Andréas Perret, « Bases de données relationnelles (notions et souvenirs d'un amateur). Une approche STS »

L'exposé d'Andréas Perret est le témoignage d'un professionnel formé « sur le tas » qui s'est pris d'affection pour ces objets étranges que sont les bases de données relationnelles. À quel mode d'organisation renvoient-elles ? Comment sont-elles construites ? Quelles sont leurs composantes internes ? Quels agencements sont-elles à même de suggérer ? Voici quelques questions sur lesquelles nous allons tenter ici de revenir.

Le lift du garagiste

D'abord une petite anecdote, souvenir du milieu des années 1990 lorsqu'Andréas Perret travaillait au sein de l'unité de planification d'un grand hôpital. Le groupe auquel il contribuait était chargé de mettre en place un système d'information destiné aux cadres de cet hôpital qui compte plusieurs milliers d'employés. Ce système devait notamment être capable de dialoguer avec la base de données que l'école d'infirmières était en train de sélectionner. On charge alors Andréas Perret d'assister à l'exposé de présentation de ce nouveau logiciel afin de s'assurer de la compatibilité des deux systèmes. Durant la présentation, de multiples démonstrations visant à démontrer les nombreux *outputs* (formulaires, récapitulatifs) que cet outil onéreux était capable de produire. Mais ça n'est pas ce qui intéressait Andréas Perret qui ose finalement poser une question plutôt inhabituelle : « Puis-je voir votre **diagramme de structure de données** ? ». Froid dans la salle jusqu'alors enthousiaste : « Hum, d'accord », répond le représentant, « mais seulement une fois que le logiciel aura été acheté »…

Que fait Andréas Perret lorsqu'il demande à voir le diagramme de structure de données ? Réponse : il demande à voir une partie des *constituants internes* de la base de données. Que fait le représentant lorsqu'il vante les mérites (clarté, facilité de commande) des *outputs* de sa base de données ? Réponse : il donne à voir son *enveloppe*. L'analogie avec le monde automobile fonctionne ici assez bien : lorsqu'un revendeur présente une voiture d'occasion, il ne montre généralement que son enveloppe (carrosserie, sièges, puissance, nombre de kilomètres parcourus). Par contre, lorsqu'une voiture est présentée à un revendeur, en plus de son enveloppe, celui-ci va également vérifier son organisation interne. Comment s'y prend-il ? Il va

UNIL | Université de Lausanne
LADHUL - Laboratoire de cultures et humanités digitales de l'UNIL

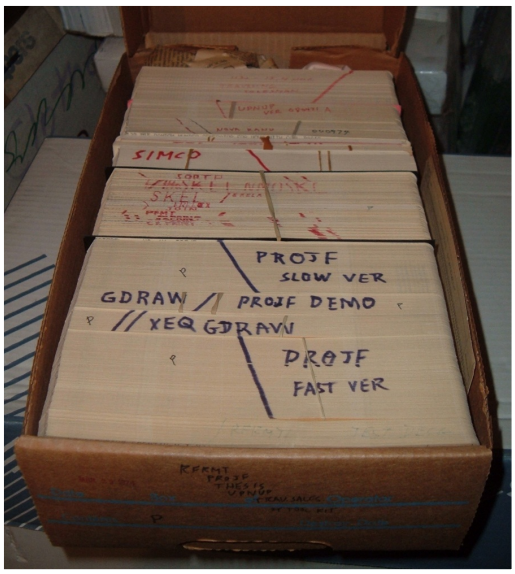
généralement la mettre sur son lift et examiner minutieusement ses constituants internes *ainsi que leurs relations*. Tous deux sont honorables : l'enveloppe tout comme l'état des relations entre les constituants internes. Pourtant, une vue qui n'embrasserait qu'un seul aspect serait à coup sûr incomplète. Mais encore faut-il être équipé pour parvenir à cette vue ! C'est le défi de l'exposé d'Andréas Perret : nous fournir un lift et un (début de) regard capable de mieux comprendre et d'évaluer les relations entre les constituants internes d'une base de données relationnelle.

Au-delà du « mode plan »

Au début des années 1990, Andréas découvre un logiciel de traitement de texte novateur nommé *Microsoft Word*. Ce logiciel permettant de hiérarchiser facilement le contenu de ses textes grâce à l'insertion de chapitres, sous-chapitres et autres sous-sous chapitres, rappelle à notre conférencier fraichement diplômé ce qu'Erwin Panofsky nommait « pensée scolastique » (en référence à l'apparition d'une architecture gothique). En effet, de par la hiérarchisation de l'information qu'il rend possible, ce mode d'organisation permet de séparer la structure du contenu. A partir de là, deux niveaux se créent : les chapitres (structure) et le texte (contenu).
Ce mode de classification est pratique à plusieurs égards mais comporte néanmoins un grand défaut : il ne permet pas de traiter des *relations* car la structure décline le contenu de façon irrémédiablement hiérarchique. Pour faire exister des relations, pour comparer le contenu selon les mêmes termes, il est nécessaire de recourir à *autre mode d'organisation* qui ne serait pas strictement arborescent.
C'est là qu'interviennent les bases de données relationnelles dont le logiciel *Access*, publié par Microsoft en 1993, rend les fonctionnalités les plus importantes accessibles aux utilisateurs novices. Comme nous le verrons plus loin, l'originalité de ces objets multiples est qu'ils permettent une organisation en tables dont les contenus indexés peuvent être facilement mis en relation. A partir de là, s'établit une organisation davantage horizontale que verticale car un élément n'est plus obligé de recevoir ses informations d'une unité supérieure. En somme, ces objets établissent une *carte* (avec hameaux et routes d'accès) plutôt qu'une *liste hiérarchique*.

Mettre en lien nécessite un index

Etats-Unis, fin des années 1950. Le hangar des archives administratives, situé alors à Alexandria (Virginie), commence à être sérieusement encombré par des milliers de boîtes de cartes perforées. L'image de cet amas est représentative d'une situation qui a conduit l'industrie informatique à trouver des solutions pour optimiser le stockage de l'information: Comment mettre un maximum d'informations dans un minimum d'espace?
Le moyen mis au point ressemble pour beaucoup – dans son principe – à ce qui rend possible les bases de données relationnelles. Pour s'y retrouver dans ce capharnaüm de cartes perforées, on range les cartes dans des boîtes qui comportent chacune un nom unique (renvoyant entre autres à une allée et une étagère). Dans ces boîtes, on trace des lignes au feutre pour délimiter les différentes séquences du programme codé en cartes perforées et on nomme ensuite chacune de ces séquences.



Ce faisant, on *indexe* l'ensemble des cartes perforées de la boîte et même du hangar ! On rend possible le pointage de son index sur une partie précise d'un programme ainsi que sa nomination : « ici se trouve la partie du programme nommée ″PROJF FAST VER″ ». Boîte unique (et chemin d'accès), séquenciation au feutre et nomination de la séquence : voilà les ingrédients nécessaires et suffisants pour localiser chaque carte perforée du hangar d'Alexandria. Oubliez de séquencer le contenu d'une boîte : les données seront toujours là, bien à l'abri dans leur boîte, mais l'accès permettant de savoir à quoi elles sont liées n'existera pas. Les données auront ainsi un statut étrange d'existant-non-existant-car-non-localisable, bien connu des ordinateurs d'aujourd'hui.

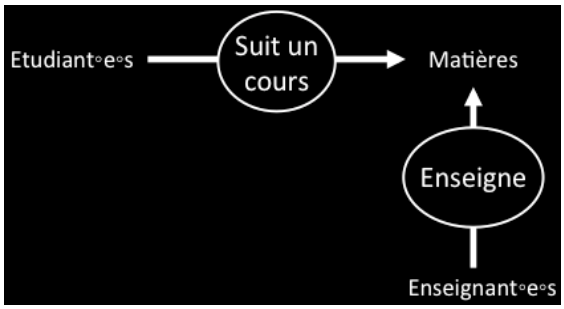
Quel rapport avec les bases de données relationnelles ? Tout et rien, comme souvent. Rien, car très peu de feutres, de bois ou d'étagères poussiéreuses entrent dans la composition d'une base de données relationnelle. Tout, car sans opérations pratiques consistant à ajouter une information *sur* les données afin de pouvoir – ensuite – les situer et les mettre en relation *facilement*, les bases de données relationnelles n'auraient jamais vu le jour. Boîtes remplies de cartes perforées et bases de données relationnelles : ces deux types d'*objets intermédiaires* ont besoin d'être *équipés* en passant par des procédures pratiques d'indexation leur permettant in fine de produire des relations[1]. Car un index suggère toujours un *mouvement de mise en relation* : mise en relation d'un groupement de cartes perforées à (par exemple) mon désir de rafraichissement de mémoire, bien aidé par cette ligne au feutre et la nominalisation de cette séquence. À coup sûr, il vaut mieux préférer le verbe « indexer » que le nom trop statique d' « index » : un index suggère toujours un *mouvement de mise en relation*. Mon désir *est indexé* à ces éléments ; j'accède à ces éléments désirés en en passant par un index (qui me facilite grandement la tâche).

Exemple 1 : indexer = traduire en tables, clés et clés étrangères

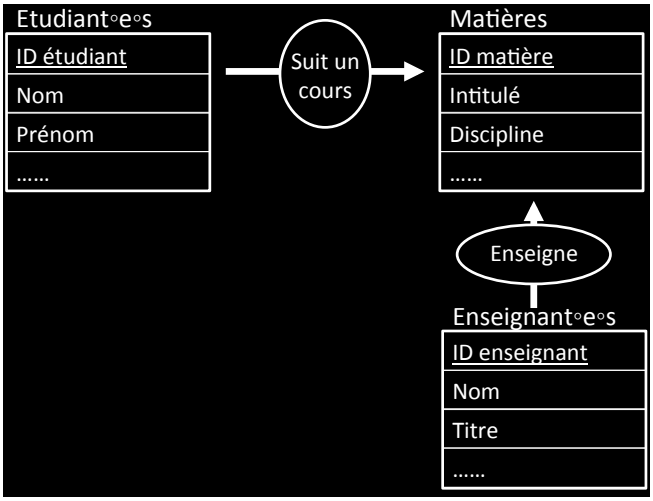
Imaginons un directeur d'école. Il souhaite mettre en place un outil lui permettant d'extraire rapidement des informations sur les élèves de son établissement : cursus suivi, enseignants rencontrés, notes obtenues, etc. Il explique à un informaticien mandaté pour l'occasion que dans son école, *il y a des élèves qui suivent des cours portant sur des matières, lesquelles sont prodigués par des enseignants* « Rien de bien surprenant », se dit l'informaticien qui voit se

[1] Sur ce thème de l'équipement des objets intermédiaires, voire VINCK D., 2011, Taking intermediary objects and equipping work into account in the study of engineering practices, Engineering Studies, vol 3, n 1, p. 25-44.

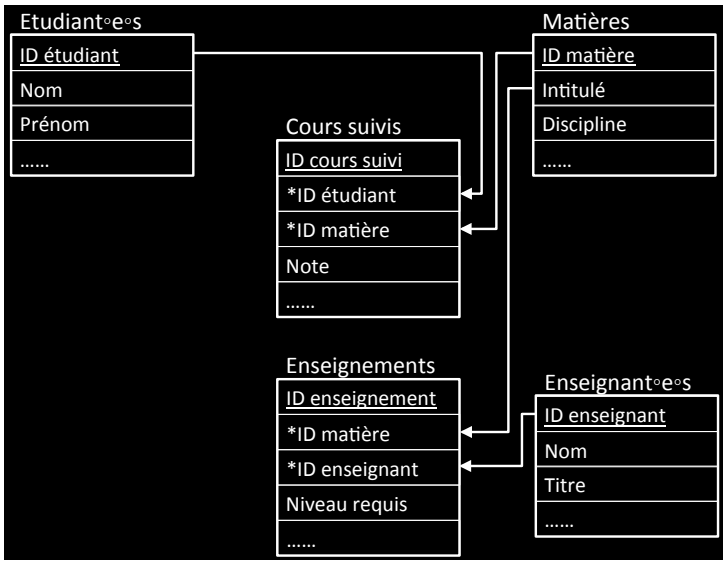
dessiner un **modèle conceptuel de données**. « Reste à voir comment ordonner cette affirmation afin de la transformer en différents ″paquets″ entretenant des relations les uns aux autres. En somme, reste à voir comment transformer ce modèle conceptuel en **modèle de données logiques** ».
Un étudiant suit un cours ; un enseignant enseigne une matière. L'étudiant est lié à la matière qu'enseigne l'enseignant par le cours qu'il suit. Schématiquement, cela représente une relation de ce type :



Evidemment, les trois termes de ce schéma nucléaire sont tous des ensembles qui regroupent plusieurs éléments. Heureusement, afin de garantir l'unicité de chaque individu (matières y compris), l'école leur a donné à chacun un identifiant (ID). A partir de là, il devient facile de transformer ces ensembles flous en des **tables**, soit des sortes de fichiers listés :



En langage « bases de données relationnelles », l'identifiant d'une table est appelé **une clé**. Notre base de données est maintenant formée de trois tables disposant chacune d'une clé se déclinant en identifiants permettant de particulariser chaque étudiant, professeur et matière. Reste le problème des verbes : en l'état, ni « suit un cours », ni « enseigne » ne lieront quoi que soit. Pour le permettre, il faut les transformer en tables disposant d'une clé qui leur est propre **ainsi que d'une « clé étrangère » faisant le pont entre les tables qu'il s'agit de relier**. Table par table – c'est-à-dire clé par clé étrangère – la base de données relationnelle est en train de prendre forme :

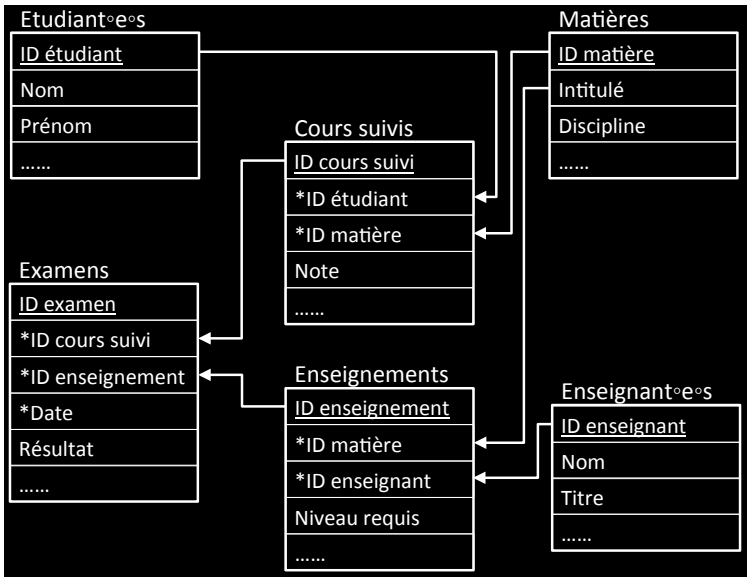


La table « Etudiant-e-s » est ainsi **indexée** à la table « Matières » *via* la table « Cours suivis » en vertu de ses clés étrangères « ID étudiant » et « ID matière » (annotées d'un astérisque). Pareil pour la table « Enseignant-e-s » : elle est **indexée** à la table « Matières » (et donc *in fine* à la table « Etudiants ») *via* la table « Enseignements » en vertu de ses clés étrangères « ID matières » et « ID enseignements ». Sous cette forme, l'ensemble des informations évoquées dans le **modèle conceptuel de données** est retranscrit sous la forme du **modèle logique de données**, qui pourra être implémenté sous forme d'une base de données informatique.

Une telle représentation offre un moule apte à réceptionner des informations, à condition qu'elles adoptent la forme qu'il prévoit. Tout modèle de données va simultanément exclure tout information dont il ne peut recevoir le reflet. Le modèle figurant ci-dessus interdit par exemple de mettre en lien les étudiants qui suivent un cours et l'enseignant qui le dispense. Pour stocker une telle information (et l'extraire ultérieurement), il faut altérer le modèle, par exemple en introduisant le concept d'examen.

On l'aura compris, pour rendre compte de ce concept d'examen, il faut 1) le traduire en une **table** disposant d'une **clé** pouvant se décliner en **identifiants** permettant de particulariser les différents examens et 2) inclure une ou plusieurs **clés étrangères** capables de faire le pont entre les tables que l'on souhaite relier :

UNIL | Université de Lausanne
LADHUL - Laboratoire de cultures et humanités digitales de l'UNIL



« Grâce à ce diagramme de structure de données, donnez-moi l'identifiant d'un élève et je vous pourrai vous sortir les cours qu'il a suivi, les enseignants qu'il a côtoyés, les matières qui l'ont intéressé où encore les dates des examens qu'il a passés. Pareil pour les autres tables : donnez-moi l'identifiant d'un enseignant et je pourrai vous sortir ces matières de prédilections, le nom de chacun de ses cours ou encore la popularité de ses cours auprès des élèves », dit l'informaticien. Toutefois ce nouveau modèle aura toujours des limites et restera incapable de répondre à des situations qui n'auront pas été identifiées par le directeur et l'informaticien; il ne permet par exemple pas d'intégrer la possibilité qu'un enseignant soit également étudiant, ce qui pourrait poser de sérieux problèmes dans le cadre universitaire qui connaît le statut d'étudiant-assistant.
L'exemple est trivial mais montre néanmoins que les liens entre les tables définissent ce que la base de données relationnelle sera capable de produire ou non. Cette étape de traduction du modèle conceptuel en modèle logique est donc cruciale et une erreur de conception logique impliquera de lourdes opérations de corrections.

Exemple 2 : Formes normales et redondances

Deuxième anecdote : une collègue demande gentiment à Andréas Perret de l'aider à créer une base de données relationnelle à partir de cette table :

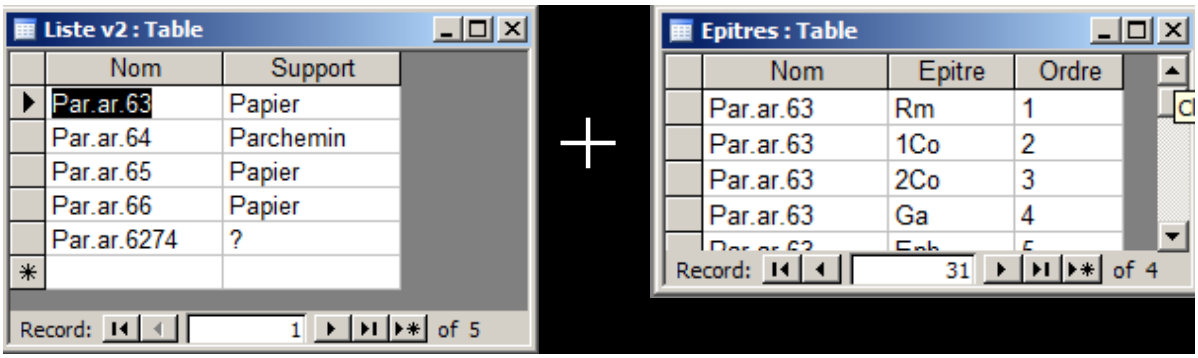
Liste originale : Table

	Nom	Ordre des épitres	Support
	Par.ar.63	Rm;1Co;2Co;Ga;Eph;Phi;Col;1Th;2Th;1Tim;2Tim;Tite;Phm;He	Papier
	Par.ar.64	2Tim;Tite	Parchemin
	Par.ar.65	Rm;1Co;2Co;Ga;Eph;Phi;Col;1Th;2Th;1Tim;Tite;Phm;He	Papier
	Par.ar.66	Rm;1Co;2Co;Ga;Eph;Phi;Col;1Th;2Th;He;1Tim;2Tim;Tite;Phm	Papier
▶	Par.ar.6274	1Tim	?
*			

Record: 5 of 5

Ici, chaque ligne du fichier représente un document disposant d'un nom, d'un ordre d'épitres et d'un support. Du point de vue de la conception d'un modèle de données logique, ce mode d'organisation pose problème puisque – en l'état – il n'est pas possible d'indexer adéquatement les informations contenues dans la colonne « Ordre des épitres ». En effet, chaque ligne de cette

colonne comprend trop d'informations disparates et en langage « base de données relationnelles », on dira qu'elle ne répond pas à la **première forme normale**. Un peu comme pour le premier exemple lorsqu'il s'agissait de traduire des verbes en tables, il faut maintenant décomposer cette colonne de façon à *ordonner* chacune de ses lignes et rendre plus *localisables* les informations qu'elle contient :

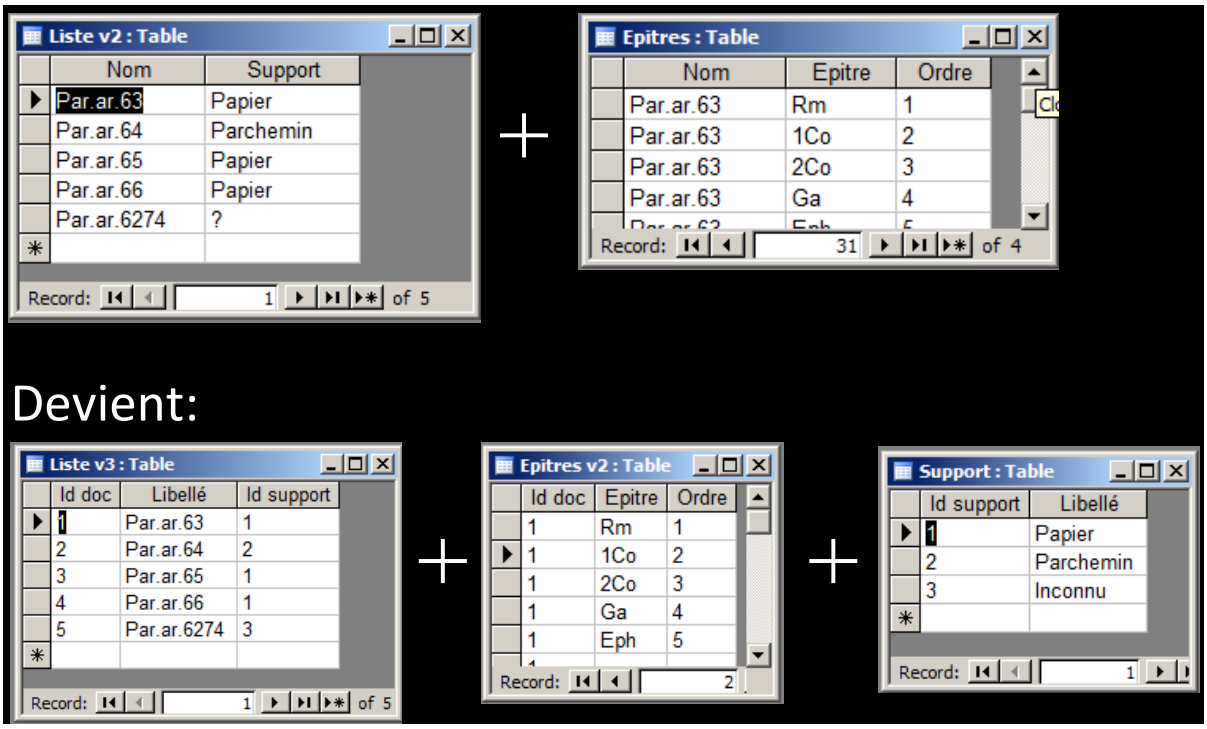


La table initiale est donc décomposée en deux tables provisoires : la première (à gauche) reprenant le nom des parchemins et leur support, la deuxième décomposant la colonne « ordre des épitres » en une colonne « épitre » et une colonne « ordre » *en fonction du nom de chaque document*. En effectuant ce premier ordonnancement, Andréas Perret effectue le **passage en première forme normale**.

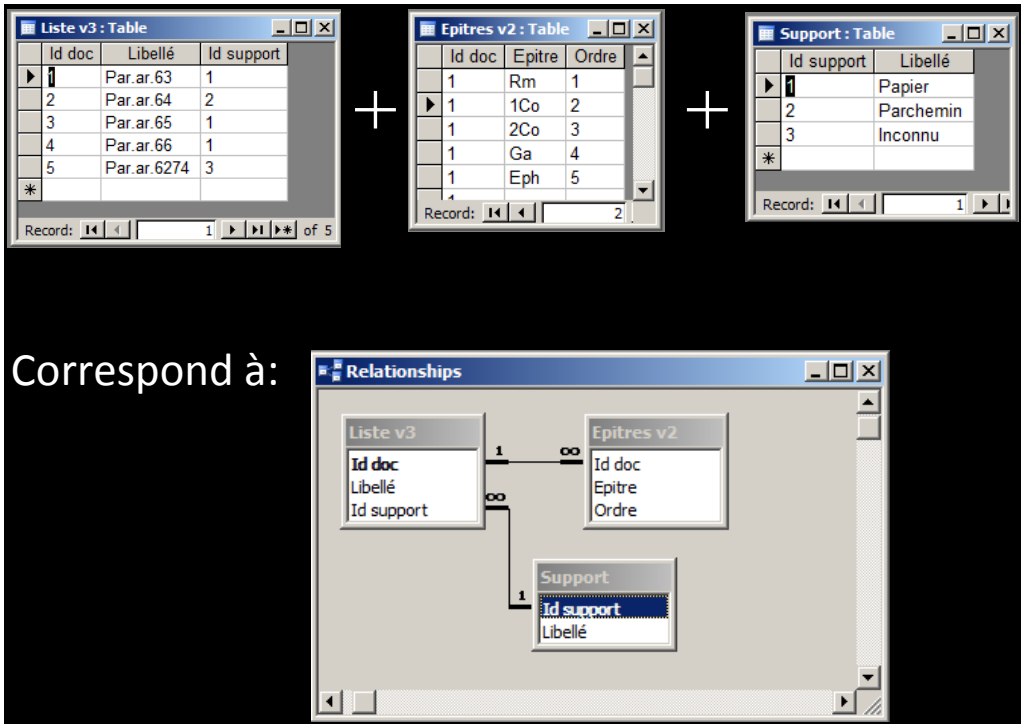
A partir de là, notre conférencier remarque deux problèmes. Le premier est que le nom des documents contient d'avantage d'informations qu'il n'est nécessaire pour identifier le document du point de vue de la base de données: un chiffre croissant (de 1 à 5) suffirait. En quoi est-ce un défaut pour un nom de document de contenir trop d'informations ? Premièrement parce que « Par.ar.63 » et consorts ne sont pas assez ergonomiques pour être, plus tard, traités par ce que l'on appellera ici des *petits nains*. Deuxièmement, un identifiant de cette taille occupera beaucoup trop de place-mémoire (le traumatisme du hangar d'Alexandria est encore frais). Ergonomie et économie : deux raisons de ne pas conserver ces identifiants et les remplacer par d'autres moins volumineux et plus faciles à analyser.

Le deuxième problème est que la table de gauche contient une **redondance** : la colonne « support » contient des entrées « Papier », « Parchemin » et « ? » qui pourront plus tard apparaître *équivoques* et nécessiter ainsi des modifications. Afin de garantir l'univocité de ces informations, leur fiabilité, ainsi qu'une façon aisée et peu coûteuse de les modifier, mieux vaut les fractionner et les indexer à une autre table. Chaque état des supports renverra ainsi à un identifiant et – en plus de contribuer à l'allègement de la base de données – les modifications potentielles de leurs termes s'en verront grandement facilitées :

UNIL | Université de Lausanne
LADHUL - Laboratoire de cultures et humanités digitales de l'UNIL



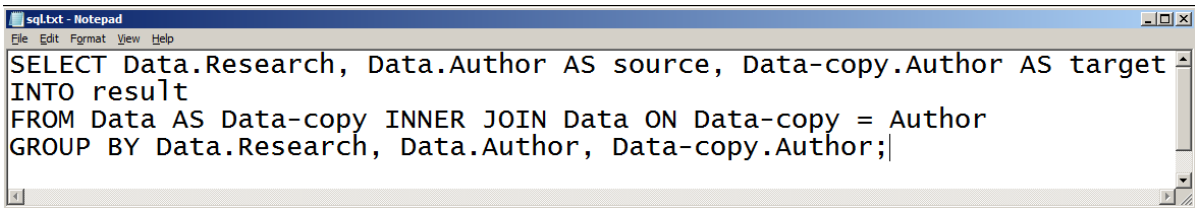
On voit qu'en transformant-décomposant les colonnes trop « touffues » en des tables distinctes mais reliées par des **clés**, les informations s'ordonnancent assez rapidement. A la première table, pour chaque document est associé un identifiant chiffré (la clé), un libellé et un identifiant de support (clé étrangère). A la deuxième table, chaque clé étrangère « id doc » est déclinée en fonction de l'ordre de ses « Epitres », constituant ainsi une **clé composée**. A la troisième table sont précisés les identifiants des supports par un libellé (un peu comme dans la table de gauche). A partir de là, le diagramme de structure des données de cette base de donnée relationnelle peut facilement être exprimé :



Ici, on voit que la table « Liste v3 » est reliée à la table « Support » par sa clé étrangère « Id support ». Cette table « Liste v3 » est également reliée à la table « Epitres v2 » par sa clé « Id doc », commune aux deux tables. Enfin, la table « Support » est liée à la table « Epitre v2 » *via* sa clé « Id support », clé étrangère de la table « Liste v3 » et sa clé « Id doc ». En ce sens, les tables « Epitres v2 » et « Support » sont liées indirectement *via* la clé de la table « Liste v3 ». Rien qu'avec ce travail de mise en ordre basique, on a effectué le **passage en deuxième forme normale** qui vient entourer la première comme une sorte de poupée russe. Plusieurs formes normales déclinant des règles de complexité croissante peuvent ainsi s'emboîter jusqu'à ce que le diagramme corresponde aux usages souhaités et s'approche d'une forme d'orthodoxie du stockage idéal.
La normalisation des informations fait intrinsèquement partie du processus de construction du modèle logique de données. Mais une fois cette tâche terminée, le gros du travail est fait ! Une fois le modèle relationnel créé, l'utilisateur pourra interagir avec la base de données et lui ordonner d'extraire ou de restituer des informations par l'entremise de rapports, d'écrans et de formulaires qui vont produire autant d' "obtenues". Reste encore à savoir comment ordonner au logiciel de gestion de base de données de nous produire les résultats souhaités.

Langage déclaratif et petits nains besogneux

A ce stade, nous avons donc des **tables indexées les unes aux autres par différentes clés**. Mais en elles-mêmes, ces tables indexées ne font rien et ne peuvent produire aucun *résultat*. Pour qu'elles soient capables de participer à la production des résultats souhaités par l'utilisateur, elles doivent se laisser explorer par d'autres éléments « mobiles » pouvant récolter, croiser, triturer leurs données mises en relation. Et c'est précisément là qu'interviennent les **logiciels de gestion de base de données**.
Il est trompeur de dire qu'un utilisateur *travaille* sur des tables de données indexées ; en réalité, il *fait travailler* un logiciel de gestion de base de données (par exemple *Access*, *Oracle* ou *FileMaker*) qui, en se basant sur les résultats souhaités par l'utilisateur, va faire des unions, des appartenances, des exclusions, bref des opérations *entre* les différentes tables grâce l'*indexation* précédemment établie. Comment l'utilisateur du logiciel peut-il transmettre ses désirs à ces êtres machiniques capable d'explorer un paysage fait de tables indexées ? Grâce à une vieille invention assez grandiose : le *Structured Query Language* (SQL), le langage de requête structuré. **Le SQL est un langage de programmation déclaratif** qui décrit ce que l'utilisateur souhaite obtenir sans décrire la manière de l'obtenir. *Fiat lux*, peut importe comment ! Le terme « langage déclaratif » peut paraître mal choisi : c'est bel et bien du *vocatif*, qui instaure ce qu'il énonce !



Exemple d'une requête en langage SQL

Soit. L'utilisateur déclare un résultat mais reste encore à faire effectivement le travail déclaré-souhaité. C'est la mission des « petits nains », travailleurs de l'ombre, sortes de globules rouges assurant la vascularisation électronique de la base de données. Mis à part le fait qu'ils sont petits (et donc peu capables de travailler sur de gros blocs) et aidés dans leur tâche par un **optimisateur de requête** qui va déterminer automatiquement le plus rapide chemin pour satisfaire la requête SQL de l'utilisateur, impossible ici de parler d'eux précisément. Petits travailleurs besogneux, soucieux de répondre à une requête précise et aidés par des algorithmes sophistiqués leur permettant de ne pas trop se perdre dans les dédales de tables et de clés, ils resteront dans ce compte-rendu des être mystérieux (mais indispensable à l'écologie d'une base de données relationnelle).

Solutions bureautiques VS solutions professionnelles

Retour au milieu des années 1990 : afin de remplir plus efficacement son cahier des charges, le responsable de l'unité de planification dont fait partie Andréas Perret demande à la direction du grand hôpital l'octroi de licences pour un logiciel de gestion de bases de données relationnelles tel qu'*Oracle*. La direction refuse, bien aidée par l'avis des informaticiens de la maison qui jugent inapproprié de laisser des non-professionnels utiliser ce genre de solution: « Pas de ça ici ; les informaticiens, c'est nous. Vous, vous utilisez des solutions bureautiques aux fonctionnalités moins poussées ».

Un peu vexés tout de même, Andréas Perret se rabat sur *Access*, la solution bureautique incluse dans la suite *Microsoft Office Professionnel*. Rabattre ? Vraiment ? Pas tant que ça en fait, puisqu'au fur et à mesure de son utilisation d'*Access*, il se rend compte que les fonctionnalités de ce logiciel – méprisé par beaucoup d'informaticiens car ne répondant pas aux standards du domaine – sont en fait extrêmement larges. Et il n'est pas le seul à s'en rendre compte ! D'opérations en opérations ; de tâches en tâches ; de projets en projets : *Access* et les autres solutions bureautiques ont permis aux employés non-initiés mais bricoleurs de répondre rapidement à des problèmes complexes. Ils bricolent, certes, et n'œuvrent certainement pas dans le sens d'une interopérabilité généralisée des bases de données relationnelles (ce qui a le don d'énerver les informaticiens professionnels) mais ils acquièrent une maîtrise, développent des compétences, proposent rapidement des solutions locales et deviennent même difficilement *remplaçables*. C'est là où le bât blesse pour certains employeurs car les rapports constitutifs des solutions bureautiques ont permis à de nombreux non-initiés de maîtriser un outil de travail capable de répondre rapidement à des requêtes auparavant impensables mais dont le substrat repose sur des solutions locales et dépendantes de leurs architectes. Au vu de cet exemple, il semble donc que la non-interopérabilité suggérée par la maîtrise de solutions bureautiques contribue tout de même à un certain *empowerment* des employés non-initiés.

Florian Jaton