



Systèmes de Gestion de Bases de Données

Editeurs successifs :

Jean-Pierre CHEINEY, Philippe PICOUEY, Jean-Marc SAGLIO,
Talel ABDESSALEM

Extraits pour le l'UE INF225

Septembre 2011

TABLE DES MATIERES

Table des Matières	3
ChapitreS 1 & 2 : INTRODUCTION AUX bases de données.....	7
I.1. Pourquoi des bases de données ?	7
I.2. Le principe d'un modèle de données.....	10
I.2.1. Le modèle entité-association	11
I.2.2. Modèles, langages et schémas	13
I.3. Précisions sur les niveaux de description	13
I.4. Du modèle Entité-Relation a UML	15
I.4.1. Entités et individus ou classes d'objets	15
I.4.2. Liens, rôles et cardinalités	16
I.4.3. Associations identifiantes et/ou entités faibles	18
I.4.4. Agrégations.....	19
I.4.5. Généralisation et spécialisation	20
I.4.6. Autres concepts.....	20
I.4.7. Demain, un seul modèle conceptuel?	21
I.5. Les objectifs des SGBD	21
I.5.1. Offrir différents niveaux d'abstraction.....	21
I.5.2. Assurer l'indépendance physique des données	22
I.5.3. Assurer l'indépendance logique des données	22
I.5.4. Contrôler la redondance des données	23
I.5.5. Permettre à tout type d'utilisateur de manipuler des données.....	23
I.5.6. Assurer l'intégrité des données	23
I.5.7. Assurer le partage des données.....	24
I.5.8. Assurer la sécurité des données	24
I.5.9. Optimiser l'accès aux données	24
I.6. Conclusions.....	25
I.7.Bibliographie GENERALE.....	25
Chapitre 2 : Le modèle relationnel	27
III.1. Le modèle relationnel	27
III.1.1. Une première approche du relationnel	28
III.1.2. La construction du modèle relationnel	28
III.1.3. Une manipulation ensembliste des données	31
III.2. L'algèbre relationnelle	32
III.2.1. Les opérateurs ensemblistes	32
III.2.2. Les opérateurs relationnels	35
III.2.3. Opérateurs de base et opérateurs dérivés.....	38
III.2.4. Des exemples de requêtes.....	38
III.2.5. Les arbres algébriques et l'optimisation	39
III.3. Conclusions.....	46
III.4. Références.....	47

Chapitre 4 : Les langages relationnels	49
IV.1. Le standard SQL	49
IV.1.1. Eléments d'un langage pour les SGBD relationnels	50
IV.1.2. Expression des opérations relationnelles	51
IV.1.3. Sous-questions, prédicats quantifiés, composition de questions	55
IV.1.4. Fonctions et groupements	57
IV.1.5. Mises à jour	59
IV.1.6. Le langage de déclaration de données	60
IV.1.7. Intégration d'un langage de manipulation de données dans un langage de programmation.....	61
IV.2. Autres langages relationnels	65
IV.2.1. QUEL.....	65
IV.2.2. Query By Example (QBE).....	65
IV.3. Conclusions.....	67
IV.4. Annexe : sémantique et syntaxe de SQL	67
IV.5. Références.....	69
Chapitre 5 : Bien concevoir une base de données.....	71
V.1. La théorie de la normalisation.....	73
V.1.1. Les dépendances fonctionnelles	74
V.1.2. Dépendance fonctionnelle élémentaire.....	76
V.1.3. Graphe des dépendances fonctionnelles	76
V.1.4. Fermeture transitive	77
V.1.5. Couverture minimale	78
V.1.6. Clé d'une relation	79
V.1.7. Décomposition des relations.....	79
V.2. Les trois premières formes normales.....	80
V.2.1. La première forme normale 1FN	80
V.2.2. La deuxième forme normale 2FN.....	80
V.2.3. La troisième forme normale 3FN	81
V.2.4. Algorithme de décomposition en troisième forme normale	81
V.2.5. Insuffisance de la troisième forme normale	82
V.3. La quatrième forme normale.....	83
V.3.1. Les dépendances multivaluées.....	83
V.3.2. Quatrième forme normale.....	84
V.4. La cinquième forme normale.....	85
V.4.1. Les dépendances de jointure.....	86
V.4.2. Cinquième forme normale	86
V.5. Conclusions.....	87
V.6. Références.....	87
Chapitre 6 : La protection de l'information.....	89
VI.1. Les vues externes	90
VI.1.1. Les objectifs : voir le monde comme il n'est pas !	90
VI.1.2. Le relationnel simplifie les vues	91
VI.2. Manipulation au travers des vues	93
VI.2.1. Consultation au travers des vues	93
VI.2.2. Influence sur l'optimisation des requêtes.....	94

VI.2.3.	Mise à jour à travers les vues.....	96
VI.3.	Confidentialité.....	97
VI.3.1.	Autorisations sur une relation ou sur une vue	98
VI.3.2.	La cession de droits	99
VI.4.	Intégrité sémantique.....	99
VI.4.1.	Les différents types de contraintes	99
VI.4.2.	Comment définir une contrainte d'intégrité ?	101
VI.4.3.	Comment regrouper (classer) les contraintes ?.....	101
VI.4.4.	Contraintes d'intégrités et transactions bases de données.....	101
VI.5.	Conclusions.....	102
VI.6.	Références.....	102

CHAPITRES 1 & 2 : INTRODUCTION AUX BASES DE DONNEES

Les Bases de Données occupent aujourd'hui une place de plus en plus importante dans les systèmes informatiques. Les Systèmes de Gestion de Bases de Données (SGBD) remplacent les anciennes organisations où les données, regroupées en fichiers, restaient liées à une application particulière. Ils assurent le partage, la cohérence, la sécurité d'informations qui, de plus en plus, constituent le cœur de l'entreprise. Des premiers modèles hiérarchique et réseau au modèle relationnel, du mainframe au micro, du centralisé au réparti, les ambitions des SGBD augmentent d'année en année. Mais à quoi peuvent bien ressembler ces logiciels de plusieurs dizaines de milliers de lignes de code qui gèrent avec une relative facilité quelques milliards d'octets d'information ?

I.1. POURQUOI DES BASES DE DONNEES ?

Comme c'est le cas pour de nombreuses innovations technologiques, une importante pression des besoins est à l'origine de l'émergence des Systèmes de Gestion de Bases de Données. Dans l'environnement informatique traditionnel des gros systèmes d'exploitation, le seul mode de gestion de données reste le gestionnaire de fichiers. Les données traitées par une application (gestion de la paie, des stocks, de la comptabilité, des locaux...) demeurent spécifiques à cette application.

L'organisation des données en fichiers telle qu'elle est traditionnellement conçue répond à des besoins précis : les services utilisent des informations le plus souvent distinctes pour des traitements différents. Pour cette raison, chaque équipe de l'entreprise dispose de ses propres fichiers de données où ne figurent que les informations la concernant. La forme sous laquelle cette information est stockée dépend de facteurs nombreux et variés : matériel disponible, bonne volonté et harmonisation des équipes de développement (choix du langage de programmation, choix de la structure de stockage...) ; surtout, les accès à l'information déterminent le plus souvent l'organisation choisie pour les données : le choix des informations regroupées dans un même fichier, le choix de l'organisation du fichier (séquentiel, aléatoire...).

Dans ce contexte, on constate l'apparition de nombreuses difficultés :

Redondance des données

La répétition en différents endroits de données identiques pose de difficiles problèmes lors des mises à jour ; chaque modification doit être répercutée sur toutes les occurrences de l'objet concerné. Par exemple, le changement de nom d'un employé (c'est fréquemment le cas lors des mariages) doit être effectué aussi bien au service du personnel qu'à la comptabilité, etc.

Difficulté d'accès aux données

L'accès aux données nécessite l'intervention d'un informaticien (nécessité de connaître la machine où résident les informations, les structures utilisées pour stocker ces informations, les chemins d'accès disponibles...). La seule manipulation possible d'informations stockées sur fichiers est une manipulation "programmée" ; les informations sont en effet décrites de façon très physique et ne permettent pas un niveau d'abstraction suffisant pour envisager une interrogation directe et interactive par un utilisateur final non spécialiste.

Problèmes de partage des données

Le problème est bien connu des concepteurs de systèmes d'exploitation : comment réagir à plusieurs ouvertures simultanées d'un même fichier. Des problèmes identiques se posent dès que plusieurs utilisateurs désirent effectuer des modifications sur les mêmes données. De surcroît, l'existence de plusieurs occurrences d'une même donnée en plusieurs endroits (plusieurs fichiers différents) provoque une complexité accrue si le système doit garantir la cohérence pour des utilisateurs manipulant la même information au même moment.

Risques d'incohérence

Des liens sémantiques existent très souvent entre les données : le poste d'un employé, figurant dans le fichier du service du personnel, n'est pas sans rapport avec le salaire qui lui est versé par la comptabilité. Pourtant la modification de l'un n'entraîne pas automatiquement celle de l'autre. De telles situations nécessitent souvent force courrier et coups de téléphone ; des incohérences apparaissent trop fréquemment du fait de la difficulté et du coût qu'implique le contrôle de "contraintes d'intégrité" liant les données entre elles.

Problèmes de sécurité

Garantir de la sécurité physique de l'information est une des tâches de base d'un système informatique. Qu'arrive-t-il si une panne interrompt un programme augmentant tous les salaires de 3% ? Combien de modifications ont été prises en compte ; où reprendre le travail ? La présence de la même information en plusieurs exemplaires n'est évidemment pas de nature à faciliter la tâche du SGBD et des programmes de reprise après panne. De même, la confidentialité de l'information est plus facile à assurer avec une seule occurrence de l'information à protéger.

Manque de portabilité des applications

Que se passe-t-il lorsque l'on change de système d'exploitation ou de support de stockage ? Dans la plupart des cas, les applications développées sont si fortement liées aux possibilités propres du système et du gestionnaire de fichiers qu'il est parfois aussi difficile de les actualiser que de les réécrire entièrement. Si la normalisation des langages de programmation est de plus en plus fréquente, les accès aux fichiers dépendent la plupart du temps de la machine ; le manque de séparation entre le code de l'interface et celui du traitement des données interdit de faire une simple modification du code.

Problèmes de maintenance

Enfin, quand la modification de la structure des données manipulées devient nécessaire, il faut assurer le passage des données depuis les anciennes vers les nouvelles structures choisies et presque toujours réécrire complètement les programmes d'application dont la plupart réalisent pourtant la même opération qu'auparavant.

Au total, les organisations traditionnelles des données sous la forme de fichiers liés aux applications présentent de sérieux inconvénients dès qu'un ensemble de programmes manipulent et partagent des informations qui ne sont pas complètement indépendantes. Le défaut de base des organisations traditionnelles provient du fait que *les données et les traitements ne sont pas explicitement séparés*. L'ancienne vision du monde faisait en effet tourner les données autour des traitements (tout comme le soleil tournait autour de la terre !). Aujourd'hui, on sait que le centre du monde est constitué par l'information manipulée (les données dont elle est constituée) : le centre c'est le système d'information de l'entreprise !

Une bonne solution à ces problèmes passe par le regroupement et l'unicité des données, ainsi que par la centralisation des moyens de gestion de ces données. L'administration unique et centralisée des données à l'échelle de l'entreprise garantit la cohérence de l'information et permet l'*indépendance des*

programmes et des données . Elle est mise en œuvre par un Système de Gestion de Bases de Données (SGBD).

L'ensemble de toutes les données peut être regroupé sous l'appellation de *base de données*. Cette base de données représente l'information de l'entreprise. Elle existe indépendamment de l'usage qu'en font les programmes d'application ou les utilisateurs finals. L'information doit donc pouvoir être décrite indépendamment de l'usage qui en est fait. Cette description nécessite un *modèle de données*, c'est à dire un ensemble de concepts et de règles assez généraux et riches de sémantique pour pouvoir donner de façon formelle un *schéma* de la structure permanente de cette information.

Le système logiciel qui met en œuvre le modèle de données (qui stocke les données, gère les accès, les droits des utilisateurs, et plus généralement traite au mieux l'ensemble des problèmes que nous venons d'évoquer) forme véritablement le SGBD.

I.2. LE PRINCIPE D'UN MODELE DE DONNEES

Dès 1965 apparaît l'idée de distinguer les données de leurs traitements. Cela exige une description préalable des données qui doit être fournie par les utilisateurs.

Pour bien comprendre cela, observons le travail que devait réaliser le programmeur et que le système doit maintenant prendre en charge : le programmeur devait transformer en structures informatiques une certaine vision de la réalité (par exemple une personne est stockée sous la forme d'un enregistrement composé d'un champ "numéro de sécurité sociale", d'un champ "nom", d'un champ "prénom" et d'un champ "date de naissance"). C'est à présent le SGBD qui va effectuer ce travail : il recevra en entrée une *description abstraite* des données. Disposant de règles déclarées par l'administrateur, il choisira la façon de les stocker et de gérer les accès.

Les outils offerts par le SGBD pour décrire les données qu'il aura à stocker repose sur un ensemble de règles et de concepts permettant de décrire le monde réel (en fait une toute petite partie du monde réel : celle sur laquelle portent les traitements). Ces règles et concepts constituent un *modèle de données*. Le modèle entité-association peut ici être donné à titre d'exemple.

I.2.1. Le modèle entité-association

Ce modèle de données correspond à une modélisation assez naturelle du monde réel. Le modèle se compose de trois concepts élémentaires : l'entité, l'association et les contraintes portant sur ces associations.

Une entité peut être :

- un objet du monde réel ;
- des données composées ;
- un ensemble d'entités.

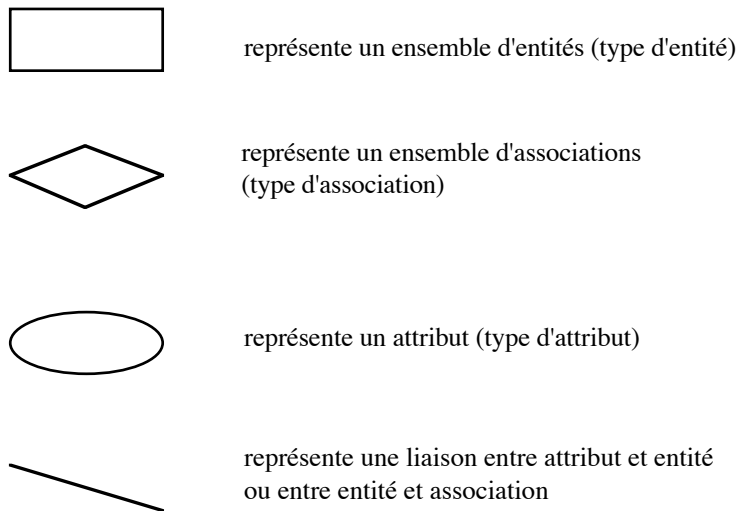
Une association peut être :

- un lien entre entités ;
- un ensemble d'associations.

Une contrainte peut être :

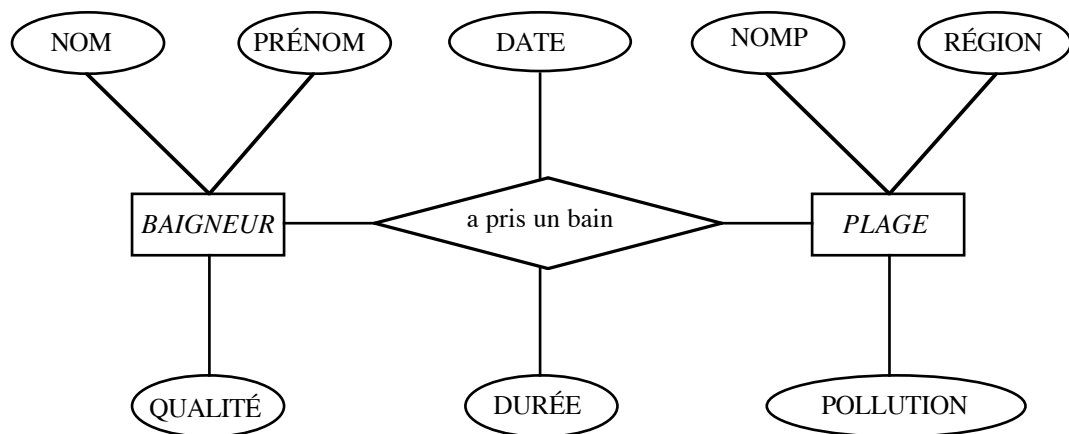
- une propriété précisant le nombre d'entités qu'une association peut unir. On parle alors de cardinalité d'association qui peut être de type 1-1, 1-N, ou N-M ;
- une propriété propre à l'association et sans rapport avec les entités rapprochées par ces associations.

Ce modèle est mis en œuvre par un langage de description. Ce langage est ici graphique :



Exprimons avec ce modèle la réalité suivante : des nageurs, dont les caractéristiques sont un nom, un prénom et une qualité, prennent des bains d'une certaine durée à une certaine date, sur certaines plages dont les caractéristiques sont le nom de la plage, la région et la pollution :

Le schéma entité/association est le suivant :



Les contraintes à définir (dont les cardinalités d'association) posent, par exemple, les problèmes suivants :

- un baigneur peut-il prendre plusieurs bains? sur des plages différentes?
- une ou plusieurs personnes peuvent-elles se baigner sur une plage très polluée ?
- autorise-t-on les mauvais nageurs à se baigner sur une plage des Landes ?

I.2.2. Modèles, langages et schémas

Plus généralement, nous dirons que le modèle de données est un mode de formalisation du monde réel. Il doit permettre la description et la représentation :

- des entités et des données qui les constituent ;
- des liens (association, relations, correspondances) qui les relient ;
- de certaines assertions (propriétés ou contraintes d'intégrité) que doivent vérifier les données de la base.

Définition : modèle de données

un modèle de données est l'ensemble des concepts et des règles de composition qui permettent de décrire les données.

Définition : langage de description

un langage de description est un langage supportant un modèle de données et permettant de définir ces données.

Définition : schéma

Un schéma est la description au moyen d'un langage déterminé d'un ensemble particulier de données.

Un Système de Gestion de Bases de Données est donc un système logiciel qui met en œuvre un modèle de données particulier et qui offre des outils de définition et de manipulation de données à des utilisateurs possédant des connaissances plus ou moins approfondies sur le modèle et sur le logiciel : administrateur, développeurs d'applications ou utilisateurs finals non spécialistes.

Les principaux modèles de données sur lesquels s'appuient les SGBD sont le modèle hiérarchique, le modèle réseau, et le modèle relationnel que nous examinerons ultérieurement.

I.3. PRECISIONS SUR LES NIVEAUX DE DESCRIPTION

La description des données peut s'effectuer à plusieurs niveaux d'abstraction qu'on appelle niveaux de description et de schémas.

Le premier niveau concerne la réalité informatique. C'est l'administrateur de la base de données qui effectue ce travail. Les objets décrits sont ici l'environnement matériel et système (organisation et partitionnement du disque, système d'exploitation...), les fichiers (nom, taille, organisation...), les articles (nom, longueur, placement...), les champs (nom, format, localisation...), les chemins d'accès (clé, lien, type...).

Le second permet la description conceptuelle. La réalité au niveau de toute l'entreprise y est décrite. L'administrateur ou l'utilisateur donne *une vue canonique des données* : c'est de ce schéma, unique, que seront dérivées *différentes vues externes*. Les objets décrits restent ici abstraits : entité (nom, attributs...), association (nom, cardinalité...), contrainte (objet, assertion...), etc.

Enfin le niveau externe permet de décrire plusieurs visions différentes d'une même réalité (le schéma conceptuel); chacune de ces visions correspondra à une application ou à un groupe particulier d'utilisateurs (service du personnel, service de comptabilité, comité d'entreprise...). Le niveau d'abstraction de la description est ici le même que celui du niveau conceptuel.

Des règles de correspondance sont utilisées pour transformer les données d'un niveau de schéma dans un autre. Ces règles sont en général fixées par l'administrateur et permettent de libérer l'utilisateur des contraintes liées à "l'informatique profonde".

Historiquement, c'est en 1971, qu'une recommandation du DBTG-CODASYL (DB Task Group) isolait un niveau interne (spécialisé dans le stockage et l'accès aux informations) et un niveau externe, proche de l'utilisateur, qui gère les rapports avec les programmes d'application. La recommandation de trois niveaux (interne, conceptuel et externe) date de 1979.

Dans cette introduction, nous ne nous préoccupons que peu du niveau interne (voir cependant le chapitre 7 qui présente différentes possibilités d'implantation physique des données dans un SGBD), mais nous pouvons profiter de ce paragraphe pour en dire quelques mots.

Précisons en effet qu'un SGBD gère des milliards d'octets de données et que, pour ce faire, il n'utilise pas les structures de données offertes par les langages de programmation (tels l'article en Pascal ou la structure en C). Il n'utilise pas non plus, en général, les fichiers offerts par les systèmes d'exploitation pour stocker les informations sur disque ; il préfère utiliser ses propres techniques de stockage. Un SGBD réserve une partie du disque et de la mémoire, y manipule des pages mémoire (chargement, déchargement, lecture, écriture) grâce aux primitives de bas niveau offertes par le système

d'exploitation. Sur ces pages mémoire, l'information est vue comme une suite d'octets dont l'organisation est connue du système¹ qui sait ainsi lire les informations stockées.

Dans l'avenir nous n'aborderons plus les détails de structure de cette couche interne, mais il faut garder à l'esprit qu'un SGBD, à son plus bas niveau, ne fait que lire ou écrire des pages mémoire. Il a de telles quantités de données à manipuler qu'il doit mettre en œuvre le maximum de techniques pour diminuer le nombre d'entrées/sorties nécessaires. Ces méthodes, que nous nous bornerons à présenter rapidement dans ces chapitres, s'appellent optimisation de requêtes, méthodes de stockage, etc.

I.4. DU MODELE ENTITE-RELATION A UML

Le modèle "Entité-Association" ("Entity-Relationship" ou ER), présenté plus haut, est contemporain des premiers SGBD. C'est un modèle de conception², généralement utilisé en mode graphique, indépendant des possibilités (logiques et physiques) des SGBD. Pour cette raison il reste très populaire, d'autant que les schémas de données qu'il permet de construire peuvent être traduits presque automatiquement dans les modèles logiques des SGBD des différentes générations : hiérarchique, réseau (CODASYL), relationnel (SQL) et même orienté objet (ODMG).

Il a été au fil du temps "enrichi" (Enhanced => EER) ou "complété" (ERC). Les nouveaux modèles de conception orientés objet (OO) ont toutefois vocation à le remplacer, surtout depuis leur récente³ unification avec l' "Unified Modeling Language" (UML).

Le foisonnement des variantes de l'ER et des modèles OO (OMT, Booch, OOSE...) ont entraîné une certaine confusion chez ceux-là mêmes qui appréciaient la clarté des schémas graphiques produits par l'ER ou ses successeurs. Aussi tenterons-nous, en quelques courts paragraphes, de montrer que pour l'essentiel (les concepts) ils présentent, au delà des différences de notation, une progression continue et sans perte des acquis principaux.

I.4.1. Entités et individus ou classes d'objets

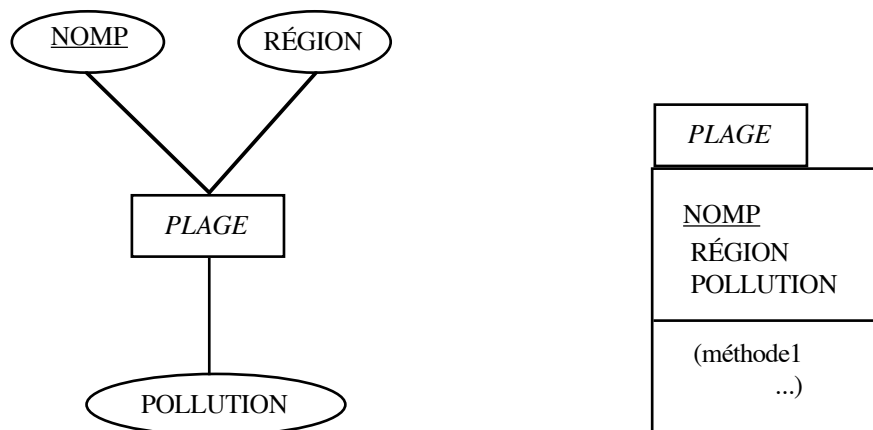
L'*entité* , comme ensemble d'*individus* de même type, est un concept auquel on peut facilement substituer celui de *classe* d'*objets* ayant les mêmes types (ou "structures") de données (et les mêmes

¹nous désignons, ici et dans les pages qui suivent, par "système" le SGBD qui, l'utilisant ou non, masque le système de gestion de fichiers - SGF - du système d'exploitation - OS.

²qualifié aussi de "sémantique"...

³ l'Object Management Group (OMG) en a accepté en 1997 la version 1.0

"méthodes" pour les manipuler). Les individus "instancient" les entités comme les objets instancient les classes. Les individus comme les objets doivent avoir un nom commun (celui de l'entité ou classe) et *un identifiant unique et invariant* de leur naissance à leur mort, que le concept physique de clef sur un article ne suggère qu'imparfaitement (on peut changer une clef pas un identifiant). Ayant le même type ils sont de surcroît décrits par les mêmes variables nommées : *caractéristiques* ou *attributs* (synonymes). Ces variables peuvent être de structures simples ou complexes : n-uplets (variables composites par ex. une Adresse) ou ensembles (variables multivaluées par ex. un Prénom). Les formes graphiques représentant entités ou classes sont différentes mais traduisibles l'une dans l'autre comme illustré ci-dessous par l'exemple de l'entité PLAGE :



N.B. l'attribut souligné est un identifiant externe, il appartient au SGBD de créer le véritable identifiant invariant (appelé pour cette raison identifiant interne ou "surrogate")

I.4.2. Liens, rôles et cardinalités

Entre les objets (ou les entités) peuvent exister des liens nommés porteurs d'une sémantique commune : le rôle. Ils peuvent lier des objets de classes différentes (comme BAIGNEUR et PLAGE, pour exprimer le rôle AIMER ou le rôle SE_BAIGNER ou un autre rôle) ou de même classe (comme BAIGNEUR, pour exprimer un rôle comme PARENT_DE ou comme ENSEIGNE_A). Dans le modèle ER ces liens sont représentés par des associations (il est interdit de relier directement des entités entre elles), même si elles n'ont pas d'attributs propres. Dans le modèle UML, si aucun attribut n'est associable au lien, celui-ci se représente par une simple arc entre les classes qu'il lie, sinon une classe association est ajoutée. Exemple :

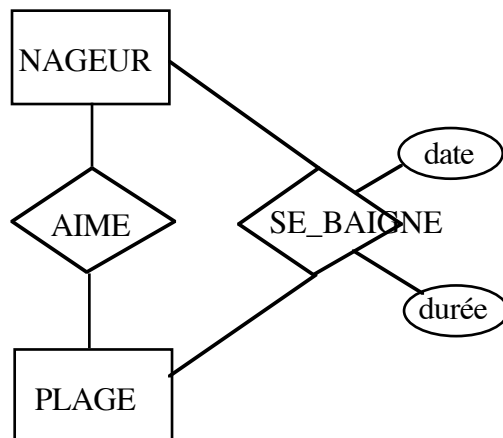


Schéma ER

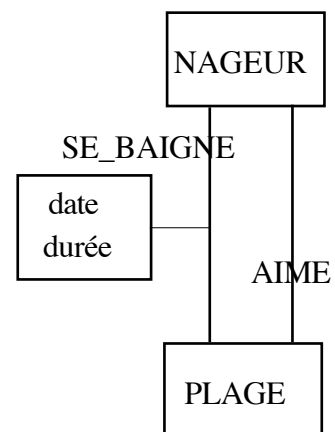
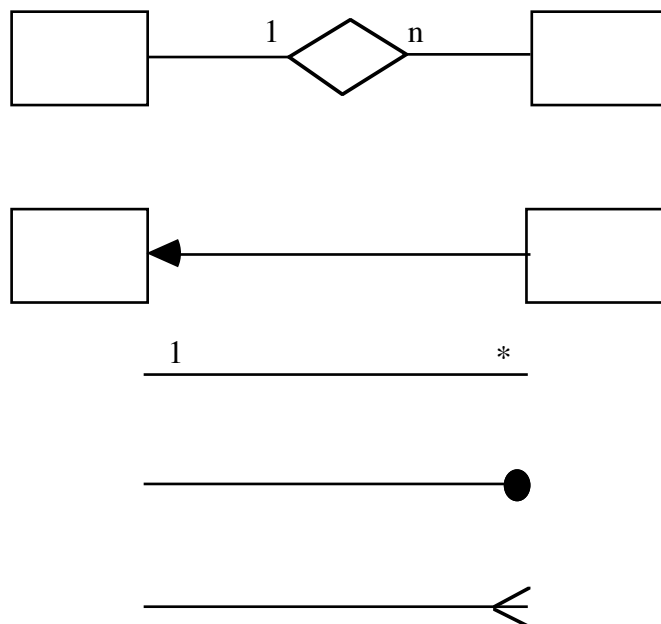


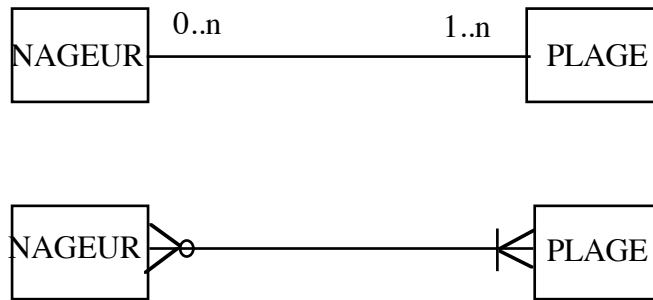
Schéma UML

Les cardinalités de ces liens ou associations spécifient, pour chaque extrémité d'un lien, c.à.d. pour chaque classe liée, le nombre d'objets pouvant être reliés à un même objet de la classe à l'autre extrémité. Les différentes notations ci-dessous sont équivalentes :



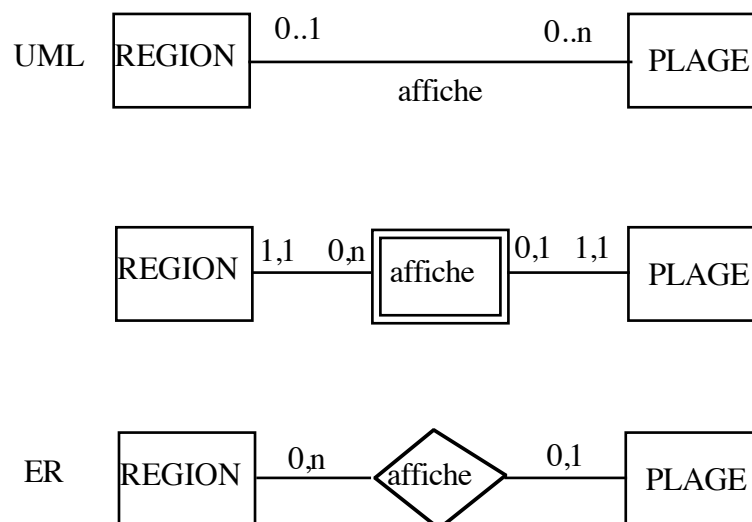
On peut être plus précis en spécifiant des cardinalités minimum et maximum, c.à.d.

au lieu de 1 l'une des paires (0,1) ou (1,1), au lieu de n l'une des paires (0,n) ou (1,n), on peut aussi afficher cette valeur n maximale, par exemple (0,3) ou (1,10)... Les dernières notations de la figure précédente pour l'association - de plusieurs à plusieurs, "n à n" - NAGEUR-aime-PLAGE pourraient être utilisées comme ci-dessous :



(dans la 2e notation le rond est un zéro et la barre verticale un 1)

Attention: dans le modèle ER classique, quand les cardinalités (min,max) sont indiquées, elles le sont généralement à l'inverse de toutes les notations précédentes. En effet les liens directs entre entités étant impossibles et l'association étant toujours considérée comme une classe intermédiaire instanciable, cette inversion s'explique pourvu que l'on passe bien par toutes les étapes de la traduction, ce que nous illustrons ci-dessous pour l'association - de 1 à n - REGION affiche PLAGE :

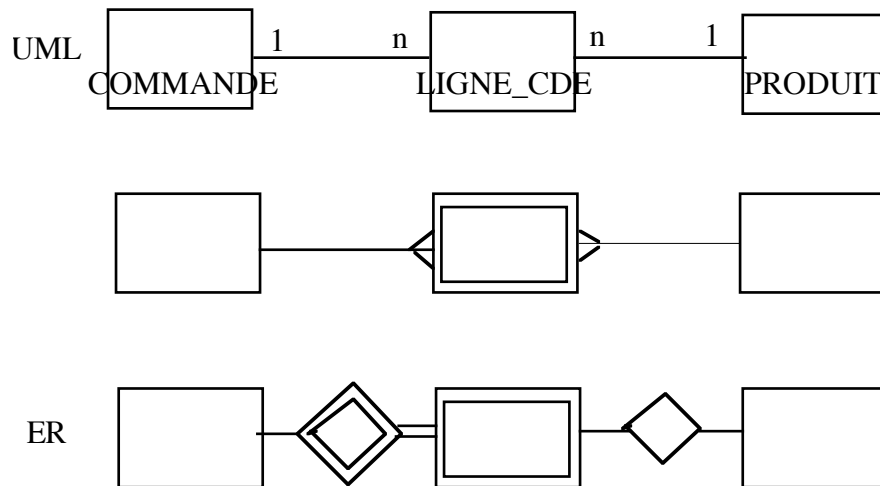


(la raison pour laquelle la "classe association" est notée par un double rectangle sera expliquée plus loin)

I.4.3. Associations identifiantes et/ou entités faibles

Les entités ou les classes s'opposent aux associations ou aux liens en ce qu'elles sont définissables en elles-mêmes. De même les identifiants de leurs individus ou instances sont autonomes. Or il existe de nombreux cas dans les univers à modéliser où certaines entités ne sont définies que *relativement* à d'autres, et les identifiants de leurs individus composés d'un identifiant d' "*individu parent*" suivi d'un identifiant propre en tant qu' "*individu enfant*". Dans ce cas on dit que l'entité correspondante est "faible" et que l'association parent-enfant est "identifiante". Pensez par exemple aux différents

comptes bancaires d'une même client d'une banque, aux différentes personnes à la charge d'un même assuré social, aux différents joueurs d'une même équipe de foot, aux différents bureaux dans un même couloir, etc... La suppression de l'individu parent entraîne celle de tous les individus enfants liés (suppression en "cascade"). Les différentes notations en usage sont illustrées par l'exemple classique ci-dessous :

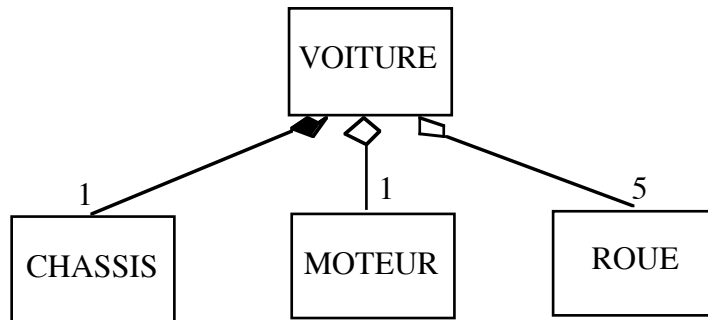


Dans cet exemple les COMMANDE et PRODUIT sont des entités "fortes" par opposition à LIGNE_CDE (rectangle doublé) qui est une entité "faible, dont le lien (trait continu) ou l'association (losange doublé) identifiant à gauche se distingue du lien (trait discontinu) ou de l'association (losange simple) non identifiant à droite.

I.4.4. Agrégations

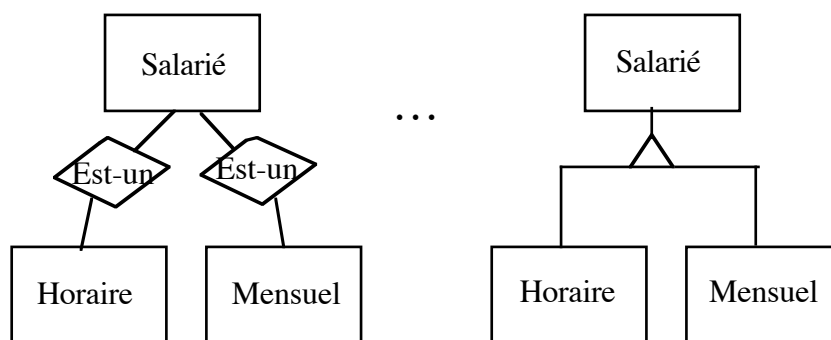
Les agrégations sont des associations ou liens particuliers ayant la sémantique "composé-composant".

La suppression d'un "composé" peut ou non entraîner celle des "composants" ("pièces détachées" irrécupérables ou récupérables!). Réciproquement la suppression d'un "composant" peut ou non entraîner la suppression du "composé" (une voiture sans roue ne change pas d'identité, une voiture avec un nouveau châssis est une autre voiture...). La notation ER classique était insuffisante pour rendre compte de ces surcharges sémantiques. La notation UML en rend compte assez précisément comme illustré ci-dessous :



I.4.5. Généralisation et spécialisation

C'est le couple de concepts le plus important pour enrichir le modèle ER. C'est un des concepts fondamentaux de l'approche orientée objet. Entre une classe, par exemple d'Employés, et une classe plus générale, par exemple de Personnes, le lien ou l'association est du type "EST-UN" (IS-A in english) : un Employé est une Personne. Même si les individus de la classe spécialisée (Employés) peuvent avoir une identification locale à cette sous-classe, il est entendu qu'ils peuvent d'abord être identifiés en tant qu'individus de la super-classe (Personnes). Plus généralement un individu d'une sous-classe "hérite" de tous les attributs de la super-classe. Le lien réciproque de la généralisation est la spécialisation. Selon que les sous-classes d'une même classe forment ou non une partition de cette classe, on dit que la spécialisation est complète ou non. Différentes classes se généralisant dans une classe par l'opération d'union sont appelées des "catégories" pour cette classe. Les notations varient un peu entre les modèles ERC, ..., OMT, UML. Le principal changement est apparu entre l'ER classique qui dessinait autant d'associations que de liens "Est-un" et les modèles suivants qui ont créé un nouveau symbole pour un nœud arborescent liant une super-classe à plusieurs sous-classes.



ER classique

UML

I.4.6. Autres concepts

Il est très vite apparu nécessaire d'ajouter des *contraintes* à une association, comme, par exemple pour une association parent-enfant, d'ordonner les individus liés, ou des contraintes entre associations,

comme, par exemple une contrainte d'exclusion pour le couple d'associations "défendu_par" et "accusé_par", ou, pour les associations de type "Est-un", une contrainte de partitionnement éventuel. Le fait que toute association peut ou non créer une *association inverse* (affirmer qu'un père "connaît" ses enfants, ce n'est pas automatiquement affirmer que chaque enfant "connaît" son père...) est aussi une surcharge que les modèles ajoutent souvent. D'autres concepts, comme celui de "qualificatif" (dans le modèle UML) ou celui d' "assertion" (dans l'ERC) ont aussi été ajoutés pour représenter d'autres contraintes que celles évoquées ci-dessus. Tous les modèles actuellement utilisés butent sur l'infinie richesse du langage naturel pour l'expression des contraintes...

I.4.7. Demain, un seul modèle conceptuel?

Les modèles pour produire des schémas de bases de données ont suivi deux histoires parallèles: les modèles de définition *conceptuelle*, qui cherchent la plus grande richesse sémantique plus que la facilité d'implémentation sur un SGBD, ont suivi une voie propre depuis l'ER de Chen; les modèles d'implémentation *logique*, liés aux structures de stockage et aux langages de manipulation des SGBD, ont connu une histoire faite de continuités et de ruptures (environ tous les 10 ans) du modèle hiérarchique au modèle objet, fortement contrainte par les phénomènes de diffusion et de parc installé. A l'heure de la "convergence relationnel-objet" (et du SQL3) on se plaît à espérer une "convergence logique-conceptuel".

I.5. LES OBJECTIFS DES SGBD

De façon plus formalisée, voici reprise et complétée la liste des objectifs des SGBD.

I.5.1. Offrir différents niveaux d'abstraction

Niveau physique :

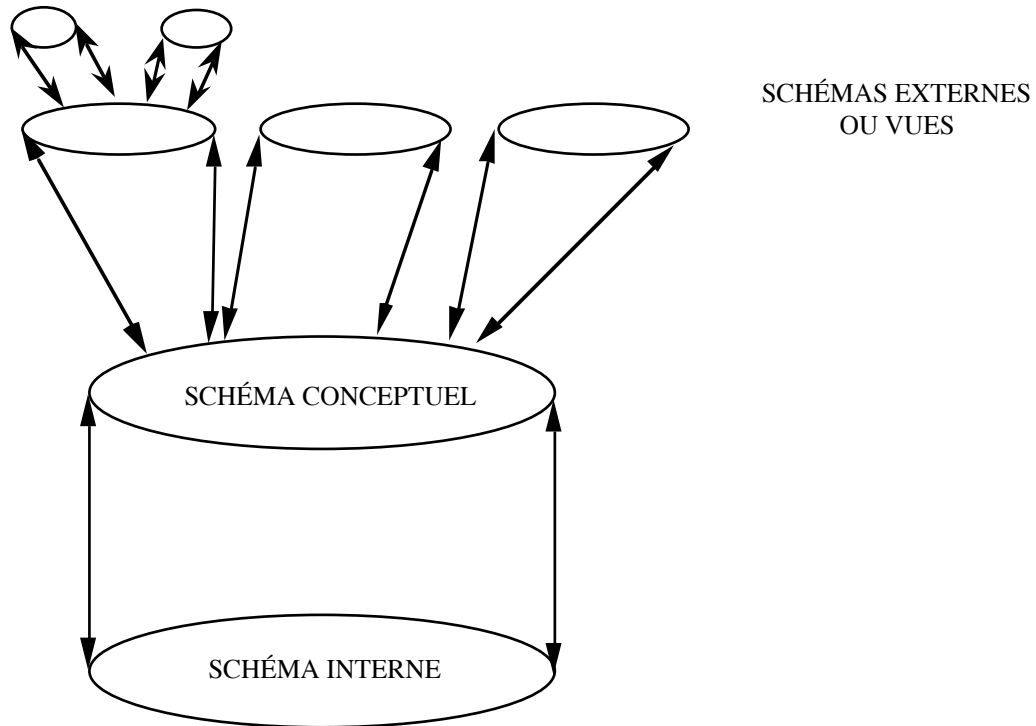
Ce niveau appelé aussi niveau interne, gère le stockage et l'accès aux données. Il n'y a qu'un seul niveau physique par SGBD.

Niveau conceptuel :

C'est à ce niveau, également appelé le niveau logique, que l'on parle de modèle (conceptuel) de données. Ce modèle décrit l'ensemble des données de l'entreprise. La description conceptuelle d'une base de données - BD - est unique.

Niveau externe (vues) :

C'est le plus haut niveau d'abstraction de la base de données. Il est aussi appelé niveau externe et est propre à un utilisateur ou à un groupe d'utilisateurs et ne lui présente qu'une vue partielle de la réalité : celle qui intéresse son service. Il y a bien entendu plusieurs vues d'une même BD.



I.5.2. Assurer l'indépendance physique des données

Le but est ici de permettre à l'utilisateur du niveau conceptuel d'ignorer la structure du niveau physique. Cela nécessite une transformation entre niveau logique et physique, mais présente deux avantages considérables :

- les programmes d'applications sont plus simples à écrire, du fait de ne pas avoir à manipuler des entités complexes (structures d'enregistrement, méthodes d'accès...) ;
- dans le cas d'une modification des caractéristiques du niveau physique, les applications n'ont pas à être modifiées.

I.5.3. Assurer l'indépendance logique des données

Le but est ici de permettre à l'utilisateur du niveau vue (typiquement les programmeurs d'application ou les utilisateurs finals) d'ignorer la structure du niveau conceptuel. Cela nécessite une transformation entre niveau externe et conceptuel, mais offre là encore deux avantages de poids :

- les programmes d'applications du niveau externe n'ont pas à avoir la vision globale de toute l'entreprise. Ils agissent à partir des vues ;

- les applications du niveau externe, en cas d'une modification du schéma au niveau conceptuel, ne sont à réécrire que si cette modification entraîne celle de la vue, ce qui est rarement le cas.

I.5.4. Contrôler la redondance des données

Un des objectifs de base des SGBD, la suppression de la redondance des données, vise à garantir la cohérence de l'information et à simplifier les mises à jour. Cependant, la redondance est parfois nécessaire pour garantir la fiabilité et les performances, ou pour la répartition des données.

Les données sont réparties quand plusieurs SGBD situés sur des sites distincts reliés par un réseau partagent les informations dont ils disposent. Posséder certaines informations sur tous les sites est alors parfois indispensable (par exemple la description des données présentes sur les différents sites pour savoir où, comment, et dans quel ordre aller les chercher en cas de besoin).

En conséquence, la redondance anarchique des données doit être éliminée et la redondance existante doit être contrôlée en propageant la mise à jour d'une donnée redondante.

I.5.5. Permettre à tout type d'utilisateur de manipuler des données

Le but est d'offrir aux différents types d'utilisateurs des moyens d'accès à la base adaptés à leurs besoins et à leurs connaissances. Nous devons ainsi distinguer :

- un ou plusieurs administrateur de la base qui doivent pouvoir décrire les données aux niveaux physique (administrateur BD et ingénieur système) et conceptuel (administrateur BD et concepteur) ;
- un ou plusieurs développeur d'applications qui écrivent, à partir du niveau conceptuel ou des niveaux externes, des programmes d'application pour eux-mêmes ou pour les utilisateurs finals ;
- un ou plusieurs utilisateur final ont besoin d'un langage simple (si possible proche du langage naturel) pour manipuler les données de manière interactive ou à partir de programmes d'application.

I.5.6. Assurer l'intégrité des données

L'intégrité logique de l'information est souvent vérifiée par les programmes d'applications dans les organisations traditionnelles à bases de fichiers. Dans une approche base de données, elle fait partie de la description de la réalité conceptuelle du système d'information. La vérification de l'intégrité est une composante du modèle de données et une tâche du SGBD qui le met en œuvre. L'intégrité sémantique correspond à des règles explicitant des contraintes du monde réel.

Nous pouvons donner comme exemple :

- il est interdit et impossible de se baigner sur une plage très polluée ;
- un baigneur ne peut pas se baigner sur une plage non recensée ;
- un baigneur ne peut pas se baigner à Binic en mars.

Toute requête de mise à jour des données (insertion, modification ou suppression) ne respectant pas l'ensemble des contraintes d'intégrité doit être rejetée par le SGBD.

I.5.7. Assurer le partage des données

Composant transactionnel d'un système informatique, un SGBD doit permettre le partage des données entre différents utilisateurs et applications. Un utilisateur n'a pas à se demander si quelqu'un d'autre travaille sur les mêmes informations au même moment et doit pouvoir accéder aux données en consultation ou en mise à jour comme s'il était seul : le système doit gérer les conflits, en refusant ou en retardant éventuellement un ou plusieurs accès.

I.5.8. Assurer la sécurité des données

Cela consiste à protéger les données contre les pannes et à refuser les accès aux personnes non autorisées. Le système doit présenter un mécanisme de vérification des droits d'accès aux objets de la base. Il doit garantir des reprises après panne en restaurant la base de données dans le dernier état cohérent avant la panne. La fiabilité est traditionnellement sur les gros systèmes mise en œuvre des techniques très sophistiquées. La qualité de l'implantation de ces techniques a d'importantes conséquences sur les performances en cas d'utilisation intensive.

I.5.9. Optimiser l'accès aux données

En permettant aux utilisateurs d'ignorer les structures physiques et les chemins d'accès à l'information, le SGBD prend à sa charge un lourd travail d'optimisation. En utilisant les meilleurs chemins d'accès, mais aussi le parallélisme ou des algorithmes de recherche sophistiqués, il permettra de minimiser le volume des données accédées et le temps d'exécution des questions.

I.6. CONCLUSIONS

Cette introduction nous a permis de présenter les motivations et les buts des Systèmes de Gestion de Bases de Données. Placés ici sur un pied d'égalité, les différents objectifs sont atteints à des degrés divers : les aspects transactionnels (concurrency, sécurité physique,...) sont bien implantés et assurent aujourd'hui une efficacité très importante aux systèmes utilisés en production et destinés à écouler un nombre important de transactions par seconde ; les indépendances physique - conceptuel / interne - et logique - externe / conceptuel - sont bien assurées par les systèmes relationnels, mais mal ou pas du tout dans les produits s'appuyant sur les modèles réseau ou hiérarchique ; la détermination par les systèmes des meilleurs algorithmes et chemins d'accès aux données ont fait faire des progrès considérables aux méthodes d'accès des systèmes informatiques modernes ; la vérification automatique des contraintes d'intégrité générales est, en revanche, encore peu implantée dans les produits.

I.7.BIBLIOGRAPHIE GENERALE

- | | |
|-----------------------|---|
| S. ABITEBOUL, R.HULL | FUNDATIONS OF DATABASES, |
| AND V.VIANU | ADDISON-WESLEY, 1995 |
| M. ADIBA ET C.DELOBEL | BASES DE DONNEES ET SYSTEMES RELATIONNELS, |
| | DUNOD, 1982 |
| C.J. DATE | AN INTRODUCTION TO DATABASE SYSTEMS, |
| | ADDISON-WESLEY, VOLUME 1, 4TH EDITION, 1987, |
| | VOLUME 2, 1983 |
| C.J. DATE | A GUIDE TO THE SQL STANDARD |
| | ADDISON-WESLEY, 1987 |
| R. ELMASRI | |
| AND S.B. NAVATHE | FUNDAMENTALS OF DATABASES SYSTEMS |
| | ADDISON-WESLEY, WORLD STUDENT SERIES, 2ND ED, |
| | 1994 |
| G. GARDARIN | BD: LES SYSTEMES ET LEURS LANGAGES, |
| | EYROLLES, 3E EDITION, 1985 |
| G. GARDARIN | MAITRISER LES BD: MODELES ET LANGAGES, |
| | EYROLLES, 1993 |
| J.MELTON AND A.SIMON | UNDERSTANDING THE NEW SQL: A COMPLETE GUIDE, |
| | MORGAN KAUFMANN, 1993 |
| J.D. ULLMAN | PRINCIPLES OF DATABASE SYSTEMS, |

COMPUTER SCIENCE PRESS, 1982

G. VOSSEN

DATA MODELS, DB LANGUAGES AND DBMS, ADDISON-
WESLEY, 1991

CHAPITRE 2 : LE MODELE RELATIONNEL

Début des années 70, le modèle relationnel fait son apparition [Codd 70]. La recherche se passionne : impossible de nier les progrès apportés concernant la représentation et la manipulation des données par les systèmes. Dix ans passent, les spécialistes déchantent : ce top-model engendre en définitive des systèmes commerciaux bien moins performants que leurs concurrents fondés sur les modèles réseau ou hiérarchique. Deux ans plus tard et voilà que les produits relationnels peuvent prétendre relayer les "vieux" systèmes. Leurs apports sont fondamentaux : les nouvelles fonctionnalités permettent un confort d'utilisation sans précédent. Les systèmes commerciaux s'emparent des concepts de ce nouveau modèle. Celui-ci, désormais, s'impose.

Mais quels sont ces concepts, leurs avantages, leurs limites ? Certains sont déjà bien répandus. Ce sont les notions de base que nous détaillerons dans la première partie : domaine, relation, attribut ; puis la manipulation ensembliste des relations par les opérateurs de l'algèbre relationnelle (deuxième partie) sur lesquels sont construits des langages non procéduraux comme SQL (chapitre 4) ; l'importante base théorique du modèle enfin fournit des méthodes pour la conception des bases de données (chapitre 5). D'autres concepts sont toutefois moins connus qui constituent pourtant un progrès essentiel : les vues relationnelles permettent à chaque utilisateur de personnaliser sa vision des données et les contraintes d'intégrité complètent la description de l'information (chapitre 6). Les chapitres suivants présentent chacun des ressorts qui font du modèle relationnel la référence obligée en matière de gestion de bases de données.

III.1. LE MODELE RELATIONNEL

Le modèle relationnel, contrairement à ceux présentés dans le chapitre précédent, ne manipule pas des structures de données figées, mais des valeurs : aucun chemin d'accès n'est préalablement défini (on ne parlera désormais plus de déplacements ou de langages navigationnels), toute manipulation des données est désormais possible. L'important bagage théorique, et la vision tabulaire des informations, qui est agréable à l'utilisateur, assurent le succès du modèle relationnel.

III.1.1. Une première approche du relationnel

Nous présentons dans ce paragraphe un moyen simple de visualiser la structure de base du modèle relationnel : la relation. Nous décrivons ci-dessous une extension de la base des nageurs sous la forme de tableaux de colonnes de valeurs typées et nommées.

La relation des nageurs :

NAGEURS	NN	NOM	PRENOM	QUALITE
	100	Plouf	Jean	Mauvaise
	110	Coule	Paul	Excellente
	120	Brasse	Jean	Bonne

La relation des plages :

PLAGES	NP	NOMP	TYPE	REGION	POLLUTION
	110	Trégastel	sable	Bretagne	absente
	119	Nice	galets	Côte d'Azur	forte
	107	Oleron	sable	Atlantique	moyenne
	118	Binic	sable	Bretagne	forte

La relation des baignades :

BAIGNADES	NN	NP	DATE	DUREE
	110	118	14/07/89	2
	110	118	15/07/89	10
	120	119	12/07/89	120

III.1.2. La construction du modèle relationnel

Nous allons dans ce paragraphe introduire les notions utilisées dans la construction théorique du modèle relationnel: domaine, relation, tuple, attribut, schéma.

La notion de domaine

Définition : domaine

Un domaine est un ensemble de valeurs (distinctes).

Cette définition correspond à l'ensemble des valeurs que peut prendre une certaine manifestation du monde réel. Elle s'apparente souvent à un type (entier, réel), mais peut également être hétéroclite (date). Dans l'exemple précédent, un domaine est l'ensemble des valeurs d'une colonne d'un tableau : {'Mauvaise', 'Excellente', 'Bonne'} est un domaine.

Exemples de domaines :

- l'ensemble des entiers est un domaine ;
- {3, 5.7, 124} est un domaine ;
- $[-3, 4] \cup \{5, 7\}$ est un domaine ;
- ('Jean', 'Paul', 'Louis', 'Arthur') est un domaine ;
- (14/07/89, 15/07/89, 15/07/89) est un domaine ;
- Tout ensemble de valeurs est un domaine.

Produit cartésien de plusieurs domaines :

Définition : produit cartésien

Le produit cartésien d'un ensemble de domaines D_1, D_2, \dots, D_n , noté $D_1 \times D_2 \times \dots \times D_n$, est l'ensemble de n-uplets (ou tuples) $\langle v_1, v_2, \dots, v_n \rangle$ tels que $v_i \in D_i$.

Exemple :

Réalisons le produit cartésien des domaines suivants, (Plouf, Coule, Brasse), (Jean, Paul).

Plouf	Jean
Plouf	Paul
Coule	Jean
Coule	Paul
Brasse	Jean
Brasse	Paul

Comme on le voit, si un domaine représente l'ensemble des valeurs possibles d'une manifestation du monde réel (exemples : les cheveux sont blonds, noirs, bruns ou blancs ; les âges sont compris entre 0 et 130, etc.), le monde réel "possible" est constitué par le produit cartésien des domaines et le monde réel "réel" est forcément un sous ensemble du produit cartésien.

La notion de relation

Définition : relation

une relation est un sous-ensemble du produit cartésien d'une liste de domaines.

Exemple :

la relation NAGEURS est un sous-ensemble du produit cartésien des domaines suivants :
(100, 110, 120), (Plouf, Coule, Brasse), (Jean, Paul), (Mauvaise, Bonne, Excellente).

La notion de n-uplet

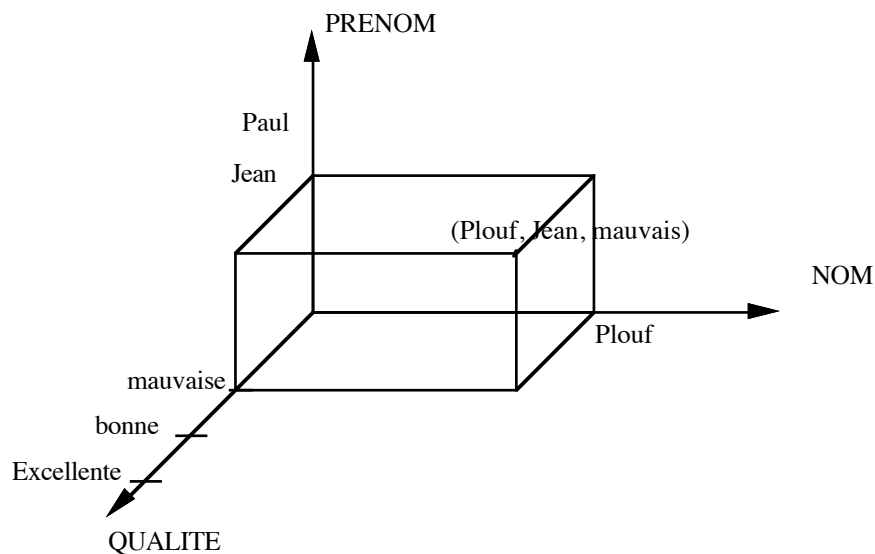
Définition : n-uplet ou tuple

un n-uplet (élément) correspond à une ligne d'une relation.

Une autre représentation des relations

Une autre façon de considérer une relation à N domaines D_1, D_2, \dots, D_N est de représenter un espace à N dimensions. Dans cet espace chaque domaine correspond à l'une des dimension, chaque *n-uplet* ou *tuple* correspond à un point de l'espace.

Exemple



La notion d'attribut

Définition : attribut

Un attribut est le nom donné à une colonne d'un tableau représentant une relation.

Exemple :

les attributs de la relation PLAGES sont NP, NOMP, REGION, TYPE et POLLUTION.

La notion de schéma

Définition : schéma d'une relation

Le schéma d'une relation est composé de son nom suivi du nom de ses attributs et de leurs domaine :

$R(A_1 \subset D_1, A_2 \subset D_2, \dots, A_N \subset D_N)$.

Lorsque le choix des domaines est évident, on simplifie l'écriture de la façon suivante :

$R(A_1, A_2, \dots, A_N)$.

Exemple :

PLAGES (NP, NOMP, TYPE, REGION, POLLUTION).

Définition : schéma d'une base de données relationnelle

Le schéma d'une base de données relationnelle est l'ensemble des schémas des relations qui la composent..

Une base de données relationnelle est constituée par l'ensemble des tuples de toutes les relations définies dans le schéma de la base.

III.1.3. Une manipulation ensembliste des données

Nous avons étudié au chapitre précédent les premiers modèles de SGBD. Que ce soit le segment dans le modèle hiérarchique ou l'article dans le modèle réseau, la manipulation des informations s'effectue enregistrement par enregistrement.

Dans un système relationnel, les informations ne sont pas forcément repérées individuellement ; on sait appliquer le même traitement à un ensemble d'enregistrements caractérisés, non par la liste des identifiants individuels, mais par le critère que vérifie chacun des enregistrements qui composent l'ensemble. On dit que la manipulation est ensembliste.

En particulier, pour rechercher des tuples, il suffit de préciser un critère de sélection ; le système déterminera l'ensemble des tuples satisfaisant ce critère et rendra un résultat. Les tuples de ce résultat, extraits de relations de la base, constituent eux-même une relation qu'il sera ainsi aisé de conserver, si nécessaire, pour le plus grand confort de l'utilisateur.

Par exemple, pour connaître les dates et durées des bains pris par Paul Coule ainsi que les noms des plages, on obtient le résultat suivant, qui est une relation RESULTAT à trois attributs NOMP, DATE, DUREE :

RESULTAT	NOMP	DATE	DUREE
	Binic	14/07/89	2
	Binic	15/07/89	10

Typiquement, on peut classer les requêtes en deux grandes catégories : la mise à jour de tuples dans une relation et la recherche de tuples vérifiant une certaine condition. Eventuellement, ces deux types de requêtes peuvent être combinés.

La manipulation ensembliste est très utile en mise à jour. Elle permet, par exemple, de modifier directement la qualité de tous les nageurs ayant pris un bain sur la plage de Binic le 14 juillet 1989,

sans avoir à déterminer préalablement la liste des nageurs qui présentent cette caractéristique. Cette puissance d'expression explique largement le succès du relationnel.

Les insertions/suppressions sont réalisées à l'aide de relations temporaires internes et d'opérateurs ensemblistes d'union et de différence (détails au paragraphe suivant). La combinaison de ces opérateurs et des opérateurs relationnels de sélection sur des critères de recherche facilite sensiblement les opérations de modification. C'est le cas en particulier des modifications calculées, c'est-à-dire des modifications portant sur un ensemble de tuples résultant eux-mêmes d'une sélection.

Exemples

- insertion ou suppression de Paul Brasse, mauvais nageur, qui a pris un bain de 2 minutes le 14/07/1989 sur la plage de sable très polluée de Binic, en Bretagne.
- recherche des noms des nageurs ayant pris des bains de plus d'une minute en Bretagne.
- suppression de tous les nageurs ayant pris, en février, des bains de plus de 2 minutes en Bretagne (hydrocution ?).

Pour manipuler ces relations, nous avons besoin d'un langage adapté dont la particularité est de savoir manipuler aisément ces tableaux de données. Ce langage constitue l'algèbre relationnelle.

III.2. L'ALGÈBRE RELATIONNELLE

L'algèbre relationnelle est le langage interne d'un SGBD relationnel. Elle se compose d'opérateurs de manipulation des relations. Ces opérateurs sont regroupés en deux familles : les opérateurs ensemblistes et les opérateurs relationnels. Chacune de ces familles contient quatre opérateurs.

III.2.1. Les opérateurs ensemblistes

L'union

Noté $R \cup S$, où R et S représentent deux relations de même schéma, cet opérateur permet de réaliser l'insertion de nouveaux tuples dans une relation. Par exemple, ajouter à la relation permanente RP...

RP	NP	NOMP	TYPE	REGION	POLLUTION
	110	Trégastel	sable	Bretagne	absence
	119	Nice	galets	Côte d'Azur	forte
	107	Oleron	sable	Atlantique	moyenne
	118	Binic	sable	Bretagne	forte

... la relation de travail RT...

RT	NP	NOMP	TYPE	REGION	POLLUTION
	110	Trégastel	sable	Bretagne	absence
	115	Deauville	sable	Normandie	forte
	117	Trouville	sable	Normandie	forte

... on obtient la nouvelle relation RESULTAT = $RP \cup RT$...

RESULTAT	NP	NOMP	TYPE	REGION	POLLUTION
	110	Trégastel	sable	Bretagne	absence
	119	Nice	galets	Côte d'Azur	forte
	107	Oleron	sable	Atlantique	moyenne
	118	Binic	sable	Bretagne	forte
	115	Deauville	sable	Normandie	forte
	117	Trouville	sable	Normandie	forte

Remarquons que la manipulation est réellement ensembliste : les éléments qui pourraient être dupliqués (insertion de la plage de Trégastel qui figure déjà dans la relation permanente) ne le sont pas.

L'intersection

Opérateur noté $R \cap S$, où R et S représentent deux relations de même schéma, qui permet de retrouver les tuples identiques dans deux relations .

La différence

Opérateur noté $R - S$, où R et S représentent deux relations de même schéma, qui permet de réaliser la suppression de tuples dans une relation. Par exemple, retrancher à la relation permanente RP...

RP	NP	NOMP	TYPE	REGION	POLLUTION
	110	Trégastel	sable	Bretagne	absence
	119	Nice	galets	Côte d'Azur	forte
	107	Oleron	sable	Atlantique	moyenne
	118	Binic	sable	Bretagne	forte
	115	Deauville	sable	Normandie	forte
	117	Trouville	sable	Normandie	forte

... La relation de travail RT...

RT	NP	NOMP	TYPE	REGION	POLLUTION
	110	Trégastel	sable	Bretagne	absence
	107	Oleron	sable	Atlantique	moyenne

... Pour obtenir la relation RESULTAT = RP - RS...

RESULTAT	NP	NOMP	TYPE	REGION	POLLUTION
	119	Nice	galets	Côte d'Azur	forte
	118	Binic	sable	Bretagne	forte
	115	Deauville	sable	Normandie	forte
	117	Trouville	sable	Normandie	forte

Le produit cartésien

Si R1 et R2 sont des relations de schémas respectifs (att_{1,1}, att_{1,2},...,att_{1,N}) et (att_{2,1}, att_{2,2},..., att_{2,M}) contenant respectivement T₁ et T₂ tuples, alors la relation R = R₁ X R₂ résultant du produit cartésien des deux relations a pour schéma (att_{1,1}, att_{1,2},...,att_{1,N}, att_{2,1}, att_{2,2},..., att_{2,M}) et contient T₁ * T₂ tuples obtenus par concaténation des T₁ tuples de R₁ et des T₂ tuples de R₂.

III.2.2. Les opérateurs relationnels

La restriction

Noté $\sigma_E(R)$ où E exprime un prédicat sur une relation R, cet opérateur permet de ne conserver, dans une relation, que les tuples dont les attributs vérifient une certaine condition. Par exemple, la relation permanente RP...

RP	NP	NOMP	TYPE	REGION	POLLUTION
	110	Trégastel	sable	Bretagne	absence
	119	Nice	galets	Côte d'Azur	forte
	107	Oleron	sable	Atlantique	moyenne
	118	Binic	sable	Bretagne	forte
	115	Deauville	sable	Normandie	forte
	117	Trouville	sable	Normandie	forte

... réduite aux plages à forte pollution (le prédicat E se note POLLUTION = 'forte') donne la relation RESULTAT...

RESULTAT	NP	NOMP	TYPE	REGION	POLLUTION
	119	Nice	galets	Côte d'Azur	forte
	118	Binic	sable	Bretagne	forte
	115	Deauville	sable	Normandie	forte
	117	Trouville	sable	Normandie	forte

La projection

Noté $\pi_{A_1, \dots, A_N}(R)$ où A_1, \dots, A_N représentent des attributs particuliers de la relation R, cet opérateur permet de ne conserver que certains attributs d'une relation. Par exemple, les différentes régions et leurs niveaux de pollution sont obtenus par projection de la relation PLAGES sur les attributs REGION et POLLUTION...

RESULTAT	REGION	POLLUTION
	Bretagne	absence
	Côte d'Azur	forte
	Atlantique	moyenne
	Bretagne	forte
	Normandie	forte

Notons l'aspect ensembliste de cette opération qui ne duplique pas le tuple ('Normandie', 'forte') qui figurait en double dans la relation intermédiaire résultat d'une projection brutale de la relation initiale.

La jointure

Définition :

La jointure de deux relations R et S selon une qualification multi-attributs Q est l'ensemble des tuples du produit cartésien $R \times S$ satisfaisant la qualification Q.

Notation

Il existe plusieurs notations possibles de cet opérateur. Les plus répandues sont :

- JOIN (R, S/Q) ;
- $R \bowtie_Q S$

équi-jointure

La qualification Q consiste simplement en l'égalité entre différents attributs des deux relations. C'est le cas le plus courant de jointure.

Exemple :

Pour connaître les bains pris par les différents nageurs, il suffit d'effectuer l'équi-jointure entre les deux relations NAGEURS et BAINADES sur chacun de leur attribut NN

RESULTAT	NOM	PRENOM	QUALITE	NN	NN	NP	DATE	DUREE
	Coule	Paul	Excellente	110	110	118	14/07/89	2
	Coule	Paul	Excellente	110	110	118	15/07/89	10
	Brasse	Jean	Bonne	120	120	119	12/07/89	120

dans le cas d'une équi-jointure, on peut éliminer la colonne en double ; on parle alors d'équi-jointure naturelle.

Exemple d'inéqui-jointure :

Quels sont les nageurs ayant pris des bains sur une plage de numéro NP supérieur à leur propre numéro d'identification NN ?

RESULTAT	NOM	PRENOM	QUALITE	NN	NP	NN	DATE	DUREE
	Coule	Paul	Excellente	110	118	110	14/07/89	2
	Coule	Paul	Excellente	110	118	110	15/07/89	10

Autre exemple de jointure :

Quels sont les nageurs ayant pris des bains de durée égale à leur numéro ?

RESULTAT	NOM	PRENOM	QUALITE	NN	DUREE	NN	NP	DATE
	Brasse	Jean	Bonne	120	120	120	119	12/07/89

La division

Noté $R \div S$ où R et S désignent deux relations, cet opérateur, plus complexe, permet de connaître toutes les valeurs d'un domaine qui sont en correspondance avec toutes les valeurs d'un autre domaine. Par exemple, existe-t-il des nageurs qui ont pris des bains à la fois le 14/07/89 et le 15/07/89 ? Pour le savoir, il suffit d'effectuer la division de la relation Baignade projetée sur (NN, DATE) (notée RES1) par la colonne DATE restreinte aux 14/07/89 et au 15/07/89 :

BAIGNADES	NN	NP	DATE	DUREE
	110	118	14/07/89	2
	110	118	15/07/89	10
	120	119	12/07/89	120

RES1	NN	DATE
	110	14/07/89
	110	15/07/89
	120	12/07/89

DATES	DATE
	14/07/89
	15/07/89

RESF	NN
	110

Les tuples du résultat RESF (NN), concaténés à tout tuple de DATES (DATE) donne un tuple de RES1 (NN, DATE). Ainsi, l'ensemble des tuples résultats du produit cartésien de RESF et DATES sont *dans* RES1.

III.2.3. Opérateurs de base et opérateurs dérivés

Cinq de ces huit opérateurs forment les opérateurs de base (ce sont l'union, la différence, le produit cartésien, la restriction et la projection) tandis que les trois autres, appelés opérateurs dérivés, s'obtiennent plus ou moins facilement par combinaison des opérateurs de base :

$$R \cap S = R - (R - S)$$

$$R \bowtie S (A_i, B_j) = \sigma_{i \theta j} (R \times S)$$

$$R \div S = \pi_{i_1, \dots, i(r-s)} (R) - \pi_{i_1, \dots, i(r-s)} ((\pi_{i_1, \dots, i(r-s)} (R) \times S) - R)$$

Les cinq opérateurs de base permettent de répondre à toutes les questions que l'on peut poser avec la logique du premier ordre (c'est à dire sans les fonctions) : on dit que l'algèbre relationnelle est complète.

En réalité, nous n'utiliserons dans nos requêtes que les opérateurs les plus maniables : ce sont l'union et la différence pour l'insertion et la suppression de tuples dans la base et la restriction, la projection et la jointure pour la recherche sélective de tuples.

III.2.4. Des exemples de requêtes

Exemple 1 :

Donner les noms des plages fortement polluées.

Cette requête comprend deux opérations (où \leftarrow désigne l'affectation)

TEMP \leftarrow Rest (PLAGES, POLLUTION = 'forte')

RESU \leftarrow Proj (TEMP, NOMP)

Ce qui se note encore :

RESU \leftarrow Proj (Rest(PLAGE, POLLUTION = 'forte'), NOMP)

Exemple 2 :

Donner les noms des personnes ayant pris un bain en Bretagne.

T1 \leftarrow Rest (PLAGE, REGION = 'Bretagne')

T2 \leftarrow Join (T1, BAIGNADE, T1.NP = BAIGNADE.NP)

T3 \leftarrow Join (T2, NAGEURS, T2.NN = NAGEURS.NN)

RESU \leftarrow Proj (T3, NOM)

Exemple 3 :

Insertion de la plage de sable "Fort bloqué", en Bretagne, faiblement polluée avec le numéro 120.

PLAGES \leftarrow PLAGES \cup (120, 'Fort bloqué', 'sable', 'Bretagne', 'faible').

Exemple 4 :

Suppression des nageurs de qualité médiocre.

NAGEURS \leftarrow NAGEURS - Rest(NAGEURS, QUALITE = 'médiocre')

III.2.5. Les arbres algébriques et l'optimisation

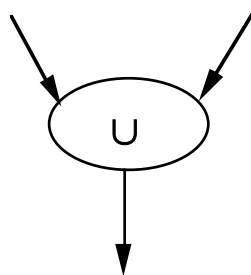
Nous venons de voir comment décrire une requête au moyen des opérateurs de l'algèbre relationnelle. Toutefois, l'écriture de cette combinaison d'opérateurs est malaisée à déchiffrer. Il existe en contrepartie une description graphique plus lisible : l'arbre algébrique.

Un arbre algébrique est un arbre dont :

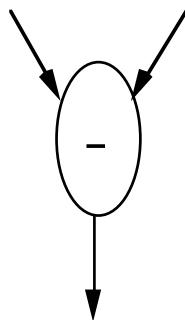
- les feuilles sont les relations de base ;
- les nœuds sont les opérateurs ;
- la racine est le résultat ;
- les liens représentent les flux de données.

Ainsi, des nœuds différents représentent les cinq principaux opérateurs relationnels : l'union, la différence, la restriction, la projection et la jointure.

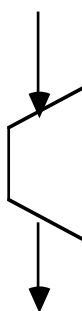
L'union :



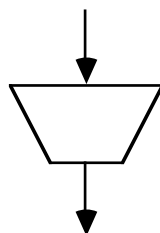
La différence :



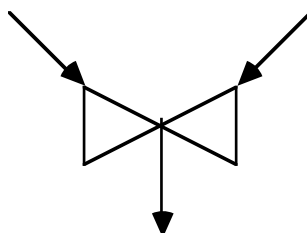
La restriction :



La projection :



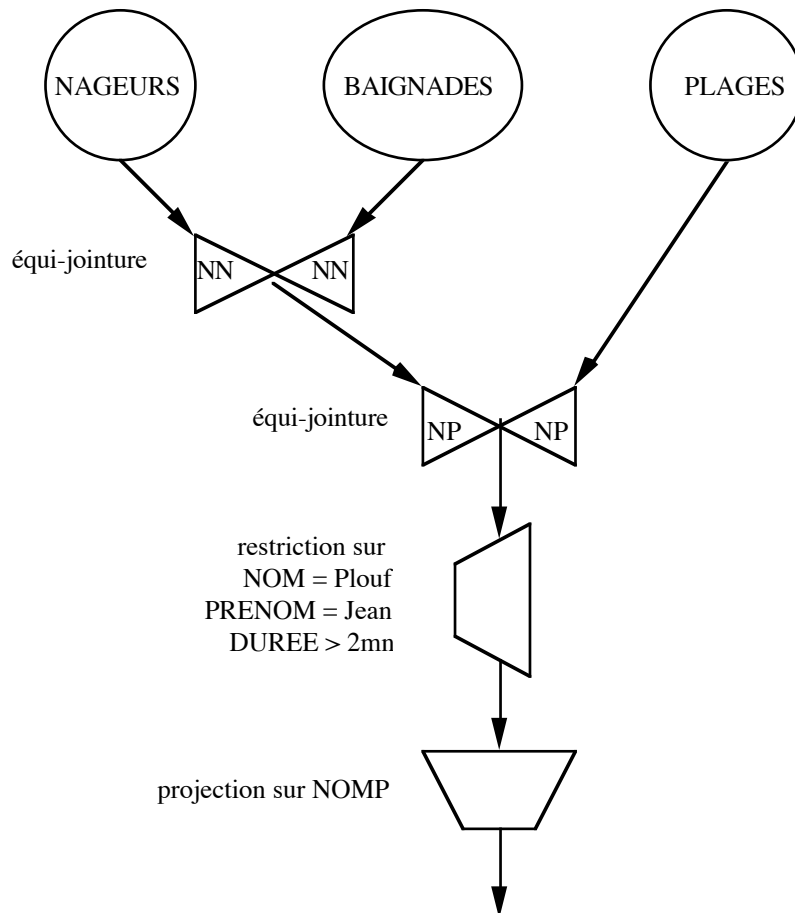
La jointure :



Nous pouvons dès lors facilement représenter la requête :

"Sur quelle plage (NOMP) Jean Plouf a-t-il pris des bains de plus de deux minutes ? "

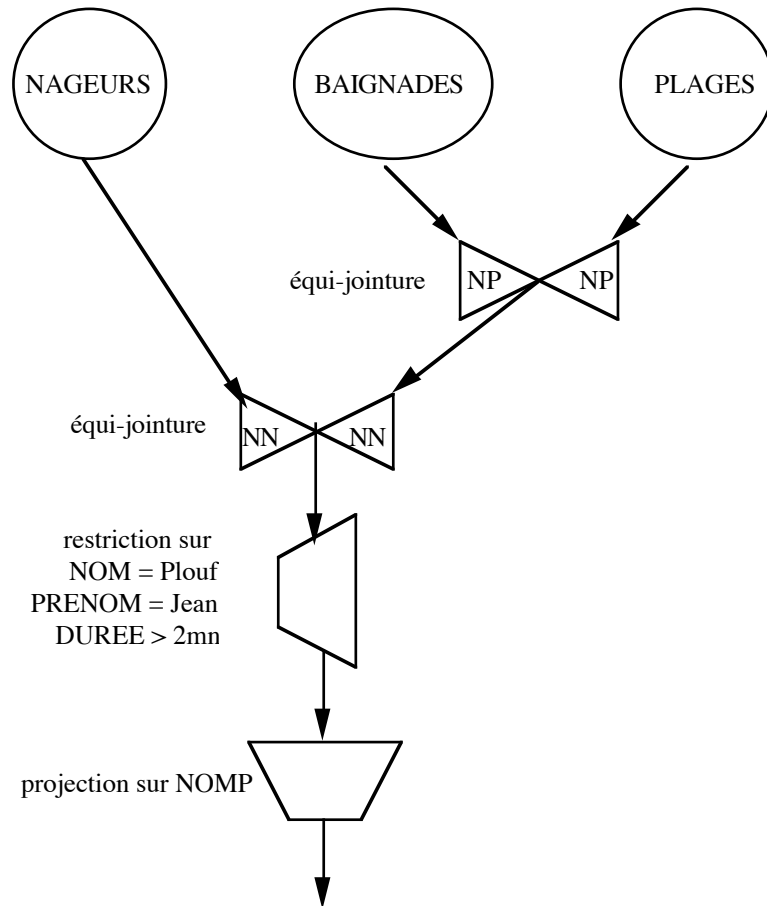
Remarquons que cette requête peut s'exécuter de diverses façons :



Première méthode

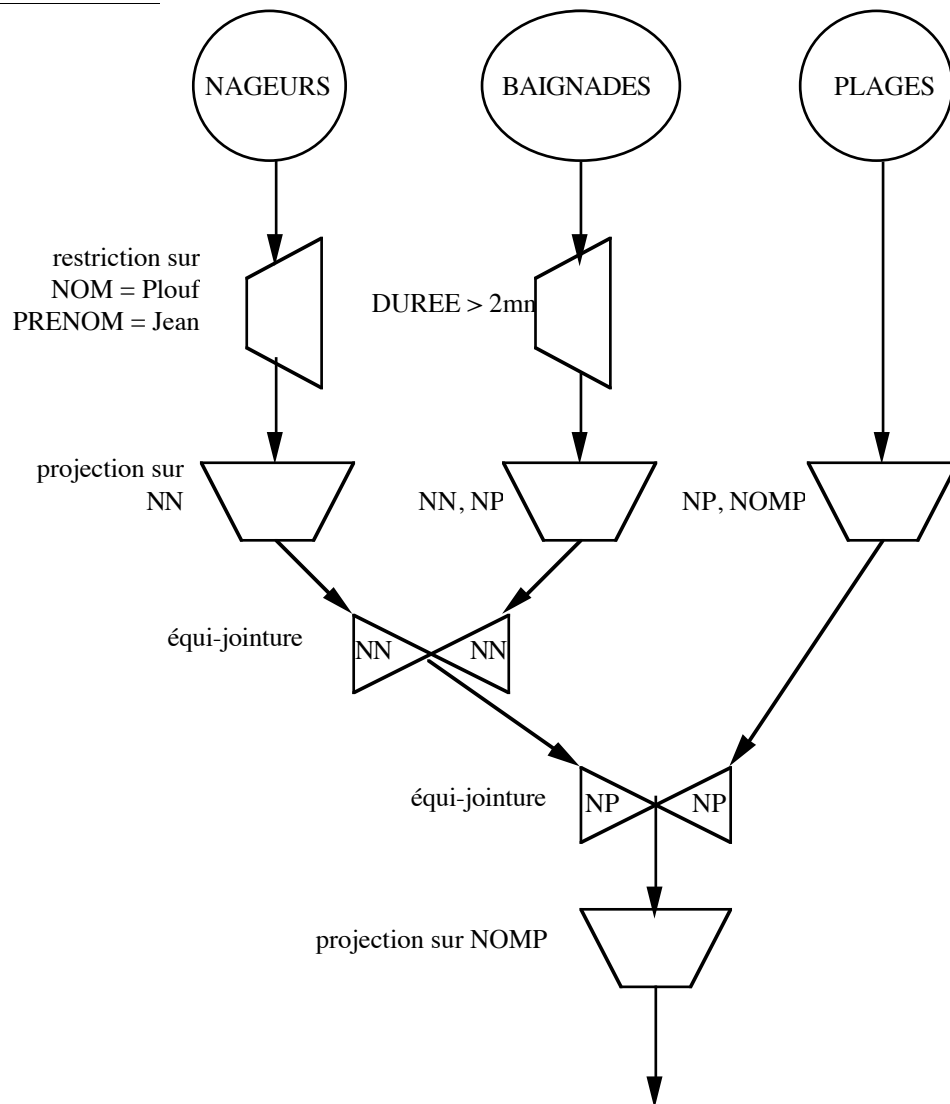
Une première méthode d'exécution de requête peut minimiser le nombre d'opérations à effectuer. C'est ce que nous faisons ici en exécutant les jointures dans un premier temps, puis en repoussant à la fin la restriction et la projection qui permettent d'obtenir le résultat désiré. On minimise ainsi le nombre d'opérations à effectuer, mais on réalise des jointures sur des relations très volumineuses. ce qui augmente considérablement les tailles des relations intermédiaires et donc le coût global d'exécution de la requête.

Deuxième méthode



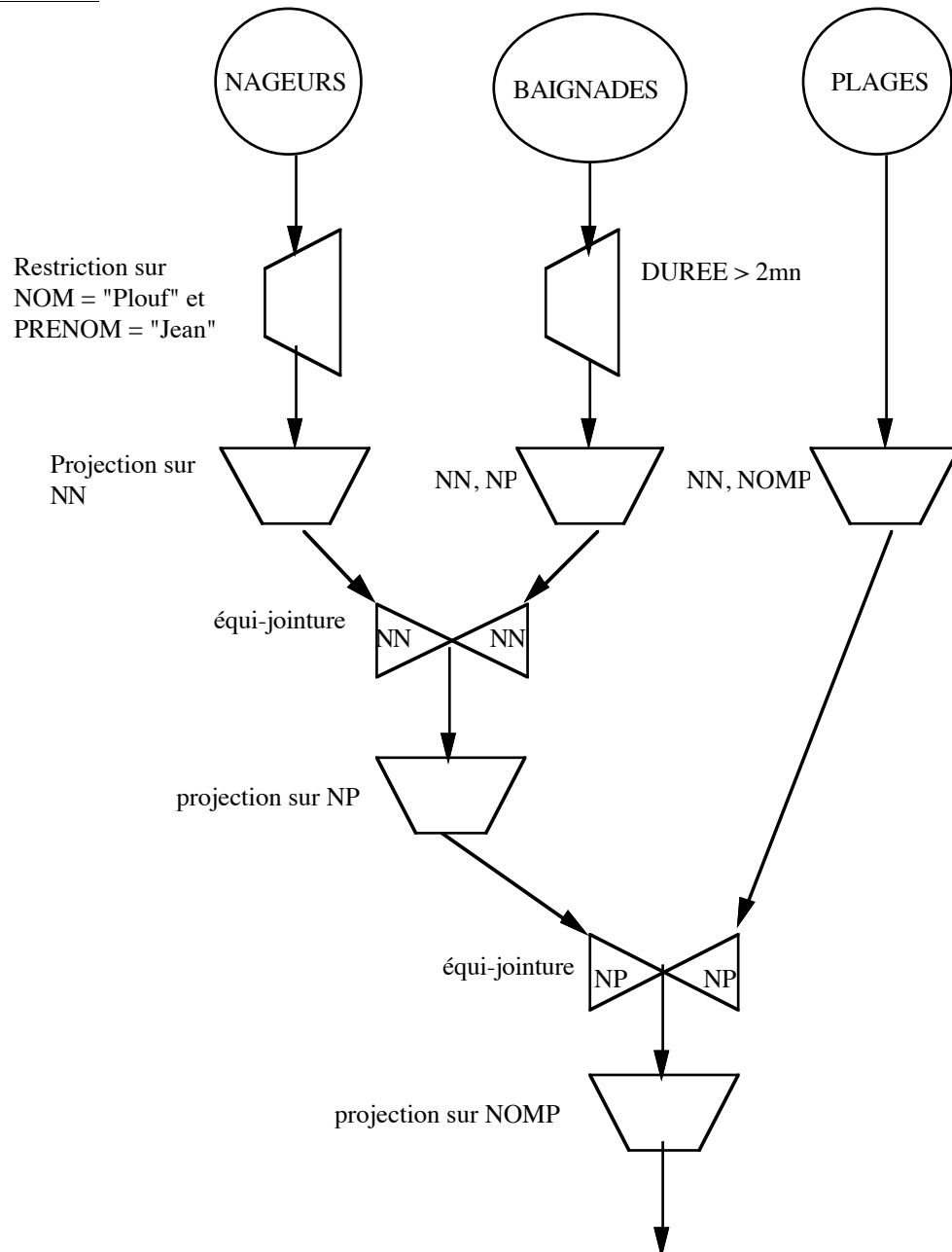
Quelle que soit l'heuristique choisie pour exécuter la requête. Déterminer dans quel ordre l'exécution des jointures entraîne un coût minimum est intéressant. Suivant la taille des relations mises en jeu, la différence de coût Correspondant à une inversion dans l'ordre des jointures peut être considérable.

Troisième méthode



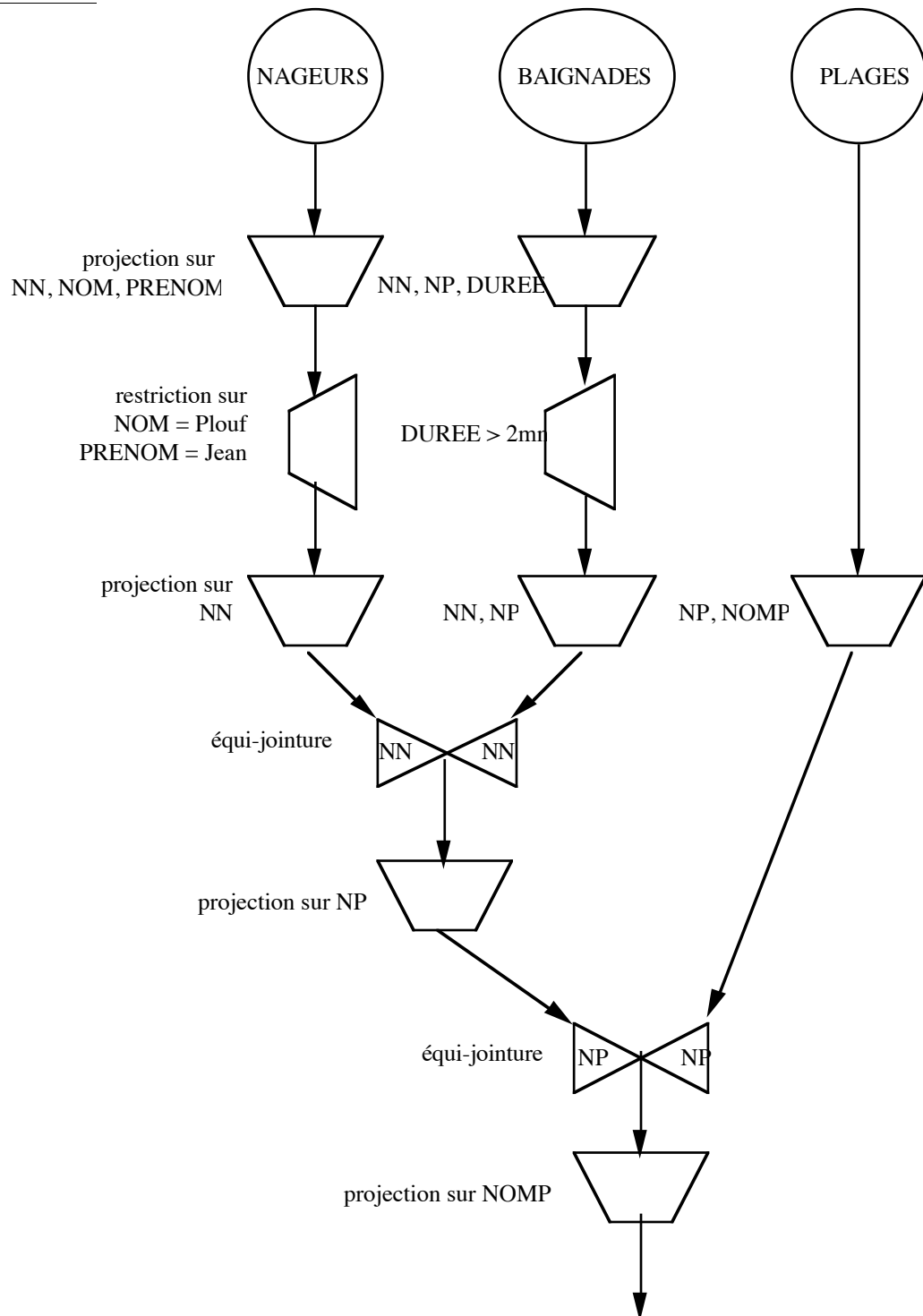
Toutefois, compte tenu du coût très important des jointures et de l'importance de la taille des relations mises en jeu, il vaut mieux rajouter des étapes intermédiaires qui diminuent la taille des relations à l'entrée des jointures. Le coût supplémentaire est compensé par une baisse sensible du coût des jointures. C'est ce que nous faisons ici : les opérations de jointure agissent sur des relations réduites au maximum par des restrictions et des projections.

Quatrième méthode



Nous pouvons affiner cette méthode en éliminant après chaque jointure les attributs devenus inutiles pour l'obtention du résultat final.

Cinquième méthode



Enfin, nous pouvons également être amenés à rechercher les critères de variation du coût entre la méthode décrite ci-dessus et la précédente. Nous rajoutons ici une première étape de projection avant la première restriction.

A partir de cet exemple, nous constatons qu'à une même requête correspond de multiples plans d'exécution. Le choix d'un plan par rapport à un autre a de nombreuses implications quant à la durée d'exécution de la requête.

Enfin, si la vérification que l'ordre des opérateurs permet de réaliser la question initialement posée, choisir parmi les nombreuses solutions qui n'affectent pas le résultat, mais seulement la rapidité d'exécution de la requête est plus délicat .

En effet, se préoccuper activement du coût d'exécution d'une requête est nécessaire : un SGBD est jugé sur sa rapidité.

Une heuristique simple consiste à restreindre le plus vite possible la taille des relations intermédiaires pour diminuer le coût . Pour ce faire, nous utilisons les propriétés des différents opérateurs :

- associativité des jointures ;
- commutativité restriction/jointure ;
- commutativité restriction/projection (si le critère de restriction porte sur les attributs projetés);
- commutativité projection/jointure (si le critère de jointure porte sur les attributs projetés).

Une méthode relativement aisée à mettre en œuvre consiste à disposer en entrée des jointures de relations les plus petites possibles, ce qui revient à faire précéder les jointures, opérations les plus coûteuses, du maximum de restrictions et de projections possibles sans altérer le résultat final.

De façon générale ; il est indispensable de disposer :

- de relations initiales et intermédiaires les plus petites possibles ;
- de techniques de placement adaptées qui permettent de ne pas rapatrier toute une relation pour isoler les informations intéressantes (voir chapitre 7) ;
- d'algorithmes performants implantant les opérateurs relationnels.

III.3. CONCLUSIONS

Les SGBD existant à ce jour mettent en œuvre des techniques performantes pour la manipulation des données.

Par exemple, tous les SGBD offrent aujourd'hui un langage assertionnel de requêtes, SQL. Un langage assertionnel permet de poser la requête sous la forme d'une ou de plusieurs conditions (assertions) à

remplir sans se préoccuper de l'ordre des différentes opérations à effectuer au niveau du système. C'est au SGBD d'analyser la condition et de la décomposer en une requête de l'algèbre relationnelle.

L'optimiseur de requêtes est un composant très important d'un SGBD relationnel. L'étape fondamentale est la construction d'un arbre algébrique optimisé, en déterminant le meilleur ordre d'exécution des diverses opérations. Cependant, les SGBD utilisent d'autres critères (contraintes d'intégrité, taille des relations candidates, méthodes de placement...) soit pour aider à la construction de l'arbre optimisé, soit pour réfuter logiquement une requête et pour choisir l'algorithme adéquat à la réalisation d'une opération particulière (une même opération dispose couramment de différents algorithmes dont les performances varient suivant les caractéristiques des relations mises en jeu).

L'opération de jointure ou la prise en compte des temps de transit de données dans des systèmes répartis sont un bon exemple de la multiplicité des algorithmes existants.

Les améliorations sur le marché suivent de près les progrès de la recherche et les produits évoluent vite : les premières versions de nombreux produits offraient des algorithmes peu performants, mais la situation a bien changé depuis. Les SGBD demeurent toutefois des produits délicats à évaluer, comme c'est souvent le cas des produits informatiques. L'utilisation d'un algorithme ne suffit pas à garantir sa bonne utilisation. Notez en particulier l'importance essentielle des méthodes de stockage et des primitives de bas niveau offertes par le système d'exploitation pour l'efficacité des SGBD. Pour toutes ces raisons, le modèle relationnel, malgré sa "beauté conceptuelle", a mis beaucoup de temps à s'imposer sur le marché.

III.4. REFERENCES

[Astrahan 76] M. ASTRAHAN et al : "*System R : A Relational Approach to Database Management*", ACM Transactions on Database Systems, Vol 1, N°2, 1976.

[Codd 70] E.F. CODD, "*A Relational Model of Data for Large Shared Data Banks*", Communications ACM, V13, N6, Juin 1970, pp 377-387.

CHAPITRE 4 : LES LANGAGES RELATIONNELS

Les opérateurs algébriques, qui effectuent la manipulation interne des relations dans un SGBD relationnel, forment un premier langage de requêtes. Ils présentent cependant l'inconvénient majeur d'exprimer les manipulations à effectuer de façon procédurale. En effet, la spécification directe par l'utilisateur d'une séquence d'opérations relationnelles correspond au choix d'une exécution particulière ; plusieurs exécutions sont possibles pour une même question. L'utilisateur à qui l'on offre un langage algébrique a donc plusieurs choix pour formuler sa question : certains sont performants, d'autres induisent des temps de réponses impressionnants, car ils génèrent un volume considérable de données intermédiaires et de très nombreux accès.

Les concepteurs de systèmes ont donc cherché à proposer un langage de manipulation de données qui ne demande précise pas l'ordre des opérations. Un tel langage est donc non-procédural. Pour cela, la complétude de l'algèbre relationnelle est exploitée, c'est-à-dire qu'elle est équivalente à la logique des prédicats du premier ordre (sans fonctions). Il s'agit d'un résultat important que nous admettrons simplement ici. Une requête s'exprime donc comme une assertion sur la base de données. Le critère logique que doivent vérifier les tuples du résultat est spécifié ; cette assertion décrit une caractéristique du résultat, indépendamment de la façon de l'obtenir. C'est le système qui, à partir de l'expression d'une assertion, génère une exécution particulière, c'est-à-dire une certaine séquence d'opérateurs relationnels.

La première tentative pour construire un véritable langage de requêtes, transformable en séquence d'opérations relationnelles, date de 76 : le langage SEQUEL du laboratoire IBM de San José. C'est lui qui donnera plus tard naissance à SQL (Structured Query Language).

IV.1. LE STANDARD SQL

Hier norme de fait, SQL est devenu une norme de droit quand l'International Organization for Standardization l'a pris en compte en 86 [ISO 86]. SQL est arrivé à maturité. Sa version normalisée par l'ISO fige ses différents aspects : définition des concepts, du vocabulaire, du langage de définition de données (déclaration du schéma, des vues, du contrôle des autorisations) et du langage de manipulation sont désormais stabilisés. Les procédures d'intégration des appels SQL dans un langage hôte (de type

Cobol, Pascal, C, Fortran...), regroupées sous le nom de "Embedded SQL", souvent noté ESQL, sont également définies. Presque tous les constructeurs proposent le langage SQL. Plus de 70 produits du marché le fournissent, sur des machines qui vont du PC au plus importants mainframes. Celui-ci étend même sa percée vers des produits micros qui, sans devenir de véritables SGBD relationnels complets, intègrent déjà la partie de SQL qui concerne la manipulation des données.

SQL est un facteur important de promotion du modèle relationnel. Il apporte une vision unifiée et une large communauté de concepts et de vocabulaire. Reposant sur une base théorique solide (la logique des prédicats), il concerne à la fois les administrateurs, les programmeurs d'applications, et les utilisateurs finals pour la définition et la manipulation des données.

IV.1.1. Eléments d'un langage pour les SGBD relationnels

La norme SQL comporte trois parties distinctes concernant :

- la manipulation de l'information ; le (DML : Data Manipulation Language) concerne l'ensemble des requêtes de recherche et de mise à jour des informations : selection, insertion, modification et suppression. Il est essentiellement destiné à l'utilisateur final ;
- la définition des structures de données ; le DDL (Data Definition Language) contient l'ensemble des commandes de création, de suppression, de modification des relations. Elle intègre la créations, la suppressions de chemins d'accès (index) et les commandes concernant la gestion des droits et des autorisations. Elle est largement destinée à l'administrateur ;
- l'accès aux informations stockées dans le SGBD depuis un langage de programmation ; l' Embedded SQL définit l'ensemble des interfaces, l'utilisation des ordres SQL dans un langage de programmation, ainsi que l'ensemble des codes d'erreur renvoyés par le SGBD au programme d'application défini. L' Embedded SQL est destiné au programmeur d'applications.

La liste des avantages et inconvénients de la norme SQL telle qu'elle se présente aujourd'hui peut être rapidement dressée : à son actif, citons sa simplicité, sa souplesse et ses fonctionnalités étendues ; à son passif, un manque de rigueur (il autorise - et encourage - certaines formulations procédurales), une utilisation lourde (mot clés, pas d'utilisation de moyens modernes d'interface homme/machine), le fait qu'il constitue, comme toute norme, un frein à la créativité des concepteurs de nouveaux systèmes. Rappelons enfin qu'il s'agit d'un langage inventé par IBM, argument stratégique délicat à placer dans les qualités ou les défauts du langage.

La normalisation de SQL, dont la première version a été achevée en 86, conduit également à se poser la question "à qui peut servir la normalisation ?". En effet, si l'on examine les différentes parties de la norme, on constate que l'administrateur doit disposer d'outils moins primitifs et plus spécifiques pour gérer convenablement son SGBD ; l'utilisateur final a bien du mal à utiliser directement SQL en interactif : il utilise des interfaces spécifiques (menus, écrans...) plus conviviales ; enfin le programmeur d'applications est aujourd'hui largement sollicité par les Langages de quatrième Génération (LAG) qui, sans être normalisés, sont plus pratiques et procurent les mêmes services de base que du Embedded SQL. Certaines mauvaises langues en concluent donc que la normalisation ne sert qu'aux normalisateurs... L'apport de la normalisation du SQL est cependant indiscutable pour les utilisateurs désirant garantir la portabilité de leurs applications d'un système SQL sur un autre. C'est l'interface standard de tout véritable SGBD relationnel. Il précise, d'autre part, un niveau d'interface précis lorsqu'on envisage des systèmes répartis et hétérogènes.

Comme pour les opérateurs de l'algèbre relationnelle, nous illustrerons l'utilisation de SQL sur la base-jouet suivante :

PLAGE (NP, NOMP, TYPE, REGION, POLLUTION)

NAGEUR (NN, NOM, PRENOM, QUALITE)

BAIGNADE (NN, NP, DATE, DUREE)

où NP désigne le numéro de plage, NN le numéro de nageur.

IV.1.2. Expression des opérations relationnelles

SQL permet d'exprimer simplement les opérations relationnelles de base.

Le bloc SQL se compose du mot-clé **SELECT**, suivi des attributs qu'on désire voir figurer dans le résultat, du mot clé **FROM**, suivi de la liste des relations touchées par la question, enfin du mot clé **WHERE**, suivi d'une qualification. Deux blocs SQL peuvent être conjugués par l'un des opérateurs binaires de l'algèbre relationnelle : **UNION**, **INTERSECT**, **MINUS**.

SELECT <liste d'attributs projetés>

FROM <liste de relations>

WHERE <qualification>

Expression de projections

L'obtention de colonnes déterminées s'effectue simplement en omettant de spécifier une qualification (le prédicat absent derrière le WHERE est alors supposé 'vrai' pour tous les tuples).

```
SELECT NOMP, REGION
```

```
FROM PLAGE
```

permet d'obtenir l'ensemble des couples (nom_de_plage, région) présents dans la relation PLAGE. Cependant l'élimination des doubles n'est pas automatique : pour lister l'ensemble des régions distinctes on utilise le mot clé DISTINCT:

```
SELECT DISTINCT REGION
```

```
FROM PLAGE
```

Enfin, les mots clés ORDER BY, ASC, DESC permettent de compléter le langage et d'ordonner les résultats pour l'utilisateur. Ainsi, pour obtenir toutes les plages, triées par ordre alphabétique décroissant sur les pollutions et croissant sur les noms de plage, on écrit :

```
SELECT *
```

```
FROM PLAGE
```

```
ORDER BY POLLUTION DESC, NOMP ASC
```

L'utilisation du "*" permet de demander tous les attributs de la relation.

Expression des restrictions

La restriction consiste à sélectionner l'ensemble des tuples vérifiant le critère placé dans la clause WHERE. Ce critère de restriction est une combinaison de critères élémentaires, connectés à l'aide de AND et de OR et pouvant présenter des NOT. Le critère atomique est de la forme <Attribut θ valeur> ou <Attribut1 θ Attribut2>. Le comparateur θ appartient à l'ensemble {=, <>, >, ≥, <, ≤}.

Un bloc permettant d'exprimer la restriction est par exemple:

```
SELECT *
```

```
FROM NAGEUR
```

```
WHERE QUALITÉ = 'médiocre' OR PRÉNOM <> 'Jean'
```

La forme <Attribut1 θ Attribut2> exprime un critère de restriction s'il s'agit d'une comparaison entre la valeur de l'Attribut1 et la valeur de l'Attribut2 *du même tuple*. Le critère est évalué pour chaque tuple en comparant deux valeurs d'attributs différents du même tuple. Par exemple, pour connaître les bains pris sur des plages de même identifiant (NP) que celui du baigneur (NN) - sémantiquement peu intéressant! - on écrirait :

```
SELECT *
```

```
FROM BAIGNADE
```

WHERE NP = NN

La sélection mono-relation générale utilise un bloc SQL qui combine une restriction et une projection. Ainsi, pour obtenir les plages de sable de Bretagne, triées par ordre alphabétique décroissant sur les pollutions et croissant sur les noms de plage, on écrira :

```
SELECT NP, NOMP, POLLUTION  
FROM PLAGE  
WHERE TYPE = 'sable' OR REGION <> 'Bretagne'  
ORDER BY POLLUTION DESC, NOMP ASC
```

Expression des jointures

L'expression de questions multi-relations en SQL , c'est-à-dire de questions mettant en œuvre, dans le résultat ou dans la qualification, des informations issues de plusieurs relations, nécessite d'exprimer l'opération de jointure.

SQL propose deux façons d'exprimer la jointure : l'une est véritablement assertionnelle, l'autre reste procédurale et utilise une imbrication de blocs de base (un bloc de base est un ensemble SELECT...FROM...WHERE...).

Exemple : donner les noms des nageurs ayant pris un bain

a) Expression assertionnelle :

```
SELECT NOM  
FROM NAGEUR, BAIGNADE  
WHERE NAGEUR.NN = BAIGNADE.NN
```

Pour distinguer deux attributs portant le même nom, ceux-ci sont préfixés par le nom de la relation d'origine

b) Expression procédurale

```
SELECT NOM  
FROM NAGEUR  
WHERE NN IN (SELECT NN  
              FROM BAIGNADE)
```

Cette manière d'exprimer la jointure présente deux inconvénients :

- elle est procédurale et demande par conséquent au système d'exécuter une séquence d'opérations particulières. L'optimisation et l'ordonnancement ne sont plus mis en œuvre par le système ; une question mal posée peut, par conséquent, entraîner des temps de réponse catastrophiques. Néanmoins certains bons systèmes acceptent une question ainsi formulée, mais la transformer en une assertion de façon à pouvoir ensuite utiliser leur optimiseur.
- elle ne permet pas d'exprimer une 'vraie' jointure : en effet seuls les attributs originaux d'une seule des deux relations peuvent être présents dans le résultat ; il s'agit en fait d'une semi-jointure

Définition : semi-jointure:

la semi-jointure de R1 par R2, notée $R1 \bowtie R2$, est la jointure de R1 et de R2 projetée sur les attributs de R1.

La semi-jointure peut être considérée comme une généralisation de la restriction. Ceci explique en effet son coût relativement faible : l'opération n'exige qu'une seule passe sur chaque relation. La relation R2 est d'abord lue pour obtenir les valeurs de l'attribut de jointure ; puis la relation R1 est restreinte avec les valeurs de l'attribut de jointure apparaissant dans R2. Chacun des tuples de R1 et R2 n'est lu qu'une seule fois.

L'utilisation de blocs imbriqués dans SQL permet cependant d'exprimer simplement des prédicats quantifiés (\forall , \exists) grâce aux sous-questions (voir § IV.1.3).

Remarque: restriction et auto-jointure

La présence de prédicat de la forme $\langle \text{Attribut1} \theta \text{ Attribut2} \rangle$ dans une qualification peut correspondre soit à un prédicat de jointure, soit à un prédicat de restriction. En fait, la question est de savoir si la comparaison porte sur la valeur des attributs 1 et 2 d'un même tuple ou de tuples différents. Si l'Attribut1 et l'Attribut2 sont préfixés par une variable relation identique (ou ne le sont pas dans les cas où il n'y a pas d'ambiguïté sur la relation), il s'agit des valeurs du même tuple. C'est donc d'un critère de restriction. Si l'Attribut1 et l'Attribut2 sont préfixés par des variables relations différentes, il s'agit de la comparaison d'une valeur de l'Attribut1 avec toutes les valeurs de l'Attribut. C'est donc d'un critère de jointure.

Question :

SELECT NN

FROM NAGEUR

WHERE NOM = PRENOM

exprime la requête '*quels sont les nageurs dont le nom est identique au prénom ?*'

(restriction);

en explicitant par une variable N1 le lien attributs- relation, la question devient :

SELECT N1.NN

FROM NAGEUR N1

WHERE N1.NOM = N1.PRENOM

alors qu'en écrivant:

SELECT N1.NN

FROM NAGEUR N1, NAGEUR N2

WHERE N1.NOM = N2.PRENOM AND N1.NN <> N2.NN

on exprimerait '*quels sont les nageur ayant pour nom le prénom d'un autre nageur ?*'

(auto-jointure). Deux variables distinctes N1 et N2 sont déclarées cette fois-ci sur la

relation NAGEUR. SQL, on le voit, oblige a plus de précision que la langue "naturelle".

IV.1.3. Sous-questions, prédicats quantifiés, composition de questions

L'utilisation de blocs imbriqués permet d'exprimer simplement et de façon très compréhensible des expressions quantifiées. Le bloc interne est alors une *sous-question*.

Définition : sous-question

Une sous question est une question SQL qui rend une seule colonne

Cette colonne est un ensemble de valeurs qui constitue l'argument d'un prédicat IN, ALL, ANY ou EXISTS du bloc externe (dans le cas du EXISTS la sous-question peut rendre un nombre quelconque de colonnes).

Illustrons ces possibilités sur des exemples :

a) Prédicat IN : quels sont les nageurs qui se sont baignés (au moins une fois) sur une plage très polluée ?

SELECT NN FROM BAIGNADE

WHERE NP IN

(SELECT NP FROM PLAGE

WHERE POLLUTION = 'élevée')

IN est le test de présence de la valeur d'un attribut dans un ensemble de valeur du même type. On l'utilise pour exprimer la semi-jointure.

b) Prédicat EXISTS : quels sont les nageurs qui se sont [ne se sont pas] baignés en 89 ?

```
SELECT *  
FROM NAGEUR N  
WHERE [NOT] EXISTS (SELECT * FROM BAIGNADE  
                     WHERE DATE BETWEEN '01-JAN-89' AND '31-DEC-89'  
                     AND B.NN = N.NN )
```

c) Prédicat ALL : quelle est la plus longue baignade ?

```
SELECT *  
FROM BAIGNADE  
WHERE DUREE >= ALL ( SELECT DISTINCT DUREE  
                      FROM BAIGNADE)
```

d) Prédicat ANY : quels sont les noms des nageurs qui se sont baignés plus longtemps que Dupond ?
(qui se sont baignés au moins une fois plus longtemps qu'au moins une des baignades de Dupond)

```
SELECT N1.NOM  
FROM NAGEUR N1, BAIGNADE B1  
WHERE N1.NN = B1.NN  
AND B1.DUREE > ANY ( SELECT DISTINCT DUREE  
                      FROM BAIGNADE B2, NAGEUR N2  
                      WHERE B2.NN = N2.NN  
                      AND N2.NOM = 'Dupond')
```

Enfin, on peut composer logiquement - par union, intersection, différence, cf. § III.2.1 - deux questions SQL. Si, par exemple, on voulait comparer notre table de nageurs à une table d'élèves qui aurait le schéma ELEVE(NOM, PRENOM, AGE) alors les expressions SQL suivantes sont possibles - et leur sens est identique à celui des expressions algébriques équivalentes :

```
SELECT NOM , PRENOM FROM ELEVE UNION SELECT NOM , PRENOM FROM NAGEUR  
                                     INTERSECT  
                                     MINUS
```


IV.1.4. Fonctions et groupements

La norme SQL présente un ensemble de fonctions prédéfinies : COUNT, SUM, AVG, MAX et MIN. Ces fonctions opèrent sur une collection de valeurs scalaires d'une colonne (éventuellement plusieurs pour COUNT) d'une relation.

COUNT	nombre de valeurs de la colonne ;
SUM	somme des valeurs de la colonne (argument de type numérique) ;
AVG	moyenne des valeurs de la colonne (argument de type numérique) ;
MAX	plus grande valeur de la colonne ;
MIN	plus petite valeur de la colonne.

Cette collection de valeurs scalaires d'une colonne est obtenue en effectuant un agrégat.

Définition : agrégat

Un agrégat est un partitionnement horizontal (les tuples sont répartis en plusieurs groupes) d'une relation selon des valeurs d'attributs, suivi d'un regroupement par une fonction de calcul (somme, moyenne, minimum, maximum, compte...)

Les fonctions s'appliquent sur un agrégat défini par les attributs figurant derrière un mot clé GROUP BY, et après application d'un éventuel critère figurant derrière un mot clé WHERE.

La clause HAVING permet d'appliquer un critère de restriction sur des groupes.

Exemple :

Donner, par numéro de plage, la durée moyenne et la somme des durées des baignades effectuées après le 1/1/88, lorsque les durées cumulées dépasse 200 minutes.

```
SELECT NP, AVG(DURÉE), SUM(DURÉE)
FROM BAIGNADE
WHERE DATE ≥ '01-JAN-88'
GROUP BY NP
HAVING SUM(DURÉE) > 200
```

Le système exécute d'abord la restriction, puis effectue le regroupement, le calcul des fonctions et la restriction sur le résultat du calcul ; exemple exécution de la question précédente :

BAIGNADE	DATE	NP	DURÉE	NN
	12/1/88	30	72	67
	2/12/87	43	24	24
	16/2/88	30	46	45
	3/3/88	43	12	12
	10/3/88	30	92	24

WHERE DATE \geq 01/01/88



BAIGNADE	DATE	NP	DURÉE	NN
	12/1/88	30	72	67
	16/2/88	30	46	45
	3/3/88	43	12	12
	10/3/88	30	92	24

GROUP BY NP



BAIGNADE	DATE	NP	DURÉE	NN
	12/1/88	30	72	67
	16/2/88	30	46	45
	10/3/88	30	92	24
	— —	—	—	—
	3/3/88	43	12	12

CALCUL DES FONCTIONS



BAIGNADE	NP	AVG(DURÉE)	SOM(DURÉE)
	30	70	210
	— —	— — —	— — —
	43	12	12

HAVING SOM(DURÉE) > 200



BAIGNADE	NP	AVG(DURÉE)	SOM(DURÉE)
	30	70	210

IV.1.5. Mises à jour

SQL propose trois clauses correspondant aux différents types de mises à jour possibles sur une base de données : l'insertion (clause INSERT), la modification (clause UPDATE) et la suppression (clause DELETE).

Insertion

Une insertion est l'ajout d'un ou plusieurs tuples dans une relation. On peut insérer, par la commande INSERT, des tuples explicitement mais un par un; le mot-clé VALUES précède les valeurs de ses attributs.

INSERT INTO NAGEUR

VALUES (145, 'BOUJUS', 'Jacques', 'excellent')

On peut également insérer directement un ensemble de tuples. Cet ensemble de taille quelconque doit être le résultat d'une requête ensembliste. La seule condition est que le schéma de la relation résultat (nombre et types des attributs) soit le même que celui de la relation dans laquelle on ajoute des valeurs. On parle dans ce cas d'*insertion calculée*.

La puissance d'expression d'un langage relationnel s'affirme ici clairement. Ainsi la requête :

INSERT INTO BAIGNADE

SELECT '10', NP, DATE, DUREE

FROM BAIGNADE

WHERE NN = '20'

ajoute pour le nageur 10 les baignades effectuées par le nageur 20.

Mise à jour

L'aspect ensembliste du langage est ici encore particulièrement utile pour les applications. Accéder aux tuples un par un n'est pas nécessaire. Il suffit d'exprimer correctement la condition qui conduit à la modification.

Par exemple :

```
UPDATE PLAGE
SET POLLUTION = 'élevée'
WHERE REGION = 'Bretagne'
AND NP IN ( SELECT NP FROM BAIGNADE
              GROUP BY NP, DATE
              HAVING COUNT (NN) > 5000 )
```

permettra de prendre en compte le fait qu'une forte fréquentation des plages de Bretagne (les plages ayant vu plus de 5000 baignades en un jour !) entraîne une pollution élevée.

Suppression

Enfin, la suppression ensembliste s'exprime elle aussi très simplement. On supprime directement les mauvais nageurs considérés comme noyés au delà de 300 minutes dans l'eau.

```
DELETE FROM NAGEUR
WHERE QUALITE = 'exécration'
AND NN IN ( SELECT NN FROM BAIGNADE
              WHERE DURÉE > 300 )
```

ou, dans un exemple plus sérieux, on supprime les baignades de Dupond comme ceci :

```
DELETE FROM BAIGNADE
WHERE NN IN ( SELECT NN FROM NAGEUR
               WHERE NOM = 'Dupond' )
```

IV.1.6. Le langage de déclaration de données

Rappelons qu'une base de données est définie par un schéma. Ce schéma est composé d'un ensemble de relations. La création d'une nouvelle relation (vide) est réalisée par l'instruction :

```
CREATE TABLE nom_de_relation ( liste de noms_d'attributs )
```

Chaque nom d'attribut a un type (CHARACTER, NUMERIC, DECIMAL, INTEGER, SMALLINTEGER, FLOAT, REAL). Une contrainte NOT NULL ou UNIQUE peut, de surcroît, être imposé à un attribut . Déclarer une clé utilise simultanément ces deux contraintes.

Exemple :

```
CREATE TABLE PLAGE (      NP INTEGER NOT NULL UNIQUE,
                           NOMP CHARACTER(30) NOT NULL,
                           REGION CHARACTER (30),
                           POLLUTION CHARACTER (20) )

CREATE TABLE NAGEUR      (      NN INTEGER NOT NULL UNIQUE,
                           NOM CHARACTER(15) NOT NULL,
                           PRENOM CHARACTER(15),
                           QUALITÉ CHARACTER(20) )

CREATE TABLE BAIGNADE    (      NN INTEGER NOT NULL,
                           NP INTEGER NOT NULL,
                           DATE CHARACTER(8) NOT NULL,
                           DURÉE INTEGER ,
                           UNIQUE (NN, NP, DATE) )
```

Remarquez l'utilisation de la clause UNIQUE dans la création de BAIGNADE : il s'agit ici de déclarer une clé multi-attributs (il n'existe pas deux triplets identiques de valeurs NN, NP, DATE). Chaque attribut NN, NP et DATE est déclaré NOT NULL, puis l'unicité du triplet est déclarée.

La clause CREATE TABLE est beaucoup plus complète que nous ne l'indiquons ici : elle permet notamment de préciser d'autres contraintes d'intégrité, des droits ainsi qu'un certain nombre de contraintes concernant l'implantation physique de la relation.

IV.1.7. Intégration d'un langage de manipulation de données dans un langage de programmation

L'intégration de SQL dans un langage de programmation apparaît très vite indispensable lorsqu'on considère l'utilisation, par des programmes d'application, des informations stockées dans la BD relationnelle. Deux approches sont possibles : la première consiste à intégrer le langage de manipulation de données dans un langage de programmation quelconque, considéré comme un langage hôte. La seconde, plus ambitieuse, étend un langage de programmation donné avec des clauses spécifiques de la manipulation des données. Elle est restée jusqu'à présent du domaine de la recherche, mais le problème, bien réel, de la "cohabitation" entre un langage de programmation procédural, privilégiant un traitement unaire et un langage de manipulation ensembliste des données, redevient d'actualité avec les Bases de Données Orientées Objet. L'approche Objet tente, en effet, de réconcilier données et traitements.

La démarche dominante d'intégration de SQL est précisée jusque dans la norme sous le nom de *Embedded SQL*. Dans cette approche, les attributs des relations manipulées sont d'abord déclarés comme des variables du langage de programmation. Un programme hôte comporte des instructions standard du langage, une section de déclaration, un ensemble de définitions de curseurs et un ensemble de traitements SQL.

L'implantation d'un Embedded SQL est généralement effectuée à l'aide d'une technique de précompilation. Le précompilateur analyse des commandes SQL et insère des appels de sous-programmes du langage hôte à destination du SGBD (cela suppose l'existence d'une bibliothèque de primitives de bas niveau entre le langage hôte et le SGBD). Il recopie telles quelles les instructions du langage hôte. Le précompilateur génère en sortie, on obtient un programme en langage hôte qui est compilable par le compilateur de ce langage.

Nous donnons ici un exemple en C-Embedded SQL d'une mise à jour sur l'attribut POLLUTION de la relation PLAGE :

```
#include <stdio.h>
EXEC SQL    INCLUDE SQLCA.H ;
main (argc,argv)
    int argc ;
    char *argv[]; {
    char *answer[1];

EXEC SQL    BEGIN DECLARE SECTION ;
            VARCHAR db_connect_string[32];
            VARCHAR xx[32];
            VARCHAR yy[16];
            VARCHAR zz[16];

EXEC SQL    END DECLARE SECTION ;
strcpy(db_connect_string, argv[1]);
EXEC SQL    CONNECT : db_connect_string;
EXEC SQL    DECLARE C1 CURSOR FOR
            SELECT nomp, pollution FROM plage WHERE region = :yy
            FOR UPDATE OF pollution;
```

```

while (printf("\nRégion? (\0 pour fin): ") && scanf("%s", zz) && strcmp(zz, '\0') != 0)
{
    EXEC SQL OPEN C1 ;
    while (sqlca.sqlcode = 0) { /* la structure sqlca est dans SQLCA.H*/
        EXEC SQL FETCH C1 INTO :xx, :zz;
        printf("\n%32s, qui était %16s, devient-elle Alarmante? (O/N)", xx, zz) ;
        scanf("%s", answer) ;
        if (answer == 'O')
            EXEC SQL UPDATE PLAGE
                SET pollution = 'Alarmante'
                WHERE CURRENT OF C1 ;
    } /* end of while cursor open */
    EXEC SQL CLOSE C1 ;
    EXEC SQL COMMIT WORK ;
} /* end of while Région indiquée */
} /* end of main */

```

Commentaires

- un traitement SQL est annoncé par EXEC SQL et terminé par un caractère point-virgule ;
- toutes les variables hôtes référencées dans un traitement SQL sont définies ; dans une "embedded declare section", délimitée par un BEGIN DECLARE SECTION et un END DECLARE SECTION ;
- le type des variables hôtes doit être compact avec celui des attributs qui les accueillent ;
- tout programme SQL doit inclure une variable hôte appelé SQLCODE. A chaque exécution d'un traitement SQL, l'indicateur numérique SQLCODE, renvoie le résultat de la requête SQL (OK, 'OK, fin curseur).
- un traitement SQL inclut des références aux variables hôtes. Celles-ci doivent être préfixées par un nom d'attribut pour les distinguer.

Des *curseurs*. sont utilisés pour traiter des ensembles de tuples dans le langage. Le mécanisme de curseur permet un traitement unaire (tuple à tuple) d'un ensemble de tuples fourni par le SGBD, par le langage hôte, en réponse à une requête SQL. Les curseurs peuvent être ouverts (OPEN C) , fermés (CLOSE C), positionnés en lecture sur un tuple (ordre FETCH) et utilisés comme position de référence (CURRENT OF C).

Remarquez que cette approche, à présent normalisée, est assez riche en possibilités ; elle est cependant relativement lourde à mettre en œuvre et l'on peut s'attendre dans les prochaines années à son abandon progressif au profit des langages de quatrième génération (L4G).

Langage de quatrième génération

Nous terminerons cette section par quelques mots sur les langages de quatrième génération. Concernant directement les développeurs d'applications, ces langages visent à faire gagner du temps et à baisser les coûts d'écriture des programmes. Ils utilisent largement les possibilités des SGBD relationnels à travers une manipulation essentiellement ensembliste des données. Ils ne constituent cependant pas simplement un langage de manipulation de données comme SQL, mais intègrent un ensemble complet de possibilités de traitement en fournissant des structures de contrôle d'un langage structuré de haut niveau (boucle, condition, parcours...). Ils s'efforcent, de surcroît, d'intégrer un ensemble d'outils (générateurs de rapports, de graphiques, d'applications...) souvent disponibles comme des logiciels à adjoindre au SGBD. Orientés vers la programmation visuelle, ils constituent parfois un véritable environnement de développement et sont aujourd'hui l'objet d'une forte demande.

Il ne faut cependant pas perdre de vue que ces langages modernes ne constituent qu'une couche de manipulation des informations stockées dans un SGBD. La qualité, les performances, la puissance d'expression résulte, en dernière analyse, essentiellement à la qualité du SGBD relationnel sous-jacent.

Signalons de surcroît qu'un L4G ne fournit aucun support à une gestion de transactions (voir chapitre 6), nécessaire à tout usage multi-utilisateurs : cette gestion de transactions doit alors être prise en compte au niveau de l'application. Les L4G ne sont pas l'objet d'une définition standard acceptée par tous : à ce jour, le choix d'un L4G particulier condamne l'utilisateur à une dépendance totale envers son fournisseur. Certains L4G génèrent cependant du SQL standard, élément favorable qui peut être pris en compte dans le choix.

IV.2. AUTRES LANGAGES RELATIONNELS

Deux autres langages méritent d'être mentionnés : QUEL et QBE.

IV.2.1. QUEL

Le langage QUEL fut le concurrent malheureux de SQL sur le terrain de la normalisation. Très proche, il constitue le langage natif du prototype INGRES de BERKELEY (INGRES, à présent propose les deux langages QUEL et SQL).

Sa syntaxe oblige à déclarer des variables sur les relations manipulées:

RANGE OF variable IS nom-de-relation

Plusieurs variables peuvent être associées à une relation. Les recherches élémentaires s'expriment par les clauses :

RETRIEVE (liste résultat)

WHERE qualification

Exemple :

donner le nom des nageurs qui se sont baignés à la fois à Deauville et à Bayonne en 1983.

RANGE OF P, P', N, B, B' IS PLAGE, PLAGE, NAGEUR, BAIGNADE, BAIGNADE

RETRIEVE N.NOM

WHERE N.NN = B.NN AND B.NP = P.NP AND B.DATE = '1983' AND P.NOM = 'Deauville'

AND N.NN = B'.NN AND B'.NP = P'.NP AND B'.DATE = '1983' AND P'.NOM = 'Bayonne'

Seules les recherches avec fonctions et agrégats s'éloignent de SQL par exemple :

RETRIEVE AVG (B.DURÉE WHERE B.DATE = '02-07-89')

permet de demander la durée moyenne des baignades du 2 juillet 1989.

IV.2.2. Query By Example (QBE)

La langage QBE est conçu comme un langage graphique. Développé par IBM à Yorktown Height, et aujourd'hui commercialisé par IBM. Il est basé sur une mise en œuvre visuelle des questions et des résultats ; il constitue à ce titre un précurseur des langages visuels d'aujourd'hui.

Lorsqu'un utilisateur désire utiliser les informations d'une relation, il en demande l'affichage du squelette, c'est-à-dire le schéma. C'est dans ce squelette qu'il spécifie ses requêtes. Nous illustrons très rapidement les possibilités de QBE par quelques questions simples.

a) Quelles sont les durées des baignades du baigneur N°10 ?

BAIGNADE	NN	NP	DATE	DURÉE
	10			P.

b) Quelles sont les pollutions des plages où s'est baigné Durand ?

BAIGNADE	NN	NP	DATE	DURÉE
	<u>100</u>	<u>200</u>		

NAGEUR	NN	NOM	PRENOM	QUALITÉ
	<u>100</u>	Durand		

PLAGE	NP	NOMP	TYPE	RÉGION	POLLUTION
	<u>200</u>				P.

on utilise ici la notion de variable exemple (une chaîne quelconque soulignée est un exemple) pour exprimer la jointure entre deux relations. Le "P." indique la colonne demandée en résultat.

c) Lister les noms des plages de sable moins polluées que la plage n° 10.

PLAGE	NP	NOMP	TYPE	RÉGION	POLLUTION
	10				Q
		P.	sable		< Q

On a exprimé ci-dessus une auto-jointure.

IV.3. CONCLUSIONS

Les langages relationnels d'aujourd'hui s'articulent tous autour du SQL. Sa standardisation par l'ISO - SQL1 ou SQL89, puis SQL2 ou SQL92 - est en effet incontournable, mais présente des inconvénients et des lourdeurs liées à l'absence, de prise en compte des outils moderne d'interface (menus, multi-fenêtrage, souris...). De surcroît, SQL, malgré son apparente simplicité, présente des subtilités qui le rendent difficilement utilisable par l'utilisateur final. Pour toutes ces raisons, l'ensemble des constructeurs proposent aujourd'hui des outils (easy SQL, générateur de SQL...) destinés à faciliter l'utilisation interactive de SQL.

IV.4. ANNEXE : SEMANTIQUE ET SYNTAXE DE SQL

a) Sémantique de SQL

Afin de bien comprendre la sémantique d'une requête SQL, il faut se rappeler qu'à chaque ligne d'une question correspond une opération. Les opérations sont interprétées dans l'ordre 1/ 2/ 3/ 4/ 5/ 6/.

<u>SELECT</u> <liste d'attributs>	5/ projection finale ;
<u>FROM</u> <liste de relations>	1/ produit cartésien de toutes les relations citées ;
<u>WHERE</u> <qualification>	2/ applications des restrictions sur les tuples (après le produits cartésien, tous les prédicats élémentaires de la qualification sont des prédicats de restriction) ;
<u>GROUP BY</u> <liste d'attribut>	3/ calcul des agrégats et des fonctions ;
<u>HAVING</u> <qualification>	4/ application d'une restriction sur les groupes résultant d'un calcul de fonction ;
<u>ORDER BY</u> <liste d'attribut>	6/ tri du résultat.

Attention : il s'agit de l'interprétation sémantique de la requête SQL ; il ne faut surtout pas en conclure qu'un SGBD relationnel exécute la requête de cette façon.

b) Règles syntaxiques en grammaire BNF

Query expressions

query-exp	::=	query-term query-exp [UNION INTERSECT MINUS query-term]
query-term	::=	query-spec (query-exp)
query-spec	::=	SELECT [ALL DISTINCT] selection table-exp
selection	::=	scalar-exp-commalist *
table-exp	::=	from-clause [where-clause] [group-by-clause] [having-clause]
from-clause	::=	FROM table-ref-commalist
table-ref	::=	table [range-variable]
where-clause	::=	search-condition
group-by-clause	::=	GROUP BY column-ref-commalist
having-clause	::=	HAVING search-condition

Search conditions

search-condition	::=	boolean-term search-condition OR boolean-term
boolean-term	::=	boolean-factor boolean-term AND boolean-factor
boolean-factor	::=	[NOT] boolean-primary
boolean-primary	::=	predicate (search condition)
predicate	::=	comparison-predicate between-predicate like-predicate test-for-null in-predicate all-or-any-predicate existence-test
comparison-predicate	::=	scalar-exp comparison {scalar-exp subquery}
comparison	::=	= <> < > ≤ ≥
between-predicate	::=	scalar-exp [NOT] BETWEEN scalar-exp AND scalar-exp
like-predicate	::=	column-ref [NOT] LIKE atom [ESCAPE atom]
test-for-null	::=	column-ref IS [NOT] NULL
in-predicate	::=	scalar-exp [NOT] IN { subquery atom [, atom-commalist] }
all-or-any-predicate	::=	scalar-exp comparison [ALL ANY SOME] subquery
existence-test	::=	EXISTS subquery
subquery	::=	(SELECT [ALL DISTINCT] selection table-exp)

Scalar expressions

scalar-exp	::=	term scalar-exp { + - } term
term	::=	factor / term { * / } factor
factor	::=	[+ -] primary
primary	::=	atom column-ref function-ref (scalar-exp)
atom	::=	parameter-ref literal USER
parameter-ref	::=	parameter [[INDICATOR] parameter
function-ref	::=	COUNT (*) / distinct-function-ref / all-function-ref
distinct-function-ref	::=	{ AVG MAX MIN SUM COUNT } (DISTINCT column-ref)
all-function-ref	::=	{ AVG MAX MIN SUM COUNT } ([ALL] scalar-exp)
column-ref	::=	[column-qualifier.] column
column-qualifier	::=	table range-variable

IV.5. REFERENCES

[Date 87] C.J. DATE : "A Guide to the SQL Standard", Addison-Wesley, 1987.

[ISO 86] ISO ANSI, "Database Language SQL", ISO/DIS 9075, Draft International Standard, 1986.

[ISO 88] ISO ANSI, "Database Language SQL2", Working Draft, 1988.

CHAPITRE 5 : BIEN CONCEVOIR UNE BASE DE DONNEES

Nous étudions dans ce chapitre comment bien concevoir une BD. Pour ce faire, à partir d'un même ensemble de connaissances ayant trait aux plages, nous proposons deux choix d'organisation des informations sous forme relationnelle. Nous étudions les qualités et les défauts de ces différents choix avant de présenter les règles de "bonne" conception d'une BD relationnelle.

Premier choix :

BAINS (NN, NOM, PRENOM, QUALITE, DATE, DUREE, NP, NOMP, TYPE, REGION, POLLUTION)

Deuxième choix :

NAGEUR (NN, NOM, PRENOM, QUALITE)

PLAGE (NP, NOMP, TYPE, REGION, POLLUTION)

BAIGNADE (NN, NP, DATE, DUREE)

NN et NP sont des numéros permettant de distinguer respectivement les nageurs et les plages.

NN est l'équivalent du numéro de sécurité sociale.

Nous constatons sur cet exemple l'existence de plusieurs façons de structurer un même ensemble d'informations. Si nous privilégions instinctivement l'une des deux solutions proposées, c'est qu'elle correspond davantage à notre perception du monde réel, dans laquelle nous distinguons naturellement certaines entités : les personnes, les plages, etc.

L'étape de conception est primordiale pour le bon fonctionnement d'un SGBD. Elle fait partie des quelques facteurs qui peuvent entraîner des incohérences dans les réponses et une diminution inacceptable des performances du système ; c'est pourquoi il est indispensable d'y attacher une attention toute particulière.

Une solution "instinctive" n'est pas suffisante pour concevoir le schéma d'une base importante. Il est donc nécessaire d'isoler les critères de décision et de formaliser des méthodes de conception des bases de données. Tel est l'objet de ce chapitre.

Les problèmes les plus courants rencontrés dans des bases de données mal conçues peuvent être regroupés selon les critères suivants :

Redondance des données

Certains choix de conception entraînent une répétition des données lors de leur insertion dans la base. Cette redondance est souvent la cause d'anomalies provenant de la complexité des insertions.

C'est, par exemple, le cas de la première organisation proposée : dès qu'une personne prend un nouveau bain, on doit non seulement répéter son numéro qui, par hypothèse, suffit à le déterminer, mais aussi toutes les informations liées à ce numéro (son nom, son prénom, sa qualité). Au contraire, dans le deuxième choix, seul le numéro indispensable à la distinction d'un baigneur est répété. La situation est identique pour les plages.

Incohérence en modification

La redondance de l'information entraîne également des risques en cas de modification d'une donnée répétée en différents endroits : on oublie fréquemment de modifier toutes ses occurrences (en général par simple ignorance des différentes places où figure l'information).

Par exemple, dans la première organisation proposée, si une personne change de nom (cas fréquent lors de mariages), il faut changer ce nom dans tous les tuples où apparaissent ses coordonnées. Dans la deuxième organisation, un seul tuple est modifié.

Anomalie d'insertion

Une mauvaise conception peut parfois empêcher l'insertion d'un tuple, faute de connaître la valeur de tous les attributs de la relation. Pour remédier à ce problème, certains SGBD implantent une valeur non typée qui signifie que la valeur d'un attribut d'un tuple est inconnue ou indéterminée. Cette valeur (appelée usuellement NULL) indique réellement une valeur inconnue et non une chaîne de caractères vide ou un entier égal à zéro (analogie avec un pointeur égal à NIL en Pascal).

Dans le premier schéma proposé, insérer une nouvelle plage où personne ne s'est jamais baigné est aussi impossible.

Anomalie de suppression

Enfin, une mauvaise conception peut entraîner, lors de la suppression d'une information, la suppression d'autres informations, sémantiquement distinctes, mais regroupées au sein d'un même schéma.

C'est ce qui se produit dans notre premier exemple, la suppression d'une plage entraîne automatiquement la suppression de tous les nageurs ne s'étant baignés que sur cette plage.

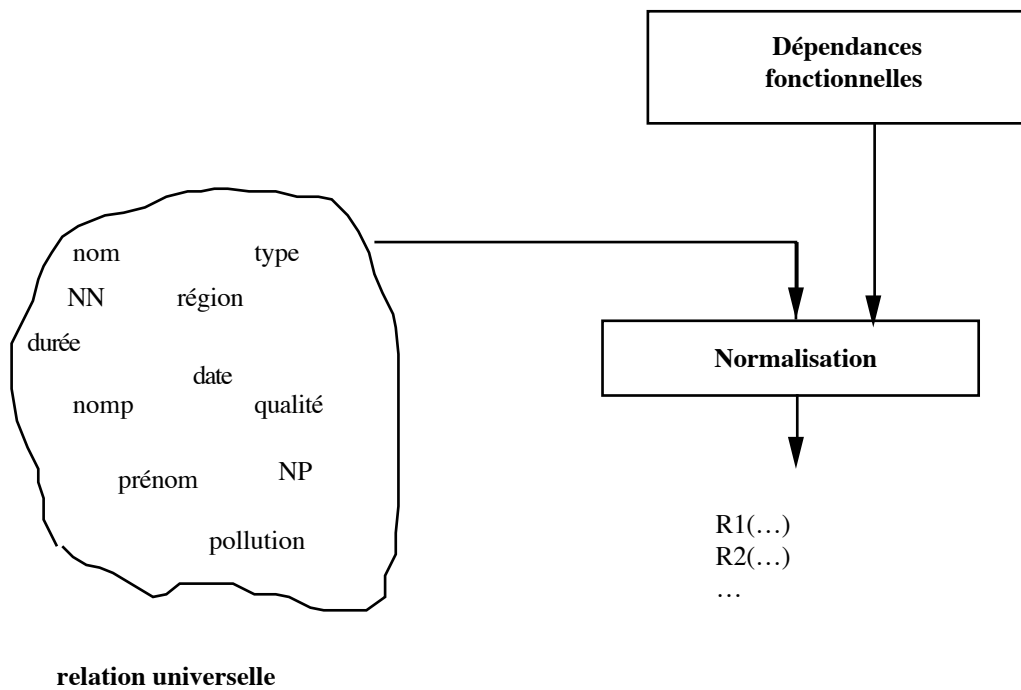
De nombreux travaux ont permis de mettre au point une théorie de conception d'une base de données relationnelle : la théorie de la normalisation, que nous allons maintenant développer.

V.1. LA THEORIE DE LA NORMALISATION

Cette théorie est basée sur les "dépendances fonctionnelles" (DF). Les dépendances fonctionnelles traduisent des contraintes sur les données (par exemple, on décide que deux individus différents peuvent avoir même nom et prénom mais jamais le même numéro NN). Ces contraintes sont représentatives d'une perception de la réalité et imposent des limites à la base.

Les dépendances fonctionnelles et des propriétés particulières, définissent une suite de formes normales (FN). Elles permettent de décomposer l'ensemble des informations en diverses relations. Chaque nouvelle forme normale marque une étape supplémentaire de progression vers des relations présentant de moins en moins de redondance. Ces étapes traduisent une compréhension de plus en plus fine de la réalité.

Chacune de ces formes normales peut être obtenue au moyen d'algorithmes de décomposition. Le point de départ de ces algorithmes est la relation universelle, c'est-à-dire la relation qui regroupe toutes les informations à stocker (dans notre exemple, le premier schéma représente cette relation universelle) ; le but est d'obtenir, en sortie, une représentation canonique des données présentant un minimum de redondance à l'intérieur de chaque relation et un maximum d'indépendance entre les différentes relations .



V.1.1. Les dépendances fonctionnelles

Définition : dépendance fonctionnelle

On dit qu'un attribut B dépend fonctionnellement d'un attribut A si, étant donné une valeur de A, il lui correspond une *unique* valeur de B (quel que soit l'instant t considéré).

Notation : $A \rightarrow B$

Exemple :

La dépendance fonctionnelle $NN \rightarrow NOM$ signifie qu'à un numéro est associé un nom unique (c'est, par exemple, le cas du numéro de sécurité sociale). Remarquons qu'une dépendance fonctionnelle n'est généralement pas symétrique, c'est-à-dire que $NN \rightarrow NOM$ n'interdit pas que deux personnes distinctes (correspondant à deux NN différents) portent le même nom.

Une dépendance fonctionnelle est une propriété définie sur tous les tuples d'une relation et pas seulement sur un tuple particulier. Elle traduit une certaine perception de la réalité (par exemple, deux personnes distinctes peuvent porter le même nom). Elle correspond à une contrainte qui doit être vérifiée en permanence.

Les dépendances fonctionnelles sont parties intégrantes du schéma d'une BD. Elles doivent donc théoriquement être déclarées par l'administrateur et contrôlées par le SGBD.

Exemple :

Nous définissons les propriétés vérifiées par notre base de baignades. Deux personnes distinctes peuvent, par exemple, porter le même nom, le même prénom, et avoir la même qualité de nage. Deux numéros de nageur différent les distinguent l'une de l'autre. Les dépendances fonctionnelles que nous venons de décrire sont donc

$NN \rightarrow NOM, NN \rightarrow PRENOM, NN \rightarrow QUALITE$

Nous pouvons supposer également que deux plages distinctes ont toujours deux numéros différent ; ce qui implique :

$NP \rightarrow NOMP$

$NP \rightarrow REGION$

que la pollution et le type sont propres à une plage :

$NP \rightarrow POLLUTION$

$NP \rightarrow TYPE$

et que deux plages d'une même région ne peuvent porter le même nom :

$(NOMP, REGION) \rightarrow NP$

Propriétés des dépendances fonctionnelles

Les dépendances fonctionnelles obéissent à certaines propriétés connues sous le nom d'axiomes d'Armstrong.

Réflexivité :

$Y \subset X \quad \Rightarrow \quad X \rightarrow Y$

Augmentation :

$X \rightarrow Y \quad \Rightarrow \quad XZ \rightarrow YZ$

Transitivité :

$X \rightarrow Y \text{ et } Y \rightarrow Z \quad \Rightarrow \quad X \rightarrow Z$

D'autres propriétés se déduisent de ces axiomes :

Union :

$X \rightarrow Y \text{ et } X \rightarrow Z \quad \Rightarrow \quad X \rightarrow YZ$

Pseudo-transitivité :

$X \rightarrow Y \text{ et } YW \rightarrow Z \quad \Rightarrow \quad XW \rightarrow Z$

Décomposition :

$X \rightarrow Y \text{ et } Z \subset Y \quad \Rightarrow \quad X \rightarrow Z$

L'intérêt de ces axiomes et des propriétés déduites est de pouvoir construire, à partir d'un premier ensemble de dépendances fonctionnelles, l'ensemble de toutes les dépendances fonctionnelles qu'elles génèrent. On parle alors de dépendance fonctionnelle élémentaire.

V.1.2. Dépendance fonctionnelle élémentaire

Définition : dépendance fonctionnelle élémentaire

Une dépendance fonctionnelle $X \rightarrow A$ est élémentaire si

- A n'est pas inclus dans X ;
- il n'existe pas X' inclus dans X tel que $X' \rightarrow A$.

Cette notion de dépendance fonctionnelle élémentaire est primordiale car elle permet de construire une sorte de famille génératrice minimale (appelée couverture minimale) des dépendances fonctionnelles utiles pour structurer la base.

Exemple :

Deux plages d'une même région ne peuvent pas porter le même nom (contrairement à deux plages de régions différentes) ; le degré de pollution d'une plage dépend exclusivement de la plage et non de la région. On a alors

$(NOMP, REGION) \rightarrow POLLUTION$

mais on n'a aucunement

$NOMP \rightarrow POLLUTION$

ni

$REGION \rightarrow POLLUTION$

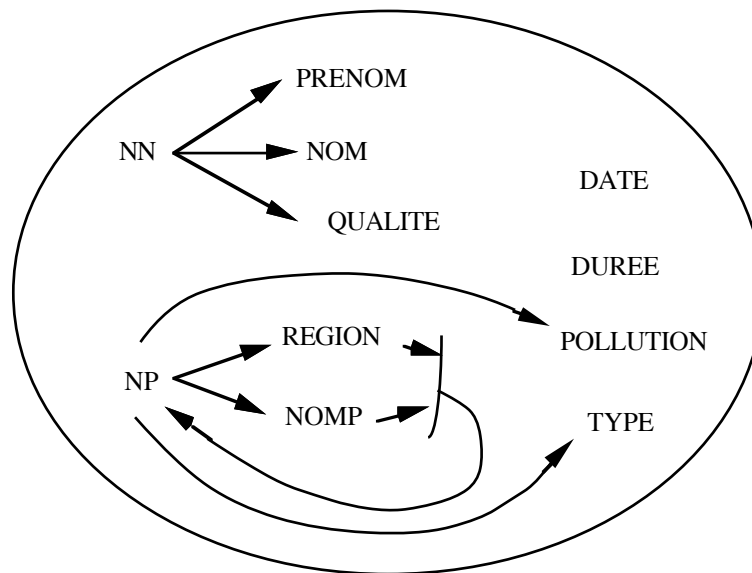
donc $(NOMP, REGION) \rightarrow POLLUTION$ est une dépendance fonctionnelle élémentaire.

V.1.3. Graphe des dépendances fonctionnelles

C'est une représentation graphique permettant de visualiser aisément toutes les dépendances fonctionnelles et d'isoler les principales (i.e. les DF élémentaires).

Exemple :

Toutes les dépendances fonctionnelles citées précédemment peuvent être représentées de la façon suivante.



V.1.4. Fermeture transitive

Définition : fermeture transitive

La fermeture transitive d'un ensemble de dépendances fonctionnelles est ce même ensemble enrichi de toutes les dépendances fonctionnelles déduites par transitivité.

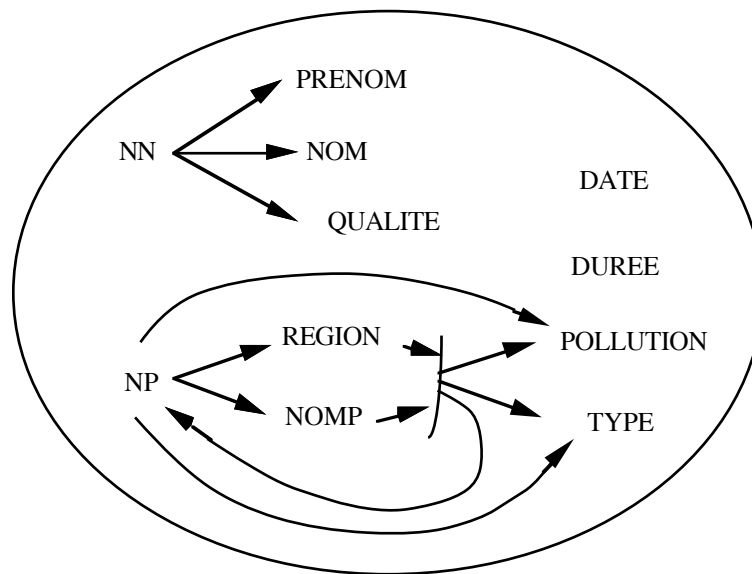
Exemple :

De l'exemple précédent , on déduit par transitivité deux nouvelles dépendances fonctionnelles :

$(NOMP, REGION) \rightarrow TYPE$

$(NOMP, REGION) \rightarrow POLLUTION$

qui enrichit le graphe de la façon suivante :



V.1.5. Couverture minimale

Définition : couverture minimale

La couverture minimale d'un ensemble de dépendances fonctionnelles est un sous ensemble minimum de dépendances fonctionnelles élémentaires permettant de générer toutes les autres.

Exemple :

L'ensemble des dépendances fonctionnelles suivant :

$(NN \rightarrow NOM, NN \rightarrow PRENOM, NN \rightarrow QUALITE, NP \rightarrow NOMP, NP \rightarrow REGION, (NOMP, REGION) \rightarrow POLLUTION, (NOMP, REGION) \rightarrow NP)$

est une couverture minimale de l'ensemble des dépendances fonctionnelles.

Théorème :

Tout ensemble de dépendances fonctionnelles admet une couverture minimale, en général non unique.

Exemple :

L'ensemble des dépendances fonctionnelles suivant :

$(NN \rightarrow NOM, NN \rightarrow PRENOM, NN \rightarrow QUALITE, NP \rightarrow NOMP, NP \rightarrow REGION, NP \rightarrow POLLUTION, (NOMP, REGION) \rightarrow NP)$

est une autre couverture minimale de l'ensemble des dépendances fonctionnelles.

La recherche de la couverture minimale d'un ensemble de dépendances fonctionnelles est un élément essentiel du processus de normalisation, dont le but est de décomposer une relation en plusieurs relations plus petites.

V.1.6. Clé d'une relation

Définition : clé d'une relation

Soit $R(A_1, A_2, \dots, A_N)$ un schéma de relation, soit F^+ l'ensemble des dépendances fonctionnelles associées à R , soit X un sous-ensemble de A_1, A_2, \dots, A_N , on dit que X est une clé de R si et seulement si

- $X \rightarrow A_1, A_2, \dots, A_N$;
- il n'existe pas de sous ensemble Y inclus dans X tel que $Y \rightarrow A_1, A_2, \dots, A_N$.

Une clé d'une relation est un ensemble minimum d'attributs qui détermine tous les autres.

Exemple :

NN est clé de la relation PERSONNE (NN, NOM, PRENOM, QUALITE) ;
(NOMP, REGION) est clé de la relation (NP, NOMP, REGION, POLLUTION, TYPE).

Plusieurs clés peuvent être candidates pour une même relation.

Exemple :

NP et (NOMP, REGION) sont deux clés candidates à la relation (NP, NOMP, REGION, POLLUTION, TYPE).

Dans l'écriture des schémas de relations, on indique les clés en soulignant les attributs constitutifs.

Exemple :

PLAGE (NP, NOMP, REGION, POLLUTION, TYPE)

V.1.7. Décomposition des relations

La théorie de la normalisation repose sur un principe de décomposition des relations.

Définition : décomposition d'une relation

La décomposition d'un schéma de relation $R(A_1, A_2, \dots, A_N)$ est son remplacement par une collection de schémas de relations (R_1, R_2, \dots, R_i) telle que :

$$\text{SCHEMA}(R) = \text{SCHEMA}(R_1) \cup \text{SCHEMA}(R_2) \cup \dots \cup \text{SCHEMA}(R_i).$$

Définition : décomposition sans perte

Une décomposition d'une relation R en N relations R_1, R_2, \dots, R_N est sans perte si et seulement si, pour toute extension de R , on a :

$$R = R_1 \bowtie R_2 \bowtie \dots \bowtie R_N$$

Théorème : décomposition sans perte

Une décomposition en deux relations est sans perte si l'attribut de jointure de la recombinaison est clé d'une au moins des deux relations.

Définition : décomposition préservant les dépendances fonctionnelles

Une décomposition (R_1, R_2, \dots, R_N) de R préserve les dépendances fonctionnelles si la fermeture des dépendances fonctionnelles de R est la même que celle de l'union des dépendances fonctionnelles des relations R_1, R_2, \dots, R_N .

V.2. LES TROIS PREMIERES FORMES NORMALES

Nous définissons ici des règles de décomposition la relation universelle sans perdre d'information en utilisant les dépendances fonctionnelles. Le but est d'obtenir une représentation du monde réel qui minimise la redondance et les risques d'anomalies lors des mises à jour.

V.2.1. La première forme normale 1FN

Définition : première forme normale

Une relation est en première forme normale si tout attribut est atomique.

Conséquences :

- un attribut représente une donnée élémentaire du monde réel ;
- un attribut ne peut désigner, ni une donnée composée d'entités de nature quelconque, ni une liste de données de même nature.

La première forme normale correspond à un souci de simplicité au niveau du langage de manipulation. Elle a pour particularité d'éviter toute hiérarchie dans une relation en interdisant les domaines composés de plusieurs valeurs.

Exemple :

Nous pourrions imaginer une relation BAINS de schéma BAINS (NN, NP, DATE, DUREES) où DUREES serait la liste des durées des bains pris par le nageur NN sur la plage NP à la date DATE. A lieu de cela, la première forme normale nous oblige à décomposer cette relation en BAINS (NN, NP, DATE, DUREE) ce qui induira autant de tuples qu'il y a de baignade d'un même baigneur, sur la même plage, le même jour ; avec (nn, np, date, durée_i) pour le $i^{\text{ème}}$ bain de la journée.

V.2.2. La deuxième forme normale 2FN

Définition : deuxième forme normale

Une relation est en deuxième forme normale si et seulement si :

- elle est en première forme normale ;
- tout attribut n'appartenant pas à une clé ne dépend pas que d'une partie de cette clé.

Exemple :

Considérons la relation PLAGE de schéma suivant :

PLAGE (NOMP, REGION, TYPE, POLLUTION)

où la clé est (NOMP, REGION). Supposons que la pollution est bien dépendante de la plage mais que le type est, quant à lui, dépendant de la région. La deuxième forme normale nous impose de distinguer deux relations R1 et R2 de schémas respectifs :

R1 (NOMP, REGION, POLLUTION) ;

R2 (REGION, TYPE).

V.2.3. La troisième forme normale 3FN

L'objectif de cette troisième forme normale est l'élimination des redondances dues aux dépendances fonctionnelles déduites par transitivité.

Définition : troisième forme normale

Une relation est en troisième forme normale si :

- elle est en deuxième forme normale ;
- tout attribut n'appartenant pas à une clé ne dépend pas d'un attribut non clé.

Théorème :

Toute relation R admet au moins une décomposition (R_1, R_2, \dots, R_N) en troisième forme normale telle que :

- la décomposition conserve les dépendances fonctionnelles ;
- la décomposition est sans perte.

Exemple :

Considérons maintenant la relation PLAGE de schéma PLAGE (NP, REGION, TYPE, POLLUTION) où la clé est NP. Supposons maintenant comme dans l'exemple précédent que le type est dépendant de la région. La troisième forme normale nous impose de distinguer deux relations R1 et R2 de schémas respectifs :

R1 (NP, REGION, POLLUTION) ;

R2 (REGION, TYPE).

V.2.4. Algorithme de décomposition en troisième forme normale

Cet algorithme accepte en entrée la relation universelle à décomposer et les dépendances fonctionnelles de la relation.

Principe de l'algorithme :

- 1- à partir du graphe G des dépendances fonctionnelles, CALCULER une couverture minimale C ;

- 2- **EDITER** l'ensemble des attributs isolés dans une même relation (tous les attributs sont clés) ;
- 3- **RECHERCHER** le plus grand ensemble X d'attributs qui détermine d'autres attributs ;
- 4- **EDITER** la relation $R(X, A_1, \dots, A_N)$ qui est en troisième forme normale ;
- 5- **SUPPRIMER** toutes les dépendances fonctionnelles figurant dans R du graphe de couverture minimale C ;
- 6- **SUPPRIMER** les attributs isolés de C (c'est-à-dire les attributs non sources ou cibles de dépendances fonctionnelles) ;
- 7- **REPETER** l'opération de réduction du graphe C à partir de l'étape 3 jusqu'à ce que C soit vide.

Remarque : Cet algorithme fournit bien une décomposition en 3FN qui préserve les DF mais qui n'est pas nécessairement sans perte. Pour avoir une décomposition sans perte, il suffit d'ajouter à la décomposition finale une relation composée des *attributs formant la clé de la relation universelle* (ou ajouter ces attributs à la relation créée à l'étape 2, si elle existe).

V.2.5. Insuffisance de la troisième forme normale

Il s'agit ici de détecter les boucles pouvant exister dans le graphe des dépendances fonctionnelles. Supposons, par exemple une relation PLAGE indiquant le nom, la région et le canton où se situent les plages. Supposons en outre que deux plages d'une même région ne puissent pas porter le même nom et qu'il n'existe pas deux cantons de même nom. La relation se note :

PLAGE (NOMP, REGION, CANTON)

et possède les dépendances fonctionnelles suivantes :

$(NOMP, REGION) \rightarrow CANTON$ et $CANTON \rightarrow REGION$

Bien que la relation soit en 3ème forme normale, il existe encore des redondances dues au fait qu'un attribut non clé détermine *une partie* de la clé. Afin d'éliminer ce type de redondance, BOYCE et CODD ont introduit une nouvelle forme normale :

Définition : Forme Normale de BOYCE-CODD (BCNF) :

Une relation est en BCNF si et seulement si les seules dépendances fonctionnelles élémentaires sont celles dans lesquelles une clé détermine un attribut. Pour l'exemple précédent une décomposition en BCNF serait :

PLAGE (NOMP, CANTON) et GEO(CANTON, REGION)

Théorème de décomposition en BCNF :

Toute relation admet au moins une décomposition en BCNF qui est sans perte ; cependant, une telle décomposition ne préserve généralement pas les dépendances fonctionnelles. Dans l'exemple précédent on ne préserve pas la première DF.

V.3. LA QUATRIEME FORME NORMALE

La notion de dépendance fonctionnelle nous a conduit à décomposer les relations en troisième forme normale et en forme normale de BOYCE CODD. Ceci est pourtant insuffisant pour éliminer les redondances et les anomalies de mises à jour. Considérons, par exemple, une relation PREFERENCES où nous indiquons, pour chaque nageur, ses différents domiciles et les plages qu'il a l'habitude de fréquenter :

PREFERENCES (NN, VILLE, NP)

Il n'existe aucune dépendance fonctionnelle entre les différents attributs, ce qui conduit à des situations du type :

PREFERENC	NN	VILLE	NP
	1	Caen	115
	1	Caen	117
	2	Nice	119
	2	Nice	115
	2	Evreux	119
	2	Evreu	115

Dans cette relation, tout est clé : elle est en 3ème forme normale. Pourtant, il subsiste de nombreuses redondances. La relation n'est pas décomposable selon les critères que nous venons d'évoquer, ce qui nous conduit à introduire la notion de dépendance fonctionnelle multivaluée.

V.3.1. Les dépendances multivaluées

Définition : Dépendance Multivaluée

Soit $R(A_1, A_2, \dots, A_n)$ un schéma de relation. Soient X et Y des sous-ensembles de A_1, A_2, \dots, A_n . On dit que $X \twoheadrightarrow Y$ (X multi-détermine Y , ou il y a une dépendance multivaluée

de Y sur X) si, étant données des valeurs de X, il y a un ensemble de valeurs de Y associées et cet ensemble est indépendant des autres attributs $Z = R - X - Y$ de la relation R.

$$(X \twoheadrightarrow Y) \Leftrightarrow ((xyz) \text{ et } (xy'z') \in R \Rightarrow (xy'z) \text{ et } (xyz') \in R)$$

Propriété :

Les dépendances fonctionnelles sont des cas particuliers de dépendances multivaluées :

$$(X \rightarrow Y) \Leftrightarrow ((xyz) \text{ et } (xy'z') \in R \Rightarrow y = y')$$

De même que pour les dépendances fonctionnelles, il existe des axiomes d'inférence de dépendances multivaluées. Ce sont les suivants :

- Complémentation : $(X \twoheadrightarrow Y) \Rightarrow (X \twoheadrightarrow R - X - Y)$
- Augmentation : $(X \twoheadrightarrow Y) \text{ et } (V \subset W) \Rightarrow (XW \twoheadrightarrow YV)$
- Transitivité : $(X \twoheadrightarrow Y) \text{ et } (Y \twoheadrightarrow Z) \Rightarrow (X \twoheadrightarrow Z - Y)$

Dont on peut déduire d'autres règles telles l'union :

- Union : $(X \twoheadrightarrow Y) \text{ et } (Y \twoheadrightarrow Z) \Rightarrow (X \twoheadrightarrow YZ)$

A partir de ces axiomes, on introduit la notion de Dépendance Multivaluée Élémentaire :

Définition : Dépendance Multivaluée Élémentaire

Une dépendance multivaluée élémentaire $X \twoheadrightarrow Y$ d'une relation R est une dépendance multivaluée telle que :

- Y n'est pas vide et est disjoint de X ;
- R ne contient pas une autre Dépendance Multivaluée du type $X' \twoheadrightarrow Y'$ telle que $X' \subset X$ et $Y' \subset Y$.

Ainsi, dans l'exemple précédent nous isolons les Dépendances Multivaluées Élémentaires suivantes :

$$NN \twoheadrightarrow VILLE \text{ et } NN \twoheadrightarrow NP$$

V.3.2. Quatrième forme normale

Définition : quatrième forme normale

Une relation est en quatrième forme normale si et seulement si, lorsqu'il existe une dépendance multivaluée élémentaire, celle-ci est unique.

Propriété :

Toute relation a une décomposition en quatrième forme normale qui est sans perte. Cette décomposition n'est pas forcément unique.

Du fait qu'une dépendance fonctionnelle est un cas particulier de dépendance multivaluée, nous pouvons déduire qu'une relation en quatrième forme normale est en forme normale de BOYCE-CODD et donc en troisième forme normale.

Sur l'exemple que nous venons de voir (PREFERENCES), la clé est l'ensemble des trois attributs et il existe deux dépendances multivaluées : la relation n'est donc pas en quatrième forme normale. Une décomposition de cette relation en quatrième forme normale donne deux relations PREF1 (NN, VILLE) et PREF2 (NN, NP)

V.4. LA CINQUIEME FORME NORMALE

La quatrième forme normale n'est pas encore suffisamment poussée pour éliminer tous les risques de redondances et d'anomalies.

Exemple :

Nos précédents nageurs sont amateurs de fruits de mer et en consomment couramment quand ils vont sur leurs plages préférées. Voyons une extension de la relation CONSOMMATION (NAGEUR, CRUSTACES, VILLE) :

CONSOMMATION	NAGEUR	CRUSTACES	VILLE
	Plouf	Tourteau	Binic
	Plouf	Tourteau	Trouville
	Plouf	Langouste	Trouville
	Coule	Tourteau	Trouville

Nous constatons aisément que cette relation présente de la redondance. Cette redondance provient d'une contrainte d'intégrité implicite dans la conception du monde réel :

"Tout nageur ayant consommé un type de crustacés et ayant séjourné dans une ville les cultivant a consommé de ce type de crustacés dans cette ville."

Ainsi, le fait que Plouf ait mangé du tourteau est répété à plusieurs reprises, etc.

Toutefois, cette relation est en quatrième forme normale car il n'y a pas de dépendance multivaluée. Pour nous en assurer, nous pouvons considérer les projections B1, B2, B3 de la relation CONSOMMATION sur les différents couples d'attributs C1(NAGEUR, CRUSTACES),

$C2(\text{NAGEUR}, \text{VILLE})$ et $C3(\text{CRUSTACES}, \text{VILLE})$, et constater que la jointure de deux de ces relations ne redonne jamais la relation CONSOMMATION. La relation n'est donc pas décomposable en deux autres relations.

V.4.1. Les dépendances de jointure

Le problème provient du fait que nous avons jusqu'alors toujours essayé de décomposer une relation en deux relations. Comme nous allons le montrer ci-dessous, certaines relations sont décomposables, non pas en deux, mais en trois relations. C'est le cas dans notre exemple en raison de la contrainte d'intégrité que nous rappelons ci-dessous :

"Tout nageur ayant consommé un type de crustacés et ayant séjourné dans une ville les cultivant a consommé de ce type de crustacés dans cette ville."

qui se note de façon plus formelle :

$(\text{nageur}, \text{crustacés}) \in C1$ et $(\text{nageur}, \text{ville}) \in C2$ et $(\text{crustacés}, \text{ville}) \in C3$
 $\Rightarrow (\text{nageur}, \text{crustacés}, \text{ville}) \in \text{CONSOMMATION}$

Définition : dépendance de jointure

Soit $R(A_1, A_2, \dots, A_n)$ un schéma de relation et X_1, X_2, \dots, X_m des sous-ensembles de (A_1, A_2, \dots, A_n) . On dit qu'il existe une dépendance de jointure $*(X_1, X_2, \dots, X_m)$ si R est la jointure de ses projections sur X_1, X_2, \dots, X_m , c'est-à-dire si :

$$R = \pi_{X_1(R)} |x| \pi_{X_2(R)} |x| \dots |x| \pi_{X_m(R)}.$$

Par exemple, la relation CONSOMMATION obéit à la dépendance de jointure suivante :

$*(\text{NAGEUR CRUSTACES}, \text{NAGEUR VILLE}, \text{CRUSTACES VILLE})$

Propriété :

Les dépendances multivaluées sont des cas particuliers de dépendances de jointure :

$$(X \twoheadrightarrow Y) \Rightarrow ((xyz) \text{ et } (xy'z') \in R \Rightarrow y = y')$$

V.4.2. Cinquième forme normale

La cinquième forme normale est une généralisation de la quatrième forme normale qui nécessite de prendre en compte les dépendances de jointure induites par la connaissance des clés d'une relation.

Définition : Cinquième Forme Normale

Une relation R est en cinquième forme normale si et seulement si toute dépendance de jointure est impliquée par des clés candidates de R.

V.5. CONCLUSIONS

Dans ce chapitre, nous avons présenté la théorie de la normalisation des schémas relationnels. Cette normalisation est très importante dans la pratique si l'on veut éviter de stocker des informations redondantes. On considère, en général, que la troisième forme normale est suffisante dans les cas courants.

Dépendant de la sémantique des données, le processus de normalisation reste à la charge de l'utilisateur du SGBD. Cette phase, délicate, est aujourd'hui largement facilitée par des outils d'aide à la conception que proposent différents constructeurs.

V.6. RÉFÉRENCES

- | | |
|--------------|--|
| [Codd 74] | E.F. CODD, " <i>Recent Investigations in Relational Database Systems</i> ", IFIP Congrès 1974, North-Holland Ed., pp 1017-1021. |
| [Fagin 79] | R. FAGIN, " <i>Normal Forms and Relational Database Operators</i> ", ACM SIGMOD 1979, Boston, Juin 1979, pp 153-160. |
| [Nicolas 78] | J.M. NICOLAS, " <i>Mutual Dependencies and Some Results on Undecomposable Relations</i> ", 5 th Very Large Data Bases, Berlin 1978, IEEE Ed., pp 360-367. |

CHAPITRE 6 : LA PROTECTION DE L'INFORMATION

Comme nous l'avons évoqué dans le premier chapitre, un SGBD comprend trois niveaux d'abstraction différents. Le troisième chapitre est consacré au niveau conceptuel qui regroupe et gère l'ensemble des informations de l'entreprise. Le prochain s'intéresse aux méthodes de stockage du niveau interne. Nous nous plaçons ici au niveau externe, là où intervient l'utilisateur final dont on doit pouvoir contrôler les actions sur les données de la base.

La démarche consistant à centraliser les données induit un certain nombre de problèmes, tant pour l'utilisateur final d'un SGBD que pour les programmes chargés de contrôler la protection, et la cohérence des informations de la base.

La vision de l'entreprise qu'à l'utilisateur final est limitée à l'ensemble des informations liées à son travail (par exemple, le service du personnel ne se préoccupe pas de la gestion des stocks). Elle ne correspond donc pas au schéma global tel qu'il est décrit au niveau conceptuel. Le SGBD doit prendre en compte ce problème en recréant l'environnement de travail habituel de chaque utilisateur.

En outre, il est indispensable de limiter l'accès des utilisateurs aux informations de l'entreprise. De même que de nombreux systèmes d'exploitation offrent des possibilités de protection sur les fichiers par affectation de droits de lecture, d'écriture ou d'exécution, des moyens d'assurer la confidentialité des informations contenues dans la base doivent exister, dans les SGBD.

Toutefois, à la différence des systèmes d'exploitation, les objets sur lesquels ces droits doivent pouvoir être distribués sont plus variés. L'utilisateur d'un système d'exploitation possède un droit sur l'ensemble d'un fichier. L'exigence typique d'un utilisateur de SGBD est de demander des droits d'accès, non pas à une relation du niveau conceptuel, mais à un ensemble d'informations. Celles-ci sont issues d'un certain nombre de relations de ce niveau. La seule particularité de cet ensemble est sa conformité à un prédicat (cet ensemble de données répond à un critère de sélection).

En outre, la confidentialité n'est pas le seul facteur de contrôle des données d'un SGBD. Introduisons maintenant la notion d'intégrité des données qui résulte de l'existence de liens

sémantiques (appelés contraintes d'intégrité) entre les données de la base, lesquels doivent être vérifiés après chaque modification de la base.

VI.1. LES VUES EXTERNES

Dans les modèles à base de pointeurs (hiérarchique et réseau), sont limitées les possibilités d'accès de l'utilisateur à des sous-schémas stricts du schéma de la base. Un certain nombre de champs du schéma initial peuvent être cachés ou renommés, mais la structure de l'information reste la même. Il est, par exemple, impossible à l'utilisateur final de définir son propre schéma provenant du "rapprochement" (ou jointure en relationnel) de deux schémas du niveau conceptuel. Ce qui signifie par exemple :

- que l'on peut connaître les noms, prénoms et niveaux de tous les nageurs (masquage du numéro NN et remplacement de QUALITE en NIVEAU) ;
- que l'on ne peut pas connaître les noms et prénoms des nageurs d'excellent niveau (car on ne peut pas isoler les enregistrements des excellents nageurs des autres) ;
- que l'on ne peut pas connaître uniquement les noms de tous les nageurs et les dates auxquelles ces nageurs se sont baignés (impossibilité de changer la structure des enregistrements visibles donc de visualiser un résultat du type jointure dans le modèle relationnel).

La solution idéale à ces nombreux problèmes est de recréer, pour chaque utilisateur, sa propre vision de l'entreprise (sans avoir bien entendu à répéter l'information au sein de la base) en laissant au SGBD le soin de gérer la confidentialité et l'intégrité des données.

Le mécanisme des vues des SGBD relationnels a l'ambition de résoudre un grand nombre de problèmes que les modèles précédents n'ont pas su traiter. Nous verrons successivement les objectifs de ce système et constaterons ensuite la facilité de leur mise en œuvre. Nous évoquerons enfin, les limites de ce mécanisme.

VI.1.1. Les objectifs : voir le monde comme il n'est pas !

Le mécanisme des vues permet de répondre aux besoins suivants :

- adaptation aux applications

Les applications ne doivent pas dépendre du schéma adopté au niveau conceptuel. L'utilisateur manipule plus naturellement les données sous leur forme habituelle plutôt que de devoir les retrouver et les traiter au sein d'une quantité importante de données parasites.

- intégration des applications existantes

Les données vues par l'utilisateur étant les mêmes que celles disponibles avant d'avoir accès à un SGBD, il doit pouvoir, grâce à de faibles modifications, aisément adapter ses anciennes applications.

- dynamique du schéma de la base

Les modifications du schéma conceptuel, parfois nécessaires pour des raisons de performances ou de nouveaux besoins, ne doivent pas impliquer de changements des schémas du niveau externe si ces derniers n'évoluent pas. Si cette règle est respectée, ces modifications n'entraînent pas non plus la réécriture des applications du niveau externe.

- confidentialité et sécurité

L'administrateur de la base définit les vues et les utilise pour définir précisément les droits d'accès d'un utilisateur. Ces derniers peuvent ainsi dépendre de la valeur des informations.

- décentralisation de l'administration d'une BD

Chaque utilisateur ou chaque équipe de développement évoluant au niveau externe doit être responsable des données qu'il gère avec ses propres droits. Il devient administrateur de sa vue et peut déléguer ses droits à un autre utilisateur.

- hétérogénéité des modèles

Pouvoir présenter un accès homogène et unifié à un ensemble de bases de données de types différents est particulièrement intéressant. Le mécanisme de vue permet cette vision unique.

VI.1.2. Le relationnel simplifie les vues

Le modèle relationnel, grâce à la manipulation ensembliste des données, autorise la définition d'un ensemble particulier de tuples à l'aide d'un critère de recherche. Ces tuples peuvent être composés à partir de plusieurs relations de base. Le résultat de n'importe quelle requête est une relation au même titre que n'importe quelle relation effectivement implantée en machine ; les résultats d'une requête constituent ainsi une relation ordinaire qui peut servir de base à une nouvelle manipulation.

Définir une vue revient alors à déterminer le critère qui permet de rassembler les tuples désirés et de les présenter sous la forme souhaitée. Une telle opération se réalise aisément à l'aide de l'algèbre relationnelle, mais est encore simplifiée par l'utilisation des langages assertionnels des systèmes relationnels actuels.

Dans un système relationnel, la notion d'une vue est donc quasi identique à une interrogation de la base. Attention cependant : la création d'une vue *définit* le critère d'obtention de la vue

désirée. Les tuples résultats ne sont pas réellement constitués ; le mécanisme reste entièrement virtuel. Nous allons maintenant en donner quelques exemples.

Exemple 1 :

Créer la vue des nageurs dont la qualité est bonne ou excellente.

```
CREATE VIEW BONS_NAGEURS AS  
SELECT NN, NOM, PRENOM, QUALITE  
FROM NAGEURS  
WHERE QUALITE IN ('Excellente', 'Bonne');
```

Exemple 2 :

Des vues peuvent être définies à partir d'autres vues . Créer la vue des nageurs appartenant à BONS_NAGEURS et dont la qualité est excellente.

```
CREATE VIEW CHAMPIONS AS  
SELECT NN, NOM, PRENOM  
FROM BONS_NAGEURS  
WHERE QUALITE = 'Excellente';
```

Exemple 3 :

On peut aussi créer la vue des mauvais nageurs :

```
CREATE VIEW MAUVAIS_NAGEURS AS  
SELECT NOM, PRENOM  
FROM NAGEURS  
WHERE QUALITE = 'Mauvaise';
```

Exemple 4 :

Des vues peuvent également être créées à partir de critères plus complexes. Par exemple, créer la vue des exploits, c'est-à-dire des nageurs ayant pris des bains de plus de 10 heures. On souhaite conserver des informations sur le nom et le prénom du nageur, le nom de la plage où a eu lieu l'exploit, ainsi que la date et la durée du bain.

```

CREATE VIEW EXPLOITS AS
SELECT N.NN, N.NOM, N.PRENOM, N.QUALITE, P.NOMP, B.DATE, B.DUREE
FROM NAGEUR N, BAINADE B, PLAGE P
WHERE N.NN = B.NN
AND B.NP = P.NP
AND B.DUREE > 600mn ;

```

VI.2. MANIPULATION AU TRAVERS DES VUES

VI.2.1. Consultation au travers des vues

L'interrogation exprimée sur une vue est toujours possible. Une vue est une relation virtuelle. Chaque requête portant sur la vue est transformée en une requête portant sur les relations réelles du niveau conceptuel. Celles à partir desquelles la vue a été définie.

Exemple 1 :

Rechercher, dans la vue BONS_NAGEURS, les noms des excellents nageur se prénommant Paul.

```

SELECT NOM
FROM BONS_NAGEURS
WHERE QUALITE = 'Excellente'
AND PRENOM = 'Paul';

```

Cette requête, exprimée au niveau conceptuel, devient :

```

SELECT NOM
FROM NAGEURS
WHERE QUALITE IN ('Excellente', 'Bonne')
AND QUALITE = 'Excellente'
AND PRENOM = 'Paul';

```

Après optimisation elle devient :

```

SELECT NOM
FROM NAGEURS
WHERE QUALITE = 'Excellente'
AND PRENOM = 'Paul';

```

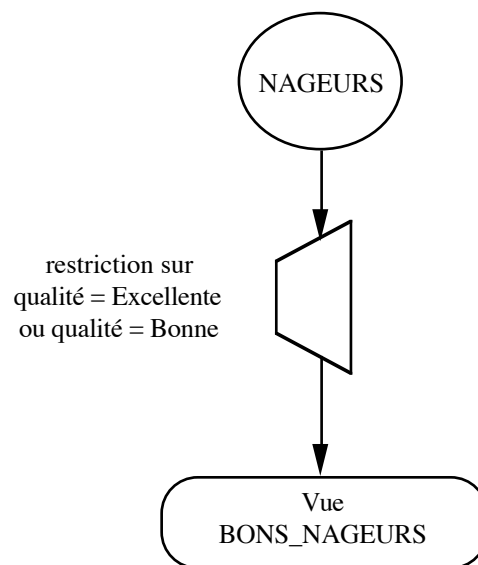
VI.2.2. Influence sur l'optimisation des requêtes

Nous avons déjà mis en évidence dans le modèle relationnel la simplicité du concept de vue et de sa mise en œuvre. Dans ce paragraphe, nous revenons sur cette simplicité en étudiant son influence sur l'optimisation des requêtes.

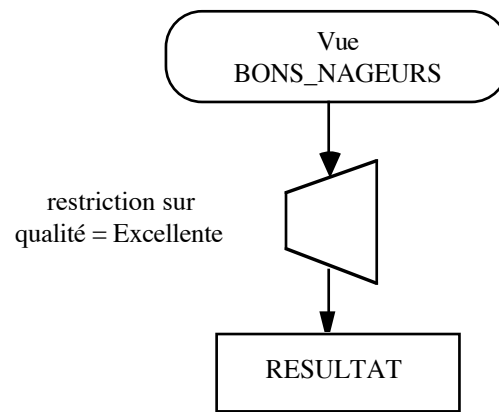
Les vues sont des relations fictives (ou virtuelle, ou dérivée). Les requêtes formulées sur ces vues sont transformées, en requêtes à appliquer à des relations de base réellement implantées en machine.

Comment le SGBD transforme-t-il le niveau vue en niveau conceptuel ? Il concatène simplement la requête de l'utilisateur à la requête créant la vue et optimise l'ensemble. Nous visualisons aisément ceci à l'aide des arbres algébriques.

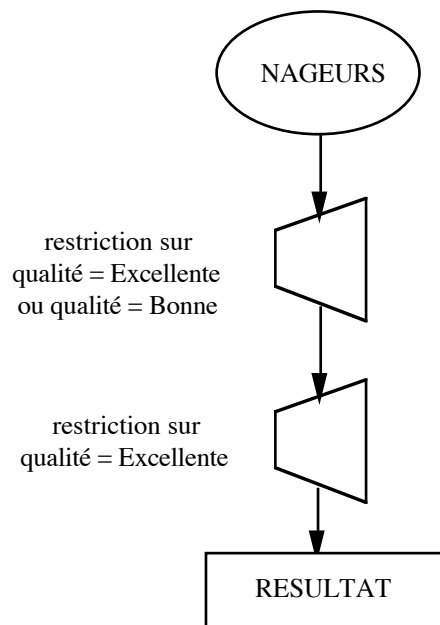
Créer la vue correspond à un arbre algébrique :



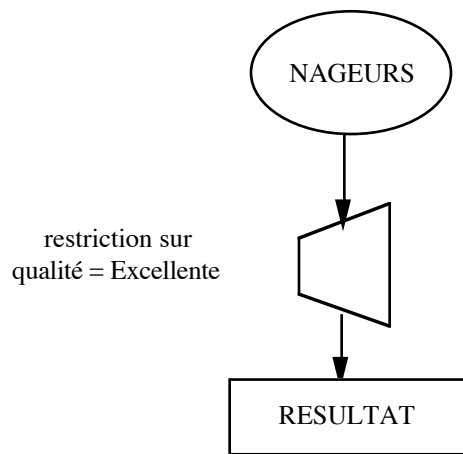
la requête sur la vue correspond également à un arbre algébrique :



La requête transformée finalement appliquée aux relations correspond à un arbre algébrique résultant de la concaténation des deux précédents.



L'optimiseur peut en conséquence modifier l'arbre pour accélérer son exécution.



Cet exemple d'optimisation, ici très simple, ne met en jeu que des restrictions. Cette étape est cependant importante et parfois complexe : mettre bout à bout deux arbres optimisés séparément ne donne pas nécessairement un arbre global optimal. Repasser par l'optimiseur pour déterminer l'arbre optimal (descente des opérateurs unaires, ordonnancement des jointures...) est indispensable.

VI.2.3. Mise à jour à travers les vues

De nouvelles difficultés surgissent lors de la mise à jour. Même si l'utilisateur possède des droits de modification sur la vue, les modifications qu'il souhaite apporter à la base demeurent indéfinies dans de nombreux cas. Voyons quelques exemples ;

Exemple 1 :

Peut-on insérer par la vue BONS_NAGEURS le tuple (73, 'Albert', 'COULE', 'Excellente') ? (on suppose que la clé NN=73 est libre). Cette insertion ne pose pas de problème si l'utilisateur possède un droit d'accès en insertion sur la vue. Tous les champs de la relation conceptuelle NAGEURS dont dépend la vue sont définis. L'insertion peut avoir lieu.

Exemple 2 :

Peut-on insérer à travers la vue CHAMPIONS le tuple (73, 'Albert', 'COULE') ? (on suppose que la clé NN=73 est libre). Ce cas est déjà plus complexe, mais cette vue correspond à une seule valeur de l'attribut manquant QUALITE. L'insertion du tuple devient celle du tuple (73, 'Albert', 'COULE', 'Excellente') dans la relation NAGEURS.

Exemple 3 :

Voyons maintenant un exemple de modifications. Si la modification au sein de la vue des BONS_NAGEURS du tuple (73, 'Albert', 'COULE', 'Excellente') en (73, Albert', 'COULE',

'Bonne') semble naturelle, transformer ce tuple en (73, 'Albert', 'COULE', 'Mauvaise') est moins évident : la modification qui fait disparaître un tuple de la vue des BONS_NAGEURS est-elle autorisée? La question de savoir si ce droit doit être donnée ou non est intéressant. Un utilisateur a-t-il le droit de "faire disparaître" des tuples de sa vue ?

Exemple 4 :

Peut-on maintenant insérer par la vue MAUVAIS_NAGEURS le tuple ('Albert', 'COULE') ? Comme dans l'exemple numéro 2, la valeur de l'attribut QUALITE de la relation NAGEURS. Mais, dans le cas présent, la valeur de la clé, l'attribut NN, doit également être complétée tout d'abord, Albert COULE peut déjà être recensé dans une autre catégorie de la base. L'insertion peut n'être en fait qu'une modification. Par exemple, Albert COULE peut être recensé en tant que bon nageur. Il s'agit alors d'une modification avec les risques que cela comporte. Nous les avons évoqués dans l'exemple précédent. Albert COULE peut être une personne différente de toutes celles qui sont déjà recensées dans la base. Dans ce cas le traitement doit être l'ajout d'un tuple dans la relation NAGEURS sous un numéro différent.

Exemple 5 :

Enfin, pour insérer un tuple du type (73, 'Albert', 'COULE', NULL) depuis la vue BONS_NAGEURS on ne sait pas si le niveau d'Albert COULE est bon ou excellent. Doit-on accepter une telle insertion ? En effet, pour l'utilisateur de la relation NAGEURS, rien n'indique que Albert COULE est bon ou excellent nageur et non pas de niveau faible ou médiocre.

L'étude de ces exemples nous montre que le mécanisme des vues externes, apporte des avantages en confidentialité et en confort de travail considérables, mais pose des problèmes non résolus dans le domaine des mises à jour. L'utilisateur qui dispose des droits nécessaires, n'est pas certain de pouvoir modifier certaines données auxquelles il a pourtant accès.

Les produits actuels limitent les mises à jour à travers les vues de façon drastique : seules les vues mono-relation sans fonction dont la définition conserve la clé sont concernées. Toutes les autres ne peuvent être manipulées qu'en interrogation.

VI.3. CONFIDENTIALITE

La confidentialité assure la protection des données de la base contre les accès non autorisés.

Elle est généralement obtenue en définissant des autorisations diverses sur les objets susceptibles d'être manipulés par les usagers. Le choix de l'objet détermine la méthode de vérification des droits d'accès. Ces droits sont généralement de plusieurs sortes (lecture,

modification et droit d'administration) car ils doivent répondre aux différentes exigences des utilisateurs.

Quelque soit la nature du SGBD, cette protection doit porter sur la plus petite unité d'information accessible par un utilisateur. Le modèle relationnel définie n'importe quel objet (relation, vue, ensemble de tuples, l'attribut d'une relation, d'une vue ou d'un tuple) par un simple critère permettant sa sélection. La définition des droits sur tout objet, aussi petit soit il est théoriquement possible, mais l'encombrement occasionné par de telles informations nécessite une gestion trop gourmande en performances. Ainsi, la gestion de droits dans le système relationnels d'aujourd'hui est elle réduite aux seules relations et vues.

La plupart des Systèmes de Gestion de Bases de Données offrent, outre les droits habituels de lecture et de modification, des possibilités d'administration comparables aux services offerts par les systèmes d'exploitation. Un utilisateur peut ainsi transmettre ses propres droits, droits d'administration inclus, à un autre utilisateur. Les SGBD compensent ainsi partiellement les inconvénients de la centralisation qu'ils imposent.

VI.3.1. Autorisations sur une relation ou sur une vue

Les droits accordés - ou autorisations - sur les relations ou vues sont de trois types :

- droit de consultation ;
- droit de modification (mises à jour, suppression, insertion) ;
- droit d'administration (définir les contraintes d'intégrité, attribuer les autorisations, détruire ou modifier le schéma de la relation).

Insistons plus particulièrement sur le droit d'administration qui est la seule contre-partie à la centralisation à outrance des données. Chaque utilisateur retrouve un certain pouvoir de décision : il a le droit de déléguer tout ou partie de ses droits sur les relations créées.

Forcer tout utilisateur à passer par une vue, conduit à considérer que l'utilisateur a des droits uniquement sur ce qu'il voit. Le mécanisme est très souple et permet de définir un droit en fonction d'un critère de sélection. Ainsi, le directeur d'un service ne peut, dans le cas d'une relation EMPLOYE lire que les salaires des membres de son propre service. De nombreuses limitations, dues à l'impossibilité des mises à jours générales à travers les vues, limitent ce mécanisme à des utilisations particulières.

VI.3.2. La cession de droits

Dans le standard SQL, le mécanisme d'attribution et de révocation des droits reste classique : les droits portent les relations de base. La requête permettant de céder ou de retirer tout ou partie de ses droits à un autre usager est la suivante :

```
GRANT | REVOKE <liste de droits> ON <nom de relation>  
TO | FROM <listes d'usagers>  
[WITH GRANT OPTION]
```

Les droits possibles sont les suivants :

```
SELECT  
INSERT  
UPDATE (nom d'attribut)  
DELETE
```

Le créateur d'une relation dispose de tous les droits sur la relation créée. Il a le droit d'administration, c'est à dire qu'il peut détruire ou modifier le schéma de sa relation. Il peut donner des droits aux autres usagers par la clause `GRANT` ; si la clause `WITH GRANT OPTION` est spécifiée, les droits sont transmissibles en aval. Notons que l'instruction `REVOKE` retire également les droits transmis en aval sur la relation, c'est à dire qu'un utilisateur peut se voir retirer ses droits, non seulement par celui qui les lui a concédés, mais aussi par un plus ancien donateur dont il ignorait l'existence... jusqu'au créateur lui-même.

VI.4. INTEGRITE SEMANTIQUE

VI.4.1. Les différents types de contraintes

On dresse ici une typologie possible des contraintes d'intégrité :

Domaine de variation

C'est la définition du domaine auquel appartient la valeur d'un attribut (entier, réel, texte, date...). Chaque donnée de la base doit vérifier cette contrainte qui peut correspondre à une simple concordance de type (c'est le cas de l'entier) ou à une condition plus complexe (c'est le cas du type date qui suppose des connaissances le nombre de jour dans un mois, les années bissextiles, etc.).

Plages de valeurs

C'est la restriction du domaine à un sous-ensemble d'un domaine. Par exemple, on peut imposer qu'un âge soit compris entre 0 et 150 ans, qu'aucun bain ne soit jamais pris entre novembre et mars...

Non nullité

La "nullité" ne correspond pas à un entier de valeur 0 ou à une chaîne de caractères vide, mais, pour tout type, à une valeur indéfinie ou non renseignée lors d'une mise à jour (NULL). Il peut être indispensable d'interdire la nullité à certains champs pour la permettre à d'autres. Par exemple, une clé de relation doit avoir une valeur non nulle, que ce soit un numéro ou une chaîne de caractère, puisqu'elle est le seul moyen de distinguer deux tuples ayant les mêmes valeurs d'attributs.

Unicité

L'unicité est également une notion importante, en particulier pour la définition des clés qui doivent être uniques.

Dépendances fonctionnelles

Les dépendances fonctionnelles sont des contraintes d'intégrité. Par exemple, un numéro de sécurité sociale détermine un tuple définissant un individu.

Dépendances référencielles

Elles indiquent que les valeurs d'un attribut (ou d'un groupe d'attributs) d'une relation R1 doivent déjà être définies dans un autre attribut (ou un autre groupe d'attributs). Par exemple, prendre un bain implique que NN et NP soient respectivement un individu et une plage et qu'ils figurent dans les tables appropriées. Un nageur peut exister sans nécessairement avoir pris de bain, mais un bain ne peut être pris que si le nageur existe.

Contraintes temporelles

Ces contraintes vérifient l'évolution des données par rapport au temps. Par exemple, le salaire affecté à un poste ne peut pas diminuer.

Contraintes avec agrégats

Ces contraintes ne portent pas sur la valeur d'un atome, mais sur une fonction concernant une collection de valeurs d'atomes. Par exemple, la durée moyenne des bains doit être supérieure à deux minutes.

Contraintes générales

Il existe également des contraintes qui n'appartiennent à aucune des catégories définies ci-dessus et sont plus générales. Par exemple : le nombre de bains pris par une personne ne peut pas décroître. Cette contrainte fait intervenir à la fois la notion de contrainte temporelle et de contrainte avec agrégats.

VI.4.2. Comment définir une contrainte d'intégrité ?

Comme nous pouvons le constater, les contraintes qui viennent d'être décrites sont mises en œuvre de différentes façons. Par exemple, les contraintes exprimant les dépendances fonctionnelles servent à la conception du schéma de la base : c'est le processus de conception qui assure automatiquement la vérification des contraintes de dépendance fonctionnelle.

Toutefois, toutes les contraintes ne sont pas liées à l'étape de conception de la base. Le langage de manipulation de données doit donc permettre la définition des contraintes d'intégrités. Ces contraintes peuvent alors être considérées comme des propriétés (assertions) indépendantes les unes des autres. Toute modification de la base de données doit les respecter. Ce formalisme est d'ailleurs le mieux adapté au modèle relationnel.

VI.4.3. Comment regrouper (classer) les contraintes ?

On peut regrouper les contraintes d'intégrité suivant différents critères : leur coût de validation, leurs caractéristiques individuelles (la contrainte s'applique tuple à tuple) ou ensembliste (la contrainte s'applique sur un ensemble de tuples), le fait qu'elles soient spécifiques d'une relation qu'elles en mettent plusieurs en jeu, etc.

VI.4.4. Contraintes d'intégrités et transactions bases de données

La notion de transaction sur une base de données est définie comme un ensemble d'actions qui font passer la base d'un état cohérent à un autre état cohérent (une BD cohérente est une BD où toutes les contraintes d'intégrité sont satisfaites). Trois fonctions d'un SGBD sont concernées : le contrôle de concurrence, la fiabilité et l'intégrité sémantique. Nous ne traitons dans ce paragraphe que le troisième aspect : le respect de règles sémantiques.

Le problème concerne l'instant de vérification des différentes contraintes : à quel moment doit-on vérifier les contraintes d'intégrité ? Quelles sont les contraintes vérifiables au gré des mises à jour de la base. Quelles sont celles qui sont vérifiées à la fin de la transaction ? Voyons quelques exemples.

Exemple 1 :

Considérons la relation NAGEURS (NN, NOM, PRENOM, QUALITE) et imposons la contrainte :

"Tous les nageurs dont le numéro est supérieur à 1000 sont d'excellent niveau."

Cette contrainte est vérifiable à chaque insertion de nouveau tuple ou à chaque modification de la relation NAGEURS.

Exemple 2 :

Considérons maintenant la même relation NAGEURS avec la contrainte d'intégrité suivante (contrainte avec agrégats) :

"Il doit y avoir autant de nageurs d'excellent niveau que de mauvaise qualité."

Supposons que l'on désire insérer dans la base un ensemble de nouveaux nageurs. A chaque insertion de nageur d'excellente ou de mauvaise qualité, la contrainte n'est pas vérifiée. Pourtant, cette même contrainte peut très bien être vérifiée lorsque tous les nouveaux nageurs ont été ajoutés.

Notifier au SGBD que l'ensemble des modifications effectuées précédemment doit être validé et que les contraintes d'intégrité peuvent désormais être vérifiées s'avère donc indispensable . L'ensemble des modifications validées forme ce qui est classiquement appelé une "transaction". Mais ceci est une autre histoire...

VI.5. CONCLUSIONS

Garantir la cohérence de l'information est un apport très important des Systèmes de Gestion de Bases de Données. L'utilisateur est libéré d'une tâche parfois difficile. Le système vérifie lui même des contraintes et des droits associés aux données. Vérifier les contraintes et les droits d'accès dans les programmes applicatifs des anciens systèmes était peu cohérent.

VI.6. RÉFÉRENCES

- | | |
|----------------|--|
| [Ullman 80] | J. ULLMAN : " <i>Principles of Databases Systems</i> " Computer Sciences Press, 1980. |
| [Bancilhon 79] | F. BANCILHON : " <i>Supporting View Updates in Relational Databases</i> " in Data Base Architectures, Bracchi Nijssen Editors, North Holland Publishing Company, 1979. |

