Notes sur la modélisation traditionnelle des systèmes d'information

Francois.Rioult@unicaen.fr

5 janvier 2021

Ce cours est largement inspiré de voire cite intégralement celui de Pierre Beust.

1 Motivations

1.1 Système d'information

C'est un ensemble d'éléments, matériels ou pas, en interaction entre eux, transformant des éléments d'entrée en éléments de sortie.

Les systèmes seront constitués par des organisations et fonctionnent en vue de la réalisation de certains objectifs.

Le SI (système d'information) représente la *mémoire* de l'organisation, relativement :

- aux flux : produits en stocks, produits commandés, bons de livraison, factures, bons de commandes
- à l'univers extérieur : clients, fournisseurs
- à l'organisation de l'entreprise : que se passe-t-il entre l'enregistrement d'une commande et sa livraison ?
- aux contraintes légales : lois, règlements, paramètres financiers...

Le SI étant la mémoire de l'organisation, on se focalise ici sur la formalisation des données et donc leur modèle, plutôt que leur traitement, qui sera délégué à un langage de programmation, comme PHP, Java, etc.

2 Analyse et conception

Trois niveaux :

1. conceptuel. C'est le plus haut niveau d'abstraction, où l'on va par exemple manipuler des clients, des produits. On décrit qui fait quoi, quand, où. On obtient un Modèle Conceptuel des Données (MCD), généralement représenté à l'aide d'UML. On pourra également trouver un Modèle Conceptuel des Traitements (MCT). Ces deux modèles forment le Schéma Conceptuel.

2. logique : ce niveau décrit le comment faire, en Modèle Logique des Données (MLD). On utilisera ici le *modèle relationnel*. On y précise également un Modèle Organisationnel des Traitements (MOT). MLD et MOT forment le Schéma Logique du SI. Le schéma logique est la traduction du schéma conceptuel selon un modèle générique existant. Ce schéma logique comprend également l'expression de contraintes d'intégrité (types de valeurs, unicité de clé) garantissant la cohérence et la vraisemblance des données. Les schémas logiques sont totalement indépendants de la technologie utilisée (matérielle et logicielle).
3. physique : Décrire le Comment faire faire par l'ordinateur. Il est question ici d'implémentation d'un gestionnaire de bases de données comme un ensemble d'objets informatique : table de hachage, arbres équilibrés, etc.

3 Conception de schémas conceptuels

On utilise les diagrammes de classe UML pour représenter les *entités* et leurs *association*, i.e. rapports entre entités. Le vocabulaire est analogue à celui de la conception objet.

On différencie :

— *classe* d'entité : regroupe un ensemble d'entités, définie en *extension* (chacun des objets de la classe) ou en *intension* (ou comprehension) : description des propriétés communes à toutes les occurrences.

— *occurrence* d'entité.

Les propriétés communes aux entités d'une classe sont les *attributs*. Certaines propriétés sont le résultat d'un calcul (par exemple le solde d'un compte est la somme de toutes les opérations sur le compte) : on appelle ce type d'opération des *méthodes*.

Une *association* est un lien logique entre plusieurs entités. On utilise des verbes pour les désigner. Certaines associations possèdent des attributs, comme une date ou une quantité par exemple lorsque qu'un client commande un produit : on parle de classe d'association.

4 UML

Le minimum à faire pour que le diagramme soit correct :

— la première lettre d'un nom de classe est une majuscule

— les attributs sont en minuscules

— on met un sens et des cardinalités aux associations

Le minimum à faire pour montrer qu'on a compris que c'est un diagramme de modélisation et pas du code informatique de documentation :

— on ne met pas les types s'ils sont évidents. Par exemple, je me doute bien qu'un nom est une String, une date une Date, etc.

— on ne met pas les identifiants techniques type clé primaire et clé étrangère. C'est une faute de mettre en modélisation des éléments qui relèvent de la logique de l'implémentation. Je suis un bon informaticien, j'ai l'habitude de mettre des clés, je ne les mets pas sur le diagramme de modélisation.

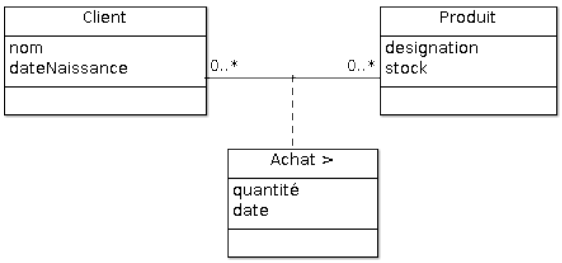

FIGURE 1 – Diagramme des classes

— on ne pinaille pas sur les cardinalités. Ce qui est important, c'est de faire la différence entre un nombre fixe (1, 2, 4 pour les roues d'une voiture) et un nombre variable (nombre de produits achetés).
— on ne dupplique pas les attributs dans les classes dérivées en cas d'héritage.
— on fait attention aux champs calculés : ce ne sont pas des attributs mais des méthodes.
— on n'utilise pas des outils automatiques de conception de diagramme à partir d'un dump de la base de données.
— on fait de l'UML et rien d'autre.

5 Rappels sur le vocabulaire du modèle relationnel

Un gestionnaire de bases de données relationnelles comme Postgres ou MySQL est organisé en *bases de données* qui contient des *tables* ou *relations*. Une relation est décrite par des *attributs* et contient des *enregistrements* ou *t-uplets*.

6 Du schéma conceptuel au schéma relationnel

On veut passer du schéma conceptuel du SI au schéma relationnel du SI, pour implémentation sur un modèle logique à base de *relations*. On s'appuie sur la modélisation réalisée à l'aide d'UML :

Règle 1 : Pour chaque classe d'entités, construire une relation ayant pour attributs ceux de la classe. Sur l'exemple précédent de la figure 1 on obtient les relations suivantes :
— Client(nom, dateNaissance)
— Produit(designation, stock)

Règle 2 : Pour chaque association, on crée une relation ayant pour attributs les identifiants des entités associées et on y ajoute éventuellement ses attributs propres :
— Achat(nom, designation, date, quantite)

Règle 3 : il s'agit d'une exception à la règle 2, dans le cas où l'une des cardinalités de l'association n'est pas multiple (i.e. variable). Dans ce cas, on ajoute une clé étrangère à la relation correspondant à la classe d'entité portant cette cardinalité finie. Par exemple, supposons que les produits sont fournis par un seul fournisseur mais que le fournisseur peut fournir plusieurs produits : dans ce cas, la relation "Produit" possèdera une clé étrangère référençant le fournisseur.

7 Contraintes d'intégrité

Les contraintes d'intégrité principales sont fournies par les *clés*.

7.1 Clé primaire

> La clé primaire est un attribut ou une combinaison d'attribut qui rend chaque enregistrement unique.

On créera parfois pour cela un attribut dédié à la gestion de la clé : un entier la plupart du temps mais parfois un attribut existant joue parfaitement le rôle de clé primaire. Par exemple, le numéro de sécurité sociale, la plaque d'immatriculation, etc. La clé primaire n'est donc pas nécessairement un entier.

Une clé peut être *composite*, comme par exemple pour un achat :
— l'identifiant du client
— l'identifiant du produit
— la date d'achat

Pour déterminer quelle est la clé primaire, il faut passer en revue les attributs et leur combinaisons et se poser la question : cette combinaison rend-elle unique chaque enregistrement ? ou Plusieurs enregistrement peuvent-ils avoir cette même combinaison d'attributs ?

L'intérêt d'avoir une clé primaire qui rend chaque enregistrement unique est double :

1. éviter de procéder plusieurs fois à un même enregistrement, car le système de gestion de la base de données indique dans ce cas une erreur
2. permettre d'indexer une collection d'enregistrement pour obtenir de bonnes performances en cas de recherche d'un enregistrement particulier.

7.2 Clé étrangère

Une clé étrangère est une clé primaire dans une relation qui permet de faire référence dans une autre relation à un enregistrement particulier. Par exemple, dans la relation des achats, une clé étrangère référence le client et une autre le produit.

L'intérêt de déclarer une clé étrangère auprès du gestionnaire est de lui offrir la possibilité de générer une erreur en cas de référence d'un enregistrement qui n'existe pas.

7.3 Aspects techniques sur les clés

En PostgreSQL, la contrainte d'unicité de la clé primaire est assurée par un index btree. On a donc une assurance de recherche rapide par cette clé. On a également l'assurance que les jointures par clés étrangères seront performantes, ce qui est primordial !

En cas de clé composite ou string, il y a également un index, mais suivant la recherche, cela peut ne pas être optimal (recherche plein texte impossible en btree par exemple).

De façon générale, il faut éviter les clés primaires sous forme de string et les clés composites. Il est conseillé de toujours créer une clé primaire synthétique, par exemple un entier. Elle sera interne à la base de données mais permettra d'obtenir de bonnes performances. On pourra, en parallèle, mettre en place des contraintes d'unicité sur des clés composites métier si besoin.

Pour les détails, voir [1].

7.4 Autres contraintes d'intégrité

unicité : `UNIQUE`

non absence : `NOT NULL`

domaine de valeurs : `CHECK (VALUE > 0)`

8 Bréviaire SQL

On pourra s'inspirer des exemples ci-dessous pour implémenter le SI en SQL :

```
DROP TABLE IF EXISTS <table>;
CREATE TABLE <table>(id VARCHAR(20), <attribut> INT, PRIMARY KEY(id));
CREATE TABLE <table>(date DATE, <attribut> VARCHAR(20) REFERENCES <table>,
                     <attribut> FLOAT, <attribut> INT REFERENCES <table>,
                     PRIMARY KEY(<attribut>, <attribut>, date));
INSERT INTO <table> VALUES(DATE '2020-02-21', 'Alphonse', 5, 1);
UPDATE <table> SET <attribut> = <attribut> + 1 WHERE <attribut> = '<valeur>';
DELETE FROM <table> WHERE <attribut> = '<valeur>' AND date = DATE '2020-02-17';
```

1. https://public.dalibo.com/exports/marketing/livres_blancs/dlb04-modeliser_avec_postgresql/DLB04_Modeliser_avec_PostgreSQL.pdf