MERISE

Merise

Analyser un Système d'Information déroute parfois le non-initié, car traduire un environnement de travail en symboles cabalistiques n'est pas très habituel pour qui ne connaît pas. Pourtant, avec une once de théorie et deux grammes de pratique, on se rend compte que le processus est très abordable, soumis à quelques règles simples, faciles à acquérir et qui s'appliquent toujours de la même manière. La méthode décrite ici est MERISE, elle est Française et a plus de 20 ans. Elle consiste à concevoir un Modèle Conceptuel de Donnée (MCD), le transposer en Modèle Logique de Données Relationnelles (MLDR), puis à générer le Modèle Physique correspondant (MPD). C'est la plus répandue des techniques d'analyse de Base de Donnée. Nous étudierons plus particulièrement aujourd'hui la construction du Modèle Conceptuel de Donnée et de ses 5 caractéristiques : Entités, Propriétés, Identifiants, Associations, Cardinalités.

Le Système d'Information

La première priorité est de transformer ce que l'on veut analyser en mots simples. L'écriture de cette petite rédaction permet à elle seule de bien comprendre ce que l'on va modéliser. Il s'agit à ce stade d'établir un lien entre l'informaticien et les utilisateurs, il ne faut donc pas hésiter à faire relire votre petit texte et à poser toutes les questions qui vous viennent à l'esprit afin de bien analyser l'existant. La difficulté principale est d'arriver à faire abstraction de vos habitudes de programmation : à ce stade, nous sommes totalement indépendant du matériel et du logiciel. Ne pensez pas en terme de tables. Pensez en terme d'entités.

Prenons l'exemple très simple d'un logiciel ayant pour but de gérer les envois de NewsLetters aux abonnés d'un site ayant plusieurs rubriques. Le service marketing veut aussi savoir quelle raison a poussé l'abonné à s'inscrire en lui proposant plusieurs choix de motivations lors de son inscription.

Le Système d'Information se décrit ainsi :
"Un abonné est inscrit à une ou plusieurs rubrique. Chaque rubrique envoie une NewsLetter chaque semaine aux abonnés de la rubrique correspondant. Un abonné a une motivation d'inscription parmi plusieurs possibles."

Ces quelques phrases, si elles sont exactes et validées par le client, sont suffisantes pour modéliser notre premier modèle. Elles contiennent en effet toutes les informations nécessaires.

Identifier les entités présentes

L'entité ABONNES représente l'ensemble des abonnés. L'entité RUBRIQUES l'ensemble des rubriques auquelles l'abonné peux s'inscrire. L'entité NEWSLETTERS représente les newsletters envoyées, MOTIVATIONS l'ensemble des motivations d'inscriptions des abonnés.

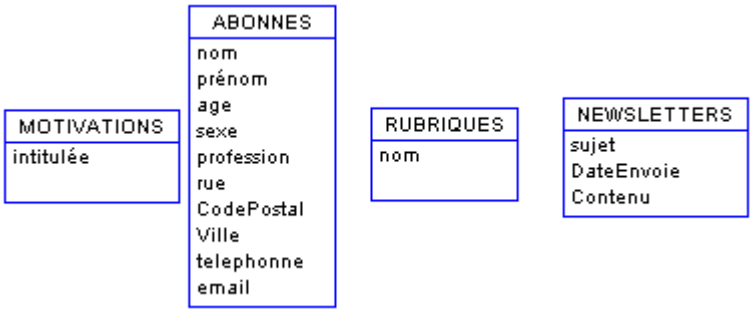
D'où les 4 entités :

Généralement, une entité est crée dans le Système d'Information si elle possède au moins 2 occurrences. Chaque 'élément d'une entité' est appelé une **occurrence** de l'entité.

Lister les propriétés des entité

Un Abonné est caractérisé par son nom, son prénom, son âge, son sexe, sa profession, sa rue, son code postal, sa ville, son pays, son téléphone et son email.
Une Newsletter est caractérisée par son sujet, sa date d'envoi et son contenu.
Une Motivation est caractérisée par son intitulé.
Une Rubrique est caractérisée par son nom.

Les 4 entités deviennent :

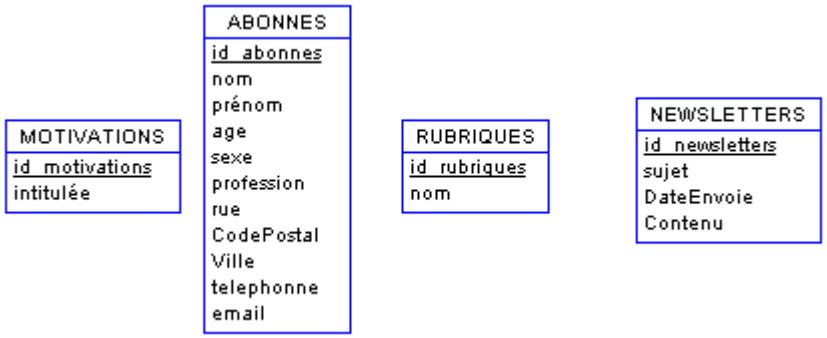


Afin de ne pas en avoir trop, on se limite généralement aux propriétés nécessaires au développement. Chaque propriété doit avoir une seule valeur possible pour chaque occurrence, sinon il s'agit d'une entité. Elle doit de plus être élémentaire et non-décomposable. Par exemple, l'adresse n'est pas une propriété élémentaire : elle comporte une rue, un Code Postal et une ville qui elles, sont 3 propriétés élémentaires.

Identifier de manière unique chaque occurrence

Imaginons que nous ayons deux abonnés qui s'appellent 'DUPOND : il est nécessaire de les distinguer sous peine de les confondre. On rajoute alors une propriété qui permettra d'identifier de manière unique chaque occurrence. Cette propriété est appelé l'**identifiant** de l'entité. Cela peut être une référence interne, un code, ou plus généralement un nombre entier. Cette propriété est soulignée afin de mettre en évidence son rôle d'identifiant.

Les 4 entités sont finalement :

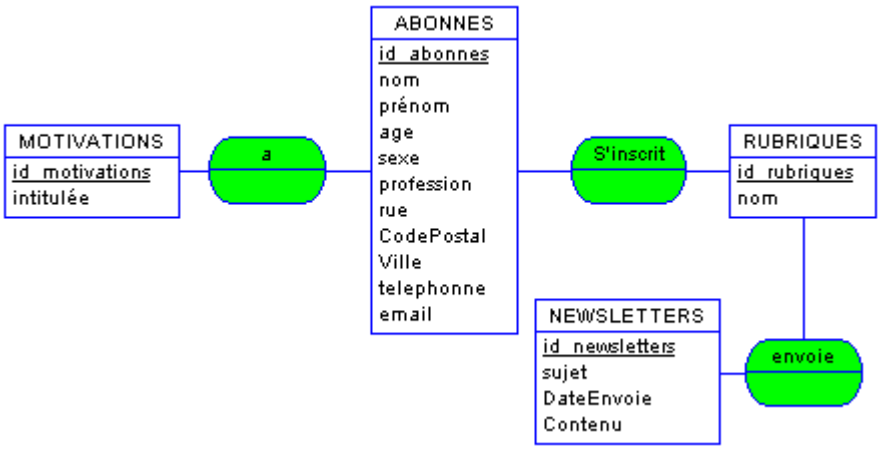


Etablir les relations entre les différentes entités

Maintenant, il s'agit d'identifier les relations entre les entités. Généralement, la simple transposition du texte suffit, les Sujets et Compléments d'Objets étants les entités, et les Verbes les relations.

Reprenons notre texte initial :
"Un Abonné a une Motivation. Un Abonné s'inscrit à une ou plusieurs Rubriques. Chaque Rubrique envoie une NewsLetter."

Les verbes sont en rouge et relient les entités. Il suffit de les intégrer au schéma :



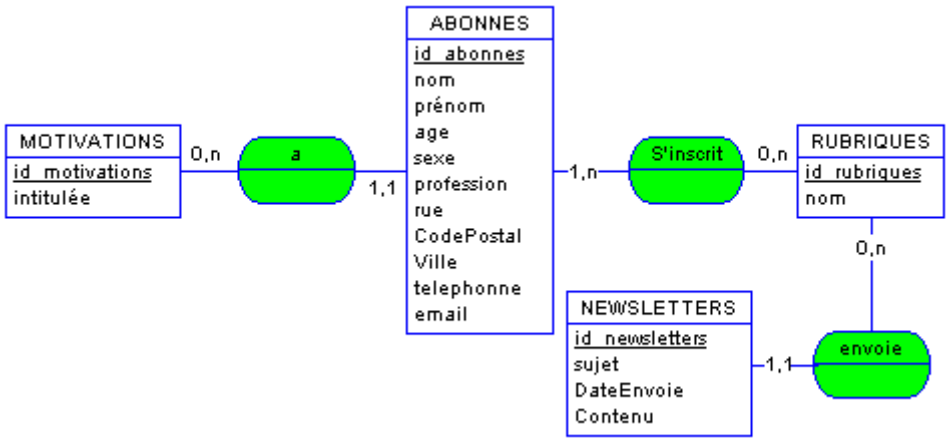
Identifier les cardinalités

Il faut maintenant établir le nombre possible d'interactions entre les entités.

Il s'agit d'un couple d'entiers de type (a ; b) .
a est la cardinalité minimum, et est égal à 0 ou 1.
b est la cardinalité maximum, et est égal à 1 ou n, n étant plus grand que 1.

Continuons notre exemple :
Un Abonné a ici une et une seule Motivation d'inscription, le marketing ayant imposé un champ obligatoire afin d'avoir cette valeur. On a donc 1 minimum, et 1 maximum. D'où la cardinalité (1;1).
Une Motivation donnée concerne 0 ou plusieurs Abonnés. On a donc 0 minimum, et n en maximum. D'où la cardinalité (0;n).
De même, un Abonné s'inscrit à une ou plusieurs Rubriques : (1;n),
Et une Rubrique possède 0 ou plusieurs Abonnés : (0;n).
Enfin, une Rubrique envoie 0 ou plusieurs Newsletters : (0;n),
Et une Newsletter appartient à une et une seule Newsletter : (1;1).

Il suffit maintenant de marquer ces couples sur le schéma, et nous avons notre Modèle Conceptuel de Donnée (MCD) :



Valider le Modèle avec le client

A ce stade, il est aisé d'aller voir encore une fois les utilisateurs du logiciel final, afin de discuter le MCD avec eux. Cela vous permettra d'entériner les propriétés qu'ils désirent utiliser, d'être bien certain des cardinalités, et de valider avec eux cette partie de votre travail. Un MCD doit pouvoir s'expliquer avec des phrases simples et être compréhensible par tout le monde. Il ne s'agit ni plus ni moins que de modéliser l'existant. Ainsi, vous serez certain de faire le développement demandé, et cela vous

permettra de vous protéger par la suite en cas de nouvelles demandes ou de modification du cahier des charges.

Il est important de bien réaliser que jusqu'à ce stade, toute cette analyse s'est déroulée totalement indépendamment de la machine ou de toute contrainte logicielle.

Ces règles fonctionnent toujours, même si il peux y avoir parfois plusieurs solutions pour chaque modèles. Le processus de modélisation, après quelques tentatives, est très simple à acquérir. Enfin, une fois le Modèle Conceptuel de Donnée établi, vous aurez fais le plus difficile. La conception de la base qui en découle est mécanique, et repose sur 6 règles strictes, nécessaires et suffisantes. Transformer un MCD en Modèle Logique, puis Physique est tellement standardisé que certains logiciels le font automatiquement....

Merise : 2ème partie

Après avoir conçu le Modèle Conceptuel de Donnée (MCD), il est maintenant temps de le transposer en Modèle Logique de Données Relationnelles (MLDR). Ce MLDR est en fait le dernier pas vers le Modèle Physique de donnée(MPD), c'est à dire la description de la base qui va être crée. Et là, deux solutions s'ouvrent à vous : soit vous laissez à un programme le soin de transformer votre MCD, soit vous le faîtes vous-même. Dans les deux cas, il est utile d'avoir un minimum de connaissance théorique sur le sujet. Après avoir définis les notions de clé primaire et de clé étrangère, nous étudierons plus particulièrement aujourd'hui les 6 règles strictes, nécessaires et suffisantes pour passer d'un MCD à un MLDR, et nous les appliquerons ensuite au schéma de Newsletter que nous avons écris la dernière fois.

Préliminaires : le Modèle Logique de Donnée (MLD)

Il s'agit du passage entre le Modèle Conceptuel de Donnée et l'implémentation physique de la base. Le MLD est lui aussi indépendant du matériel et du logiciel, il ne fait que prendre en compte l'organisation des données. C'est d'ailleurs le point primordial de la modélisation : si l'organisation des données est relationnelle (si elles sont "liées" entre elles), alors le MLD est Relationnel et devient le MLDR, ou Modèle Logique de Donnée Relationnel. Pour la petite histoire, le MLDR a été inventé par Codd en 1970, et repose sur la Théorie Ensembliste...

Un peu de vocabulaire : Les **données** sont stockées dans des **relations**. Une **relation** est un ensemble de **T-uple**, et un **T-uple** est définis par un ou plusieurs **attributs**. Dans la pratique, la **relation** est en fait la table, un
T-uple est une ligne (ou enregistrement), et les **attributs** sont les colonnes.

Exemple de la table NEWSLETTER :

id_newsletter	Sujet	DateEnvoie	Contenu	id_rubrique
25	Newsletter N°25	11/01/2001	Texte 25	10
26	Newsletter N°26	21/01/2001	Texte 26	20

Cette table est décrite par :
NEWSLETTER (id_newsletter, Sujet, DateEnvoie, Contenu, #id_rubrique)

Chaque enregistrement doit être identifié de manière unique (voir la notion d'identifiant abordée dans l'article précédent). L'attribut qui permet d'identifier de façon unique chaque ligne est appelée la **Clé Primaire**. Elle peut être composée, c'est à dire comprendre plusieurs attributs. Ici, il s'agit de l'attribut **id_newsletter**.

La table Newsletter comprend un attribut provenant de la table **RUBRIQUES**, l'attribut **id_rubrique**. Cet attribut est appelé **Clé Etrangère**.

Dans le formalisme, la clé primaire est soulignée, et la clé étrangère est précédée du signe #. D'où l'écriture définitive :

MATABLE (Cle_Primaire, Colonne1, Colonne2, #Cle_Etrangere)

Dans notre exemple :

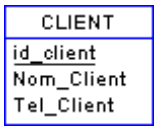
Rubrique (id_rubrique, Nom)
Newsletter (id_newsletter, Sujet, DateEnvoie, Contenu, #id_rubrique)

Ici, **id_rubrique** est la **Clé Primaire** de la table **RUBRIQUE**, et est une **Clé Etrangère** dans la table **NEWSLETTER**.

Une fois assimilée ces notions de **clés primaires** et de **clés étrangères**, nous pouvons maintenant énoncer les règles suivantes :

1 : Une entité se transforme en une relation (table)

Toute entité du MCD devient une relation du MLDR, et donc une table de la Base de Donnée. Chaque propriété de l'entité devient un attribut de cette relation, et dont une colonne de la table correspondante. L'identifiant de l'entité devient la **Clé Primaire** de la relation (elle est donc soulignée), et donc la **Clé Primaire** de la table correspondante.



<==> CLIENT (id_client, Nom_Client, Tel_client)

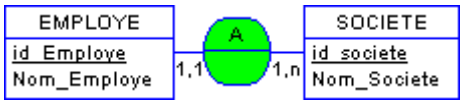
2 : Relation binaire aux cardinalités (X,1) - (X,n), X=0 ou X=1

La **Clé Primaire** de la table à la cardinalité (X,n) devient une **Clé Etrangère** dans la table à la cardinalité (X,1) :

Exemple de Système d'Information (SI) :

Un employé a une et une seule société. Une société a 1 ou n employés.

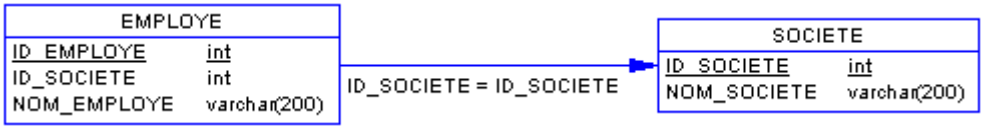
Modèle Conceptuel de Donnée (MCD) :



Modèle Logique de Donnée Relationnelle (MLDR) :

EMPLOYE (id_Employe, Nom_Employe, #id_Societe)
SOCIETE (id_Societe, Nom_Societe)

Modèle Physique de Donnée (MPD), ou schéma de base :



3 : Relation binaire aux cardinalités (X,n) - (X,n), X=0 ou X=1

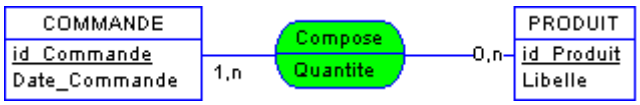
Il y a création d'une table supplémentaire ayant comme **Clé Primaire** une clé composée des **identifiants** des 2 entités. On dit que la **Clé Primaire** de la nouvelle table est la **concaténation** des

Clés Primaires des deux autres tables.
Si la relation est porteuse de donnée, celles ci deviennent des attributs pour la nouvelle table.

S.I. :

Une commande est composée de 1 ou n produits distincts en certaine quantité. Un produit est présent dans 0 ou n commandes en certaine quantité.

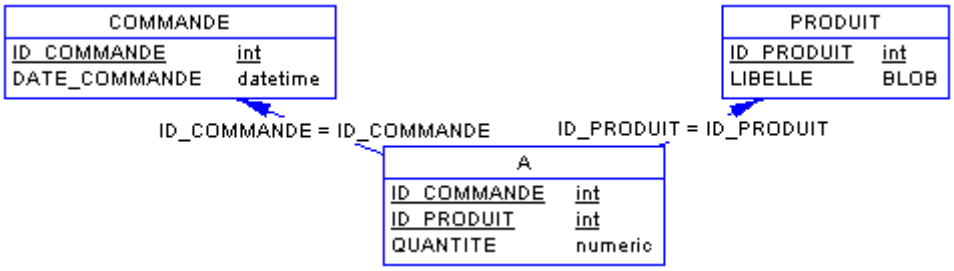
MCD :



MLDR :

COMMANDE (id_Commande, Date_commande)
PRODUIT (id_Produit, libelle)
COMPOSE (id_Commande, id_Produit, qantité)

MPD :



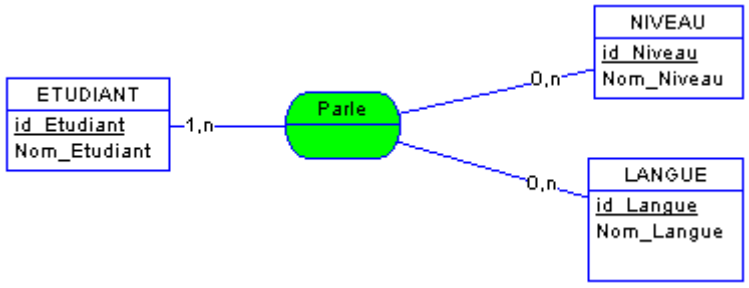
4 : Relation n-aire (quelles que soient les cardinalités).

Il y a création d'une table supplémentaire ayant comme **Clé Primaire** la **concaténation** des **identifiants** des entités participant à la relation.
Si la relation est porteuse de donnée, celles ci deviennent des attributs pour la nouvelle table.

S.I. :

Un étudiant parle une ou plusieurs langues avec un niveau. Chaque langue est donc parlée par 0 ou n étudiants avec un niveau. Pour chaque niveau, il y a 0 ou plusieurs étudiants qui parlent une langue.

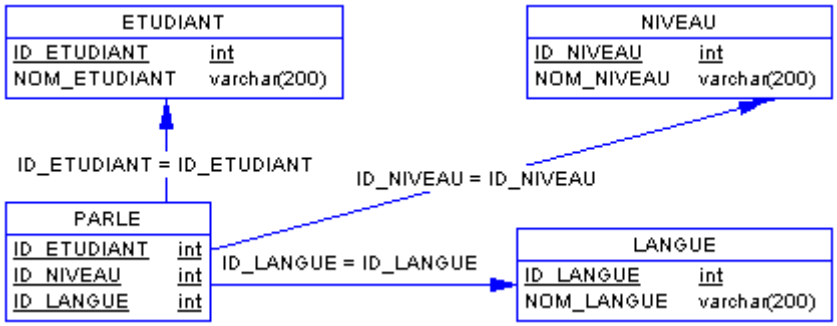
MCD :



MLDR :

ETUDIANT (id_Etudiant, Nom_Etudiant)
NIVEAU (id_Niveau, Nom_Niveau)
LANGUE (id_Langue, Nom_Langue)
PARLE (id_Etudiant, id_Niveau, id_Langue)

MPD :



5 : Association Réflexive.

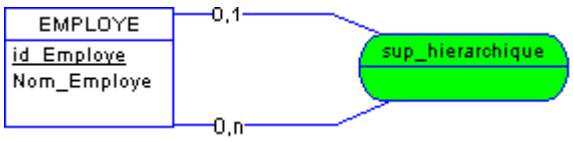
- Premier cas : cardinalité (X,1) - (X,n), avec X=0 ou X=1.

  La **Clé Primaire** de l'entité se dédouble et devient une **Clé Etrangère** dans la relation ou nouvelle table. Exactement comme si l'entité se dédoublait et était reliée par une relation binaire (X,1) - (X,n) (Cf règle 2).

  S.I. :

  Prenons l'exemple d'une société organisée de manière pyramidale : chaque employé a 0 ou 1 supérieur hiérarchique direct. Simultanément, chaque employé est le supérieur hiérarchique dir 0 ou plusieurs employés.

  MCD :



  MLDR :

  EMPLOYE (id_Employe, Nom_Employe, #id_Sup_Hierarchique)

  #id_Sup_Hierarchique est l'identifiant (id_Employe) du supérieur hiérarchique direct de l'emplo considéré.

  MPD :

-

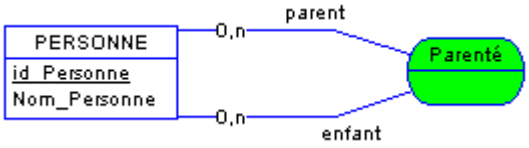
- Deuxième cas : cardinalité (X,n) - (X,n), avec X=0 ou X=1.

  De même, tout se passe exactement comme si l'entité se dédoublait et était reliée par une relation binaire (X,n) - (X,n) (Cf règle 3). Il y a donc création d'une nouvelle table.

  S.I. :

  Prenons cette fois l'exemple d'une organisation de type familiale : chaque personne a 0 ou n descendants directs (enfants), et a aussi 0 ou n descendants directs (enfants).

  MCD :

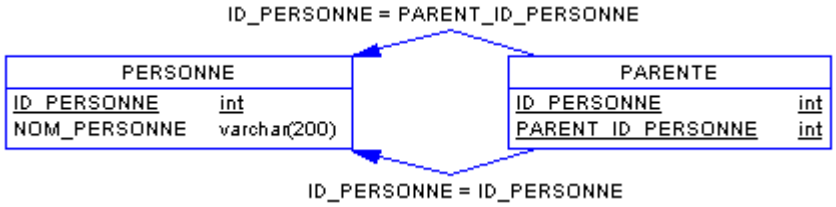


  MLDR :

  PERSONNE (id_Personne, Nom_Personne)
  PARENTE (#id_Parent, #id_Enfant)

  #id_Parent est l'identifiant (id_Personne) d'un ascendant direct de la personne. #id_Enfant est l'identifiant (id_Personne) d'un descendant direct de la personne.
  La table PARENTE sera en fait l'ensemble des couples (parents-enfants) présent dans cette fami

  MPD :



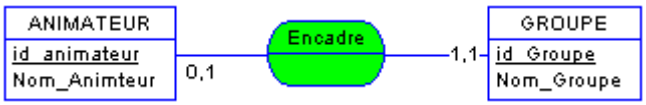
6 : Relation binaire aux cardinalités (0,1) - (1,1).

La **Clé Primaire** de la table à la cardinalité (0,1) devient une **Clé Etrangère** dans la table à la cardinalité (1,1) :

S.I. :

Dans ce centre de vacances, Chaque animateur encadre en solo 0 ou 1 groupe, chaque groupe étant encadré par un et un seul animateur.

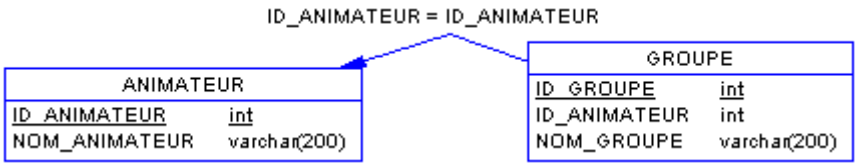
MCD :



MLDR :

ANIMATEUR (id_Animateur, Nom_Animateur)
GROUPE (id_Groupe, Nom_Groupe, #id_animateur)

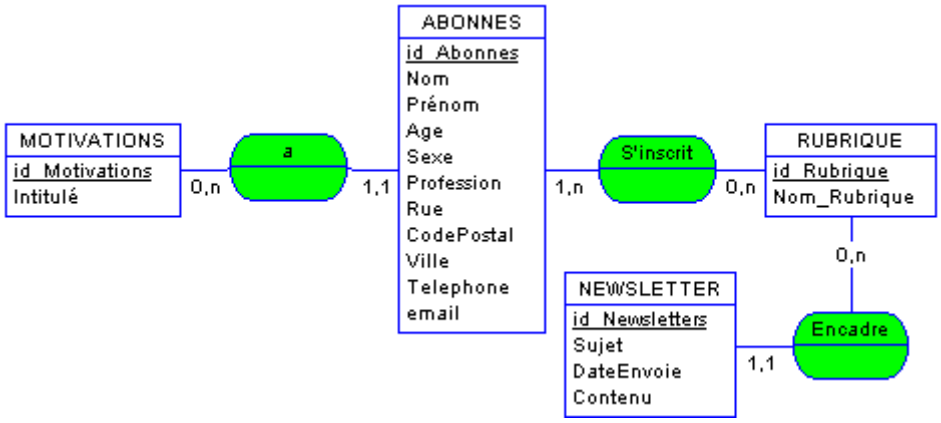
MPD :



CONCLUSION

Ces 6 règles représentent TOUS les cas que vous pourrez rencontrer. Il ne faut surtout pas se laisser impressionner par le nombre de schémas, ni se laisser intimider par le coté inhabituel du processus de modélisation. Il est très simple à acquérir. En fait, au bout de quelques modélisations et d'un ou deux développements, vous vous rendrez compte que finalement tout ceci est très logique et d'une évidence rare... Et surtout, surtout, votre base de donnée correspondra EXACTEMENT au système d'information décris dans le cahier des charges. De plus, écrire le MCD, le valider avec votre client, puis en déduire le MLDR et donc le Modèle Physique vous fera rentrer complètement dans le chantier. Vous irez ensuite beaucoup plus vite, avec très peu de risque d'être hors sujet. Après, la majorité du travail restant ne sera plus qu'une question de requètes, de mise en forme et d'ergonomie, avec une bonne gestion d'Entrée/Sortie de l'information...

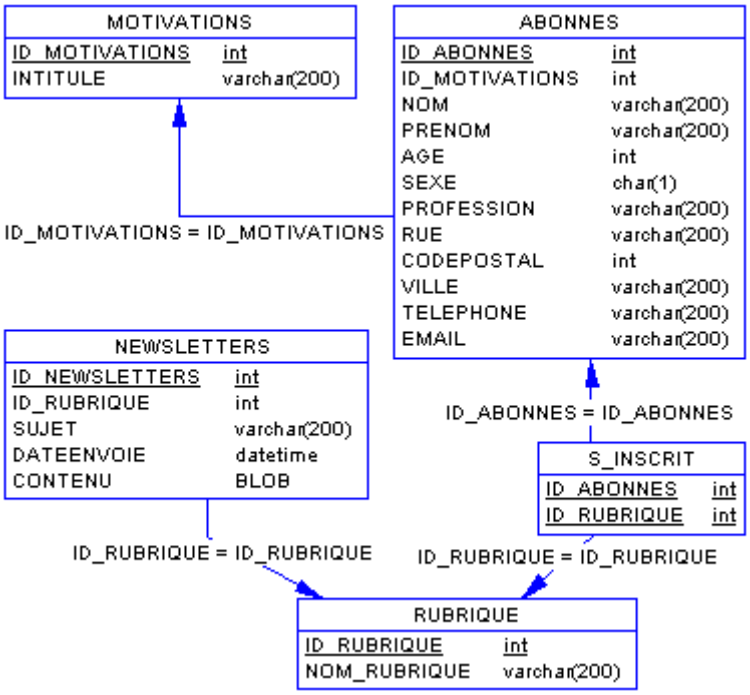
Allez, si vous êtes encore avec moi, vous avez bien mérité la fin de l'analyse de notre Newsletter du mois de décembre :



Entraîne le MLDR suivant :

MOTIVATIONS (id_Motivation, Intitule)
ABONNES (id_Abonne, #id_Motivation, Nom, Prenom, Age, Sexe, Profession, Rue, CodePostal, Ville, Telephone, Email)
S_INSCRIT (id_Abonne, id_Rubrique)
RUBRIQUES (id_Rubrique, Nom_Rubrique)
NEWSLETTERS (id_Newsletters, #id_Rubrique, Sujet, DateEnvoie, Contenu)

Qui nous mène au Modèle Physique de Donnée (MPD) ou schéma de la Base :



Merise: 3ème partie

Modéliser, c'est comprendre. Pour développer le logiciel dont les utilisateurs ont besoin, l'informaticien ne doit pas correspondre aux stéréotypes de notre imaginaire collectif. Au contraire, il lui appartient de s'ouvrir, d'aller vers les utilisateurs de son travail, de cerner quels sont leurs besoins, de discuter avec eux et de les regarder travailler. C'est ainsi qu'il cernera au mieux leurs attentes et qu'il apprendra à se mettre à la portée des utilisateurs de son travail : rien de tel qu'observer un journaliste râlant devant son interface "qui veux pas faire ce que je lui dis, euh !!!" pour se rendre compte qu'il vaut mieux se mettre à la place de l'utilisateur final afin qu'il soit satisfait de son programme. Car c'est de cette manière que l'on obtient la récompense suprême : voir un client heureux d'utiliser son nouveau logiciel, et surtout le voir travailler avec durant longtemps. Attachons-nous à ce noble objectif : après avoir commenté le MPD ou Schéma de Base de la Newsletter vue précédemment et avoir regardé ce qu'il représente véritablement, je vous proposerais un autre exemple significatif.

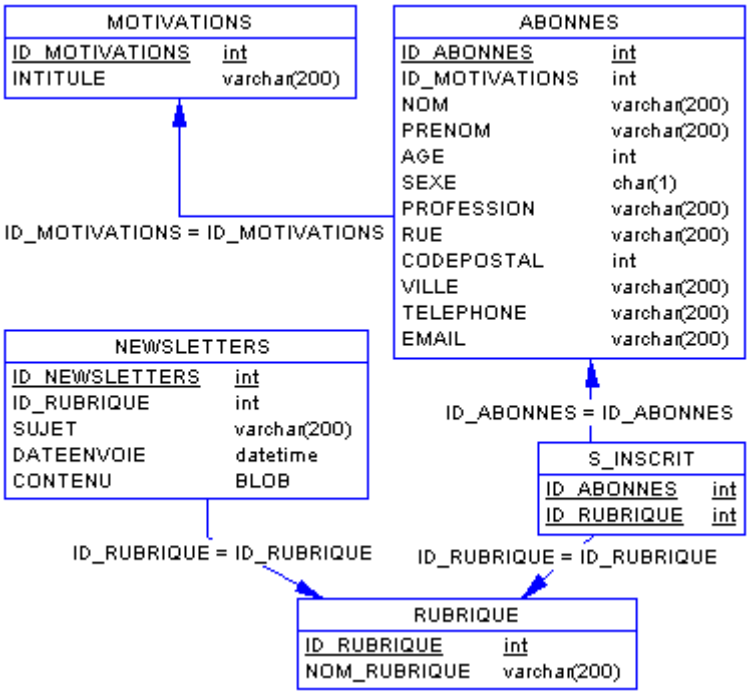
Utiliser le Modèle Physique de Donnée :

Une fois le système d'information analysé et modélisé en Modèle Conceptuel de Donnée (MCD), et après être passé par le Modèle Logique de Donnée Relationnel (MLDR), nous arrivons au Modèle Physique de Donnée (MPD). Il s'agit maintenant de créer la base correspondante à l'étude entamée. C'est à ce stade seulement que la base de donnée choisie intervient.

Le SQL (Structured Query Language), ou Langage d'Interrogation Structuré, a été reconnu en tant que norme officielle de langage de requête relationnelle par l'institut ANSI (American National Standards Institute) et par l'organisme ISO (International Standards Organization). Malgré cela, les syntaxes d'extractions des données et de créations des tables varient quelques peux d'une base à l'autre. En particulier, si la base de donnée utilisée pour le développement n'est pas véritablement relationnelle (cas de MySql dans sa version actuelle), il appartiendra au développeur de prendre lui-même en charge les limitations rencontrées, afin de s'assurer que sa base ne puisse JAMAIS être corrompue, c'est à dire contenir des données aberrantes.

APPLICATION SUR UN MODELE PHYSIQUE CONCRET :

Prenons l'exemple du schéma de base (MPD) suivant :



- La table **MOTIVATIONS** est très simple à créer : elle comporte deux champs, **ID_MOTIVATIONS** et **INTITULE**. **ID_MOTIVATIONS** est la Clé Primaire.

- **ABONNES** comporte les 12 champs du schéma. **ID_ABONNES** est la clé primaire. **ID_MOTIVATIONS** est une clé étrang7re provenant de **MOTIVATIONS**, c'est à dire que sa valeur doit être toujours égale à une valeur de **ID_MOTIVATIONS** de **MOTIVATIONS**. L'intérêt majeur des clés étrangères est surtout d'éviter les redondances, sources d'erreurs.

    - Pour les bases non totalement relationnelles : Il appartiendra au développeur de vérifier lors de chaque insertion dans **ABONNES** que l'**ID_MOTIVATIONS** fournis fais partie des valeurs existantes de **ID_MOTIVATIONS** de **MOTIVATIONS**. De même, lors de chaque suppression d'un enregistrement de **MOTIVATIONS**, il faudra vérifier qu'aucun enregistrement d'**ABONNES** n'utilise la valeur d'**ID_MOTIVATION** correspondante.

- **S_INSCRIT** comporte deux champs, **ID_ABONNES** et **ID_RUBRIQUE**. **ID_ABONNES** et **ID_RUBRIQUE** sont clé primaire de **S_INSCRIT** : **S_INSCRIT** a comme clé primaire la concaténation de ces deux champs. C'est à dire que tout couple (**ID_ABONNES**,**ID_RUBRIQUE**) de **S_INSCRIT** est unique. **ID_ABONNES** est aussi clé étrangère de **ABONNES** dans **S_INSCRIT**, et **ID_RUBRIQUE** est clé étrangère de **RUBRIQUE** dans **S_INSCRIT**.
  Une telle table est communément appelée "**Table de Lien**". L'intérêt d'une telle table est que pour chaque **ID_ABONNES** donné, il est aisé de retrouver tous les **ID_RUBRIQUE** associés, et vice et versa.

    - Pour les bases non totalement relationnelles : Il faudra vérifier lors de chaque insertion dans **S_INSCRIT** que le couple (**ID_ABONNES**,**ID_RUBRIQUE**) n'existe pas déjà dans la table **S_INSCRIT**, que **ID_ABONNES** existe dans **ABONNES** et que **ID_RUBRIQUE** existe dans **RUBRIQUE**. De même, pour chaque suppression d'un abonné, il faudra supprimer tous les couples (**ID_ABONNES**,**ID_RUBRIQUE**) ayant l'**ID_ABONNE** correspondant. Pareil pour toute suppression de **RUBRIQUE**.

- **RUBRIQUE** est elle aussi très simple à créer : elle comporte deux champs, **ID_RUBRIQUE** et **NOM_RUBRIQUE**. **ID_RUBRIQUE** est la Clé Primaire.

- **NEWSLETTERS** comprend les 5 champs du schéma. **ID_NEWSLETTER** est la clé primaire. **ID_RUBRIQUE** est une clé étrangère provenant de **RUBRIQUE**.

    - Pour les bases non totalement relationnelles : Il faudra vérifier lors de chaque insertion dans **NEWSLETTER** que **ID_RUBRIQUE** existe dans **RUBRIQUE**. De plus, pour chaque suppression d'une rubrique, il faudra s'interroger sur le sort réservé à chaque newsletter de cette rubrique : les détruire ou les archiver.

APPLICATIONS AUX BASES RELATIONNELLES :

Les vérifications détaillées précédemment n'ont lieu que pour assurer la cohérence de la base. Il est donc logique, si celle ci le permet, de déléguer et d'automatiser ces taches au niveau ce celle-ci. Généralement, les vérifications afférentes à une clé étrangère sont confiées à un Trigger (un Trigger est un ensemble d'instruction SQL s'effectuant avant ou après un événement donné, par exemple une insertion ou une suppression). Ainsi, lors de chaque commande d'insertion sur la table désignée au Trigger préalablement correctement programmé, celui ci va vérifier AVANT l'insertion que la clé étrangère est valable. Dans le cas ou elle ne le serait pas, le Trigger renvoie un message d'erreur et l'insertion ne s'effectue pas, évitant ainsi de corrompre la base. De même, certains traitements automatisés pourront être réalisés directement à l'aide de procédures stockées. Exemple : un devis validé qui entraîne la création de la facture correspondante. Et surtout, les Trigger et Procédures Stockées étant compilées directement par la Base de Donnée, leur exécution est beaucoup plus rapide qu'une série d'instruction SQL envoyées par le programme attaquant la base.

Une base de donnée correctement pensée est à envisager comme un contenant d'information "vivant", forcement cohérent, aux réactions automatisées. Une telle base se suffirait presque à elle-même pour gérer un Système d'Information. Le développement ne consisterait alors plus qu'à afficher son contenu en temps réel, et à fournir les outils d'insertion appropriés. Le rêve...

SECOND EXEMPLE : MODELISER UN DOCUMENT

Il est courant, lors du développement d'un site Web ou de l'informatisation d'un système d'information, de démarrer son analyse par un document. Captures d'écrans, photocopies, sont parfois les principales pièces jointes à la demande de devis, accompagnés du commentaire suivant :
"Je veux faire ça !!!". Bien. Alors, faisons ça...

Système d'Information :

L'entreprise "WebCash" de vente par correspondance désire ajouter à son site un système de consultation de factures visible en ligne pour ses clients. Chaque client, après authentification, pourra accéder à toutes les factures le concernant, qu'elles soient anciennes ou en cours de traitement indifféremment. Pour être sur de bien se faire comprendre, "WebCash" fournis une copie d'une facture type en disant :
"C'est ça qu'on veut sur l'écran !"

Voici une copie de cette facture :

WebCash S.A.R.L **FACTURE N° 12345**
24, Avenue des Rêves roses
75008 PARIS

Paris, le 15/10/2000

Nom :	BIDOCH
Prénom :	Robert
Adresse :	12, rue du centre
Code Postal :	70000
Ville :	Gray

N° Article	Libellé	Prix Unitaire	Quantité	Prix
234	Stylo Plume	12.5 F	1	12.50 F
568	Couteau Suisse	75.00 F	2	150 F
132	Serviette	30.00 F	1	30.00 F
TOTAL TTC :				192.50 F
Dont TVA 19.6% :				37.73 F

A PAYER : **192.50 F**

Avec nos plus cordiaux remerciements

Voilà, tous les éléments sont réunis. Il ne reste plus qu'à concevoir la Base de Donnée se cachant derrière cette innocente petite facture. Cet exemple est très conforme à la réalité. Il sera très intéressant à étudier, car il permettra d'expliquer un certain nombre de points, et de mettre en évidence certaines erreurs à ne pas commettre. N'hésitez pas à prendre le stylo et à vous entraîner, je vous fournirais une solution commentée la prochaine fois.

A bientôt...

Merise: 4 ème partie

Dans la famille "Je cherche à comprendre le Système d'Information que je modélise", je propose les documents fournis par votre client. Car s'il sera aisé de discuter avec lui de l'exactitude du MCD, comprendre son environnement de travail peut parfois être plus ou moins évident : ses explications seront rarement complètements explicites du premier coup, et il est courant que le client lui-même n'ai qu'une vague idée de ce qu'il veut faire. C'est là que, armée de toute notre diplomatie et de notre pédagogie, commence la partie la plus palpitante de notre travail : "Comprendre ce que veux le client, et comment ça marche". Car vous n'avez pas envie de recommencer 10 fois votre travail et de modifier votre code tout le temps, n'est ce pas ? Il va donc falloir poser des questions, aller voir comment il se débrouille actuellement, amasser un maximum de documents, et en retirer le maximum d'informations. Car être certain de comprendre les besoins de l'utilisateur qu'il exprime avec ses mots à lui, c'est là tout le but de la modélisation.

MODELISER UN DOCUMENT : UNE PRATIQUE COURANTE

Commençons par terminer l'exemple de la dernière fois, dont re-voici l'énoncé :

Système d'Information :

L'entreprise "WebCash" de vente par correspondance désire ajouter à son site un système de facturation visible en ligne pour ses clients. Chaque client, après authentification, pourra accéder à toutes les factures le concernant, qu'elles soient anciennes ou en cours de traitement indifféremment. Pour être sur de bien se faire comprendre, "WebCash" fournis une facture type en disant : "C'est ça qu'on veut sur l'écran !"

Voici une copie de cette facture :

WebCash S.A.R.L **FACTURE N° 12345**
24, Avenue des Rêves roses
75008 PARIS

Paris, le 15/10/2000

Nom :	BIDOCH
Prénom :	Robert
Adresse :	12, rue du centre
Code Postal :	70000
Ville :	Gray

N° Article	Libellé	Prix Unitaire	Quantité	Prix
234	Stylo Plume	12.5 F	1	12.50 F
568	Couteau Suisse	75.00 F	2	150 F
132	Serviette	30.00 F	1	30.00 F
TOTAL TTC :				192.50 F

Dont TVA 19.6% :	37.73 F
A PAYER :	**192.50 F**

Avec nos plus cordiaux remerciements

APPLICATION DE LA METHODE MERISE

Elle consiste à construire le Modèle Conceptuel de Donnée (MCD), Générer le Modèle Logique de Données Relationnelles (MLDR), et le transposer en Modèle Physique de Donnée (MPD).

- **Construire le Modèle Conceptuel de Donnée (MCD) :**

La méthode est toujours la même : Identifier les entités présentes, Lister les propriétés des entités, Identifier de manière unique chaque occurrence, Etablir les relations entre les différentes entités, et Identifier les cardinalités.

- Identifier les entités présentes :
  On relève trois entités : **CLIENT**, **FACTURE**, **ARTICLE**.
  - **CLIENT** est l'ensemble des clients de la société WebCash. Une occurrence de cette entité est présentée par Robert BIDOCH, qui est le client à qui cette facture est destinée.
  - **FACTURE** est l'ensemble des factures émises par WebCash, dont une occurrence est présente en "FACTURE N° 12345".
  - **ARTICLE** est l'ensemble des articles vendus par WebCash, dont trois occurrences sont présentes, dénommés Stylo Plume, Couteau Suisse et Serviette.
  - Une facture étant composée de plusieurs lignes, il aurait été possible de relever l'entité LIGNE_FACTURE : elle n'est utile que si l'on désire archiver pour chaque ligne son Numéro. La base déduite aurait été sensiblement la même, démontrant ainsi que plusieurs solutions sont parfois possibles.
  - La TVA est ici considérée comme constante et unique. Dans le cas contraire, elle aurait représenté l'entité TVA.
  - La société WebCash ne représente pas une occurrence d'une entité, car c'est la seule société éméttrice de facture de notre analyse.

- Lister les propriétés des entités :
  - Un **CLIENT** est caractérisé par son Nom, son Prénom, son Adresse, son CodePostal et sa Localité. Afin de pouvoir s'authentifier, il est aussi caractérisé par un Login et un Passwd.
  - Une **FACTURE** est caractérisée par son Numéro, et sa Date d'émission.
  - Un **ARTICLE** est caractérisé par son Numéro, son libellé, et son PrixUnitaire. Le prix total, de par son PrixUnitaire et sa quantité, peux être recalculé : ce n'est donc pas une caractéristique de l'**ARTICLE**.
  - Chaque propriété doit avoir une seule valeur possible pour chaque occurrence, ce qui est ici le cas. Elle doit de plus être élémentaire et non-décomposable, ce qui est aussi le cas. D'une manière générale, toute information résultant d'un calcul n'est pas une caractéristique d'une entité.

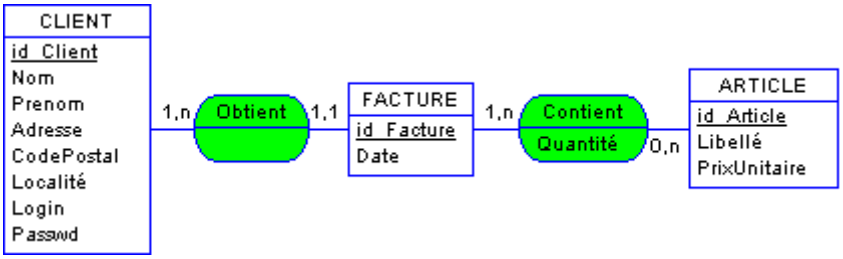
- Identifier de manière unique chaque occurrence :
  chaque occurrence de chaque entité doit pouvoir être identifiée de manière unique : cette propriété s'appele l'identifiant.
  - Un **CLIENT** sera identifié par un Numéro unique, cette caractéristique de l'entité étant appelé id_Client.
  - Une **FACTURE** sera identifiée par son Numéro qui est unique. Cette caractéristique sera appelée id_Facture.
  - Un **ARTICLE** sera identifié par son Numéro qui est lui aussi unique. Cette

caractéristique sera appelée id_Article.

- Etablir les relations entre les différentes entités :
  Un **CLIENT** obtient une **FACTURE** qui contient des **ARTICLES** en certaine quantité.

- Identifier les cardinalités :
  - Un même **CLIENT** obtient 1 ou plusieurs **FACTURE**.
  - Une même **FACTURE** est obtenue par un seul **CLIENT**.
  - Une même **FACTURE** contient 1 ou plusieurs **ARTICLE**.
  - Un **ARTICLE** est contenu dans 0 ou n **FACTURE**.

On en déduit donc le MCD suivant :



Comme d'habitude, il est alors temps de retourner voir le client et de discuter le Modèle avec lui, afin de vérifier qu'il ne manque rien et que l'analyse correspond bien à SA réalité de travail. Après validation, il est temps de passer à l'étape suivante.

- **Générer le Modèle Logique de Données Relationnelles (MLDR) :**

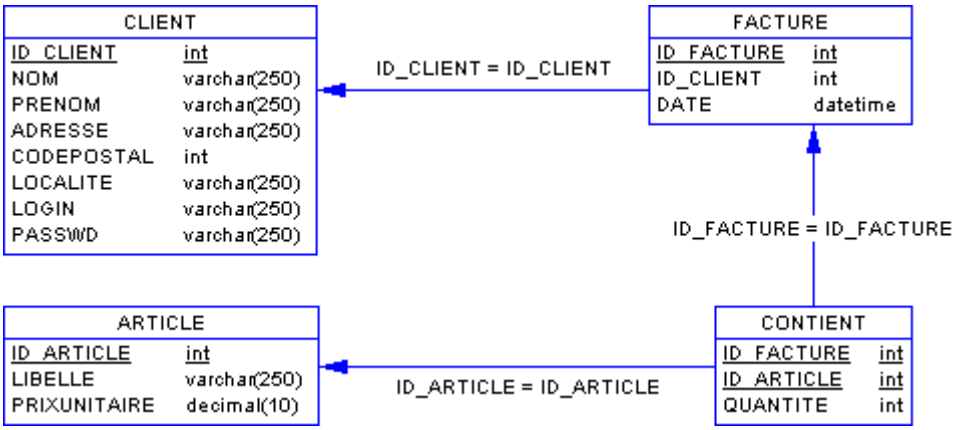
Relation (X,1)-(X,n) entre **FACTURE** et **CLIENT** :

CLIENT (id_Client, Nom, Prenom, Adresse, CodePostal, Localite, Login, Passwd)
FACTURE(id_Facture, #id_Client, Date)

Relation (X,n)-(X,n) entre **FACTURE** et **ARTICLE** :

CONTIENT(#id_Facture, #id_Article, Quantite)
ARTICLE(id_Article, Libelle, PrixUnitaire)

- **Transposer en Modèle Physique de Donnée (MPD) :**



Voilà, il ne reste plus qu'à créer la base, à la remplir, et à développer dessus.

UNE BASE DE DONNEE COHERENTE

Un tel modèle est dit **cohérent**, c'est à dire que pour chaque donnée fournie, il permet de retrouver toutes les informations s'y rattachant.

Pour chaque client donné, il sera possible d'avoir toutes ses factures, et donc tous les articles qu'il a achetés et en quelle quantité. Pour chaque facture, il sera possible de retrouver le client correspondant, ainsi que la liste des articles et leurs quantités respectives. Enfin, pour chaque article, il sera aisé de retrouver les quantités vendues, à quelle date et à quels clients.

La base ne contient aucune **redondance**, c'est à dire qu'aucune information présente ne peut être déduite d'autres informations présentes. Ceci évite grandement le risque de **corruption**, ou présence de donnée aberrante.

Les prix intermédiaires, la somme totale et autres résultats sont des **traitements**, c'est à dire qu'ils sont calculés par rapport aux informations contenues dans la base. Ainsi, toute erreur sera forcemment une erreur de calcul, et non pas une erreur de stockage. Il est plus facile de vérifier un programme d'extraction de donnée que de vérifier la cohérence du contenu d'une base.

LES AVANTAGES D'UN TEL RESULTAT

Le développement se réduit maintenant à une interface d'administration (BackOffice) ou WebCash pourra ajouter/modifier/supprimer ses **ARTICLES**, rentrer ses **FACTURES**, et consulter son fichier **CLIENT**. Ensuite, sur le site lui-même (FrontOffice), il ne reste plus qu'à faire le formulaire d'accréditation du **CLIENT** (Login/Passwd) qui permettra de retrouver son identifiant, puis à lui lister les **FACTURE** correspondant à cet identifiant, et enfin lui afficher le détail de celle qu'il aura sélectionné.

Si ce développement paraît au final aussi simple, c'est qu'il s'appuie sur une base bien pensée. Si celle-ci correspond effectivement à l'environnement de travail de WebCash, il n'y aura aucune raison de la changer.

Vous obtiendrez ainsi la meilleure des références et des publicités : concevoir un outil simple à utiliser qui marche durant longtemps sans avoir à être modifié. Et ça, pour un client, c'est vraiment le top du top.

A bientôt...

Merise: 5ème Partie

Vous êtes encore nombreux à ne pas être vraiment convaincus de la nécessité de passer par la phase d'analyse avant de commencer réellement à coder. Peut-être est-ce due à cette impression qu'a souvent l'autodidacte (ou le débutant) que ce travail n'est pas réellement rentable. De même, un client (ou un patron) peut avoir le sentiment de jauger l'évolution d'un travail au nombre de lignes écrites, à la rapidité ou les pages lui sont fournies, et donc à

ce qu'il voit. Après être revenus sur ces phénomènes très répandus, je vous présenterais une introduction à une analyse beaucoup plus conséquente, analyse qui a servie de base à un logiciel gérant la coupe du monde en France en 1998. Car quand les choses se compliquent un tout petit peu (et cela arrive très très vite, et de plus en plus couramment), il n'est plus question d'amateurisme sous peine de risquer, cette fois, le naufrage total...

"Vite ! Pas Chère ! Et tout de suite."

Comme tout un chacun, l'investisseur (client, directeur) raisonne très souvent à court terme et exige rapidement des preuves que l'argent qu'il investit est correctement utilisé, et que le temps de travail qu'il paye est rentabilisé au maximum. Lui expliquer que quelques jours ou quelques heures de réflexions sont parfois nécessaires pour l'analyse peut le plonger dans un état suspicieux, avec parfois même le sentiment d'être trompé sur le temps de travail facturé. Pourtant, cette phase préliminaire représente les fondations même de son projet : il vaut mieux qu'il ait son site quelques jours plus tard mais que celui-ci soit correctement pensé et développé de manière professionnelle. Ainsi, son bon fonctionnement, son évolutivité et sa maintenance rentabiliseront largement son très léger investissement.

"Analyser ? Réfléchir ?? Mais pour quoi faire ???"

Un autre phénomène très répandu est celui du "bidouilleur" : régulièrement autodidacte, et généralement brouillon, il pense être capable de tout faire en mettant lui-même la main à la pâte. Très souvent, il s'agit d'un commercial persuadé qu'il pourra faire seul ce qui lui semble long et cher sans raison. Dans un premier temps, il semblera y arriver, à force de persévérances et d'interventions qui lui paraîtront être de petites astuces très intelligentes et qu'il ajoutera ici et là, très content de lui-même et de son résultat immédiat. Mais cela se termine toujours de la même manière : son code et ses fichiers se révelent être devenus si foullis et si incompréhensibles qu'il finira par être incapable de faire fonctionner quoi que ce soit, et devra se résoudre à appeler au secours. Seulement, il est souvent trop tard, et la seule solution viable pour son site Web est de le re-développer entièrement et correctement. Et là, ça coûte beaucoup plus cher que quelques heures d'analyse ou les conseils d'un professionnel.

Improviser : élimination directe ?

Prenons un nouvel exemple un plus complexe : en 1998, a eu lieu en France la coupe du monde de Football. De très nombreux sites Web ont alors entamé des développements spécifiques, afin de présenter un certain nombre d'informations concernant la compétition aux internautes. Leur besoin a tout d'abord été de fournir à leurs journalistes des outils rédactionnels afin de leur permettre d'afficher en ligne les articles concernant la compétition. La plupart de ces sites étant déjà dotés d'interfaces de travail pour les autres secteurs d'actualités, cette tache s'est généralement révélée assez simple. Mais, pour ce qui concernait le stockage et la présentation des résultats et des statistiques, ce fut une tout autre histoire...

Extraits d'un petit briefing de l'équipe technique :

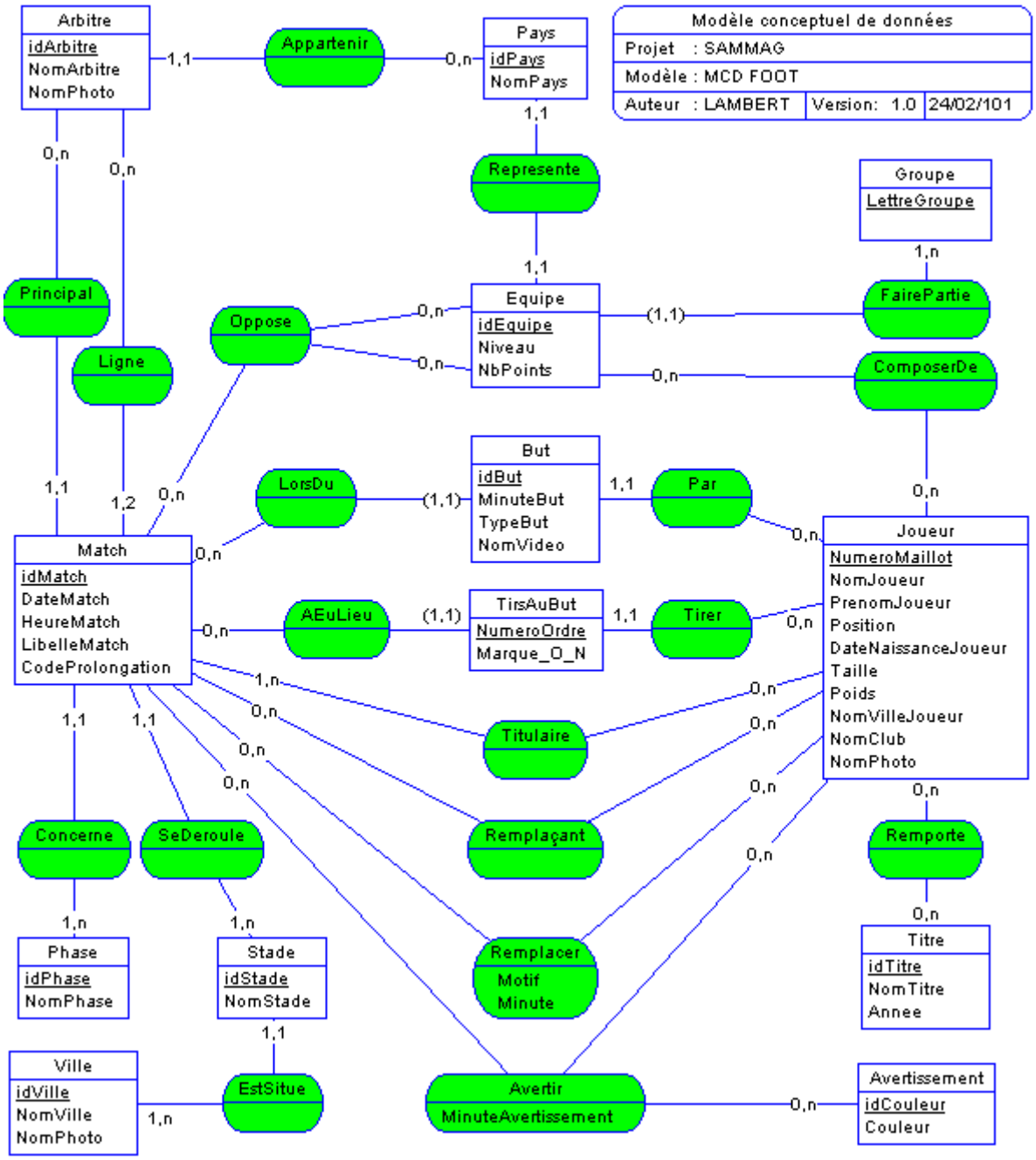
"Ecoute, le webmaster, vient voir : Là, il y a la coupe du monde qui commence la semaine prochaine. On va présenter les scores des matchs, et deux trois petits classements faciles. Tu nous fais un machin hyper-simple pour qu'on puisse faire un max d'audience avec tous les footeux. Tu vois, un petit programme où là, on rentre les buts et les cartons, et quand on clique ici, sur le site, on voit tout comme il faut. Comme ça, les journalistes, ils marquent dans une petite fenêtre que le match a eu lieu ici, avec les joueurs qui jouent, les buteurs qui marquent, les remplacés qui sortent et le type qui arbitre. Comme dans le journal, quoi.... Toi, tu leur fais gentiment une moulinette pour stocker tout ça et tout afficher tranquillement sur le site quand le surfeur il le demande. OK ? Ca va ? Tu vois, un truc comme ça, tranquille..."

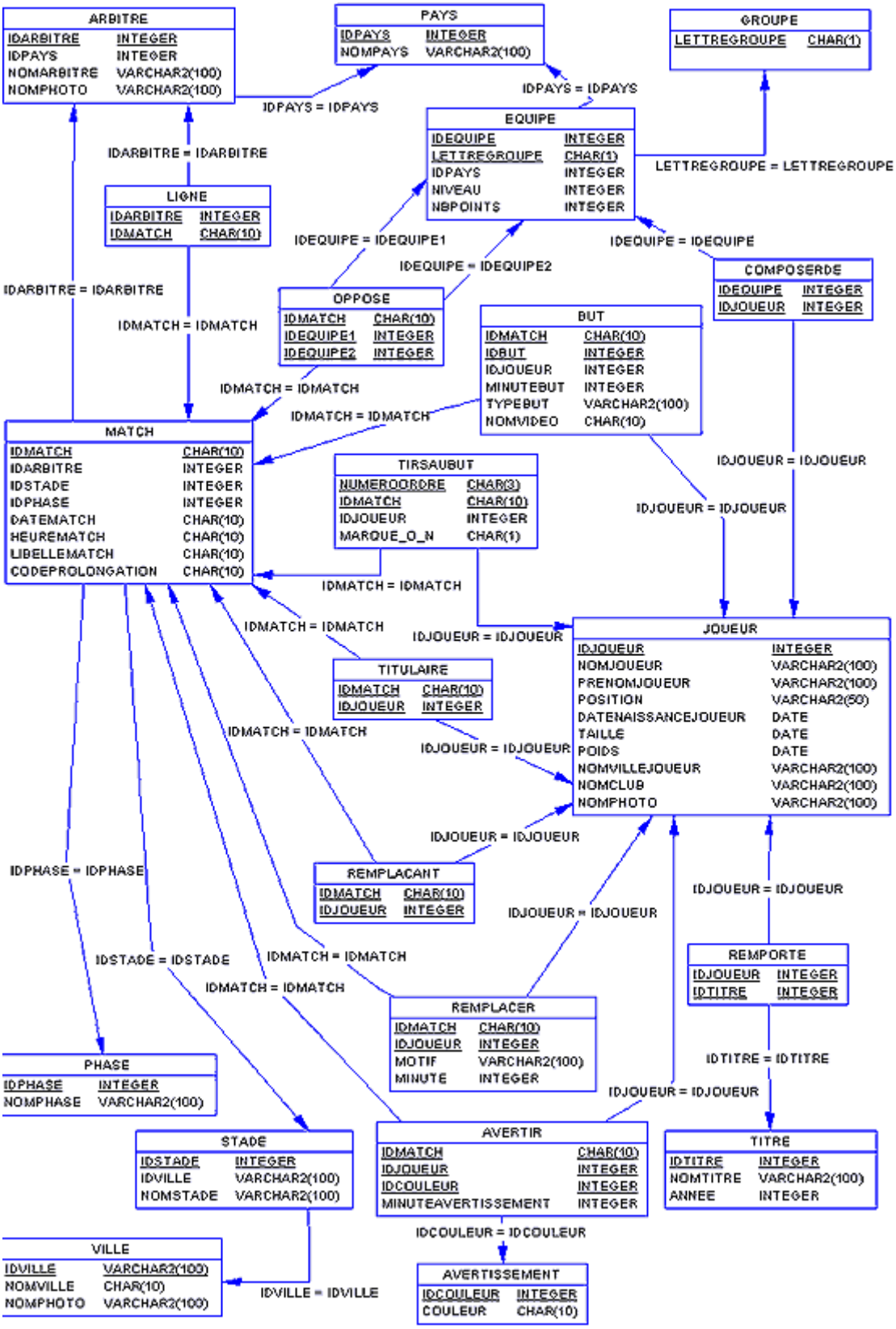
Eh ben c'était pas gagné...

Tacle en retard : carton Rouge !

Je ne sais pas si vous vous souvenez du spectacle alors offert par certains des sites Web concernés : joueurs ayant marqué 327 fois en 4 matchs, Barthez dans l'équipe du Brésil, Zidane finaliste en totalisant 23 minutes de jeux sur le terrain, résultats farfelus, quand résultats tout court il y eut. Car bien souvent, le développement se résuma en dernier ressort à afficher en catastrophe des pages statiques avec les feuilles de match, les classements et les résultats écrits directement en dur, dans des pages HTML fixes. Il y eu même des sites très connus (qu'on ne citera pas) qui furent totalement incapables de présenter le moindre résultat avant la fin de l'épreuve !

Pour ceux qui seraient interressés, j'ai concocté une petite analyse sur cette épreuve. Bien évidemment, selon les informations que l'on voudra stocker ou présenter, certaines modifications peuvent être apportées. Il ne faut pas perdre de vue que dans ce type d'analyse, il n'y a pas UNE mais bien souvent plusieurs solutions. Tout dépend de ce que l'on veut réellement faire. Néanmoins, cela devrait tout de même ressembler à cela :





Bien gérer les individualités

Un petit mot sur ce MCD et ce MPD :

- On remarque tout d'abord les cardinalités (1,1) mises entre parenthèses : elles décrivent ce que

l'on appelle des entités dépendantes. En effet, une **Equipe** est liée à son **Groupe**, de même qu'un **But** et un **TirsAuBut** ne peuvent exister sans le **Match** lors duquel ils ont été marqués. Cela s'appelle un identifiant relatif. La conséquence d'un identifiant relatif est que la table générée par l'entité a comme double clé primaire son propre identifiant ainsi que l'indentifiant de l'entité dont elle dépend.

- Le champ **NBPOINTS** de la table **EQUIPE** : ce renseignement pourrait être retrouvé par une requête appropriée prenant en compte les résultats des matchs joués précédemments. Néanmoins, le nombre de points pouvant être modifié sur tapis vert, il est judicieux de prévoir qu'il puisse ne pas correspondre aux résultats précédents. Il ne s'agit donc pas d'une redondance, un facteur externe pouvant intervenir et modifier sa valeur. Cela implique tout de même que le développeur devra prévoir à la fois de le mettre à jour lors de la saisie des résultats (par procédures stockées ou par une fonction de son script), et une interface permettant d'y accéder et d'y écrire directement.

- le champ **CODEPROLONGATION** de la table **MATCH** sert à savoir si un match a eu ou non des prolongations, renseignements non indiqués par tous les score.

- Il y a une double relation 1,1 entre les entités **Equipe** et **Pays**. Normalement, cela voudrait dire que **Pays** est un argument de Equipe, et ne devrait pas devenir une table. Mais ici, **Pays** sert aussi pour les **Arbitres**, et est donc une entité reliée à **Arbitre** par une relation (1,1)-(0,n). Cette légère disgression se trouve donc justifiée par la liaison **Pays**-**Arbitre**.

Cela paraît compliqué ? Pourtant, il n'y a que 21 tables. Certains systèmes d'informations sont bien plus complexes que cela, il n'est pas rare qu'une boutique complète repose sur une base de données de près de 70 tables. Mais déjà, là, il n'est plus question d'improviser et de créer ses tables au hasard, sous peine de se retrouver coincé et d'avoir rapidement de très gros soucis. Avec un poil d'expérience, une telle analyse peut être réalisée en quelques jours. Comme d'habitude, si elle correspond effectivement au besoin du client, le développement se résumera ensuite à une interface d'administration pour remplir correctement la base, puis à un moteur d'affichage à base de requêtes pour en afficher le contenu.

Ne pas trop exagérer

Il conviendra tout de même de ne pas tomber dans l'excès inverse : l'analyse est là pour nous faire gagner du temps, pas pour nous en faire perdre. Merise, Gare Yourdon et les autres sont des méthodes très complètes qui peuvent néanmoins se révéler excessivement lourdes et contraignantes. Il n'est pas question ici d'utiliser au pied de la lettre ces procédés dans leur intégralité. Mais les techniques du Client/Serveur sont en train d'arriver sur le Web, les sites dynamiques se compliquent et se professionalisent. Et la méthode que je vous propose dans mes articles est simple, toujours vrai, rapide, efficace, et vous rendra donc de fiers services.

A bientôt...

Merise: 6ème Partie

L'actualité nous le prouve une fois encore : ce n'est pas parce que quelque chose est nouveau que les principes fondamentaux ne sont plus valables. De même qu'une entreprise est tributaire de son chiffre d'affaire et de sa rentabilité, un développement informatique se juge selon un minimum de règles élémentaires. On pourrait déjà citer : analyse, cohérence de conception, modularité, lisibilité, et utilisation censée des ressources matérielles et logicielles. Dans cette optique, construire une belle base de donnée bien pensée, cohérente, fonctionnelle, est déjà un premier pas. L'étape suivante est de savoir comment la remplir, et surtout comment lire les données qu'elle contient. Car les requêtes SQL pour interroger la base peuvent être très gourmandes en ressources systèmes et temps machine. Alors autant les formuler correctement afin de ne pas surcharger le serveur et donc d'afficher les pages rapidement. Vous pensez connaître la fonction "SELECT", la plus simple et la plus utilisé ? Bien, alors vérifions cela...

SECTEURS NOUVEAUX ? FIASCOS IDIOTS...

Après avoir vécu plusieurs années euphoriques, certaines Start Up Internet sont actuellement à la peine.

Car même si le marché des nouvelles technologies est fabuleux, les perspectives de croissances phénoménales, les investisseurs ont perdu de vue la notions la plus fondamentale de l'économie : une entreprise doit être rentable. Et bizarrement, ce point était absent de très nombreux Business Plan, Businness Plan ayant permis à ne nombreuses sociétés de lever des sommes astronomiques. Certaines d'entre elles n'existaient et ne communiquaient que pour plaire à leurs investisseurs, et n'avaient même aucune activité commerciale.

La réaction des financiers est impitoyable : la majorité des investisseurs stoppent maintenant leurs investissements, et demandent à leurs entreprises d'être immédiatement rentables. Tout de suite. Et, comme nous sommes de bon montons de Panurges, les valeurs boursières baissent, des projets très prometteurs ne trouvent plus preneurs, et les décideurs affichent une frilosité exagérée. On oublie que de nombreuses entreprises font déjà d'énormes bénéfices, que le commerce électronique affiche de très bons chiffres pour l'an 2000, et que ceux ci vont tripler dans les deux ans. Que nous sommes en train de transposer tout notre modèle économique, notre façon de travailler, et d'organiser différemment notre société. Et surtout que c'est un sacré chantier qui va générer une immense activité...

METIERS NOUVEAUX ? SOYONS PROS !!!

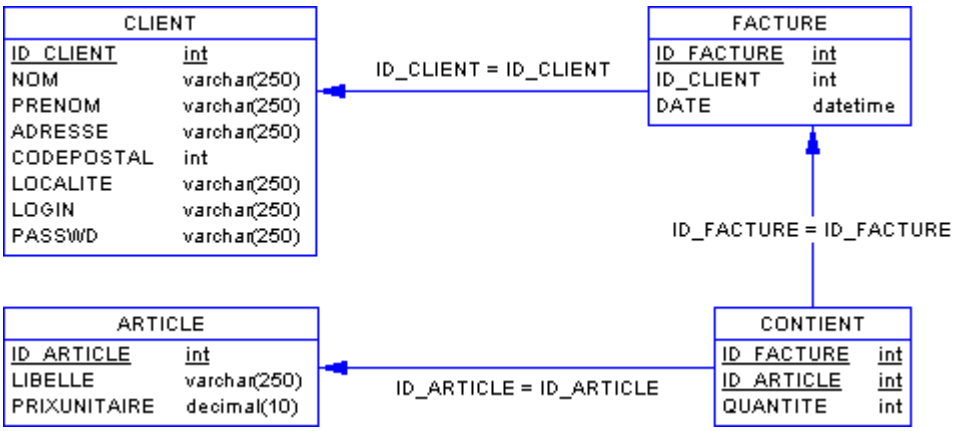
L'analogie avec le développement est évidente : Internet a permis à tout un chacun d'ouvrir un espace chez un hébergeur, et à l'aide de livres devenus très accessibles et facilement disponibles de commencer ses propres réalisations. Intellectuellement, culturellement, ce fut une nouveauté fabuleuse. Dans le même temps, les informaticiens traditionnels n'ont pas toujours pris au sérieux ces nouvelles techniques "avec des images qui bougent", et que "tout le monde touche". C'est pour cela que nous trouvons beaucoup d'autodidactes dans les équipes techniques Web. Même si les informaticiens issus de formations classiques sont maintenant de plus en plus présent au sein des équipes de développements Internet, il est important de professionnaliser les méthodes de travail. De cesser l'amateurisme, de réapprendre l'analyse et le génie logiciel. En bref, de devenir hyper professionnel afin de ne pas faire l'erreur de se décrédibiliser quand viendra pour nous aussi l'heure du bilan technique du travail réalisé ces dernières années. Car nous aussi, nous serons jugés, en quelque sorte, quand nos développements seront repris pour être adaptés aux nouveaux besoins de l'entreprise, ou convertis aux technologies futures :

Ce n'est pas parce qu'un domaine est nouveau, que les principes fondamentaux ne sont plus valables.

LANGAGE SQL : SYNTAXE D'EXTRACTION DES DONNEES

Le langage SQL (Structured Query Langage, ou Langage d'Interrogation Structuré) date de la fin des annés 70. (voir article Un peu d'histoire des SGBD sur le web et ailleurs du 13 Novembre 2000). Nous allons déjà commencer par voir comment lire les données, et ce que cela signifie réellement.

Appuyons-nous sur ce schéma de base :



Syntaxe de la commande SELECT :
SELECT [ALL | DISTINCT] liste de colonnes
FROM Table_1, table_2, ... , Table_n
WHERE ……
GROUP BY ……
HAVING ……..
ORDER BY …….
COMPUTE …….

Exemples simples :
SELECT NOM, PRENOM **FROM** CLIENT
renvoie le contenu des colonnes NOM et PRENOM de la table CLIENT.

SELECT NOM as lastname, PRENOM as firstname **FROM** CLIENT
renverra le même contenu, mais en renommant le nom des colonnes de sorties.

SELECT LIBELLE, PRIXUNITAIRE * 0.85 as prixpromo **FROM** ARTICLE
affichera le nom des articles et le prix diminué de 15%. Il est donc possible d'effectuer des calculs directement au niveau de la base de donnée, ce qui évitera de faire appel à une partie de script spécialement développée pour l'occasion.

Détail des clauses optionnelles :

La clause **WHERE** exprime les conditions de recherches.
La clause **GROUP BY** groupe les résultats par critères.
La clause **HAVING** permet de restreindre les résultats en imposant une propriété.
La clause **ORDER BY** trie les résultats suivant un critère de qualification.
La clause **COMPUTE** génère des statistiques.

LANGAGE SQL : LA CLAUSE WHERE

Les conditions de recherches peuvent faire référence à des colonnes qui ne sont pas incluse dans la liste de sélection. Elle est très utilisée pour définir ce que l'on appèle des jointures :
SELECT CLIENT.NOM, CLIENT.PRENOM, FACTURE.DATE
FROM CLIENT, FACTURE
WHERE CLIENT.ID_CLIENT=FACTURE.ID_CLIENT
renverra le nom et le prénom des clients ayant des factures, ainsi que la/les date des factures correspondantes. Il est important de faire une jointure sur ID_CLIENT dans la clause Where, pour indiquer au moteur SQL que la liaison entre les deux tables s'effectue sur cette colonne 'commune'. (Nous reviendrons dans un prochain article sur le fonctionnement d'un moteur SQL).

Nous avons plusieurs outils pour les types de condition :

Opérateur de comparaison : = , > , < , >= , <=, <>
SELECT LIBELLE, PRIXUNITAIRE
FROM ARTICLE
WHERE PRIXUNITAIRE <>'100'
renverra les articles dont le prix est différent de 100. Il est important de mettre les côtes ' ' dans la clause Where, car sinon le moteur SQL interprétera 100 comme étant une colonne.

Intervalles : BETWEEN , NOT BETWEEN
SELECT LIBELLE, PRIXUNITAIRE
FROM ARTICLE
WHERE PRIXUNITAIRE BETWEEN '100' AND '200'
renverra les articles dont le prix est compris entre 100 et 200.

Listes : IN , NOT IN
SELECT LIBELLE, PRIXUNITAIRE
FROM ARTICLE
WHERE PRIXUNITAIRE IN ('100','200','300)
renverra les articles dont le prix est soit de 100, soit de 200, soit de 300.

Correspondances de chaînes : LIKE , NOT LIKE
Quelques jokers :

- **%** joker : toute chaîne de caractère
- **-** joker simple: un caractère, n'importe lequel
- **[ ]** un caractère de l'intervalle ou de l'ensemble spécifié
- **[^]** un caractère en dehors de l'intervalle ou de l'ensemble spécifié

SELECT LIBELLE, PRIXUNITAIRE
FROM ARTICLE

WHERE LIBELLE LIKE '%teu%'
renverra les articles dont le nom contient 'teu' (pelleteuse, par exemple....)

Valeurs inconnues : IS NULL, IS NOT NULL
SELECT LIBELLE, PRIXUNITAIRE
FROM ARTICLE
WHERE PRIXUNITAIRE IS NOT NULL
renverra les articles dont le prix existera, c'est à dire aura une valeur correspondante dans la table.
ATTENTION !!! En sql, NULL ne signifie pas 0 ,il représente une valeur indéterminée, il signifie 'rien', comme l'ensemble vide en Mathématique.

Combinaisons : AND , OR
SELECT LIBELLE, QUANTITE, DATE
FROM ARTICLE , CONTIENT, FACTURE
WHERE ARTICLE.ID_ARTICLE=CONTIENT.ID_ARTICLE
AND CONTIENT.ID_FACTURE=FACTURE.ID_FACTURE
AND PRIXUNITAIRE NOT BETWEEN '100' AND '200'
OR PRIXUNITAIRE='150'
renverra les articles, leur quantité par facture, et la date, et ceci pour les articles dont le prix n'est pas compris entre '100' et '200' ou est égal à '150'.

LANGAGE SQL : LES AGREGATS

Les fonctions d'agrégats calculent des valeurs à partir des données d'une colonne particulière. Les opérateurs d'agrégat sont : sum (somme), avg (moyenne), max (le plus grand), min (le plus petit), et count (le nombre d'enregistrements retournés).
SELECT sum(PRIXUNITAIRE)
FROM ARTICLE, CONTIENT
WHERE ARTICLE.ID_ARTICLE=CONTIENT.ID_ARTICLE
AND CONTIENT.ID_FACTURE='123456'
renverra le montant total de la facture '123456'

SELECT avg(PRIXUNITAIRE)
FROM ARTICLE, CONTIENT
WHERE ARTICLE.ID_ARTICLE=CONTIENT.ID_ARTICLE
AND CONTIENT.ID_FACTURE='123456'
renverra le prix moyens par article de la facture '123456'

SELECT max(PRIXUNITAIRE)
FROM ARTICLE, CONTIENT
WHERE ARTICLE.ID_ARTICLE=CONTIENT.ID_ARTICLE
AND CONTIENT.ID_FACTURE='123456'
renverra le plus grand prix de la facture '123456'

SELECT min(PRIXUNITAIRE)
FROM ARTICLE, CONTIENT
WHERE ARTICLE.ID_ARTICLE=CONTIENT.ID_ARTICLE
AND CONTIENT.ID_FACTURE='123456'
renverra le plus petit prix de la facture '123456'

SELECT count(PRIXUNITAIRE)
FROM ARTICLE, CONTIENT
WHERE ARTICLE.ID_ARTICLE=CONTIENT.ID_ARTICLE
AND CONTIENT.ID_FACTURE='123456'
renverra le nombres d'articles figurants sur la facture '123456'

LANGAGE SQL : LA CLAUSE GROUP BY

La clause **GROUP BY**, utilisée dans une instruction **SELECT**, divise le résultat d'une table en groupe. Il y a actuellement 16 niveaux de sous-groupes possibles.
La clause GROUP BY est présente dans les requêtes qui comportent également des fonctions d'agrégats, ces dernières produisant alors une valeur pour chaque groupe, également appelé agrégat vectoriel.
SELECT NOM, avg(PRIXUNITAIRE), sum(PRIXUNITAIRE), ID_FACTURE
FROM ARTICLE , CONTIENT, FACTURE, CLIENT
WHERE ARTICLE.ID_ARTICLE=CONTIENT.ID_ARTICLE
AND CONTIENT.ID_FACTURE=FACTURE.ID_FACTURE

AND FACTURE.ID_CLIENT=CLIENT.ID_CLIENT
GROUP BY ID_FACTURE
renverra pour chaque facture, par nom de client, le prix moyens ainsi que la quantité des produits présents sur chaque facture.

LE LANGAGE SQL EST TRES SOUS-ESTIME. ET POURTANT...

Il n'est pas possible de tout voir en un seul article. La prochaine fois, je détaillerais les clauses HAVING, ORDER BY, et COMPUTE. Ensuite, je vous montrerais comment fonctionne un moteur de base de donnée, afin de bien comprendre de quoi il s'agit, et de montrer comment optimiser ses requêtes SQL. Ensuite, nous nous amuseront un peu avec quelques requêtes un peu complexes, mais qui permettront surtout d'avoir l'information en un seul accès à la base de donnée, et de retoucher ensuite cette information le moins possible dans le script lui-même. Autant limiter les accès à la base, et récupérer le maximum d'information en une seule requête. Car une fois que les requêtes principalement utilisées ont été identifiées, il y a des méthodes très simples pour accélérer les temps de réponse de la base sur ces requêtes. Et surtout, cela évitera de retraiter les données dans de longues et lourdes routine de programmation, souvent illisibles, incompréhensibles, et très très coûteuses en temps machine : il n'y a rien de pire qu'un trie de données effectuée sous forme de pile (tableaux passés en mémoires), alors autant éviter les lourdeurs inutiles.
Après, tout est question de choix technologique et logicielle : un programme ou un site reposant sur une base complexe devra peut être utiliser un moteur SQL conséquent et plus professionel...

Merise: 7ème Partie

Nous voici à la troisième étape du développement de votre projet Web : l'architecture a été définie (choix matériels et technologiques), la base de donnée (cohérente...), a été analysée et construites dans les règles de l'art... Il s'agit maintenant d'utiliser au mieux tout ceci afin de construire les pages dynamiques qui afficheront aux visiteurs les informations qu'ils ont demandés, ainsi que celles que votre employeur/client désire mettre en avant. C'est à ce stade que le langage SQL intervient. Or, ses possibilités sont sous-estimées, et donc ses fonctionalités souvent sous-employées. Pourtant, c'est le fer de lance de la partie dynamique du site Web. Une des parties les plus coûteuses en ressources systèmes, aussi... Car insérer et extraire des données sollicite fortement le(s) disque(s) dur(s) (HDD), utilise de la mémoire (RAM) et du temps système (CPU). Il importera donc non seulement d'utiliser les bonnes requêtes aux bons moments, mais aussi de les rédiger le plus proprement possible afin d'éviter au maximum le gaspillage des ressources du ou des serveurs abritant la base de donnée.

ETRE BON, C'EST SAVOIR SE PROTEGER

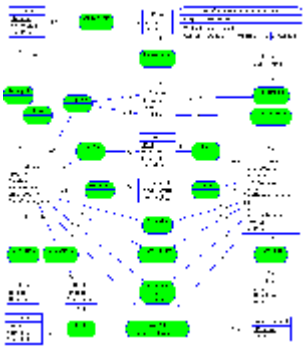
Les grands manitous du Web nous l'ont tous dit : lors de la ruée vers l'or, ce sont les marchands de pelles qui se sont enrichis. A les entendre, il semble possible d'affirmer que les grands gagnants de la course folle de l'Internet soient les informaticiens. Leurs doléances sont maintenant exprimées : difficultés à nous recruter, salaires scandaleux, obligation de nous augmenter tous les six mois, il semblerait que nous ne nous soyons jamais aussi bien porté que depuis ces dernières années. Et visiblement, ils n'ont pas aimé. Or, actuellement, certaines entreprises ferment. Il devient plus facile de recruter de la "main d'oeuvre". Et cela, par contre, à l'air de beaucoup leur plaire. Pourtant, personne ne parle jamais des journées de 14 heures dans les parcs à informaticiens (box), voir même des heures de nuit ou de Week End devant son écran "pour la bonne cause".

La conséquence est claire : maintenant plus encore qu'hier, tout informaticien devra donc rapidement apprendre à se protéger, sous peine de se laisser manger par son travail et sa direction commerciale et marketing. Il devra mettre des barrières et imposer ses délais aux décideurs (patrons, clients, ...) qui oublient un peu vite que nous sommes pas que des outils, mais qu'en revanche nous sommes totalement indispensables à la conception des jolis projets qu'ils nous demandent. Et que la fiabilité desdits projets, et donc la pérennité des entreprises qui les utilisent, dépend en grande partie de la qualité de notre production. Il existe un moyen infaillible de se protéger, et donc de rester performant dans son travail : être bon, être réellement professionnel. Afin que notre code nécessite le moins de retouches possibles, et qu'il convienne parfaitement à la demande des utilisateurs. Ainsi, nous serons tous gagnants : les programmes seront performants, et les informaticiens auront moins de pression. Il faut que les utilisateurs prennent confiance en notre travail...

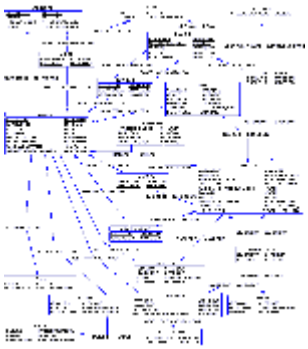
RAPPEL : SELECT ... FROM ... WHERE ... GROUP BY ...

Reprenons le modèle de la coupe du monde Football de 1998 tel que nous l'avons vue dans la partie 5 :

MCD



MPD



SELECT NOMPAYS, count(IDBUT)
FROM JOUEUR, BUT, COMPOSERDE, EQUIPE, PAYS
WHERE JOUEUR.IDJOUEUR=BUT.IDJOUEUR
AND JOUEUR.IDJOUEUR=COMPOSERDE.IDJOUEUR
AND COMPOSERDE.IDEQUIPE=EQUIPE.IDEQUIPE
AND PAYS.IDPAYS=EQUIPE.IDPAYS
GROUP BY NOMPAYS
renverra, pour chaque pays participant, le nombre total de buts marqué durant la compétition.

La requête concerne 5 tables, elle nécessite donc 4 jointures. Le détail des clauses **WHERE**, **GROUP BY** et des **Agrégats** a été vu dans la partie 6

LANGAGE SQL : LA CLAUSE HAVING

La clause **HAVING** définit les critères de la clause **GROUP BY** de la même façon que la clause **WHERE** dans une instruction **SELECT**. L'avantage de la clause **HAVING** est surtout qu'elle peut comporter des agrégats, ce que ne permets pas la clause **WHERE**.

SELECT NOMJOUEUR
FROM JOUEUR, BUT
WHERE JOUEUR.IDJOUEUR=BUT.IDJOUEUR
HAVING count(BUT.IDJOUEUR) > '2'
retournera la liste des joueurs ayant marqué plus de deux buts.

Il est possible de combiner les critères avec les opérateurs **AND** et **OR**.

LANGAGE SQL : LA CLAUSE ORDER BY

La clause **ORDER BY** permet de trier les résultats d'une requête par colonnes. Il est possible de demander un tri croissant (**asc**) , ou décroissant (**desc**). Par défaut, le tri est croissant (**asc**). Plusieurs colonnes peuvent être indiquées : dans ce cas, le tri sera effectué selon la première colonne indiquée, puis la deuxième, etc...

SELECT TAILLE, POIDS, NOMJOUEUR
FROM JOUEUR
ORDER BY TAILLE desc, POIDS desc
retournera la taille, le poids et le nom des joueurs participants au tournoi, mais du plus grand au plus petit : en cas de taille équivalente, les plus lourds seront retournés en premier.

Cette clause est très importante : en effet, elle permet à elle seule d'effectuer un classement des données, et donc d'éviter de programmer un traitement supplémentaire généralement très lourd en ressource système et en mémoire.

Il est possible d'utiliser le numéro des colonnes plutôt que leur nom.

SELECT POSITION, avg(TAILLE)
FROM JOUEUR
GROUP BY POSITION
ORDER BY 2
retournera la taille moyenne des joueurs positions par positions, de la taille moyenne la plus petite à la plus grande. Cela peut se révéler très utile, notamment pour des tableaux de statistiques. Et surtout, les données arrivent déjà trillées.

LE LANGAGE SQL EST LA BASE DU DEVELOPPEMENT

Ces deux articles n'ont pas la prétention de remplacer un ouvrage technique, ou la documentation officielle. Seulement, les requêtes SQL sont la trame du développement d'un site dynamique. Une base bien conçue permettra de gérer avec cohérence les données, et ainsi d'éviter un certain nombre d'erreurs. La requête SQL permet d'extraire ces données, le langage de développement s'occupant de la présentation de ces données et de la gestion des tests souvent nécessaires, voir même de la mise en page. De plus, il ne faut pas perdre de vue que les appels à la base de donnée sont de grands consommateurs de ressources systèmes : il importera donc de bien doser leur utilisation, et surtout d'utiliser toutes les possibilités du SQL afin d'obtenir le maximum d'informations dans la même requête.

J'en ai encore eu la confirmation lors des deux derniers évènements auxquels j'ai assisté, un déjeuner/débat avec des acteurs de l'Internet Français et Européens (entrepreneur et investisseurs), et le First Tuesday du mois d'Avril (avec le fondateur de Self Trade) : l'heure est à la crise financière pour les projets Internet non-viables, mais l'avenir est surtout au sites dynamiques. Les entreprises ne se satisfont plus de sites 'plaquettes', elles veulent maintenant une plus grande dimension informatique et technique pour leur Internet.

Merise: 7ème Partie

Nous voici à la troisième étape du développement de votre projet Web : l'architecture a été définie (choix matériels et technologiques), la base de donnée (cohérente...), a été analysée et construites dans les règles de l'art... Il s'agit maintenant d'utiliser au mieux tout ceci afin de construire les pages dynamiques qui afficheront aux visiteurs les informations qu'ils ont demandés, ainsi que celles que votre employeur/client désire mettre en avant. C'est à ce stade que le langage SQL intervient. Or, ses possibilités sont sous-estimées, et donc ses fonctionalités souvent sous-employées. Pourtant, c'est le fer de lance de la partie dynamique du site Web. Une des parties les plus coûteuses en ressources systèmes, aussi... Car insérer et extraire des données sollicite fortement le(s) disque(s) dur(s) (HDD), utilise de la mémoire (RAM) et du temps système (CPU). Il importera donc non seulement d'utiliser les bonnes requêtes aux bons moments, mais aussi de les rédiger le plus proprement possible afin d'éviter au maximum le gaspillage des ressources du ou des serveurs abritant la base de donnée.

ETRE BON, C'EST SAVOIR SE PROTEGER

Les grands manitous du Web nous l'ont tous dit : lors de la ruée vers l'or, ce sont les marchands de pelles qui se sont enrichis. A les entendre, il semble possible d'affirmer que les grands gagnants de la course folle de l'Internet soient les informaticiens. Leurs doléances sont maintenant exprimées : difficultés à nous recruter, salaires scandaleux, obligation de nous augmenter tous les six mois, il semblerait que nous ne nous soyons jamais aussi bien porté que depuis ces dernières années. Et visiblement, ils n'ont pas aimé. Or, actuellement, certaines entreprises ferment. Il devient plus facile de recruter de la "main d'oeuvre". Et cela, par contre, à l'air de beaucoup leur plaire. Pourtant, personne ne parle jamais des journées de 14 heures dans les parcs à informaticiens (box), voir même des heures de nuit ou de Week End devant son écran "pour la bonne cause".

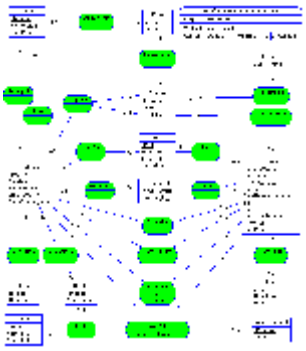
La conséquence est claire : maintenant plus encore qu'hier, tout informaticien devra donc rapidement apprendre à se protéger, sous peine de se laisser manger par son travail et sa direction commerciale et marketing. Il devra mettre des barrières et imposer ses délais aux décideurs (patrons, clients, ...) qui oublient un peu vite que nous sommes pas que des outils, mais qu'en revanche nous sommes totalement indispensables à la conception des jolis projets qu'ils nous demandent. Et que la fiabilité desdits projets, et donc la pérennité des entreprises qui les utilisent, dépend en grande partie de la qualité de notre production. Il existe un moyen infaillible de se protéger, et donc de rester performant dans son travail :

être bon, être réellement professionnel. Afin que notre code nécessite le moins de retouches possibles, et qu'il convienne parfaitement à la demande des utilisateurs. Ainsi, nous serons tous gagnants : les programmes seront performants, et les informaticiens auront moins de pression. Il faut que les utilisateurs prennent confiance en notre travail...

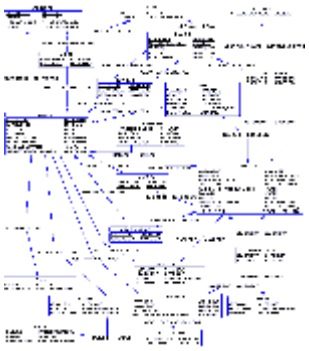
RAPPEL : SELECT ... FROM ... WHERE ... GROUP BY ...

Reprenons le modèle de la coupe du monde Football de 1998 tel que nous l'avons vue dans la partie 5 :

MCD



MPD



SELECT NOMPAYS, count(IDBUT)
FROM JOUEUR, BUT, COMPOSERDE, EQUIPE, PAYS
WHERE JOUEUR.IDJOUEUR=BUT.IDJOUEUR
AND JOUEUR.IDJOUEUR=COMPOSERDE.IDJOUEUR
AND COMPOSERDE.IDEQUIPE=EQUIPE.IDEQUIPE
AND PAYS.IDPAYS=EQUIPE.IDPAYS
GROUP BY NOMPAYS
renverra, pour chaque pays participant, le nombre total de buts marqué durant la compétition.

La requête concerne 5 tables, elle nécessite donc 4 jointures. Le détail des clauses **WHERE**, **GROUP BY** et des **Agrégats** a été vu dans la partie 6

LANGAGE SQL : LA CLAUSE HAVING

La clause **HAVING** définit les critères de la clause **GROUP BY** de la même façon que la clause **WHERE** dans une instruction **SELECT**. L'avantage de la clause **HAVING** est surtout qu'elle peut comporter des agrégats, ce que ne permets pas la clause **WHERE**.

SELECT NOMJOUEUR
FROM JOUEUR, BUT
WHERE JOUEUR.IDJOUEUR=BUT.IDJOUEUR
HAVING count(BUT.IDJOUEUR) > '2'
retournera la liste des joueurs ayant marqué plus de deux buts.

Il est possible de combiner les critères avec les opérateurs **AND** et **OR**.

LANGAGE SQL : LA CLAUSE ORDER BY

La clause **ORDER BY** permet de trier les résultats d'une requête par colonnes. Il est possible de demander un tri croissant (**asc**) , ou décroissant (**desc**). Par défaut, le tri est croissant (**asc**). Plusieurs colonnes peuvent être indiquées : dans ce cas, le tri sera effectué selon la première colonne indiquée, puis la deuxième, etc...

SELECT TAILLE, POIDS, NOMJOUEUR
FROM JOUEUR
ORDER BY TAILLE desc, POIDS desc

retournera la taille, le poids et le nom des joueurs participants au tournoi, mais du plus grand au plus petit : en cas de taille équivalente, les plus lourds seront retournés en premier.

Cette clause est très importante : en effet, elle permet à elle seule d'effectuer un classement des données, et donc d'éviter de programmer un traitement supplémentaire généralement très lourd en ressource système et en mémoire.

Il est possible d'utiliser le numéro des colonnes plutôt que leur nom.

SELECT POSITION, avg(TAILLE)
FROM JOUEUR
GROUP BY POSITION
ORDER BY 2
retournera la taille moyenne des joueurs positions par positions, de la taille moyenne la plus petite à la plus grande. Cela peut se révéler très utile, notamment pour des tableaux de statistiques. Et surtout, les données arrivent déjà trillées.

LE LANGAGE SQL EST LA BASE DU DEVELOPPEMENT

Ces deux articles n'ont pas la prétention de remplacer un ouvrage technique, ou la documentation officielle. Seulement, les requêtes SQL sont la trame du développement d'un site dynamique. Une base bien conçue permettra de gérer avec cohérence les données, et ainsi d'éviter un certain nombre d'erreurs. La requête SQL permet d'extraire ces données, le langage de développement s'occupant de la présentation de ces données et de la gestion des tests souvent nécessaires, voir même de la mise en page. De plus, il ne faut pas perdre de vue que les appels à la base de donnée sont de grands consommateurs de ressources systèmes : il importera donc de bien doser leur utilisation, et surtout d'utiliser toutes les possibilités du SQL afin d'obtenir le maximum d'informations dans la même requête.

J'en ai encore eu la confirmation lors des deux derniers évènements auxquels j'ai assisté, un déjeuner/débat avec des acteurs de l'Internet Français et Européens (entrepreneur et investisseurs), et le First Tuesday du mois d'Avril (avec le fondateur de Self Trade) : l'heure est à la crise financière pour les projets Internet non-viables, mais l'avenir est surtout au sites dynamiques. Les entreprises ne se satisfont plus de sites 'plaquettes', elles veulent maintenant une plus grande dimension informatique et technique pour leur Internet.

Merise: 8ème Partie

Il est maintenant temps de regarder comment fonctionne le moteur de la base de donnée et d'optimiser les requêtes SQL. En effet, il est possible de gagner considérablement en ressources machines rien que sur une optimisation du code SQL. Là encore, il n'est pas question de magie, mais de bien comprendre comment cela fonctionne, et d'utiliser à propos les outils choisis, en l'occurrence les bases de donnée, qui sont très coûteuses en ressources systèmes. Bien sur, elles nous rendent de fiers services mais leur utilisation abusive (ou maladroite) peut provoquer de forts ralentissements, voir même crasher le serveur. Il faut donc être très précautionneux de ce que l'on va demander à l'outil. Comme d'habitude, il faut définir une architecture en adéquation avec ses besoins, et l'utiliser le plus à propos possible.

ECONOMISER LA MACHINE, C'EST EN FAIT POUVOIR LUI EN DEMANDER PLUS

Un hébergeur Français bon marché, très populaire chez les informaticiens et proche d'un fournisseur d'accès gratuit à Internet, nous l'a cruellement rappelé à la fin de l'année dernière. Fournissant des hébergements mutualisés (plusieurs sites par machines), il s'est retrouvé avec certains sites reposant intégralement sur des requêtes SQL lourdes voir inutiles, fausses, et surtout lancées à tors et à travers. Certains de ces sites sont devenus un peu connus, ont commencé à faire de l'audience, et ont finalement faillis faire chavirer l'ensemble de la plate-forme d'hébergement. Les bases de données, surchargées, refusaient souvent de se lancer et de répondre aux requêtes, et renvoyaient régulièrement des messages d'erreurs à tous les utilisateurs. Aux heures de pointes, les sites n'étaient parfois même plus consultables. Inutile de dire que la crédibilité d'un site internet affichant des erreurs d'accès à la base SQL est sérieusement entamée, et que de nombreuses personnes ont du reprendre leur développement. Pour que le site Internet tienne la charge quand il commence à être fréquenté, il vaut mieux qu'il repose sur des fondations solides tant matérielles que logicielles. Quant à cet hébergeur, il a du refondre son architecture matérielle, mais a aussi perdu une grande part de sa crédibilité et de sa clientèle...

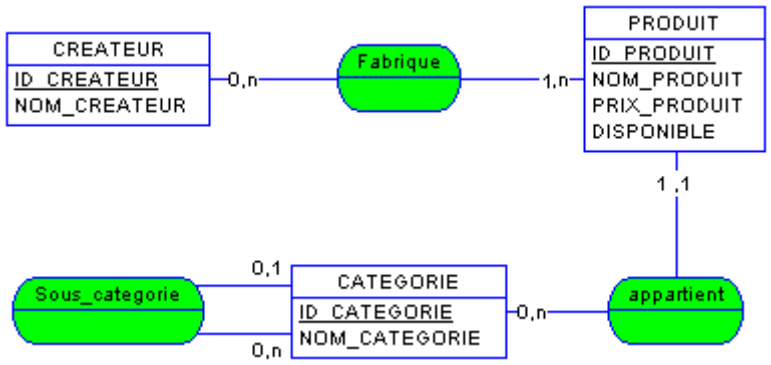
MODELISONS UN CATALOGUE : REVISIONS DE LA TECHNIQUE...

Système d'Information (SI) : Arborescence en Arbre...

Un garagiste expose sur le Web ses modèles, afin de présenter ses nouveautées et surtout son stock en temps réel (disponibilité de ses modèles). Il utilise ce que l'on appèle un catalogue, sans caddy ni paiement en ligne (ce sont des voitures...), mais avec navigation par catégorie/sous-catégorie, affichage de la fiche-produit de la voiture, et possibilité de recherche par marque de véhicule. Il limite sa fiche produit au nom, au prix, et à la disponibilité de la voiture (on simplifie...). Ce qui donne :

Un **Produit** possède 1 ou plusieurs **Créateurs** (marques, fabricants). Un **Créateur** fabrique 0 ou plusieurs **Produits**. Ces **Produits** appartiennent chacun à une et une seule **Catégorie**. Chaque **Catégorie** contient 0 ou plusieurs **Produits**. Enfin, une **Catégorie** peut avoir 0 ou plusieurs sous-**Catégories**, chaque **Catégorie** ayant 0 ou une **Catégorie** parente.

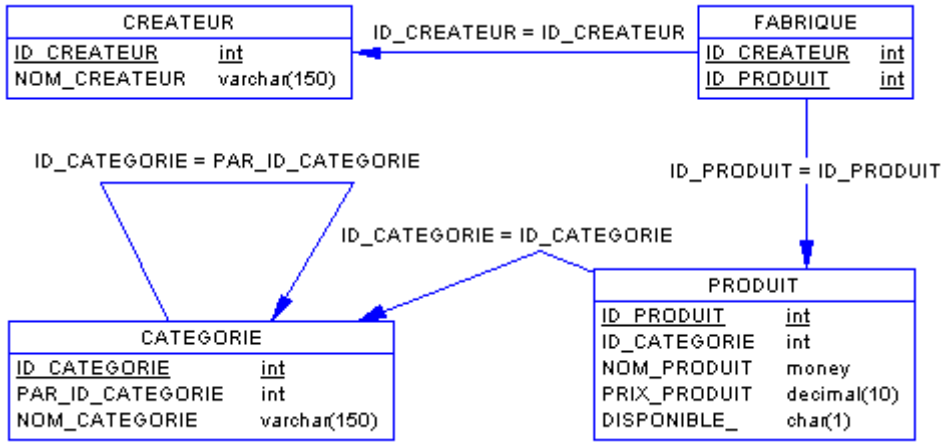
D'où le MCD :



Qui entraîne le MLDR :

CREATEUR (ID_CREATEUR, NOM_CREATEUR)
FABRIQUE (ID_CREATEUR, ID_PRODUIT)
PRODUIT (ID_PRODUIT, #ID_CATEGORIE, NOM_PRODUIT, PRIX_PRODUIT, DISPONIBLE)
CATEGORIE (ID_CATEGORIE, #ID_PAR_CATEGORIE, NOM_CATEGORIE)

Qui génère le MPD :



POUR SE PROTEGER, IL FAUT BIEN COLMATER LES JOINTURES !!!

Prenons une requête d'extraction toute simple, et regardons ce qu'il se passe : On recherche les modèles

disponibles de Véhicules de marques "Peugeot" de type "Décapotable", et leur prix. L'identifiant ID_CREATEUR de "Peugeot" est ici "4", et l'identifiant ID_CATEGORIE de "Décapotable" est "10".

```
SELECT *
FROM PRODUIT, FABRIQUE
WHERE PRODUIT.ID_CATEGORIE='10'
AND FABRIQUE.ID_CREATEUR='4'
AND PRODUIT.ID_PRODUIT=FABRIQUE.ID_PRODUIT
```

Regardons maintenant ce que fait le moteur de la base :

1°) Celui interprète la requête ligne par ligne. Pour commencer, il met dans un tableaux tous les éléments de **PRODUIT** et de **FABRIQUE**, en faisant ce que l'on appelle un **produit cartésien** : c'est à dire que pour chaque enregistrement de la première table rencontrée, ici **PRODUIT**, il mettra en face tous les enregistrements de la seconde table rencontrés un par un, et ce pour chaque ligne. Si **PRODUIT** contient 250 enregistrements, et **FABRIQUE** 110, cette table temporaire et intermédiaire comprendra 250*110=27500 lignes.

PRODUIT					FABRIQUE	
id_produit	id_categorie	nom_produit	prix_produit	disponible	id_createur	id_produit
1	3	103SP	55 000	Y	4	2
1	3	103SP	55 000	Y	4	11
...	...	...	...	...	8	15
...	...	...	...	...	...	...
1	3	103SP	55 000	Y	33	18
2	10	205 Blue	83 000	Y	4	12
2	10	205 Blue	83 000	Y	4	11
...	...	...	...	...	8	2
...	...	...	...	...	...	...
2	10	205 Blue	83 000	Y	33	18
etc...	etc...	etc...	etc...	etc...	etc...	etc...

2°) Puis, il supprime de cette même table temporaire les valeurs de id_catégories différentes de '10'.

PRODUIT					FABRIQUE	
id_produit	id_categorie	nom_produit	prix_produit	disponible	id_createur	id_produit
2	10	205 Blue	83 000	Y	4	2
2	10	205 Blue	83 000	Y	4	11
...	...	...	...	...	8	15
...	...	...	...	...	...	...
2	10	205 Blue	83 000	Y	33	18
etc...	etc...	etc...	etc...	etc...	etc...	etc...

3°) Ensuite, il supprime de cette même table temporaire les valeurs de id_createur différentes de '4'.

PRODUIT					FABRIQUE	
id_produit	id_categorie	nom_produit	prix_produit	disponible	id_createur	id_produit
2	10	205 Blue	83 000	Y	4	2
2	10	205 Blue	83 000	Y	4	11
...	...	...	...	...	...	...

etc...	**etc...**	**etc...**	**etc...**	**etc...**	**etc...**	**etc...**

4°) Enfin, il effectue la jointure demandée, et ne garde que les lignes dont PRODUIT.ID_PRODUIT=FABRIQUE.ID_PRODUIT .

PRODUIT					**FABRIQUE**	
id_produit	**id_categorie**	**nom_produit**	**prix_produit**	**disponible**	**id_createur**	**id_produit**
2	**10**	**205 Blue**	**83 000**	**Y**	**4**	**2**
etc...	**etc...**	**etc...**	**etc...**	**etc...**	**etc...**	**etc...**

5°) Eventuellement, si il le lui avait été demandé, c'est à ce stade qu'il aurait interprété les commandes des instructions **GROUP BY**, puis **HAVING** et enfin **ORDER BY**. Toutefois, il est intéressant de constater que toutes les colonnes sont présentes dans le tableaux, et que l'on a passé en mémoire à peu près 100 fois l'intégralité de la quantité de données contenues dans ces seules tables.

Imaginez un peu si le garagiste avait eu 100 000 Voitures réparties dans 250 Catégories, avec à peu près 970 marques différentes ?

"FÔ PAS GACHER", COMME DIRAIT L'AUTRE...

Reprenons la même requête, mais formulée un poil différemment :

SELECT PRODUIT.NOM_PRODUIT, PRODUIT.PRIX_PRODUIT
FROM PRODUIT, FABRIQUE
WHERE PRODUIT.ID_PRODUIT=FABRIQUE.ID_PRODUIT
AND PRODUIT.ID_CATEGORIE='10'
AND FABRIQUE.ID_CREATEUR='4'

Regardons ce que fais le moteur de la base :

1°) Tout d'abord, il récupère dans un tableaux les éléments de **PRODUIT** demandés et ceux nécessaires pour la requête, fais pareil pour **FABRIQUE**, et la jointure étant spécifiée en premier, il ne prend que les lignes dont PRODUIT.ID_PRODUIT=FABRIQUE.ID_PRODUIT .

PRODUIT				**FABRIQUE**	
id_produit	**id_categorie**	**nom_produit**	**prix_produit**	**id_createur**	**id_produit**
2	**10**	**205 Blue**	**83 000**	**4**	**2**
3	**10**	**Mégane**	**83 000**	**8**	**3**
5	**15**	**4L ME**	**83 000**	**4**	**5**
...	**...**	**...**	**...**	**...**	**...**
etc...	**etc...**	**etc...**	**etc...**	**etc...**	**etc...**

2°) Il enlève les lignes dont id_categorie n'est pas égal à '10'

PRODUIT				**FABRIQUE**	
id_produit	**id_categorie**	**nom_produit**	**prix_produit**	**id_createur**	**id_produit**
2	**10**	**205 Blue**	**83 000**	**4**	**2**
3	**10**	**Mégane**	**83 000**	**8**	**3**

...	...	...	...	...	...
etc...	etc...	etc...	etc...	etc...	etc...

3°) Puis celles où id_createur n'est pas égal à '4'

PRODUIT				FABRIQUE	
id_produit	id_categorie	nom_produit	prix_produit	id_createur	id_produit
2	10	205 Blue	83 000	4	2
...	...	...	...	...	...
etc...	etc...	etc...	etc...	etc...	etc...

4°) Enfin, il ne garde que les colonnes demandées dans la requête, c'est à dire le nom, et le prix.

205 Blue	83 000
...	...
etc...	etc...

Si j'avais su, par expérience ou connaissance du contexte, que la clause FABRIQUE.ID_CREATEUR='4' était plus réductrice en terme d'éléments que la clause PRODUIT.ID_CATEGORIE='10' , je l'aurais alors mise avant celle ci, afin de réduire le plus possible le nombre d'éléments stockés en mémoire, et donc les opérations nécessaires pour les tris et traitements ultérieurs.

Il est intéressant de remarquer que pour un obtenir un résultat similaire, on a beaucoup moins tiré sur la machine, qui pourra donc accomplir cette requête un plus grand nombre de fois simultanément, et donc accueillir un plus grand nombre de visiteur sans souffrir...

QUI VEUT ALLER LOIN, MENAGE SA MONTURE

Sans tomber non plus dans l'intégrisme inutile du coupeur de cheveux en 4, il est tout de même très clair que la simple formulation de la requête SQL est lourde de conséquence sur les ressources et le temps machine nécessaire pour sa simple exécution. On peut ainsi citer quelques précautions simples qui allègeront simplement la charge reposant sur le serveur :

- **Mettre les jointures en premier :**
  Si n tables, alors (n-1) jointures.

- **Placer les comparaisons les plus restrictives le plus tôt possible :**
  cela fera toujours autant de lignes qui ne seront plus en mémoire, et que l'ordinateur n'aura plus à traiter dans le reste de sa requête.

- **EVITER ABSOLUMENT "SELECT * FROM ..." :**
  Ne demandez que les colonnes nécessaires, c'est toujours ça de moins à garder en tableaux après la requête, et donc cela économise la mémoire. De plus, si un jour vous déplacez votre code, et que deux colonnes se trouvent inversées dans la nouvelle base, cela n'aura aucune conséquence pour votre développement.

- **Comparer des colonnes de même type :**
  Un CHAR(150) est considéré du même type qu'un VARCHAR(150), mais différent d'un CHAR(152) ou d'un VARCHAR(148). Cela oblige le moteur de base de donnée à effectuer des conversions internes.

- **Formuler les clauses de comparaison le plus précisement possible :**
  En particulier, éviter de mettre des **%** partout dans les clauses **LIKE**, c'est très lourd à traiter...

- **Mettre les identifiants en INT, et en AUTOINCREMENT :**
  L'avantage principal de l'autoincrement est que pour chaque création d'enregistrement ne comprenant pas d'office son identifiant, le moteur se charge lui-même de lui en attribuer un [du type max(id)+1], ce qui évite des manipulations supplémentaires.

- **Utilisez des INDEX :**
  Mais l'explication, là, ce sera pour la prochaine fois....