'on voulait stocker une image pour chaque personne dans l'exemple ci-dessus, on ajouterait à la définition du type d'objet *Personne* un attribut *image* de type *blob*.

Tous les types d'attributs mentionnés jusqu'à présent sont des types techniques, permettant au SGBD de vérifier que les valeurs à enregistrer appartiennent bien au type spécifié.

Certains SGBD modernes permettent la définition de types dérivés complexes avec des structures internes, par exemple *adresse* (voir discussion ci-dessus). Ou encore la définition de types sémantiques,

dont la définition de l'ensemble des valeurs valables doit être fournie au moyen d'une liste ou par référence à une table de valeurs.

Le type de chaque attribut doit être spécifié au plus tard lors du passage au modèle physique des données.

Association

Une **association** est un lien unissant deux objets.

(Il est possible de concevoir des associations liant plus de deux objets, on les appelle ternaires, quaternaires, etc. mais nous réduirons systématiquement de telles situations à plusieurs associations entre paires d'objets, donc binaires.)

Dans la méthode Merise, l'association est appelée relation. Mais ce terme possède un sens différent dans le contexte relationnel et nous évitons donc soigneusement de l'utiliser. Nous dirons donc: modèle objets – associations et non pas entités – relations.

Une association ne porte pas de nom, mais peut (devrait) être documentée au moyen d'un verbe décrivant l'association. Voir l'exemple ci-dessous. Une petite flèche peut indiquer dans quelle **direction** la description doit être lue. À chaque extrémité de l'association, on peut indiquer le **rôle** joué par l'objet qui s'y trouve.

Les **cardinalités** indiquent le nombre d'objets pouvant se situer à chaque extrémité d'une représentation d'association (contrairement à la représentation Merise ou Chen où l'on indique le nombre d'associations liées à un objet unique). Si la cardinalité n'est pas indiquée, on admet qu'elle vaut exactement 1. Une cardinalité **n** ou ***** peut prendre les valeurs 0 à l'infini, on dit alors **plusieurs**, cas spéciaux 0 et 1 compris. Le type le plus courant est l'association ayant la cardinalité 1 d'un côté et la cardinalité * (plusieurs) de l'autre: on dit que c'est une association **un à plusieurs** ou **un à n**.

Représentation d'associations

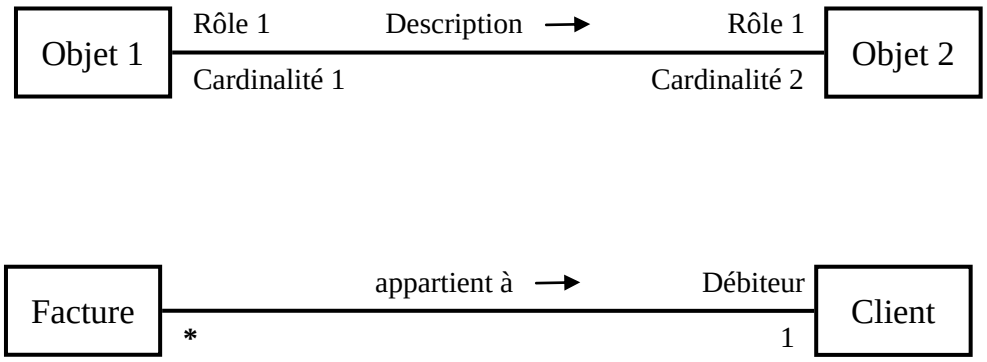

Figure 3.2. Formalisme et exemple d'associations

L'exemple ci-dessus doit être interprété comme suit: une facture appartient toujours exactement à un client. Un client peut posséder de 0 à n factures.

Dans le modèle relationnel, les associations sont implémentées au moyen du mécanisme des clés primaires et étrangères. Toujours dans ce modèle, une association ne possède pas d'attributs, c'est un

simple lien. Nous verrons par la suite comment tenir compte des attributs qui pourraient être liés à une association.

Agrégation

L'**agrégation** est une association particulière indiquant que l'objet figurant du côté **un** de l'association (l'**agrégat**) **se compose** des objets (les **composants**) figurant au côté **plusieurs**, ou **contient** ces objets. UML fait encore la distinction entre agrégation forte (les composants ne peuvent exister en l'absence de l'agrégat) et agrégation faible (les composants peuvent avoir une existence indépendante). Nous ne voyons pas de raison suffisante pour faire cette distinction dans le cadre de la modélisation des données, mais elle reste possible.

Le recours à la notion d'agrégation, sémantiquement plus précise que la simple association, se révèle souvent utile en modélisation des données.

Représentation d'agrégations

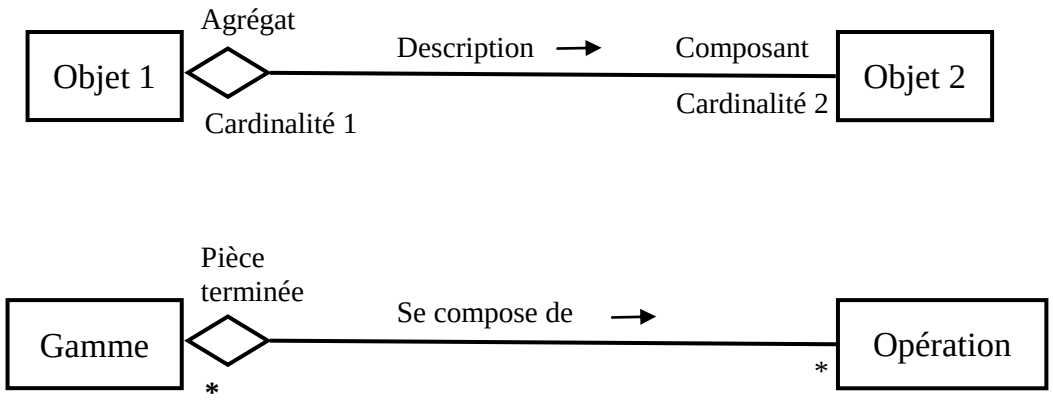

Fig. 3.3. Formalisme et exemple d'agrégations

L'exemple ci-dessus indique qu'une gamme d'opérations pour fabriquer un type de pièces se compose de plusieurs opérations qui peuvent se retrouver dans des gammes différentes.

Généralisation – spécialisation

Notion tirée de l'orientation objets, ce type de relation ne décrit **pas** une association entre objets. Le graphisme généralisation – spécialisation indique que le type d'objet figurant du côté de la flèche correspond à un sur-ensemble du type d'objet figurant de l'autre côté. On parle aussi de **super-type** et de **sous-type,** termes que nous utiliserons ici. Ainsi, *être humain* est super-type de *homme* et de *femme*, alors que *homme* et *femme* sont sous-types de *être humain*. Chaque homme, chaque femme *est un* être humain. Le sur-type correspond à une généralisation du sous-type, le sous-type à une spécialisation du sur-type. Quelques règles concernant la généralisation – spécialisation doivent être observées si l'on veut conserver la parité avec l'orientation objet classique:

a) Les ensembles d'objets correspondant à différents sous-types doivent être disjoints: un objet particulier ne peut pas appartenir à deux ensembles d'objets de sous-type différent. Exemple: une personne ne peut pas être homme **et** femme.

b) Un objet ne peut pas changer de sous-type. Il naît, vit et meurt appartenant à l'ensemble d'objets défini par un sous-type unique.

Représentation de généralisations – spécialisations

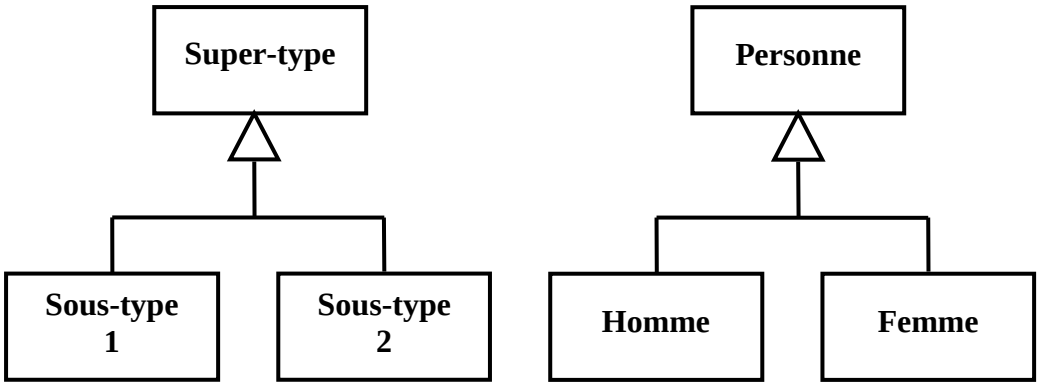

Figure 3.4. Formalisme et exemple de généralisation - spécialisation

Les sous-types d'objets doivent contenir tous les attributs du super-type, mais peuvent contenir des attributs supplémentaires que ne partagent pas les autres sous-types du même super-type. Exemple: le sous-type *homme* pourrait posséder un attribut *barbu* (oui/non) et le sous-type *femme* un attribut *nombre de grossesses*.

Malheureusement, la notion de généralisation – spécialisation, intéressante en soi, est d'une utilité très limitée dans le cadre de la modélisation des données en informatique de gestion (alors que son importance est énorme en programmation). Souvent les exemples présentés dans les ouvrages sont malheureusement discutables, voire carrément fallacieux.

Nous verrons comment traiter de telles situations en pratique. Nous verrons en particulier qu'il ne faut jamais représenter des *rôles* (par exemple tenus par des personnes dans une organisation) au moyen de cette technique.

Clé primaire

La **clé primaire** est un attribut (ou groupement d'attributs) dont la valeur est intrinsèquement unique et qui distingue donc chaque objet de tous les autres du même type.

La définition suivante du type d'objet *Personne*:

Personne (nom, prénom, sexe, date de naissance)

ne comporte pas de clé primaire. Rien n'empêche en effet deux personnes distinctes de posséder exactement les mêmes valeurs des attributs. Une clé primaire possible pour *Personne* serait:

Personne (no sécurité sociale, nom, prénom, …)

Convention: nous soulignons l'attribut qui constitue (ou le groupe d'attributs qui constituent) la clé primaire de la définition d'un type d'objet. Autre possibilité: faire suivre le nom de l'attribut de PK (primary key) ou CP (clé primaire)

Chaque définition de type d'objet devra finalement comprendre une clé primaire en vue de l'implémentation avec un SGBD relationnel, mais cette exigence est sans importance lors des premières étapes de la modélisation.

La notion de clé primaire est discutée en détail dans les chapitres de ce livre.

La notion de clé primaire (notion conceptuelle) ne doit pas être confondue avec la notion d'index sur une colonne d'une table (notion d'implémentation physique), bien que la définition d'un index unique sur la ou les colonnes correspondant à la clé primaire constitue avec certains SGBD la seule manière d'assurer l'unicité des valeurs de ce champ.

Clé étrangère

Une **clé étrangère** est un attribut (ou groupement d'attributs) faisant référence à la clé primaire d'un autre (en général) type d'objet. La valeur de la clé étrangère doit figurer parmi les valeurs des clés primaires du type d'objet concerné. Par contre la valeur d'une clé étrangère peut aussi être nulle (non définie) si les règles de gestion le permettent. La définition d'un type d'objet peut contenir plusieurs clés étrangères.

Les clés étrangères constituent le mécanisme utilisé dans le modèle relationnel pour spécifier les associations. Une clé étrangère définit en effet une association de l'objet en question avec un autre objet, généralement d'un autre type. Sans autre indication, une clé étrangère indique une association de type plusieurs à un. Chaque objet du type dans lequel figure la clé étrangère ne peut en effet être associé qu'à un seul objet du type auquel elle se réfère, mais plusieurs objets du type où figure la clé étrangère peuvent être associés à un même objet du type auquel il est fait référence. La notion de clé étrangère est discutée en détail dans les chapitres de ce livre.

Exemple de clé étrangère:

Client (no_client, raison sociale, …)

Facture (no facture, *no_client*, date, montant)

Cet exemple indique qu'il existe une association entre les objets de type de *Facture* et les objets de type *Client*. Il s'agit d'une association de type plusieurs à un. Plusieurs factures peuvent être liées à un client, mais chaque facture n'est associée qu'à un seul client.

Convention: nous représentons les clés étrangères en italique. Autres possibilités: souligner les clés étrangères en traitillés ou les faire suivre de FK (foreign key) ou CE (clé étrangère). Il est également possible de préciser le type d'objet référencé par la clé étrangère. Exemple: no client (CE Client).

La notion de **clé étrangère** (notion conceptuelle) ne doit pas être confondue avec la notion d'**index secondaire** (notion d'implémentation), quoique la définition d'un index secondaire sur une colonne d'une table correspondant à une clé étrangère soit fréquente dans la pratique; ceci de manière à permettre un accès rapide aux objets par la valeur de cette clé étrangère, par exemple pour trouver toutes les factures liées à un client dans l'exemple ci-dessus.

Intégrité référentielle

Règle précisant que la valeur d'une clé étrangère doit figurer parmi les valeurs de la clé primaire référencée par cette clé étrangère. Ainsi la valeur de *no_client* de *Facture* dans l'exemple précédent doit figurer parmi les valeurs de *no_client* dans *Client*: la facture **doit** faire référence à un client existant.

Les règles de gestion déterminent si la valeur d'une clé étrangère peut être nulle (non définie) ou non.