

Editions systèmes et information

CONCEPTION & REALISATION

DES BASES DE DONNEES :

De UML à SQL

Jacques Guyot

Copyright © 2002 Editions systèmes et information

Tous droits de traduction, d'adaptation et de reproduction par procédé réservés pour tous les pays

ISB : 2 -940105-12-x

*Editions systèmes et information
P/a Gilles Falquet
25, chemin de la californie
Cb-1222 Vésenaz*

Copyright © 2008 Editions systèmes et information :

Edition Internet

A Amélie, Maël, Noé, Alina et Thibault

Table des matières

NOTATIONS	15
1. INTRODUCTION A LA MODELISATION	17
L'INFORMATION AU CŒUR L'ENTREPRISE	17
DU STRUCTURE AU NON STRUCTURE	18
UNE VISION DE L'EVOLUTION DE L'INFORMATIQUE.....	20
QUOI, COMMENT, QUI ?.....	21
MODELE	23
POURQUOI MODELISER ?	24
UML - UN PEU D'HISTOIRE	25
DEJA DES EXERCICES	26
2. OBJETS ET CLASSES	29
C'EST QUOI UN OBJET	29
CLASSE	30
POURQUOI LES OBJETS ?.....	30
DIAGRAMME DE CLASSES EN UML	31
DIAGRAMME D'OBJETS.....	33
LIENS ENTRE LES OBJETS	33
ASSOCIATIONS	34
CARDINALITE DES ASSOCIATIONS	35
<i>Cas 1..1, 1</i>	36
<i>Cas 0..1</i>	37
<i>Cas *, 0..*</i>	38
<i>Cas 1..*</i>	38
ASSOCIATION BINAIRE, TERNAIRE,	39
CLASSE D'ASSOCIATION	41
CONTRAINTES.....	43
AGREGATION ET COMPOSITION	46
GENERALISATION	50
<i>Différence entre généralisation et composition ?</i>	50
EXEMPLES DE GENERALISATION	51
LA GENERALISATION EST CLASSIFIANTE.....	52
PRINCIPE DE SUBSTITUTION DE LISKOV	53
FACTORISER LE CODE	54
HERITAGE MULTIPLE	55
COVARIANCE	56
3. LES CAS D'UTILISATION.....	59
OBJECTIFS DU MODELE	59
PROCESSUS DE SPECIFICATION DES BESOINS	60
PROCESSUS DE DEVELOPPEMENT	61
STEREOTYPES DES CAS D'UTILISATION	62
CAS COMPARES AUX CLASSES	64
RECOMMANDATIONS	66
EXEMPLE : LE RESTAURANT	70

EXTENSION POUR LES PROCESSUS	71
EN SAVOIR PLUS SUR UML	72
LES ELEMENTS D'UML	73
UML : UN SYSTEME OUVERT	75
EXERCICE	75
4. SORTIR DE LA CONFUSION.....	77
EXERCICES.....	77
<i>Exercice sur RECETTE</i>	77
<i>Exercice sur TT³</i>	77
<i>Enoncé de TT³</i>	77
<i>Correction RECETTE</i>	79
<i>Correction TT³</i>	80
5. ENCORE UNE INTRODUCTION.....	81
DES FICHIERS AUX RELATIONS	82
PROBLEMATIQUE DES BASES DE DONNEES	85
SQL - LANGAGE DES BASES DE DONNEES RELATIONNELLES	89
TROIS DISCOURS	90
6. BASE DU MODELE RELATIONNEL	93
DOMAINES	93
CONSTITUANTS	95
RAPPELS SUR LA LOGIQUE DES PROPOSITIONS	96
RAPPELS SUR LA LOGIQUE DES PREDICATS	97
ENSEMBLES, PROPOSITIONS ET PREDICATS	99
RETOUR A LA DEFINITION DE RELATION	100
EXERCICES.....	104
<i>Questions sur TT³</i>	104
<i>Réponses sur TT³</i>	104
7. MODELISATION ET DEFINITION DES DONNEES	107
DIAGRAMMES SYNTAXIQUES	107
LANGAGE DE DESCRIPTION DE MODELISATION	107
LANGAGE DE DESCRIPTION DES DONNEES (SQL).....	111
EXERCICES.....	114
8. DES CLASSES AUX RELATIONS.....	115
CHAQUE CLASSE DEVIENT UNE RELATION	115
QUE FAIRE AVEC LES MÉTHODES ?	116
<i>Mémoriser les attributs calculés</i>	116
<i>Utiliser des vues</i>	117
<i>Utiliser les méthodes de mise à jour</i>	117
TRADUIRE LES ASSOCIATIONS.....	118
<i>Un des maximum est égal à un</i>	118
<i>Les deux maximum sont plus grand que un</i>	120
ASSOCIATION ATTRIBUEE	122
AGRÉGATION ET COMPOSITION	123
GÉNÉRALISATION	124

<i>Tout dans un</i>	124
<i>Chacun à sa place</i>	125
EXERCICES.....	128
<i>Questions sur TT³</i>	128
<i>Réponses sur TT³</i>	129
9. ALGÈBRE RELATIONNELLE.....	134
OPERATIONS ALGÈBRIQUES	134
<i>L'union</i>	135
<i>La différence</i>	135
<i>La projection</i>	136
<i>Le produit</i>	137
<i>La sélection</i>	138
JOINTURES	138
<i>La θ jointure (la théta-jointure)</i>	139
<i>Variations sur la jointure</i>	139
<i>Equi-jointure</i>	140
<i>Jointure naturelle</i>	140
<i>Semi-jointure</i>	140
<i>Jointure externe</i>	141
EXPRESSION ALGÈBRIQUE.....	141
LANGAGE ALGÈBRIQUE, UN LANGAGE D'INTERROGATION.....	142
EXERCICES.....	146
<i>Questions sur TT³</i>	146
<i>Réponses sur TT³</i>	146
10. INTERROGATION EN SQL.....	148
DU LANGAGE ALGÈBRIQUE A SQL.....	148
SYNTAXE DES REQUÊTES SQL.....	152
<i>La projection</i>	154
<i>Expression</i>	155
<i>Les relations du produit cartésien</i>	156
<i>Condition</i>	157
<i>Sous-requêtes</i>	163
<i>Regroupements</i>	165
<i>Ensembles</i>	167
<i>Arborescences</i>	167
<i>Trier</i>	169
MODELE DE LA MACHINE SQL.....	170
EXPRESSION GRAPHIQUE D'UNE REQUÊTE.....	171
EXERCICES.....	172
<i>Questions sur TT³</i>	172
<i>Réponses sur TT³</i>	175
11. MODIFICATION DES DONNÉES.....	181
PRIMITIVES DE MODIFICATION	185
<i>Création</i>	185
<i>Suppression</i>	186

<i>Mise à jour</i>	187
TRANSACTION.....	187
LANGAGE DE MANIPULATION DE DONNEES EN SQL.....	189
INTERFACES UTILISATEURS.....	193
EXERCICES.....	194
<i>Questions sur TT³</i>	194
<i>Réponses sur TT³</i>	194
12. LES REGLES D'INTEGRITE	195
DEFINITIONS ET PROPRIETES	196
LA CONSISTANCE D'UNE TRANSACTION.....	197
PORTEE DE LA REGLE D'INTEGRITE	198
7.4. LES REGLES D'INTEGRITE EN SQL.....	203
EXERCICE.....	ERREUR ! SIGNET NON DEFINI.
<i>Questions sur TT³</i>	<i>Erreur ! Signet non défini.</i>
<i>Réponses sur TT³</i>	<i>Erreur ! Signet non défini.</i>
13. DEPENDANCES FONCTIONNELLES	211
FERMETURE ET CONSEQUENCE LOGIQUE DE F	213
UN SYSTEME DE DEDUCTION	214
<i>Validité du système de déduction</i>	215
<i>Extension du système de déduction</i>	216
<i>Forme canonique de F et saturation de X</i>	217
<i>Complétude du système de déduction</i>	218
<i>Equivalence, couverture et irredondance de F</i>	219
ALGORITHME LIE AU DF	221
<i>Tester l'équivalence de F et G sans calculer les fermetures</i>	221
<i>Calculer la base complète de F</i>	222
<i>Calculer une base irredondante de F</i>	223
EXERCICES.....	223
<i>Questions sur TT³</i>	223
<i>Réponses sur TT³</i>	224
14. CLES.....	227
ALGORITHME DE RECHERCHE DE CLES.....	228
ATTRIBUTS PUIES ET SOURCE	229
<i>Proposition: sur les clés, puits et sources</i>	229
<i>Unicité de la clé dans les graphes de df acycliques</i>	230
<i>Heuristiques pour la recherche des clés</i>	231
DECOMPOSITION D'UNE RELATION	233
EXERCICES.....	233
<i>Questions sur les clés</i>	233
<i>Questions sur TT³</i>	234
<i>Réponses sur les clés</i>	235
<i>Réponses sur TT³</i>	235
15. DECOMPOSITION D'UNE RELATION.....	237
CONSERVATION DE L'INFORMATION	238

<i>Propriété de la décomposition</i>	239
DECOMPOSITION TOTALE	240
<i>Tester si une décomposition est totale</i>	240
PRESERVATION DES DF	244
<i>Tester la préservation des df</i>	245
<i>Inclusion des df dans les Ri</i>	247
ELIMINATION DES ANOMALIES DE MISE A JOUR	247
1er forme normale 1NF	248
2eme forme normale 2NF	250
3eme forme normale 3NF	251
<i>Forme normale Boyce-Codd (BCFN)</i>	252
<i>La 3FN une relaxation de la BCFN</i>	253
16. VUES	255
VUES EN SQL	256
MULTIPLIER LES REPRESENTATIONS LOGIQUES	258
<i>Créer des vues contextuelles</i>	258
<i>Interfacer un schéma</i>	259
<i>Créer des dépendances calculées</i>	259
<i>Déduire de l'information</i>	261
EXERCICES	263
<i>Le jeu des trois dés</i>	263
<i>Produit scalaire et le produit matriciel</i>	264
<i>Sur TT3</i>	265
<i>Réponses</i>	265
17. OPTIMISATION ET PERFORMANCE	269
OPTIMISATION DES REQUETES SQL	270
<i>Réécriture de la requête SQL</i>	272
<i>Chemins d'accès</i>	274
<i>Choix de la méthode de jointure</i>	277
UNE PERCEPTION HOLISTIQUE DE LA CONCEPTION	279
SPECIALISER LES VUES POUR UN ACCES PRINCIPAL	283
DENORMALISATION	285
18. INVEST	289
ENONCE	289
DOMAINE DES CONSTITUANTS	290
SEMANTIQUE	291
DIAGRAMME DES CLASSES	292
RELATIONS	292
DEPENDANCES FONCTIONNELLES	293
TABLES TYPE	293
SUR LA MODELISATION	294
REQUETES	294
REGLES D'INTEGRITE	295
19. SECUR	297
ENONCE	297
DOMAINE DES CONSTITUANTS	298

SCHEMA DES RELATIONS.....	299
DEPENDANCES FONCTIONNELLES.....	299
DIAGRAMME DES CAS D'UTILISATION.....	300
DIAGRAMME DES CLASSES.....	301
SEMANTIQUE.....	301
TABLE TYPE.....	302
SUR LA MODELISATION.....	303
REQUETES.....	303
REGLES D'INTEGRITE.....	304
20. ULTIMA.....	305
ENONCE.....	305
DOMAINE DES CONSTITUANTS.....	306
RELATIONS.....	307
DEPENDANCES FONCTIONNELLES.....	307
DIAGRAMME DES CLASSES.....	308
SEMANTIQUE.....	308
TABLES TYPE.....	309
SUR LA MODELISATION.....	309
REQUETES.....	310
REGLES D'INTEGRITE.....	310
21. BIGSMAC.....	313
DOMAINE DES CONSTITUANTS.....	315
DIAGRAMME DES CLASSES.....	316
RELATIONS.....	318
DEPENDANCES FONCTIONNELLES.....	319
SEMANTIQUE.....	319
TABLES TYPE.....	319
SUR LA MODELISATION.....	320
REQUETES.....	321
REGLES D'INTEGRITE.....	321
22. DUF.....	323
DOMAINE DES CONSTITUANTS.....	324
RELATIONS.....	325
DEPENDANCES FONCTIONNELLES.....	325
DIAGRAMME DES CLASSES.....	326
SEMANTIQUE.....	327
TABLES TYPE.....	328
SUR LA MODELISATION.....	329
REQUETES.....	329
REGLES D'INTEGRITE.....	330
23. FAME.....	331
ENONCE.....	331
DOMAINE DES CONSTITUANTS.....	332
DIAGRAMME DE CLASSES.....	333
RELATIONS.....	333
DEPENDANCES FONCTIONNELLES.....	334

SEMANTIQUE.....	334
SUR LA MODELISATION	335
REPENDRE EN SQL AUX QUESTIONS SUIVANTES:	336
VUES	336
REGLES D'INTEGRITE.....	336
24. ELECTA.....	337
DOMAINE DES CONSTITUANTS	337
DIAGRAMME DE CLASSES	338
RELATIONS	338
DEPENDANCES FONCTIONNELLES.....	339
SUR LA MODELISATION	339
25. CONFER.....	341
ENONCE	341
QUESTIONS	342
26. 39,2 LE MATIN.....	343
ENONCE	343
DIAGRAMME DES CAS D'UTILISATION	346
DIAGRAMME DES CLASSES	348
27. QCM LIGTH	349
DOMAINE DES CONSTITUANTS	350
RELATIONS	350
DEPENDANCES FONCTIONNELLES.....	351
SEMANTIQUE.....	351
QUESTIONS DE MODELISATION	352
REPENDRE EN SQL	353
28. METGE	355
DIAGRAMME DES CLASSES	360
DOMAINE DES CONSTITUANTS	360
RELATIONS	361
QUESTIONS	361
29. ETUDE DE CAS : A B C D	365
DIAGRAMME DES CAS D'UTILISATION	368
DIAGRAMME DES CLASSES	369
30. PLANS, PLANS, RATAPLAN	371
31. BIBLIOGRAPHIE.....	375
32. INDEX.....	379

Notations

$ R $	prédicat d'une relation R
R^+	ensemble de constituants de R
\in	appartient à
\subseteq	est inclus dans
\cup	union
\cap	intersection
$-$	différence
\emptyset	ensemble vide
\wedge	et logique
\vee	ou logique inclusif
\neg	négation
\forall	pour tout
\exists	il existe
\equiv	équivalence
\times	produit cartésien
$ $	tel que (dans un ensemble)
$\{$	un ensemble
\prod	séquence de produit
R,S,T, \dots	des relations
r,s,t	des n-uplets
A,B,C, \dots	des constituants
X,Y,Z, \dots	des ensembles de constituant
$R(\underline{A},B,C,D)$	Une clé de la relation est indiquée par un soulignement

1.Introduction à la modélisation

"Les images ne sont pas faites pour la lumière. Tout rêve le sait et chaque nuit le prouve" Vie secrète - Pascal Quignard

L'information au cœur l'entreprise

L'activité des entreprises est actuellement entièrement centrée sur l'information. Les systèmes les plus connus et les plus visibles sont les systèmes de base des entreprises qui sous-tendent l'ensemble des activités opérationnelles de l'entreprise (Legacy system en anglais). Parmi ces applications, nous trouvons traditionnellement la comptabilité générale, la gestion des fournisseurs, la gestion des stocks, la gestion des ventes et des clients et souvent une application centrée sur le métier de l'entreprise. Pour une régie, il s'agira d'une gestion d'immeuble, pour un cabinet de gestion de fortune, on trouvera une gestion de portefeuille. Ces systèmes opérationnels sont caractérisés par l'aspect «protocole de saisie» des informations. Par exemple, il existe généralement une seule manière de saisir une écriture comptable et l'ensemble des rapports associés à la comptabilité sont généralement prédéterminés.

Les besoins en information ont rapidement dépassé le cadre opérationnel pour atteindre celui du décisionnel. Dans ce cadre, le gestionnaire doit prendre des décisions sur la base des données opérationnelles. Ses outils sont ceux de l'analyse de données, de l'aide à la décision, les données sont agrégées par périodes, par domaines. Ces besoins sont actuellement regroupés sous le terme d'entrepôts de données (Datawarehouse en anglais), d'analyse d'espace à n-dimensions ou de cubes de données.

La dernière exploitation possible des données est celle effectuée par les outils de "DataMining" qui à partir des données opérationnelles tentent de découvrir des règles ou des lois cachées dans l'information. Ces outils sont surtout utilisés dans les études de marché pour découvrir le comportement du consommateur.

L'information d'une entreprise n'est pas uniquement contenue dans des données structurées, elle est aussi présente dans les documents manipulés par l'entreprise. Pour celles qui tendent vers le "zéro papier", elles utilisent des systèmes de gestion électronique des documents (GED) qui sont des bases de données de documents. Les documents sont généralement scannés à l'entrée du système d'information. Les documents sont ensuite analysés par un outil de reconnaissance de caractères, ce qui permet ultérieurement

d'accéder au contenu de ces documents, sinon on en a uniquement une image. Ces GED possèdent généralement de système d'indexation du contenu, des indexations par mots clés, par auteurs, par date qui permettent de retrouver le document.

La gestion de la circulation des documents et la synchronisation des activités sont gérées avec des techniques de gestion de flux (Workflow en anglais). Ceci permet d'automatiser l'administration du suivi des documents. Les documents sont associés à des états et des règles de gestion sont établies sur ces états pour autoriser les traitements appropriés.

Un pas supplémentaire est franchi avec les collecticiels. Dans le cas précédent, les documents circulaient d'un acteur à l'autre. Ici, il s'agit d'organiser la synchronisation des acteurs avec des outils de communication et des outils permettant d'élaborer à plusieurs, simultanément, des documents. A nouveau, des bases de données contenant les messages et les participations des acteurs sont mises en œuvre. Les mécanismes d'indexation sur les contenus, les auteurs, les destinataires, les versions, etc. facilitent les recherches.

Et finalement l'utilisation de serveurs WEB en Intranet ou Internet complète la mise en réseau de l'information. L'information de l'entreprise est vue comme un document hypertextuel. Les bases de données sont encore le moyen le plus fiable et le plus simple pour gérer les liens d'hypertexte entre les différents documents.

Ce bref survol des différentes techniques d'utilisation de l'information dans l'entreprise, nous montre que les bases de données sont au centre de la gestion des informations.

Du structuré au non structuré

Comme nous l'avons vu, l'information peut prendre plusieurs formes. Dans les bases de données, elle est fortement structurée. Par contre dans un serveur Web, elle est non-structurée.

structurée (BD)	non-structurée (WEB)
modélisable	indexable
domaine limité (nbr de catégories petit)	domaine illimité (nbr de catégories grand)
protocole connu (cycle de vie des objets)	pas de protocole explicite

Le structuré est généralement le résultat d'une modélisation d'un domaine limité. Le domaine étant limité, les catégories d'objets à l'intérieur de celui-ci sont peu nombreuses, il est donc possible de modéliser l'ensemble. De plus

le protocole du cycle de vie des objets est connu (c'est à dire quand les objets apparaissent, se modifient et disparaissent du champ d'application). Les bases de données contenant les applications de base de l'entreprise sont un cas typique d'information structurée.

Le non-structuré est caractérisé par un domaine aux frontières plus floues. Les catégories d'objets deviennent alors plus nombreuses et moins discernables par une forme régulière. Le domaine n'est plus modélisable, mais généralement indexable. Il n'y a pas de protocole explicite du cycle de vie des objets.

Structuré	Semi-structuré	non-structuré
modélisable	syntaxe	sémantique
auteur titre éditeur nbr pages	Chapitre partie paragraphe phrase	Sens Concept Idées Commentaires

Il existe une continuité d'états de l'information du structuré au non structuré. Un objet peut avoir plusieurs facettes informationnelles ayant différents niveaux de structuration. Par exemple un livre pourra être vu comme:

- Un objet structuré appartenant à la base de données d'un libraire ou la modélisation aura uniquement retenu son auteur, son titre, son éditeur, le nombre de pages, etc. ;
- Le même livre peut être décrit en XML avec une structure de chapitres, parties, paragraphes, phrases, etc. Ici on a affaire à un document dont la forme est structurée ;
- Enfin ce livre peut faire l'objet d'une critique sur internet, ses adorateurs et ses détracteurs peuvent créer des pages en HTML qu'ils lieront au reste du Web par des liens hypertextuels.

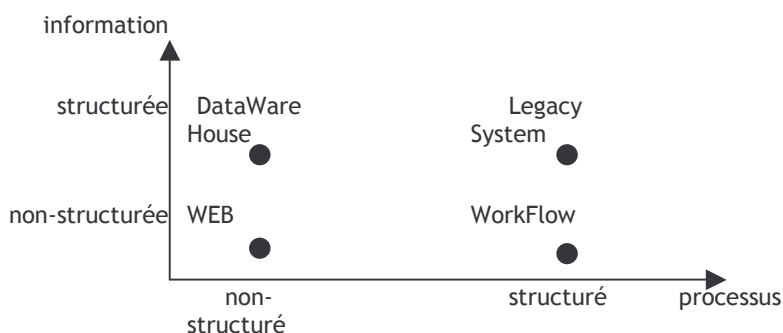


Figure 1-1 : Systèmes en fonction de leur structuration

Les processus aussi sont plus ou moins structurables. Dans la Figure 1-1, on constate que tous les cas de figure sont possibles. Les applications de base

de l'entreprise sont fortement structurées dans les données et dans les processus. Par contre, quand un gestionnaire explore, compare, analyse un entrepôt de données, son cheminement n'est pas connu, ni structuré.

A l'inverse, dans une gestion de documents, les données peuvent être faiblement structurées et les traitements complètement contraints par les règles de gestion.

La modélisation d'un système d'information explicite quand cela est possible: la structure des données, la structure des traitements et les règles de gestion. Ces trois axes de description sont la base de la modélisation.

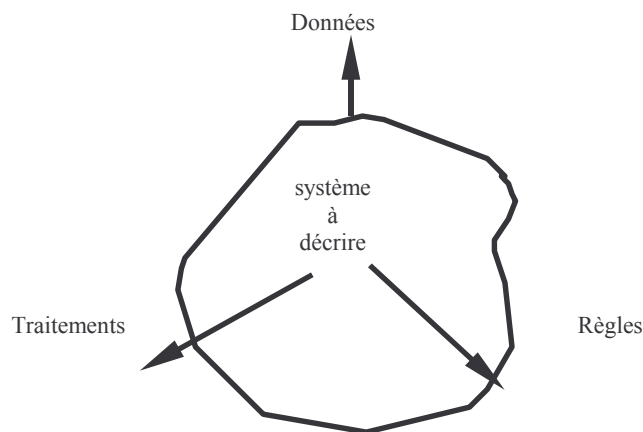


Figure 1-2 : Axes de description du système d'information

Une vision de l'évolution de l'informatique

Nous donnons dans le tableau qui suit quelques points de repère sur l'évolution de l'informatique. Au fond, seule la tendance nous intéresse. Les traitements sont de plus en plus distribués. Les informations manipulées sont de plus en plus standardisées et modélisées. L'architecture appelle à plus d'interconnexions, plus de mobilité dans les données et les applications échangées.

Les axes de l'architecture d'un système d'information sont donc en suivant ces tendances:

- Les données et les règles de gestion sont toutes gérées par une base de données ;
- Les traitements sont spécifiés dans un code supportant la mobilité, c'est à dire pouvant s'exécuter sur des plate-formes différentes et sous des systèmes d'exploitation différents ;
- Les interfaces doivent être de nature hypertextuelle.

Les applications Client/serveur, intranet/internet utilisant un mélange de Web, de code Java et de base de données possèdent tous les ingrédients décrits ci-dessus.

| CPU

| DONNEES

| ARCHI-

| TECHNO-

			TECTURE	LOGIE
60	ordinateur central	fichiers	centralisé	bande, carte perforée, centre de traitement
70	mini départemental	BD hiérarchiques réseaux	étoile	terminal, disque amovible
80	ordinateur personnel	BD relationnelles	Point réseau interne	disquette, câble coaxial
90	ordinateur réseaux	BD Objets/relation	réseau global	CD-rom, modem
00	Objets intelligent	Documents	Mobilité,	Ecran plat, téléphone portable

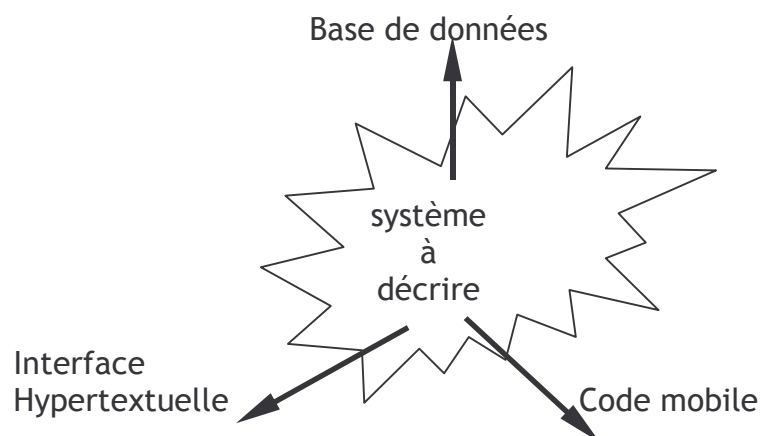


Figure 1-3 : Axes de l'architecture du système d'information

Quoi, Comment, Qui ?

Nous venons de voir que nous pouvons nous interroger sur comment décrire l'information, sur le choix des technologies à mettre en œuvre pour la gérer. Mais d'autres questions gravitent autour de l'information :

- Sur la confidentialité : qui peut accéder à l'information, quels sont les droits offerts à chacun, combien me coûte son dévoilement ?
- Sur la fiabilité : Pendant combien de temps puis-je vivre sans cette information, combien me coûte sa perte ?
- Sur son atteignabilité : comment la retrouver, quels sont les index qui la référence ?
- Sur son utilisation : qui l'utilise, dans quel processus est-elle utilisée ?
- Sur son cycle de vie, quand est-elle créée, supprimée, modifiée dans le champ d'application ?

- Sur ses associations : à quelles autres informations est-elle liée, de quelles informations dépend-elle ?
- Sur son importance pour l'entreprise ;
- Sur les services possibles : la valeur ajoutée à ces services grâce à cette information ;
- Sur les opérations possibles.

En examinant ces questions, on peut constater que plusieurs points de vue sont à prendre en compte. Cet aspect polymorphe de l'information s'étend aussi à sa représentation. Dans la premier cours après 30 minutes, je demande aux étudiants de modéliser le groupe d'étudiants qu'ils représentent. Et les résultats se répartissent généralement en plusieurs styles.

On trouve le style classification.

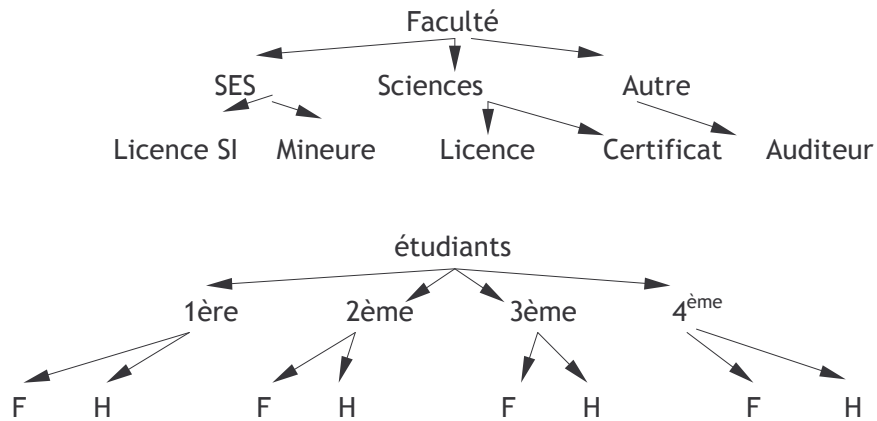


Figure 1-4 : l'arbre comme choix de classification

Le style boîtes et flèches:

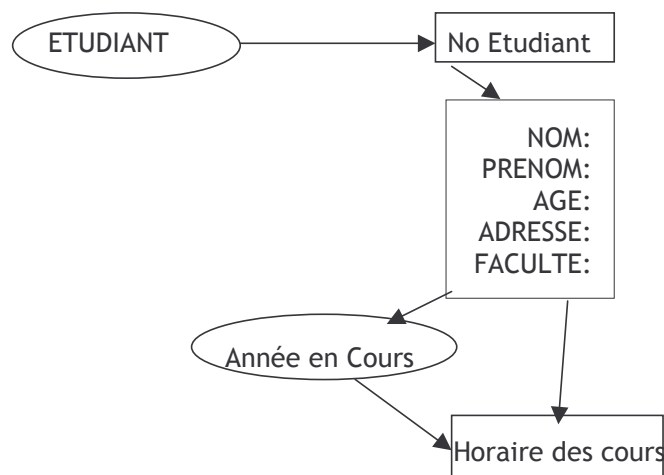


Figure 1-5 : la boîte comme choix de rangement

Le style texte et liste:

Par rapport au cours :	Par rapport à l'étudiant :
<ul style="list-style-type: none"> • horaire du cours • semestriel, annuel • Nbr d'heures de cours • quels genres d'exas (écrit, oral) • Nbr de participants • nom du prof • pré-requis 	<ul style="list-style-type: none"> • nom • prénom • âge • quelle licence suivie • quelle année d'étude

Sur la base des résultats de ce petit exercice, on peut constater que:

- **Sans objectif**, chacun a suivi des objectifs propres, chacun a suivi sa propre inspiration ;
- **Le domaine n'est pas clos par une frontière**: l'absence d'objectif ne délimite pas le SI à spécifier ;
- **La forme est molle**: chacun a représenté le SI selon son inspiration sans donner un sens précis aux symboles utilisés ou aux structures ;
- **Communicable**: toutes ces formes sont facilement communicables mais sans rigueur dans la forme, il est difficile d'être précis.

Modèle

Pour nous un modèle est une abstraction de la réalité sur laquelle on peut opérer. C'est une abstraction, car nous devons filtrer, sélectionner une partie de la réalité pour élaborer le modèle. Pour ce qui a été retenu, se pose la question de comment le représenter ? Ce choix de la représentation est directement lié à l'aspect opératoire de la modélisation. Comment manipuler le modèle ? Comment interroger le modèle ? Comment interpréter les réponses dans la réalité ? En effet, il va y avoir un retour du modèle dans la réalité. Nous faisons le détour par le modèle pour nous aider à prendre une décision dans la réalité. Tous les modèles ne sont pas adaptés à tout. Chacun est spécialisé dans une facette particulière de la représentation de la réalité. On doit donc vérifier les hypothèses d'utilisation du modèle et connaître les limites de validité du modèle.

L'utilisation d'un modèle correspond toujours à un déplacement du questionnement. Examinons l'exemple de la Figure 1-6 : dans le réel, il existe saturne. Dans la réalité, ce qui est perçu par nous et qui dépend de nos sens et de nos moyens d'investigation, nous percevons saturne comme une masse sphérique avec un anneau. Si notre questionnement est : quel est le volume de saturne ? Nous cherchons le modèle le plus adéquat, nous faisons des hypothèses simplificatrices. Par exemple, nous décidons de considérer saturne comme une sphère parfaite et de ne pas tenir compte de

son anneau. La question devient alors une question du domaine de la géométrie : quel est le volume de la sphère ? Nous utilisons la formule adéquate avec une estimation du rayon de saturne. En retour, nous obtenons un chiffre que nous interprétons dans la réalité.

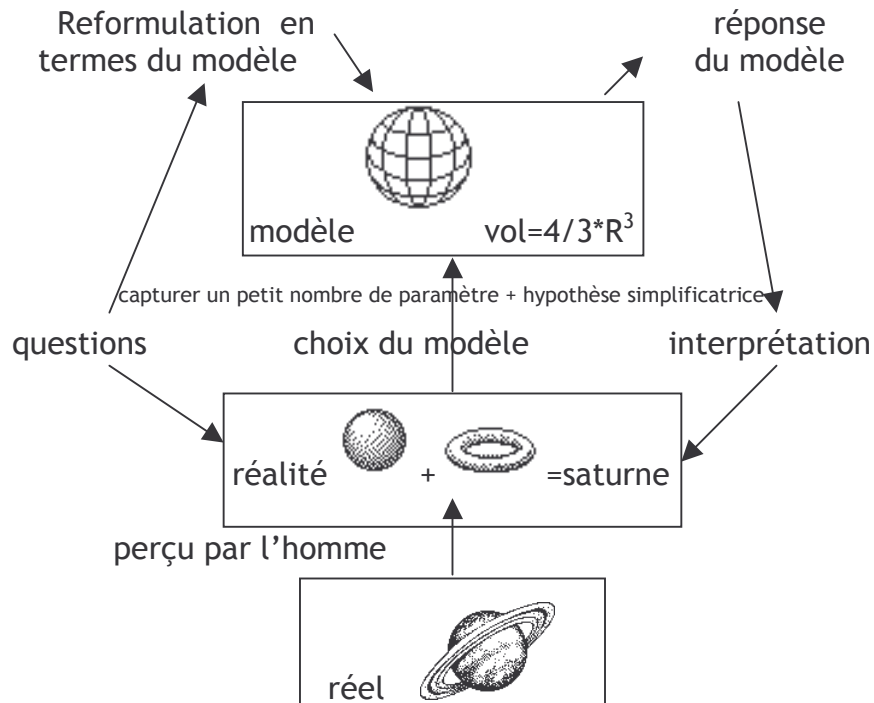


Figure 1-6 : de la réalité au modèle.

Le processus de modélisation peut être caricaturé par les étapes suivantes :

- On a dans un premier temps des questions sur la réalité, mais celle-ci ne se laisse pas interroger ;
- On modélise donc la partie qui doit être interrogée ;
- On initialise le modèle avec les valeurs provenant de la réalité ;
- On reformule la question initiale dans les termes du modèle ;
- On évalue la question dans le modèle ;
- On valide la réponse donnée par le modèle ;
- On interprète la réponse dans la réalité initiale.

Pourquoi modéliser ?

Avant de terminer cette introduction à la modélisation, on peut se demander pourquoi modéliser le système d'information. Pour nous, la modélisation du système d'information détermine un espace possible pour l'articulation de deux discours. L'un est le discours du gestionnaire en termes de responsabilité, de ressources humaines, de document, de hiérarchies d'unité d'organisation, de méthodes de management, de coûts, de valeur ajoutée, de service, etc.. L'autre discours est celui de l'informaticien en termes de technologies, de base de données, de système exploitation, de réseaux, de méga-hertz, giga-bytes, etc.. La modélisation permet de s'abstraire des deux types de discours et de se consacrer plus pragmatiquement sur les objectifs

et les besoins de l'entreprise et de ses acteurs. Le concepteur produira un système en poursuivant les objectifs suivants:

- Minimiser la complexité ;
- Maximiser l'évolutivité ;
- Autoriser l'implémentabilité ;
- Augmenter la Robustesse.

Ces objectifs seront dans le reste de l'ouvrage, une ligne de conduite lors des choix obligés de la conception.

Comme nous l'avons vu précédemment pour modéliser, il est important de fixer la notation dans laquelle nous allons exprimer. Nous avons fait le choix de travailler avec la notation UML (*Unified Modeling Language*). UML est destiné à la modélisation objet et couvre toutes les phases d'un projet. Pour nous il permettra de spécifier les objets principaux des systèmes d'information, à savoir les données, les traitements et les règles de gestion.

UML - un peu d'histoire

UML est un exemple de réussite par l'alliance. Au début des années 90, la recherche est orientée vers le monde des objets et les chercheurs en génie logiciel produisent des méthodes pour le paradigme objet. Deux acteurs G. Booch et J. Rumbaugh sont en concurrence avec leur modèle Booch91 et OMT1. La deuxième version de leur modèle complète les manques de la première version et ajoute des fonctionnalités. Mais les modèles Booch93 et OMT2 sont conceptuellement pratiquement identiques. G. Booch et J. Rumbaugh décident d'allier leur compétence plutôt que de se perdre dans une querelle stérile. En octobre 1994, J. Rumbaugh entre dans la société Rational (www.rational.com) de G. Booch. En octobre 1995, il publie une première version UM 0.8 (Unified Method) réunissant leurs travaux. En juillet 1996, sous l'influence de Ivar Jacobson et de sa méthodologie OOSE, la première version de UML 0.9 paraît. En janvier 1997, la version UML 1.0 est proposée à l'OMG (Object Modelling Group www.omg.org) afin de la standardiser. Cette standardisation est parrainée par Digital Equipment, Hewlett-Packard, Microsoft, Oracle, Texas Instruments Software, Unisys, etc. En novembre 1997, la version 1.1 est standardisée par l'OMG. Sur le site de l'OMG, on peut suivre les développements et les révisions du standard.

Les acteurs principaux de UML sont:

- Grady Booch : méthodologie, Booch91, Booch93 ;
- Jim Rumbaugh: méthodologie, OMT1, OMT2 ;
- Ivar Jacobson: méthodologie, OOSE .

Mais on y trouve aussi les travaux de:

- Harel: Statechart (automate formel) ;
- Meyer: pré et post-condition ;
- Shlaer et Mellor: cycle de vie d'objets .

En deux ans, la plupart des grandes sociétés de logiciel ont adopté UML dans leurs outils. Comme on le verra, UML n'impose une méthode mais une notation.

Les principaux objectifs d'UML sont:

- De modéliser des systèmes complets et complexes, UML ne se limite pas seulement au langage de programmation objet ;
- De coupler les concepts avec les artefacts exécutables. Il assiste donc le développement depuis la conception jusqu'au déploiement ;
- De gérer la complexité des systèmes. En utilisant, la possibilité de raffinement successif, il est possible d'utiliser UML aussi bien pour concevoir un distributeur de boisson ou qu'une navette spatiale ;
- D'être un modèle adapté aux humains et aux machines. Chaque concept possède toujours les deux représentations, l'une externe graphique et lisible par une personne, l'autre formelle et manipulable par un programme ou un outil ;
- D'être un système ouvert. UML n'est qu'une notation, il ne définit pas la méthode, ni les outils. Il reste extensible à d'autres concepts.

Avoir une unité de notation durant tout le cycle de vie du développement constitue un atout majeur en faveur d'UML. En effet, peu de méthodes couvrent complètement le cycle de vie du développement. Et la rupture ou le changement de modèle crée un risque de perte sémantique dans la continuité du développement.

UML c'est six modèles:

- Modèle des **classes** : exprimer la structure statique des objets ;
- Modèle des **cas d'utilisation** : exprimer les besoins des utilisateurs ;
- Modèle des **états** : exprimer la structure dynamique des objets ;
- Modèle des **interactions** : entre les acteurs et le système, entre les objets ;
- Modèle de **réalisation** : exprimer le regroupement et les unités logiques de réalisation ;
- Modèle de **déploiement** : exprimer la répartition physique des éléments du système.

Dans les deux chapitres suivants, nous allons examiner les deux premiers modèles.

Déjà des exercices

Chercher des modèles et vérifier qu'ils sont bien des abstractions munies d'opérations :

- en physique
- en économie
- en sociologie
- en psychologie

- etc.

Répondre aux points suivants:

- A quelles interrogations répondent-ils ?
- Quelles sont les hypothèses d'utilisation ?
- Comment reformuler sa question ?
- Comment paramétrer la modélisation ?
- Comment interpréter la réponse ?

2. Objets et classes

« Tous les systèmes y passaient, adoucis d'une certitude de triomphe facile, d'un baiser universel qui terminerait le malentendu des classes » *Germinal* – E. Zola

Dans ce chapitre, nous présentons les concepts d'objet et de classe ainsi que les notations UML qui leur sont associés.

C'est quoi un objet

C'est une entité contenant des:

Données : les données nous informent sur l'état de l'objet.

Procédures : les procédures nous informent sur le comportement de l'objet.

L'état de l'objet est généralement conservé dans des variables et le comportement de l'objet est spécifié dans des méthodes, des petits programmes associés à l'objet.

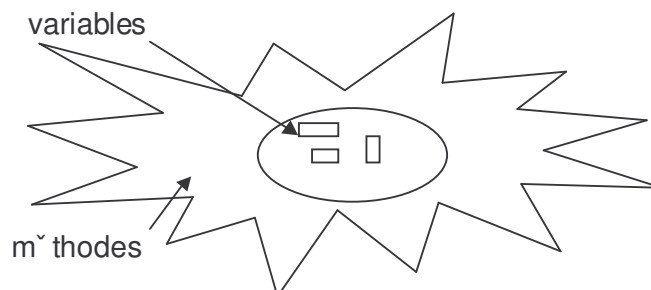


Figure 2-1 : Un objet.

Prenons par exemple l'objet *rectangle r1*, son état est défini par les deux variables largeur et hauteur valant respectivement 5 et 8. Il est possible de le manipuler pour lui demander sa surface, son périmètre ou bien de modifier son état en lui demandant de s'agrandir.

Chaque objet a une identité (Oid) qui permet de le repérer. Les objets communiquent entre eux en échangeant des messages. La réception d'un message par un objet déclenche l'exécution de la méthode invoquée avec les paramètres donnés.

Par exemple:

r1.surface() retournera la valeur 40

r1.agrandir(5) modifiera l'état de *r1* (largeur=25 et hauteur=40)

Les objets ne restent pas longtemps isolés : ils se rassemblent en collection, de ces collections se dégagent une structure commune, de l'artisanat, on passe au stade industriel.

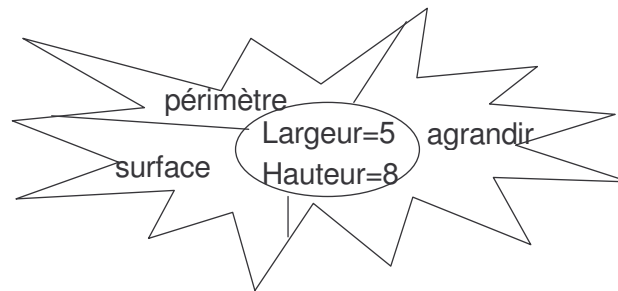


Figure 2-2 : r1 un rectangle, son état et son comportement.

Classe

Une classe est un moule pour fabriquer des objets

- De même structure ;
- De même comportement ;
- Régit par les mêmes règles.

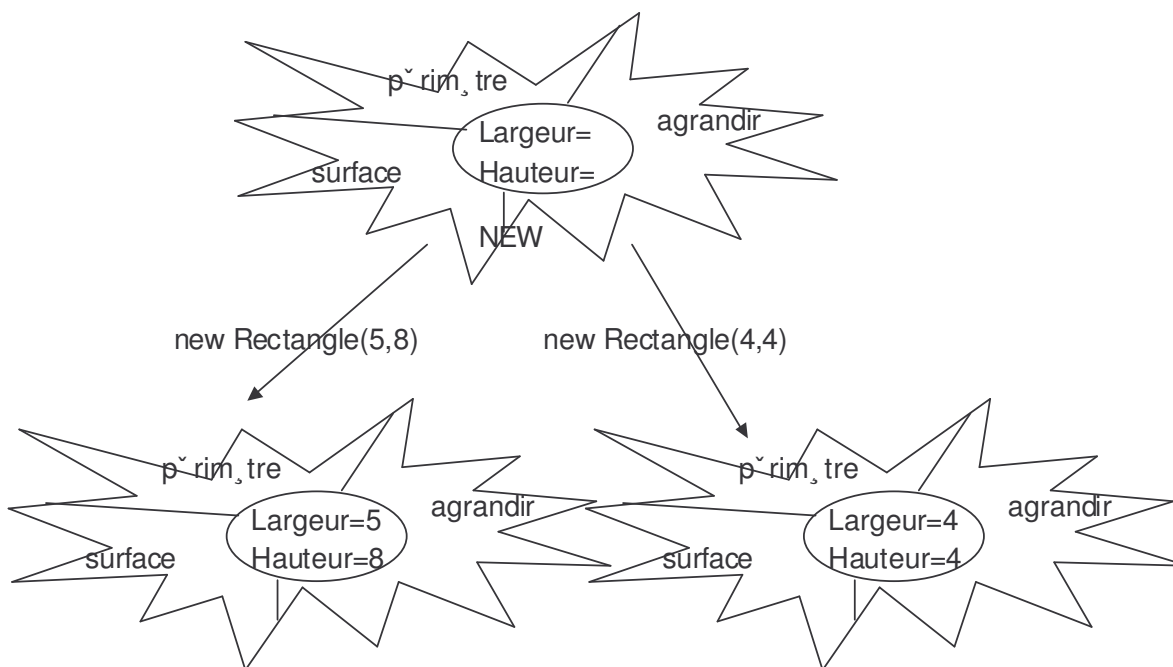


Figure 2-3 : La classe Rectangle et deux instances.

La classe Rectangle, est le moule permettant de fabriquer des objets rectangles ayant tous une largeur, une hauteur et réagissant de la même manière aux messages périmètre, surface et agrandir. La création d'un objet rectangle par la classe Rectangle s'obtient en invoquant la méthode new sur la classe elle-même. On parle alors d'instanciation d'objet à partir de la classe. Un objet est une instance d'une classe

Pourquoi les objets ?

La programmation objet a rencontré un large succès auprès des développeurs durant ces dernières années, d'abord avec Smalltalk, ensuite

avec C++ et dernièrement avec Java. La programmation objet possède des bonnes propriétés "génie logiciel".

La première est de réunir les données et traitements. Le comportement est défini dans la classe. Les changements d'états d'un objet devraient être uniquement le fait des méthodes de cet objet. La compréhension d'un objet peut donc s'effectuer dans le contexte de la classe. Dans la programmation classique, la séparation des données et des traitements ne permet pas d'avoir cette certitude sur la modification des variables.

Les objets échangent des messages entre eux et cela devrait être le seul mode de communication (il ne doit pas y avoir d'interaction directe sur une variable d'état d'un objet depuis une méthode extérieure à cet objet). Si tel est le cas, alors le comportement d'un système se décrit comme la collaboration d'un ensemble d'objets, orientée vers un objectif.

L'encapsulation est le mécanisme qui permet de cacher comment sont réalisées les méthodes et de rendre privé l'accès aux variables d'états de l'objet. Ce mécanisme permet de délimiter clairement la spécification d'un service et de son implémentation. Cela permet de rendre plus évolutif le système, la modification de l'implémentation d'une méthode n'ayant pas d'effet secondaire si la spécification est respectée.

L'héritage est l'autre mécanisme majeur de la programmation objet. Il permet d'affiner le comportement d'une classe, de factoriser le code et d'être un vecteur de réutilisation. Plus généralement, le polymorphisme permet à un objet de s'habiller avec plusieurs types de comportements différents.

L'ensemble de ces caractéristiques a assuré le succès du paradigme objet. Dans la suite de cet ouvrage, nous aborderont les propriétés conceptuelles structurantes suivantes:

- Les classes ;
- Les attributs ;
- Les méthodes ;
- Les associations ;
- Les relations d'héritage.

Nous ne commenterons pas les caractéristiques propres à la programmation objet comme la visibilité des attributs, des méthodes, les types de classe (interface, abstraite, ...).

Diagramme de classes en UML

La notation UML (Figure 2-4) dessine le concept de classe dans un rectangle, le nom de la classe apparaissent dans la partie supérieure. Le compartiment suivant contient la liste des attributs de la classe et le dernier compartiment la liste des méthodes.

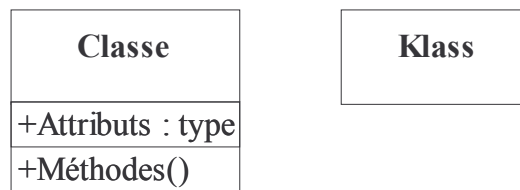


Figure 2-4 : désignation des classes en UML.

La Figure 2-5 montre les classes *Rectangles*, *Disques* et *Triangles*.

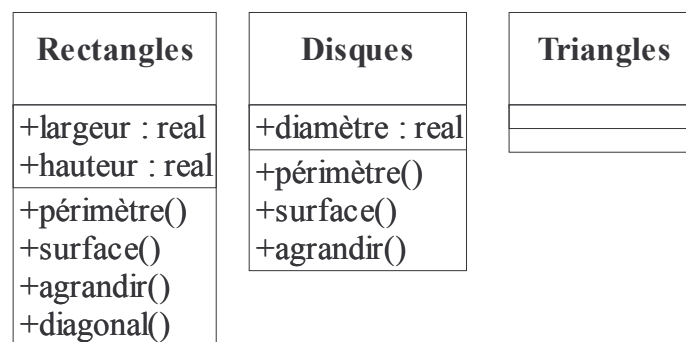


Figure 2-5 : exemples de classes en UML.

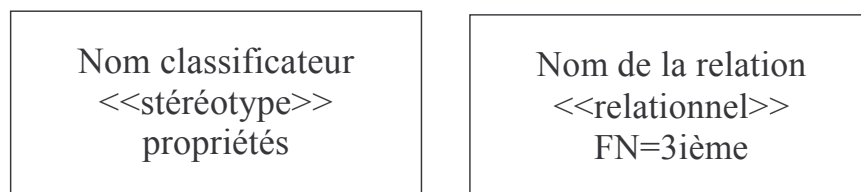


Figure 2-6 : les stéréotypes.

Les classes appartiennent à l'ensemble des classificateurs de la description statique. Il existe d'autres types de classificateurs, par exemple :

- <<*type de données*>> : un descripteur associé à un ensemble de valeurs, à un domaine ;
- <<*interface*>> : le nom d'un ensemble d'opérations qui définissent un comportement ;
- <<*sous-système*>> : un package qui est traité d'une manière unitaire avec une spécification et une implémentation ;
- <<*utilitaire*>> : une classe ne comportant que des opérations (pas d'instance).

Chaque classificateur définit un stéréotype et peut se représenter comme indiqué dans la Figure 2-6.

Il est aussi possible de créer des nouveaux stéréotypes, par exemple, on peut créer le classificateur *relation* associé au stéréotype `<<relationnel>>`, une classe ne comportant que des attributs (pas d'opération).

Diagramme d'objets

La notation UML, pour désigner les objets réutilise la forme rectangulaire. L'identité de l'objet est définie en deux parties, la première est le nom de l'instance, la deuxième le nom de la classe. Dans le deuxième compartiment, il est possible de spécifier l'état de l'objet en associant des valeurs aux attributs. La Figure 2-7 montre trois cas d'identification des objets :

- *Instance* : le cas d'un objet dont la classe n'est pas connue ;
- *Instance :Klass*, le cas d'un objet dont la classe est connue ;
- *:Klass*, un objet anonyme d'une certaine classe.

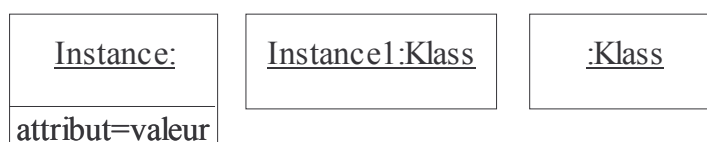


Figure 2-7 : les différentes appellations des objets.

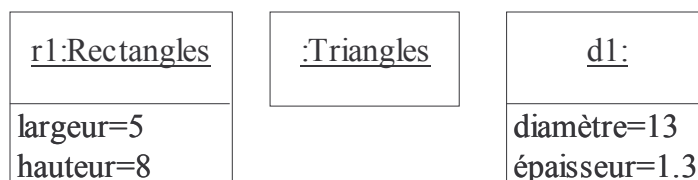


Figure 2-8 : exemples d'objets

La Figure 2-8 montre trois objets :

- Un rectangle r1 avec les états pris par ses attributs ;
- Un triangle anonyme ;
- Un objet d1 dont la classe n'est pas spécifiée.

Liens entre les objets

Les objets ont des liens entre eux. Dans l'exemple suivant, les liens entre les objets définissent la sémantique qui lie les objets. Ils décrivent comment interagissent statiquement les objets. On pourra dire que le cours est donné par un professeur, dans une tranche horaire, dans une salle. La salle est équipée avec des tables, derrière une table se trouvent deux chaises sur lesquelles sont assis Marie et Noé.

Les classes ont des **associations** entre elles. Les liens sont des instances des associations.

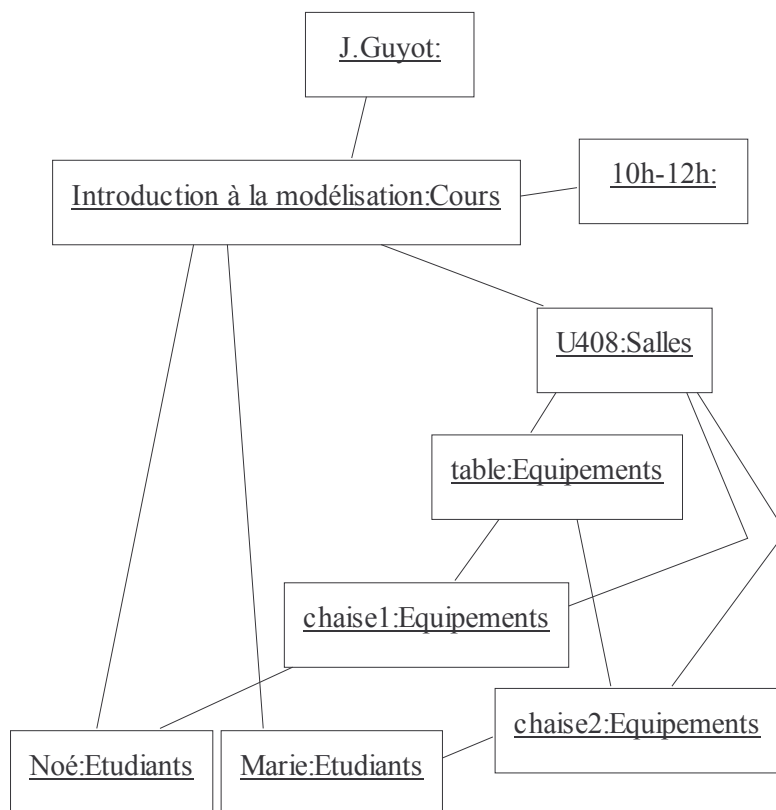


Figure 2-9 : les objets sont liés entre eux

Associations

Une association décrit la connexion entre plusieurs classes. Les classes sont indépendantes les unes des autres, dans le sens où l'existence d'une instance de ces classes ne dépend pas d'une autre. L'association détermine un couplage faible entre classes.

Les associations déterminent les interactions entre les classes. Le nom de l'association spécifie cette interaction. Pour être plus précis, on peut recourir aux rôles. Dans la Figure 2-10, on peut lire :

- Classe est associée à Klass ;
- Les cours sont suivis par des étudiants ;
- Les étudiants suivent des cours ;
- Les personnes sont les employés d'une entreprise dans une relation de travail ;
- Les entreprises sont les employeurs des personnes dans une relation de travail.

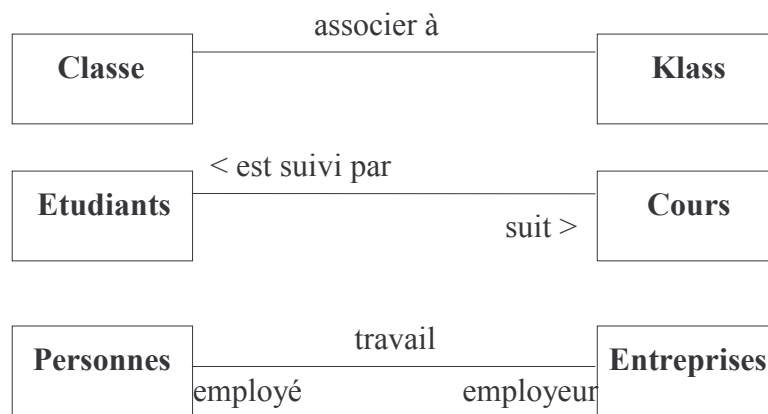


Figure 2-10 : exemples d'association

Entre deux classes, il peut exister plusieurs associations possibles, on parle aussi de rôles multiples. Les personnes peuvent entretenir des liens différents avec les villes. Elles peuvent y naître, travailler ou les habiter.

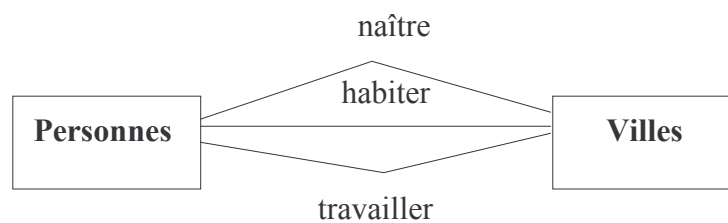


Figure 2-11 : exemple d'associations multiples.

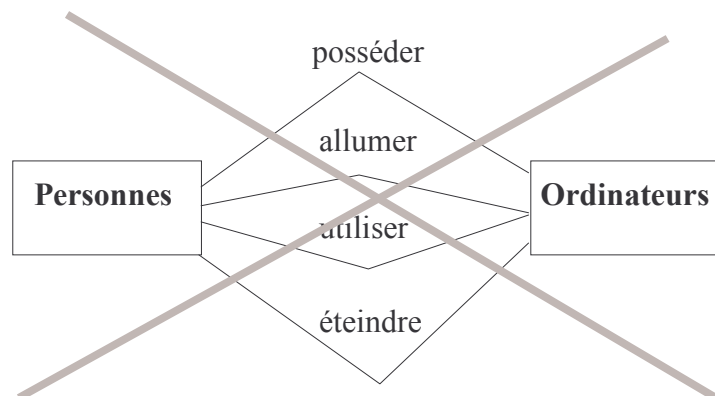


Figure 2-12 : les méthodes ne sont pas des associations.

Les associations doivent décrire une relation stable entre les classes. Par conséquent, les liens entre les objets doivent exister en dehors de l'exécution d'une méthode. Dans la Figure 2-12, *allumer*, *éteindre*, *réparer* sont des méthodes. Seul *posséder* établit un lien stable entre *Personnes* et *Ordinateurs*

Cardinalité des associations

Les associations sont contraintes par leur cardinalité. On peut imposer que tout objet de A est associé par r à un nombre minimal d'objets de B et au

plus à un nombre **maximum** d'objets de B. On indiquera cette contrainte en fixant un intervalle min..max à proximité de B.



Figure 2-13 : dénnotations de la cardinalité d'une association.

La modélisation de la Figure 2-14 dénote qu'un étudiant peut suivre plusieurs cours (0..n). Un cours doit avoir au moins 6 et au plus 30 étudiants.

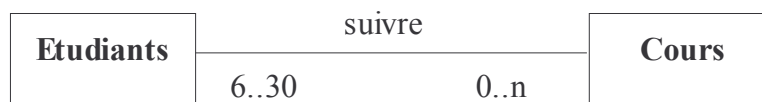


Figure 2-14 : exemples de cardinalité

Nous allons examiner plus précisément certains cas particuliers.

Cas 1..1, 1

Tout objet de A est associé exactement 1 objet de B (1 dénote 1..1)



Figure 2-15 : cardinalité 1 = 1..1

Nous avons par exemple, en acceptant ces généralités :

- Un être humain agit avec une seule personnalité ;
- Un croyant est fidèle à une seule religion ;
- Un enfant est né d'une seule mère ;
- Un ville est rattachée à un seul canton.

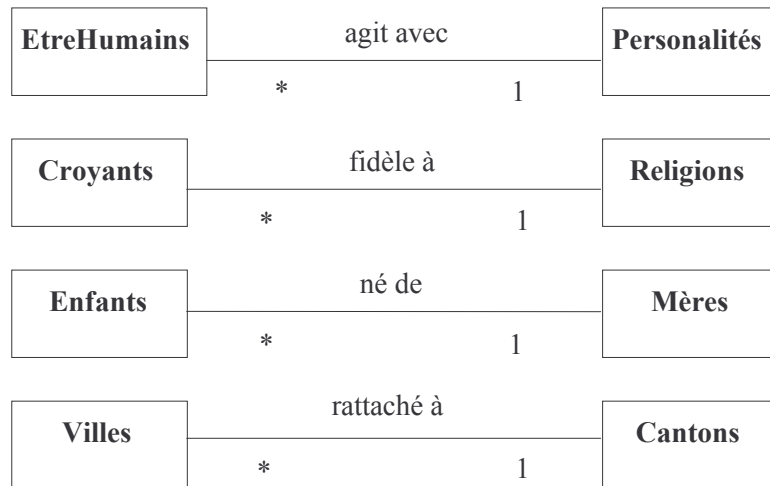


Figure 2-16 : exemple de cardinalité à 1

Cas 0..1

Tout objet de A est associé au plus à 1 objet de B



Figure 2-17 : cardinalité 0..1, l'option

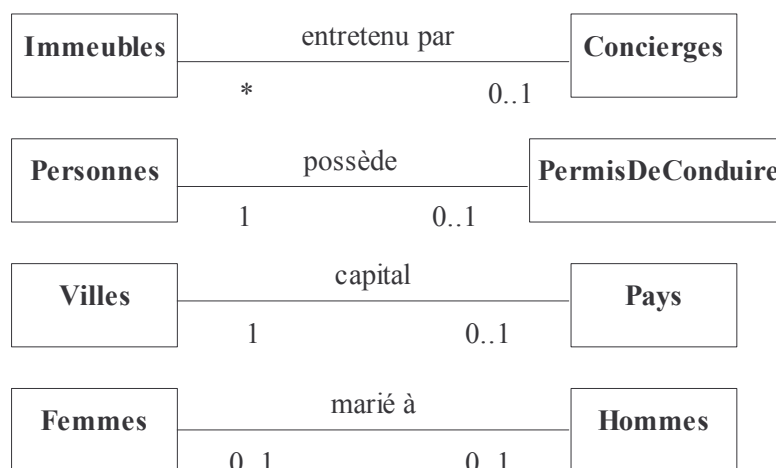


Figure 2-18 : exemple de cardinalité 0..1.

Nous avons par exemple, en acceptant ces généralités :

- Un immeuble est entretenu au plus par un concierge ;
- Une personne possède au plus un permis de conduire ;

- Une ville peut être la capitale d'un seul pays, un pays possède une et une seule capitale ;
- Un homme est marié au plus à une femme et réciproquement.

Ce dernier cas est intéressant, on conçoit aisément qu'il est possible que l'on ne distingue pas les hommes des femmes, nous aurions donc une classe *personnes*. L'association *marié à* devient une association entre *Personnes* et se dessine de la façon suivante.

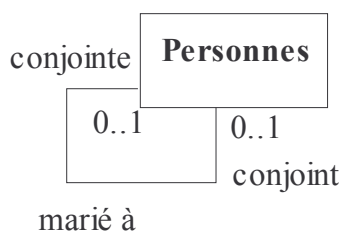


Figure 2-19 : association entre même classes

Les deux modélisations ne sont pas tout à fait équivalentes la première ne marie que des hommes à des femmes, la deuxième marie des personnes à des personnes sans distinction de sexe. Si l'on veut retrouver, la même contrainte, il sera nécessaire d'ajouter une règle d'intégrité (par exemple, demander que le sexe des conjoints soit différent)

Cas *, 0..*

Tout objet de A est associé à plusieurs objets de B. L'étoile dénote un nombre arbitraire.



Figure 2-20 : cardinalité * = 0..n

Cas 1..*

Tout objet de A est associé à au moins 1 objet de B, éventuellement plus.

Nous avons donc les exemples possibles suivants :

- Un auteur doit avoir écrit au moins un livre. Un livre peut être écrit par plusieurs auteurs et éventuellement ne pas avoir d'auteur connu ;
- Un homme peut posséder plusieurs nationalités (ou zéro pour les apatrides). Une nationalité peut être représentée par plusieurs personnes.

- Un pays peut posséder plusieurs monuments. Un monument se situe dans un et un seul pays.



Figure 2-21 : cardinalité 1..*

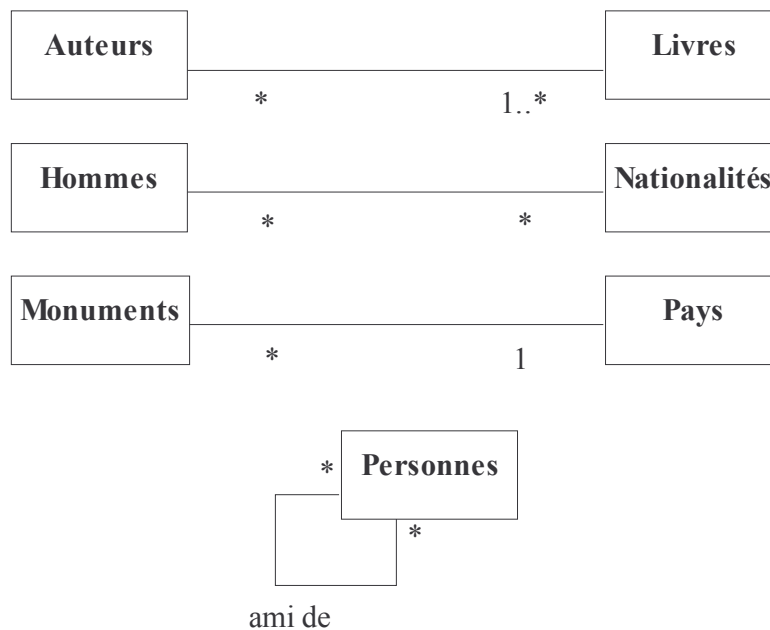


Figure 2-22 : exemple de cardinalité multiple.

Association binaire, ternaire, ...

On définit l'**arité** de l'association comme étant le nombre de classes participant à l'association. Pour le moment, nous n'avons examiné que des associations **binaires**.



Figure 2-23 : association binaire

Mais rien n'empêche de modéliser la naissance d'un individu en associant d'autres concepts à cet événement. Dans le cas suivant, l'association est **ternaire**. La naissance est l'association qui lie une personne, une ville et un

signe du zodiaque. Les cardinalités doivent être examinées, en fixant deux objets et en regardant à combien d'objets ce couple peut être associé.

Dans notre exemple, nous avons :

- Une personne et une ville ne peuvent être associées qu'à un signe ;
- Une personne et un signe ne peuvent être associés qu'à une ville ;
- Un signe et une ville peuvent être associés à plusieurs personnes.

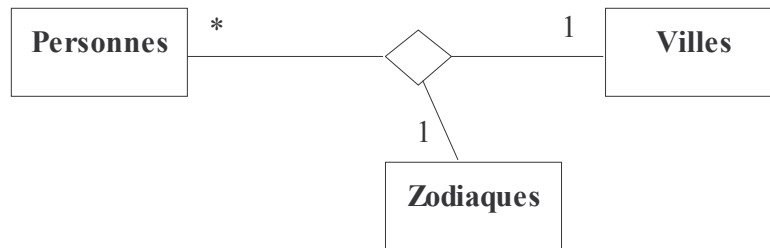


Figure 2-24 : association ternaire

On dira d'une association qu'elle est n-aire si sa cardinalité dépasse trois. Le cas suivant montre une association d'arité 5, la naissance est l'association qui lie une personne, une ville et un signe du zodiaque chinois, deux signes du zodiaque avec deux rôles distincts, l'un de signe principal, l'autre d'ascendant.

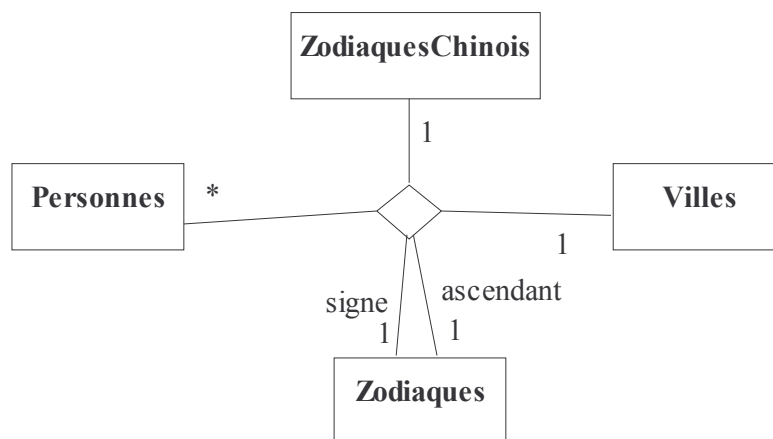


Figure 2-25 : association multiple

D'une manière générale, on évitera, les associations n-aire. En effet, elle cache souvent un concept qui peut être élevé au rang de classe. Notre exemple comprend alors une classe *Thème astral* qui est associée par des associations binaires aux classes précédentes.

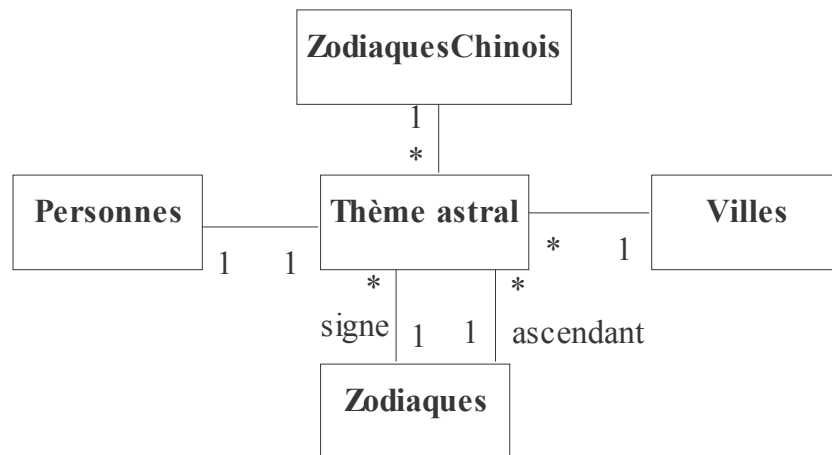


Figure 2-26 : retour aux associations binaires !

Classe d'association

Jusqu'à présent, nous avons eu le souci de modéliser uniquement le fait qu'un objet d'une classe soit associé à un autre d'une autre classe. Parfois, nous désirons ajouter des informations concernant cette association. Ces informations seront représentées par une classe d'association qui sera directement attachée à l'association (avec un type de trait brisé). Ceci va permettre de mémoriser des informations dans le lien.

Dans l'association *Travailler à*, entre *Personnes* et *Villes*, il nous est possible d'ajouter des informations précisant la période de travail avec les attributs *depuis* et *jusqua*. On remarquera qu'il est ainsi possible qu'une personne soit associée deux fois à une même ville pour des périodes différentes.

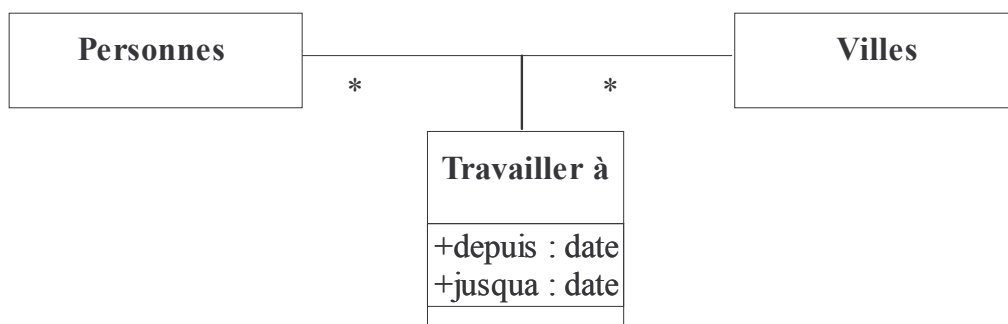


Figure 2-27 : exemple de classe d'association.

On représentera ces informations supplémentaires, en les rattachant sur le lien directement. Nous avons, dans l'exemple suivant que *Marie* travaille à *Paris* depuis le *1.1.2000* jusqu'à une date indéterminée.

Parfois la classe d'association doit perdre ce statut car elle contient des concepts qui demandent à être explicités. Nous allons examiner un exemple illustrant ce besoin d'explicitation.

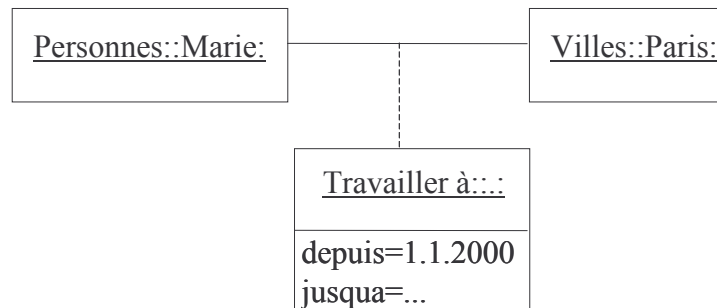


Figure 2-28 : exemple d'instance de classe d'association.

Dans la Figure 2-29, le concepteur a choisi de représenter Auteurs, Livres et l'association *Ecrit Par* qui les lie. Il a ajouté les attributs Editeur et date de parution à cette association.

Cette modélisation est erronée, car les attributs Editeur et Parution, ne concernent pas l'association, mais le Livre. Il est donc envisageable d'ajouter ces attributs à Livres. Mais cette nouvelle modélisation a pour inconvénient de ne pas pouvoir tenir compte qu'un livre peut avoir plusieurs éditions.

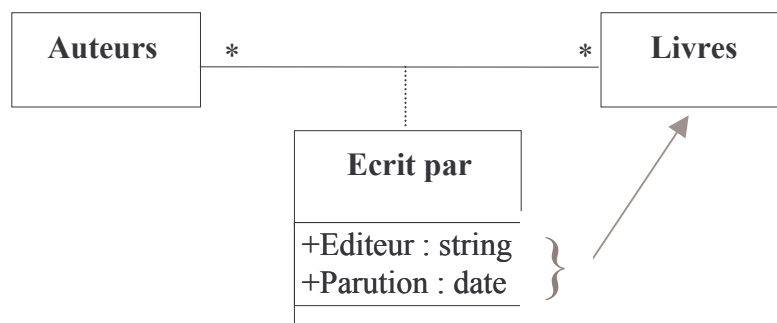


Figure 2-29 : l'association ne suffit pas.

Ici intervient le contexte de la modélisation. En effet, un particulier a rarement plusieurs versions du même livre. Par contre ce fait est courant pour un libraire. Le concept d'éditeur doit donc être pris pleinement en considération. Nous ajoutons donc une classe *Editeurs* et une association *Edition* qui contiendra les attributs *No ISBN* et *date de parution*.

La modélisation de la Figure 2-30, permet effectivement d'associer plusieurs éditions à un même livre. Mais en faisant glisser le contexte, on s'aperçoit qu'une œuvre littéraire peut être assemblée dans une édition sous diverses formes. Une nouvelle d'un auteur peut être publiée seule dans une

anthologie consacrée à un thème ou dans un recueil de nouvelles de cet auteur.

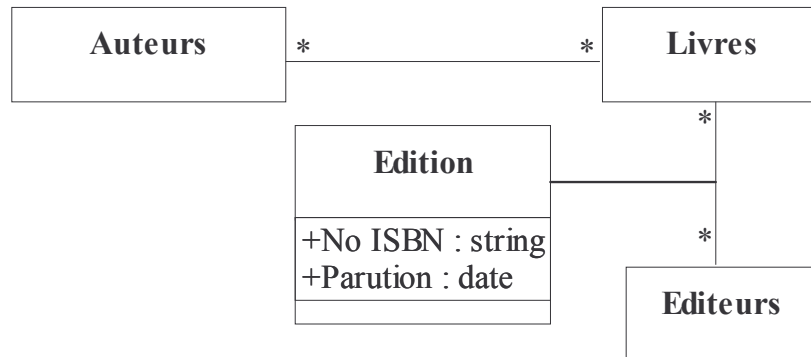


Figure 2-30 : suite de l'exemple

Pour modéliser ceci, il est indispensable de séparer l'œuvre de son mode de publication. Dans la modélisation qui suit, nous avons remplacé la notion d'édition par celle de publication. Il est même alors possible de modéliser la notion de coédition où chaque éditeur participe à un certain taux aux risques et aux bénéfices.

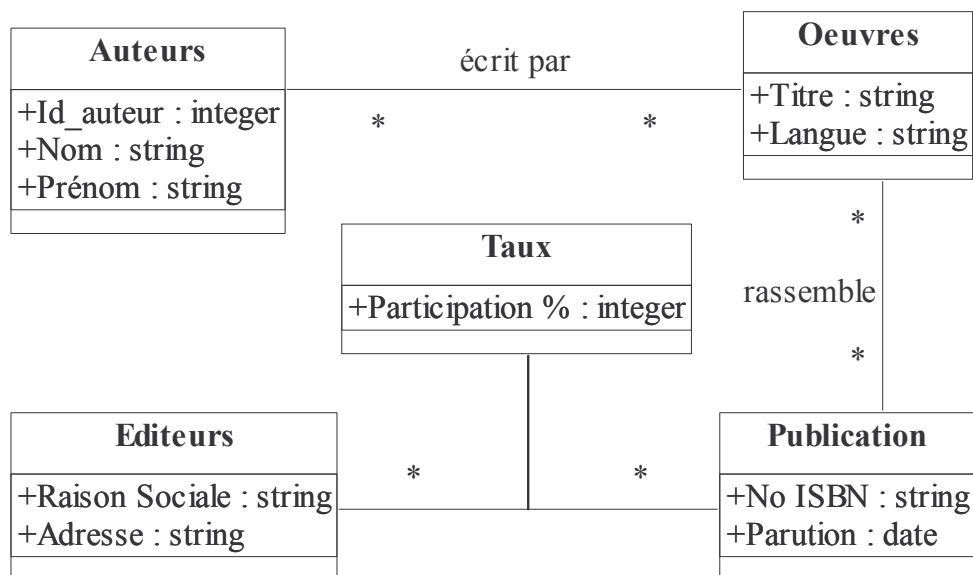


Figure 2-31 : fin de l'exemple.

Contraintes

Les cardinalités sont un cas particulier des contraintes. Pour exprimer une contrainte, on utilisera le stéréotype de contrainte (notée avec des accolades) que l'on attachera aux objets devant valider la contrainte. Les

contraintes peuvent être placées sur n'importe quel objet de la modélisation.

Dans la Figure 2-32, on montre un exemple de contrainte et une note qui sont attachés à la classe *Triangles*. Nous allons commenter quelques exemples de contraintes.

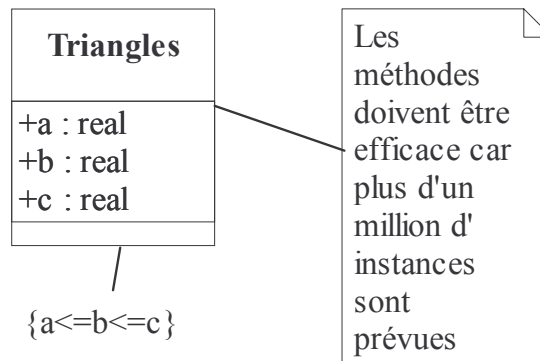


Figure 2-32 : dénotation des contraintes.

Les tirages de Loto sont associés à un nombre fixe de boules, six dans notre cas et ces associations sont ordonnées. Lors de la traduction de cette modélisation, il faudra ajouter une information pour tenir compte de cet ordre.

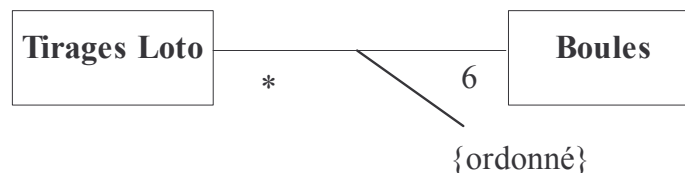


Figure 2-33 : contrainte sur l'ordre des liens.

Dans l'exemple qui suit, les employés sont associés à un vol. La règle énonce que chaque vol doit comporter un pilote et deux copilotes. Ceci est représenté par les cardinalités. Bien entendu, le pilote doit être distinct des copilotes. Il faut une contrainte externe pour l'exprimer. Une contrainte peut aussi être exprimée dans une note (par exemple, celle concernant les vols internationaux)

La contrainte qui est ci-dessous exprime que les conférenciers représentant une communication doivent être choisis parmi les auteurs de cette communication.

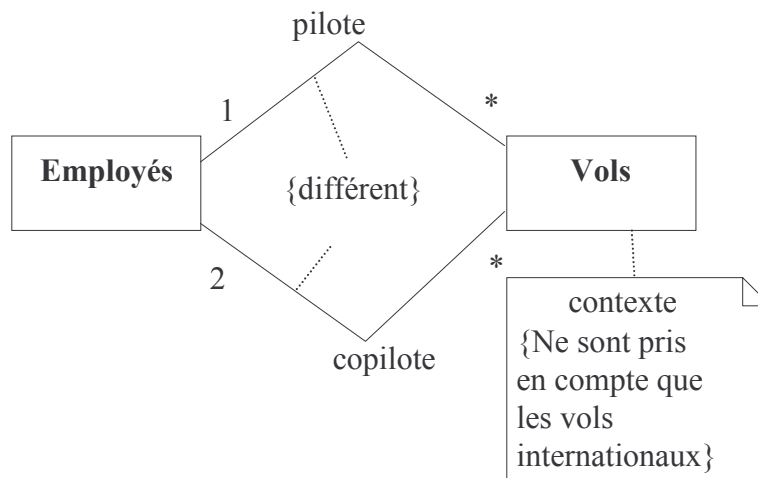


Figure 2-34 : contrainte entre rôles.

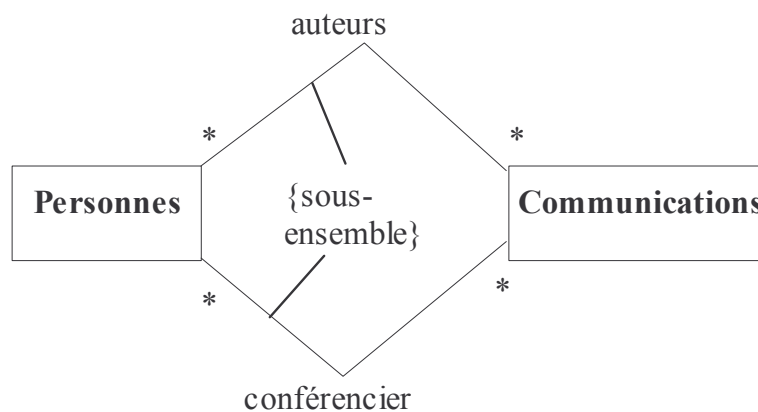


Figure 2-35 : contrainte entre rôles.

Le cas suivant exprime qu'un contribuable est soit une personne physique soit une personne morale.

On peut contraindre l'association à être parcourue que dans un sens. Le sens de navigation est indiqué par une flèche. Ce type d'information est important dans la programmation objet car les associations sont souvent implémentées avec des pointeurs et, comme l'indique le nom, il ne se parcourt que dans un sens. Pour qu'une association soit bidirectionnelle, il faut implémenter le chemin inverse.

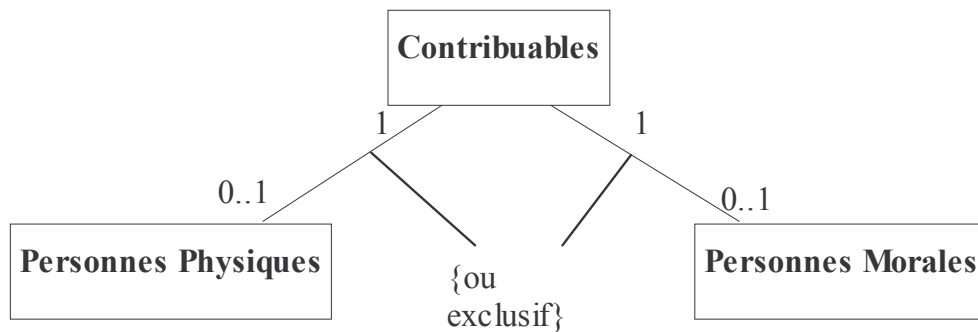


Figure 2-36 : contrainte entre associations

Cette notion de navigation n'a pas d'importance dans le cas d'une implémentation avec un système de gestion de base de données relationnelle car les associations sont représentées par des valeurs et non pas par des pointeurs.



Figure 2-37 : contrainte de navigabilité.

Pour définir plus formellement les contraintes, il est possible d'utiliser OCL (Object Constraint Language). OCL permet de définir des contraintes, des requêtes, des expressions booléennes et des expressions de navigation. Les expressions permettent de manipuler les attributs des classes, de former des ensembles, de contraindre ces derniers à posséder certaines propriétés. Pour plus de détail, nous renvoyons le lecteur à la documentation de OMG [OMG99] qui consacre un chapitre entier à OCL ou à l'ouvrage [WAR99]

Agrégation et composition

L'agrégation définit une association non symétrique, dans le sens que les classes associées ne sont plus complètement indépendantes. Nous avons des phrases du genre suivant pour marquer cette dissymétrie :

- A est formé de B ;
- A pré-existe à B ;
- B n'existe pas sans A ;
- A contient B (ensembliste).

D'autres dépendances moins structurelles peuvent exister telles que :

- Les attributs d'une classe sont dépendants de l'autre ;

- Les actions d'une classe sont dépendantes de l'autre.

L'agrégation est spécifiée par un petit losange du côté de la classe qui joue le rôle de maître dans l'association.

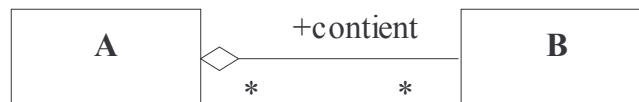


Figure 2-38 : dénotation de l'agrégation.

Voici quelques exemple d'agrégation : Une recette de cuisine est élaborée avec des ingrédients. Un polygone est constitué d'une suite de points sur le plan. Un portefeuille de titres est un ensemble de placement.

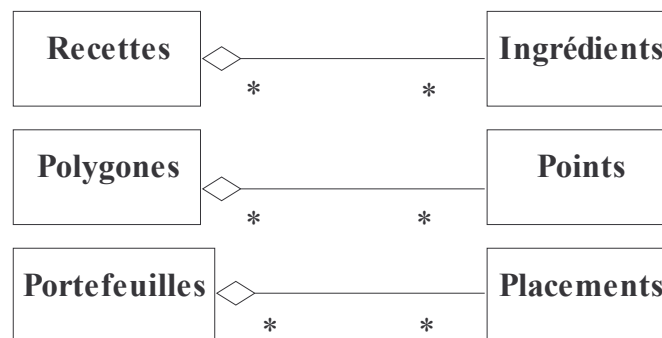


Figure 2-39 : exemples d'agrégation.

Si un objet de B n'est lié qu'à un objet de A alors l'agrégation est une composition. La cardinalité l'association est de 1 pour les objets de B. La composition se distingue graphiquement par un losange rempli. Les objets de B sont entièrement dépendant de ceux de A.

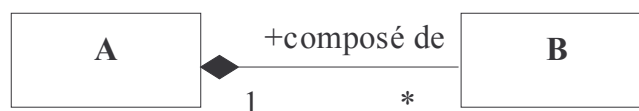


Figure 2-40 : dénotation de la composition.

En reprenant l'exemple, On peut constater qu'un polygone est composé d'un ensemble de points. En utilisant l'agrégation pour modéliser la même situation, les points du plan peuvent être partagés. Dans la composition, il n'y a pas de partage : si deux polygones partagent un sommet, il existe deux points différents qui ont des valeurs identiques comme coordonnées.

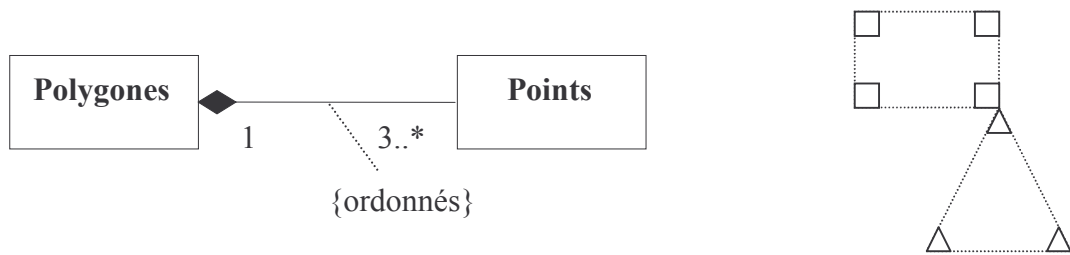


Figure 2-41 : exemple de composition

Le choix d'une modélisation dépend essentiellement des opérations à effectuer sur les polygones :

La composition favorise les déplacements indépendants des polygones, les homothéties de chaque objet. Elle sera adéquate pour la représentation d'un système de logiciel de dessin.

L'agrégation favorise le travail sur les points. En effet, modifier la position d'un point entraînera la modification de tous les polygones qui le partagent. Elle sera choisie pour la modélisation de treillis, comme dans l'exemple, montrant la modélisation d'un hélicoptère en « fil de fer ».



Figure 2-42 : « Applet » d'animation 3D «fil de fer » de Sun

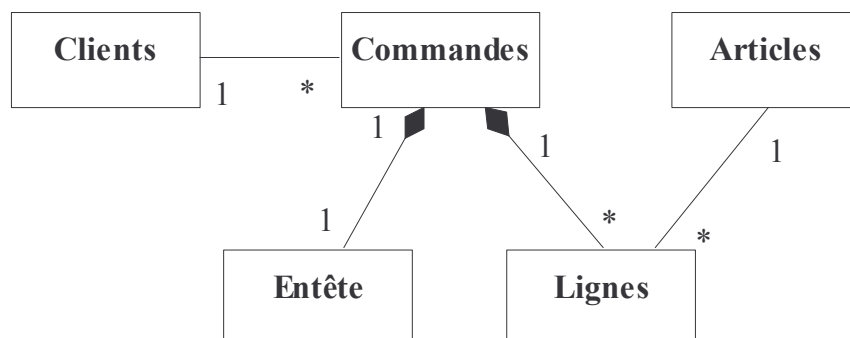


Figure 2-43 : exemple classique des clients et des commandes.

L'exemple ci-dessus est celui de la commande d'un client, une commande est composée d'une entête et d'un ensemble de lignes de commande,

chaque ligne de commande est associée à un article. Une commande est associée à un client.

Dans la programmation objet, le fait de reconnaître une composition peut modifier la nature même de la modélisation : on peut transformer une composition en un type et attribuer ce type à un attribut. Dire qu'une personne est associée à un *domicile* par un lien de composition ou spécifier qu'un attribut *domicile* de la personne est du type *Adresse* est équivalent. On doit toutefois remarquer que dans le cas où l'implémentation se fera dans une base de données relationnelle, il n'est pas possible d'associer des types complexes à un attribut. Une telle modélisation relationnelle ne serait pas en première forme normale 1NF (voir chapitre sur la décomposition). Dans notre cas, nous préférons toujours expliciter les associations.

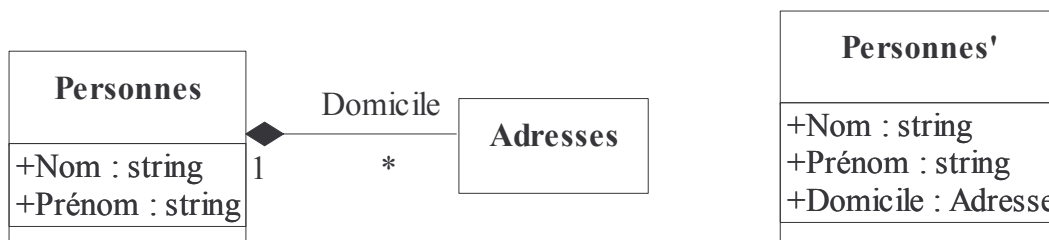


Figure 2-44 : composition ou type ?

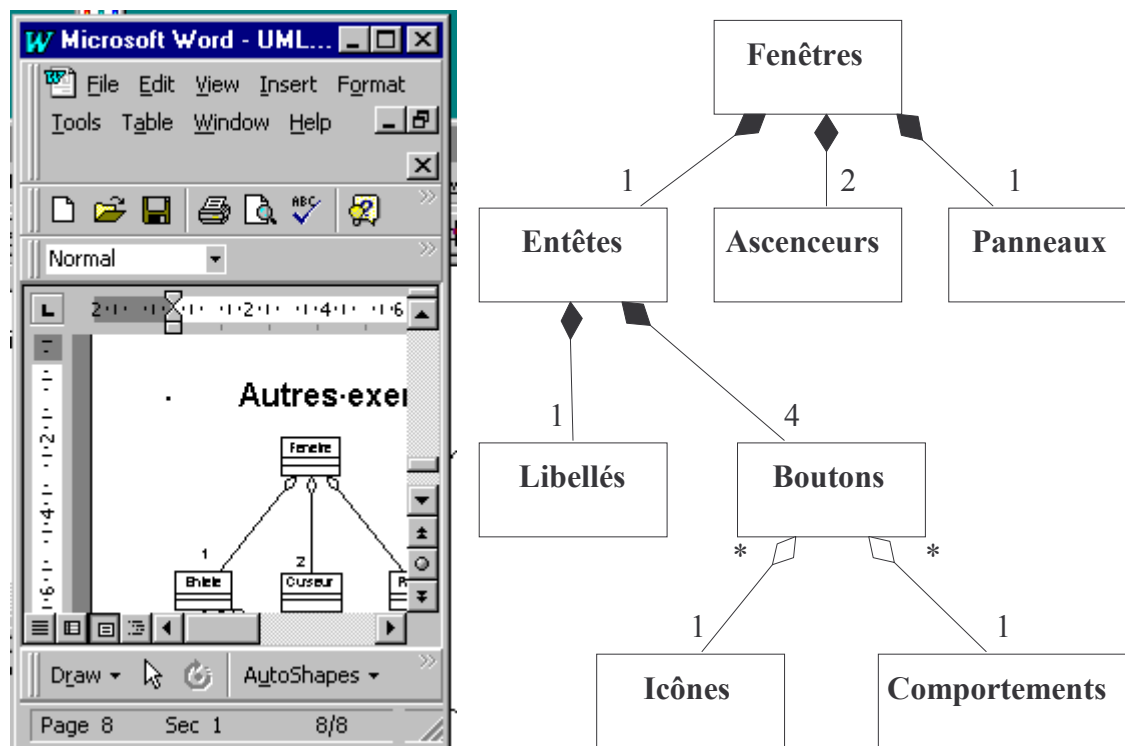


Figure 2-45 : Exemple des fenêtres.

Comme dernier exemple, nous prenons celui d'une fenêtre à afficher sur un écran d'ordinateur. Une fenêtre est composée d'une entête, de deux ascenseurs et d'un panneau. Une entête est composée d'un libellé et de

quatre boutons. Un bouton est associé à une icône et un comportement. On remarquera que l'agrégation et la composition permettent de construire des hiérarchies qui spécifient par raffinements successifs les détails du système à modéliser.

Généralisation

Le dernier concept de la modélisation des classes est celui de la généralisation. L'objectif de ce concept est de permettre de partager entre les classes des attributs et des méthodes. Les raffinements successifs de la modélisation ne concernent plus les détails structurels d'un objet formé par d'autres objets (le cas de composition) mais un même objet dont le comportement se spécialise selon la spécification de son instance. On parle aussi de relation d'héritage. Graphiquement, la sous classe est reliée par une flèche creuse à la classe principale. Un objet sera une instance d'une seule classe. Il s'agit donc d'expliciter une structure entre les classes.

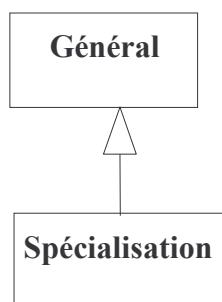


Figure 2-46 : représentation de la généralisation.

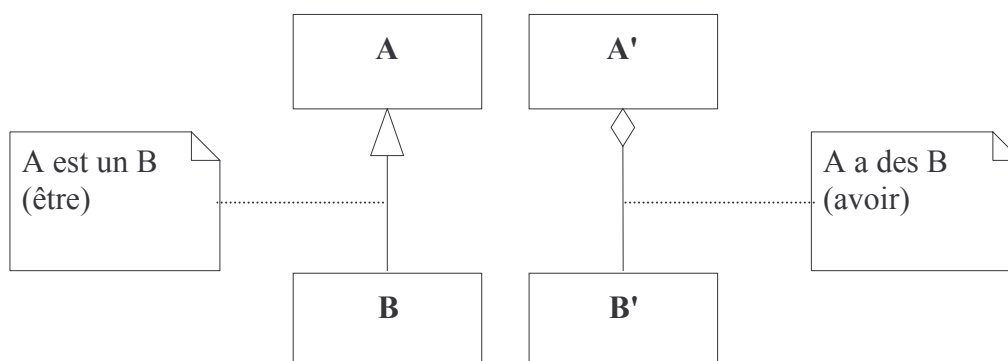


Figure 2-47 : différence entre généralisation et composition.

Différence entre généralisation et composition ?

Dans le cas de la composition, il existe des instances de A' et de B', les instances de A' sont composées par des instances de B'. Dans le cas de la généralisation, il existe des instances de A et de B, mais leurs instances ne sont pas liées. Les instances de B héritent du même comportement que les instances de A.

Exemples de généralisation

Illustrons ce nouveau concept avec des exemples :

Dans la Figure 2-48, on trouve deux spécialisations de *Boissons*, celles *Sans Alcool* et celles *Alcoolisées*. Les boissons *Sans Alcool* sont-elles mêmes spécialisées en *Eaux minérales* et *Jus de Fruit*. Pour les *Alcoolisées*, on trouve les *Vins* et les *Bières*. Il est intéressant de voir qu'il existe des instances dans chacune de ces classes, les classes les plus spécialisées héritent de tous les attributs et de toutes les méthodes de leurs ancêtres.

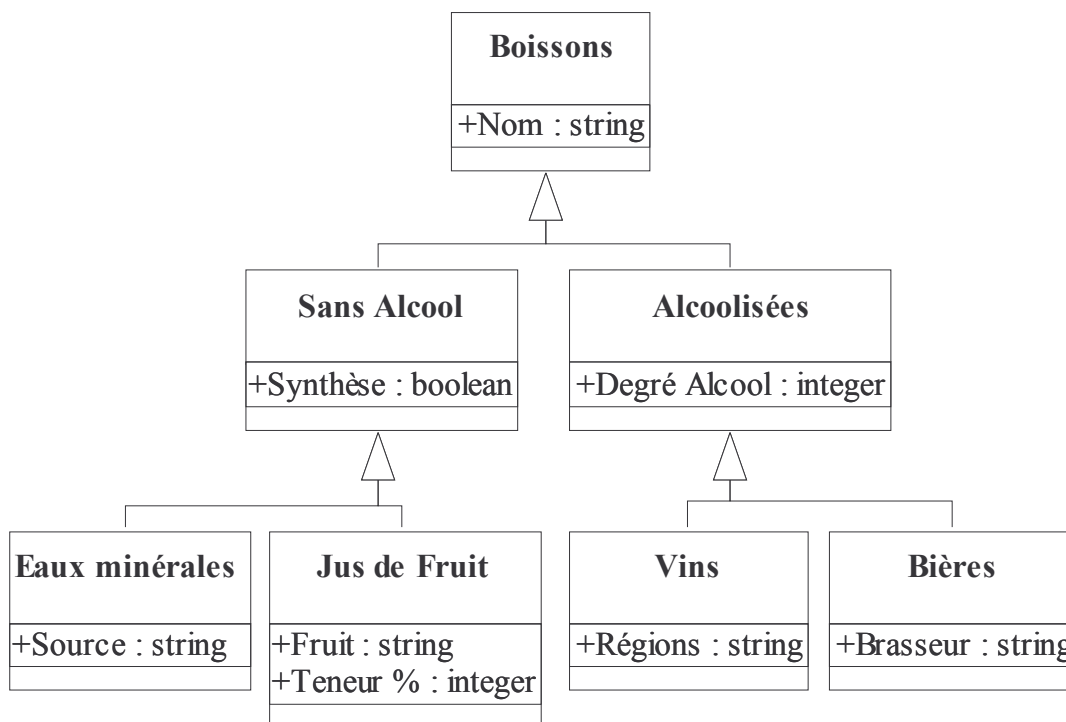


Figure 2-48 : spécialisation des boissons

Nous avons donc les instances suivantes pour les spécialisations de boisson :

- o1 : Boisson (Nom=élixir) ;
- o2 : Sans Alcool (Nom=Coca-Cola, Synthèse=oui) ;
- o3 : Alcoolisées (Nom=Whisky, Degré Alcool= 45) ;
- o4 : Eaux minérales (Nom=Evian, Synthèse=non, source=Evian) ;
- o5 : Jus de fruit (Nom=Pommos, Synthèse=non, Fruit=pomme, teneur%=15) ;
- o6 : Vins (Nom=Château Margaux, Degré Alcool= 13, Région=Bordeaux) ;
- o7 : Bières (Nom=Mort Subite, Degré Alcool= 5, Brasseur=Keermaeker).

La Figure 2-49 illustre la différence entre l'agrégation et la composition : l'eau est une boisson, mais elle contient des sels minéraux.

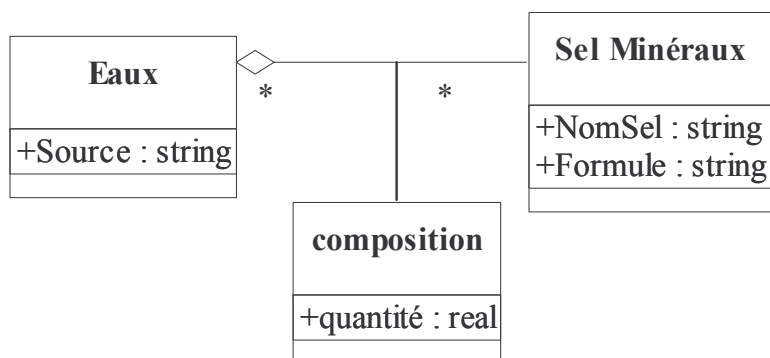


Figure 2-49 : utilisation de la composition

L'évian est une eau minérale, mais aussi une boisson sans alcool et finalement une boisson. L'évian *contient* du NaCl (sel).

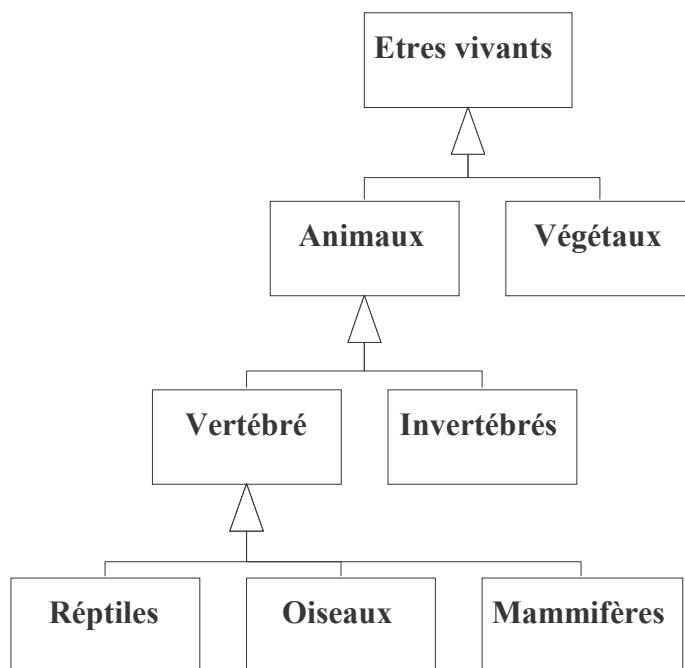


Figure 2-50 : l'héritage est classifiant.

La généralisation est classifiante

L'héritage établit un ordre classifiant entre les différentes sous-classes. Ce type de classification est bien connu dans le domaine de la biologie.

A nouveau, l'appartenance d'un individu à une classe le fait hériter de tous les attributs et de toutes les méthodes des classes parentes (aussi nommées super-classes).

Cette facette de l'héritage peut en compliquer son utilisation. Examinons la Figure 2-51 : nous avons séparé les surfaces planes en polygones et en ellipses, les polygones en rectangles et en triangles. Mais peut-on vraiment dire que :

- un carré est un rectangle ?
- un cercle est une ellipse ?

Peut-on parler de la largeur et de la hauteur d'un carré ?

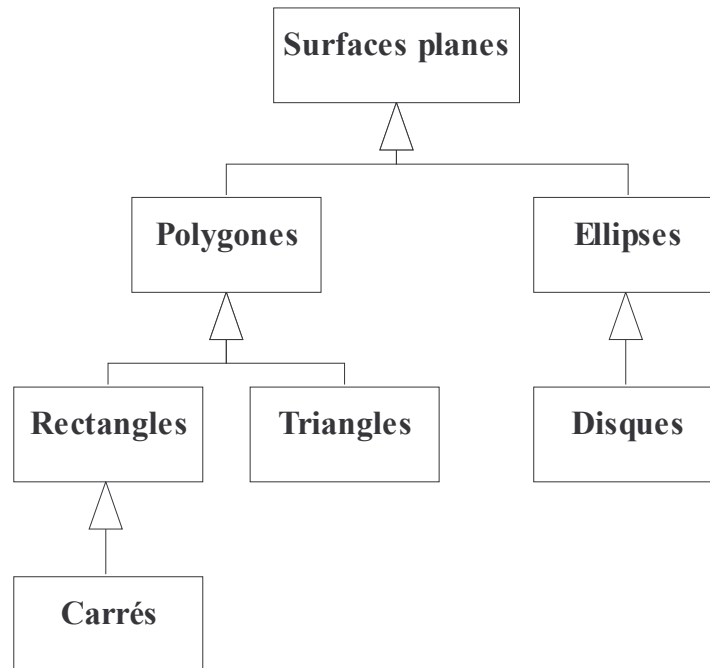


Figure 2-51 : l'héritage n'est pas toujours simple.

Principe de substitution de Liskov

Le principe de substitution de Liskov [LIS94] s'énonce ainsi « Si A est un B alors il doit être possible de substituer à un objet de la classe la plus générale (B) un objet d'une de ses sous-classes (A) sans modifier le comportement du système (un programme) »

Imaginons qu'il existe une méthode *miauler()* dans la classe *Chats* de la Figure 2-52. Une instance *minou:* de la classe *Siamois* devient après substitution une instance *minou:* de la classe *Chats* et doit donc pouvoir miauler.

Revenons à nos rectangles et à nos carrés. On peut imaginer que l'on décide que les carrés sont des rectangles. Pour masquer, le fait que le carré n'ait qu'un côté significatif, on ajoute une nouvelle méthode qui créera les carrés en initialisant la largeur et la hauteur avec la valeur du paramètre côté. La méthode *Doubler()* ne fait plus la distinction entre la hauteur et la largeur. La méthode *GetCôté()* permet de retrouver le côté. Cette implémentation a pour désavantage d'utiliser deux variables pour stocker une seule valeur. Mais admettons que nous n'ayons pas de problème d'espace mémoire.

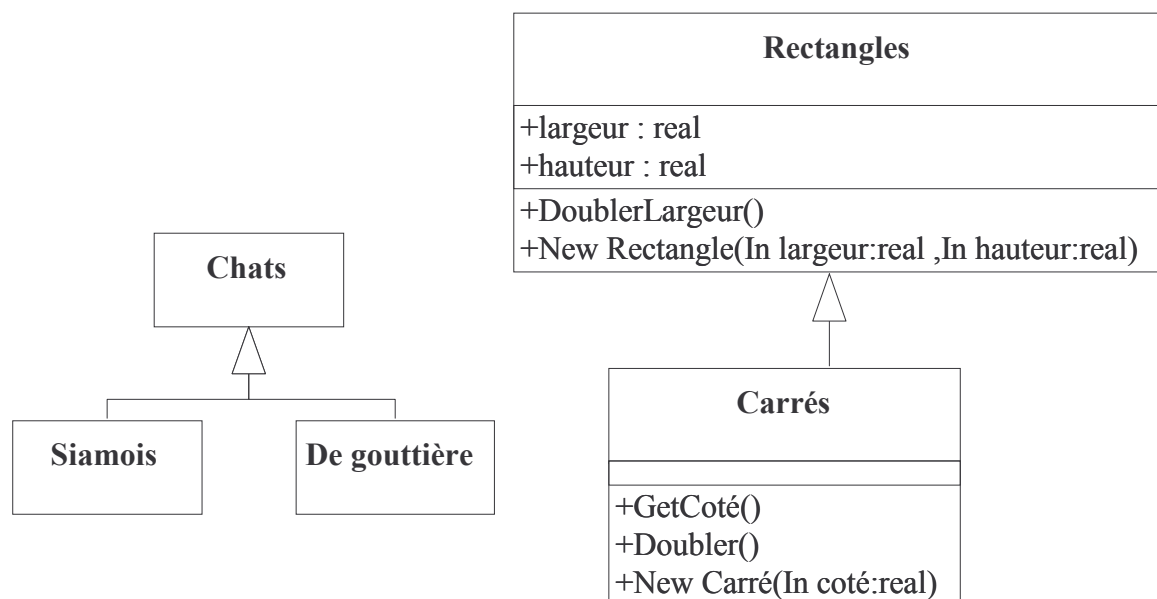


Figure 2-52 : illustration du principe de substitution de Liskov

Le principe de substitution est-il respecté? Imaginons le carré c :(largeur=8,hauteur=8). Si on le considère comme un rectangle, il est possible de lui demander de doubler sa largeur *DoublreLargeur()*. Effectivement, il peut se comporter comme un rectangle, mais se faisant, il perd sa nature de carré. En effet après l'exécution de la méthode nous avons pour c :(largeur=16,hauteur=8).

Généralement toute spécialisation qui opère par restriction, ne respecte pas le principe de substitution de Liskov.

Nous allons voir que la généralisation n'est pas toujours utilisée comme un outil conceptuellement structurant, mais qu'il peut être utilisé par les développeurs dans les cas suivants:

- Pour propager et partager du code ;
- Comme mécanisme de version ;
- Comme mécanisme de réutilisation.

Factoriser le code

Prenons le cas d'une étude qui a déjà été faite pour l'étude des trajectoires des planètes. Un peu plus tard, il est demandé d'étendre cette étude aux satellites. Le programmeur avisé constatera qu'il possède déjà une méthode pour le calcul en faisant de *Satellite* ; une sous-classe de *Planètes*, il résout probablement son problème.

Cependant, même si la trajectoire des satellites peut se calculer comme celle d'une planète, la lune n'est pas une planète.

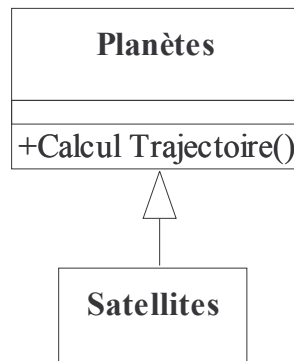


Figure 2-53 : hériter des méthodes

Une telle utilisation de la généralisation s'écarte d'une modélisation conceptuelle.

Héritage Multiple

L'héritage multiple décrit le cas où une classe hérite son comportement à partir de plusieurs classes. Le cas le plus simple est celui où ces classes ne partagent pas la même arborescence. Dans la Figure 2-54, les objets de la classe *Planètes* se comportent comme des corps célestes dont on peut établir la trajectoire, et comme des sphères dont on peut fixer le diamètre et dont il est possible de calculer le volume. On remarquera que l'on a résolu le problème d'héritage entre *Planètes* et *Satellites* en isolant la méthode *CalculTrajectoire()* dans une classe plus abstraite.

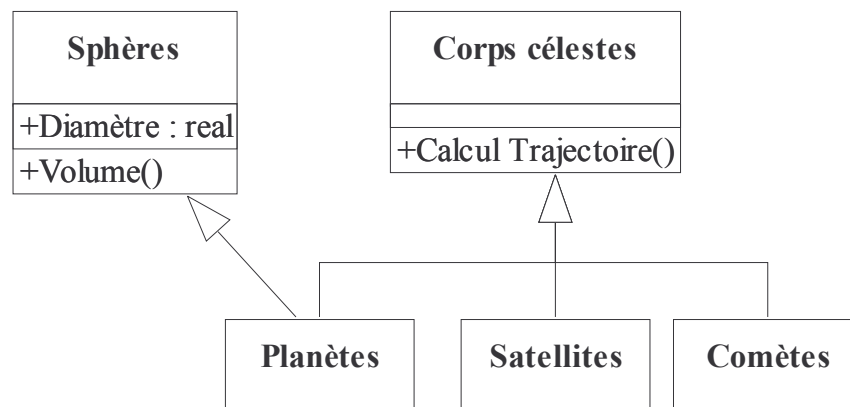


Figure 2-54 : héritage multiple.

Un cas plus complexe est celui où les super-classes partagent la même arborescence. Dans la Figure 2-55, les objets de la classe *Nectarines* se comportent comme des pêches et des prunes. Souvent, l'héritage n'est pas complet et il faut préciser ce que l'on hérite de l'un et de l'autre, et ce que

l'on masque. Métaphoriquement, pour les nectarines on veut l'arôme de la pêche et la texture de la peau des prunes.

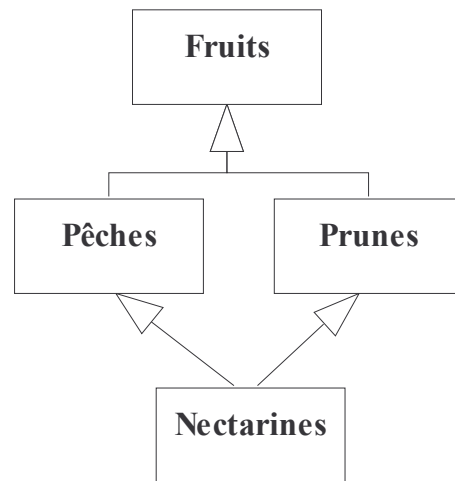


Figure 2-55 : complexité de l'héritage multiple.

Covariance

La covariance permet d'intégrer plusieurs critères indépendants dans une même arborescence. Il s'agit de classer avec deux critères, par exemple. L'emploi d'un tableau est plus adéquat que celui d'un arbre ! Jouons le jeu et classifions les véhicules avec un critère du milieu et un critère sur la propulsion.

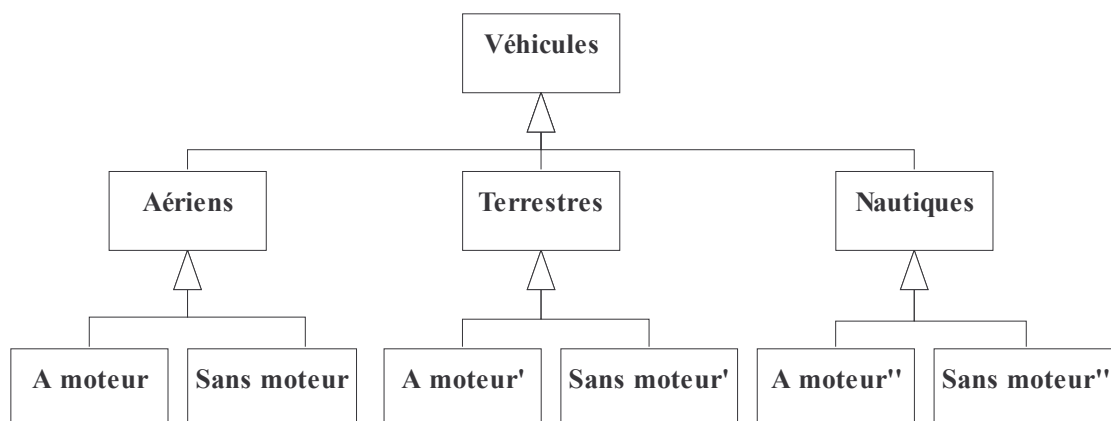


Figure 2-56 : covariance - une vision possible

Nous obtenons deux visions possibles :

- Figure 2-56, ici la classification s'effectue d'abord par le milieu et ensuite par la propulsion ;
- Figure 2-57, la classification s'effectue d'abord par la propulsion et ensuite par le milieu.

On constate que l'utilisation de la généralisation pour traiter la covariance n'est pas adéquate. Le nombre de sous-classes au dernier niveau peut devenir grand. Il est égal au produit du nombre de critères à chaque niveau.

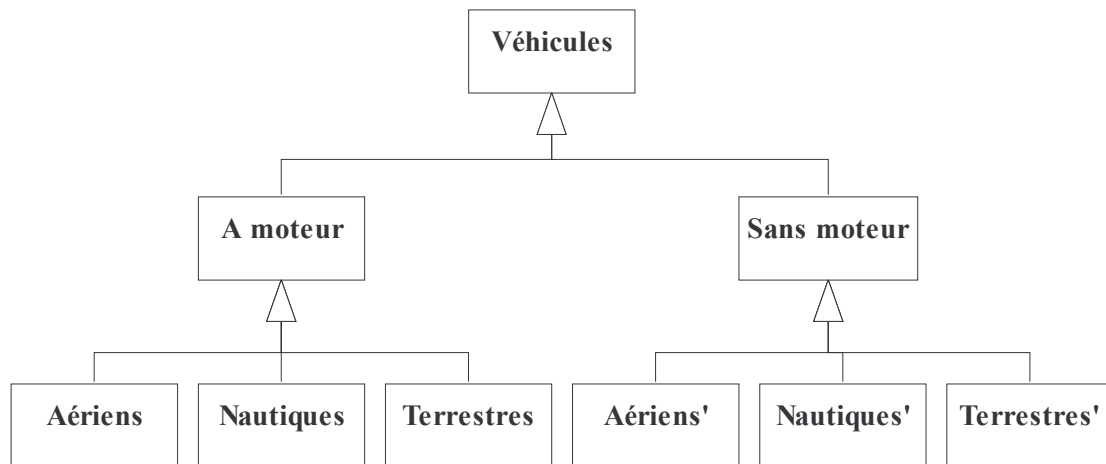


Figure 2-57 : covariance - une autre vision possible

On préférera utiliser l'héritage multiple telle que dans la Figure 2-58. Le nombre de classes est la somme du nombre de critères à chaque niveau.

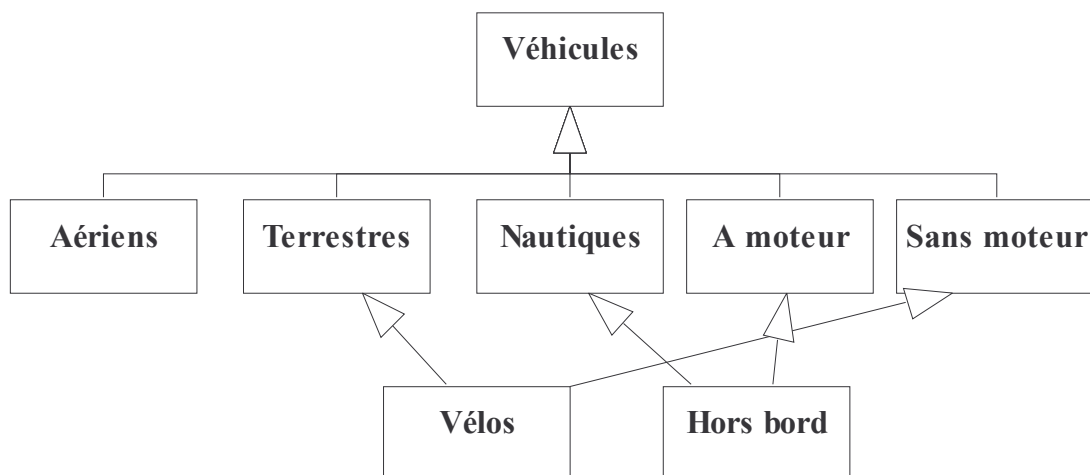


Figure 2-58 : utiliser l'héritage multiple ?

L'héritage multiple est complexe à utiliser. Les concepteurs du langage Java ont préféré l'abandonner au profit du concept d'interface. Par exemple, les classes *Nautiques* et *A moteur* deviennent des interfaces. On dira alors que la classe *Hors bord* implémente les interfaces *Nautiques* et *A moteur*. Il n'existe donc que des instances de *Hors bord* et qui se comporte comme des véhicules nautiques et à moteur. Pour une discussion plus approfondie sur ce sujet voir [BON99].

La Figure 2-59 reprend la même modélisation en utilisant le concept d'interface. On remarquera l'utilisation simultanée de l'icône interface et de la boîte standard. La dépendance d'implémentation est représentée par un trait discontinu.

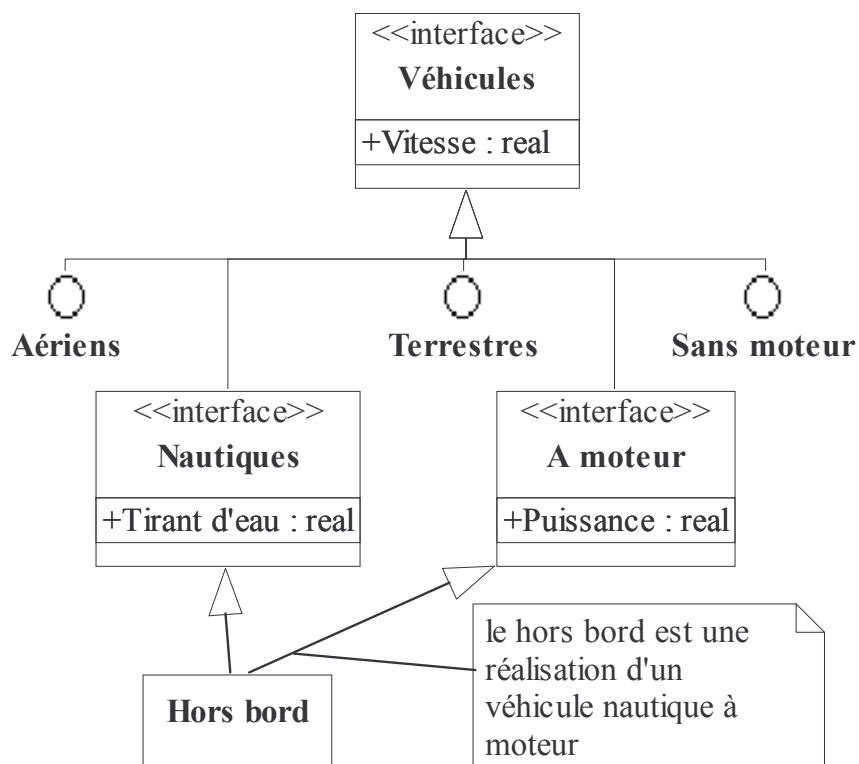


Figure 2-59 : utiliser l'héritage multiple ?

Les exemples sont souvent purement académiques. Dans la pratique, la modélisation de la Figure 2-60 en termes de données sera souvent largement suffisante.

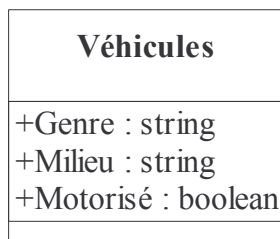


Figure 2-60 : parfois les choses ne sont pas si compliquées.

3. Les cas d'utilisation

« Faire la différence entre connaissance et opinions constitue déjà un degré d'évolution avancé » Le quatrième royaume-Luis Ansa

Les cas d'utilisation sont un modèle d'UML ; ils ont été formalisés par Ivar Jacobson. Les cas d'utilisation sont en amont de la conception des systèmes informatisés. Ivar Jacobson montre dans [JAC92] qu'ils sont un concept du génie logiciel. Les cas d'utilisation donnent une vision des services rendus par le système sans entrer dans les détails de leur réalisation. Des scénarios spécifient les interactions entre les composants du système et les acteurs déclenchent les cas d'utilisation. Il est possible de repenser à ce niveau les activités et Ivar Jacobson décrit dans [JAC94] comment les cas d'utilisation sont adaptés au BPR (Business Process Reengineering). Les cas d'utilisation sont aussi un élément de la réutilisation logicielle. [JAC97].

Le modèle des cas d'utilisation va spécifier le comportement d'un *système* (ou d'une partie d'un système, voire d'une classe) tel qu'il apparaît à un utilisateur extérieur à ce système. Les services rendus par ce système vont être identifiés et appelés *cas d'utilisation*. Les cas d'utilisation définissent une transaction entre le système et un *acteur*, souvent un utilisateur idéalisé dans un rôle particulier.

Objectifs du modèle

La spécification d'un système (informatisé) est un problème intrinsèquement complexe, car généralement la connaissance des besoins du système à réaliser et la connaissance des moyens de la réalisation du système ne sont pas dans une même unité cognitive (personne). Cette séparation rend difficile la communication. Ces difficultés sont quotidiennes entre ceux qui expriment les besoins du système à réaliser (les utilisateurs) et ceux qui connaissent les technologies pour concrétiser le système (les informaticiens).

L'objectif des cas d'utilisation est de créer un trait d'union cognitif : Les utilisateurs sont la source d'information et les créateurs de ce modèle. Ils ont la charge de :

- **Déterminer les besoins** : seuls les utilisateurs sont capables de faire ce travail, ils ont des connaissances profondes du domaine, ils savent ce qui est nécessaire, ils connaissent les raisons et l'utilité des procédures qu'ils emploient.

- **Comprendre les besoins** : en travaillant à la détermination des besoins, on va simultanément travailler sur la raison d'être de ces derniers dans le système et l'on aboutit généralement à un système dont les fonctionnalités sont nécessaires et suffisantes.
- **Délimiter le système** : finalement le résultat sera de fixer une frontière entre l'intérieur et l'extérieur du système : ce que fera le système, ce l'on peut attendre du système.
- **Intégrer les vues utilisateurs** : dès que le système atteint une certaine taille, les utilisateurs ne possèdent qu'une vue partielle de l'ensemble. La modélisation des cas d'utilisation est le lieu où ces connaissances partielles peuvent (et doivent) s'intégrer.

On peut résumer le modèle des cas d'utilisation par la fiche signalétique suivante :

- Qui participe ? Les utilisateurs ;
- Comment ? En langage naturel et avec un formalisme simple ;
- Quoi ? Ce que doit réaliser le système ;
- Pour qui ? Les informaticiens (éventuellement un BPR du système).

Processus de spécification des besoins

Commentons la Figure 3-1 : Généralement, à travers d'un schéma directeur, l'organisation se fixe des objectifs à atteindre en termes de qualité et de quantité. Ces objectifs sont communiqués aux diverses structures et utilisateurs de l'organisation. Ces objectifs demandent la collaboration de plusieurs départements pour être atteints.

Dans le cadre de la réalisation d'un système d'information, les responsables des différents départements reçoivent le mandat d'élaborer la spécification d'un système d'information remplissant les objectifs de la direction. Ces responsables vont décrire le système à réaliser au moyen des cas d'utilisation. Ces cas d'utilisation feront généralement l'objet d'une description sous forme d'un scénario, d'un diagramme de séquence ou de collaboration. Les cas d'utilisation sont déclenchés par des acteurs externes au système et ne sont donc pas concernés par le découpage interne de l'organisation. Ceci va favoriser l'intégration des vues des différents responsables.

Le processus de spécification va être réitéré afin de raffiner les cas, de présenter un niveau de détail suffisant pour que l'on puisse effectivement réaliser un système.

Parallèlement, sur la modélisation des cas d'utilisation, il est possible de revoir les procédures de gestion (Business process Reengineering ou BPR).

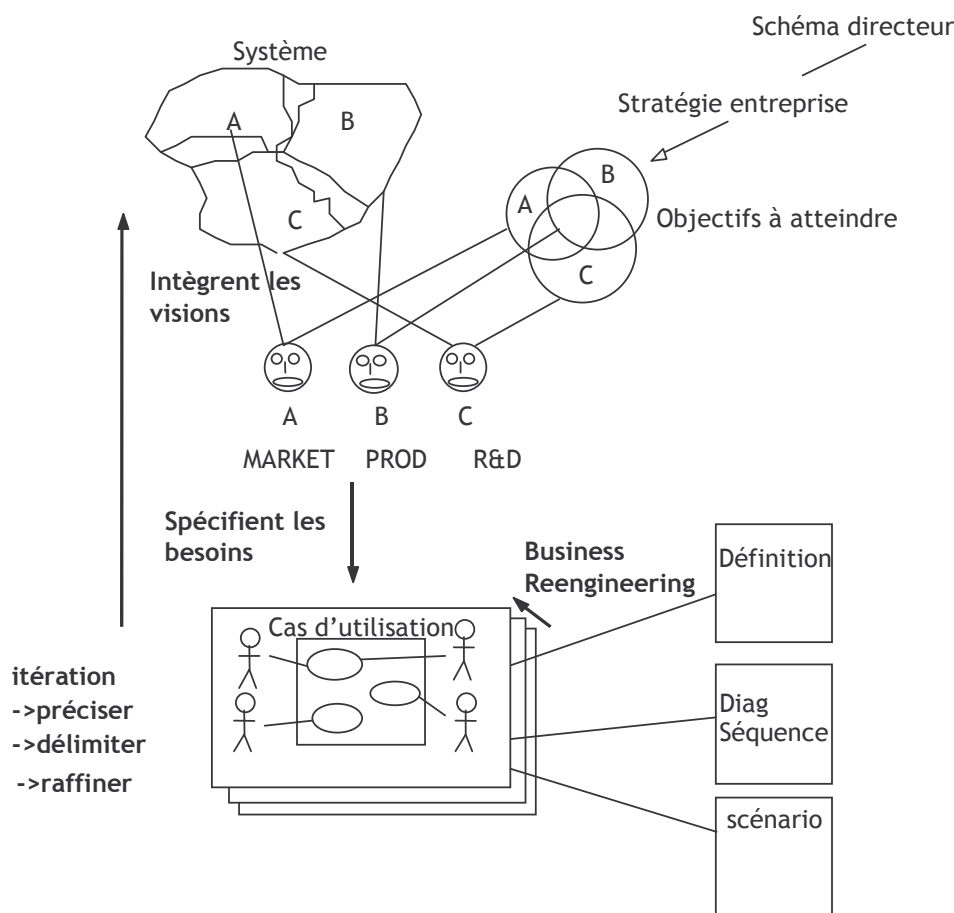


Figure 3-1 : processus de spécification des besoins.

Processus de développement

Les développeurs du système vont recevoir comme mission de réaliser un système qui peut s'utiliser comme décrit dans les cas d'utilisation. Il s'agira alors de modéliser le système avec des classes, des diagrammes d'états-transitions, de découper la complexité du système en modules (packages), de décrire le déploiement du système dans ces différents composants.

Les cas d'utilisation vont être la référence durant tout le processus de développement. Il s'agit de ne pas inventer ou introduire des comportements imaginés par les informaticiens.

La procédure de test sera simple car il suffira de suivre les scénarios des cas d'utilisation et de contrôler que le système peut s'utiliser comme il était prévu.

Précédemment, nous avons évoqué que nous cherchions satisfaire les critères de complexité, d'évolutivité, d'implémentabilité et de robustesse. Nous cherchons aussi à diminuer la complexité du système, en permettant aux utilisateurs de spécifier le système. Nous le rendons plus concret car les utilisateurs sont plus centrés sur l'action que sur les abstractions. Au départ, les descriptions peuvent paraître embrouillées, mais au fur et à mesure des itérations, les objectifs de ces actions vont se dégager. En terme

d'évolutivité, les cas d'utilisation sont bien adaptés et sont recommandés pour les BPR. L'implémentabilité n'est pas garantie, mais elle sera conçue par les besoins exprimés et généralement elle est le résultat d'une description d'une action qu'exécute déjà un utilisateur.

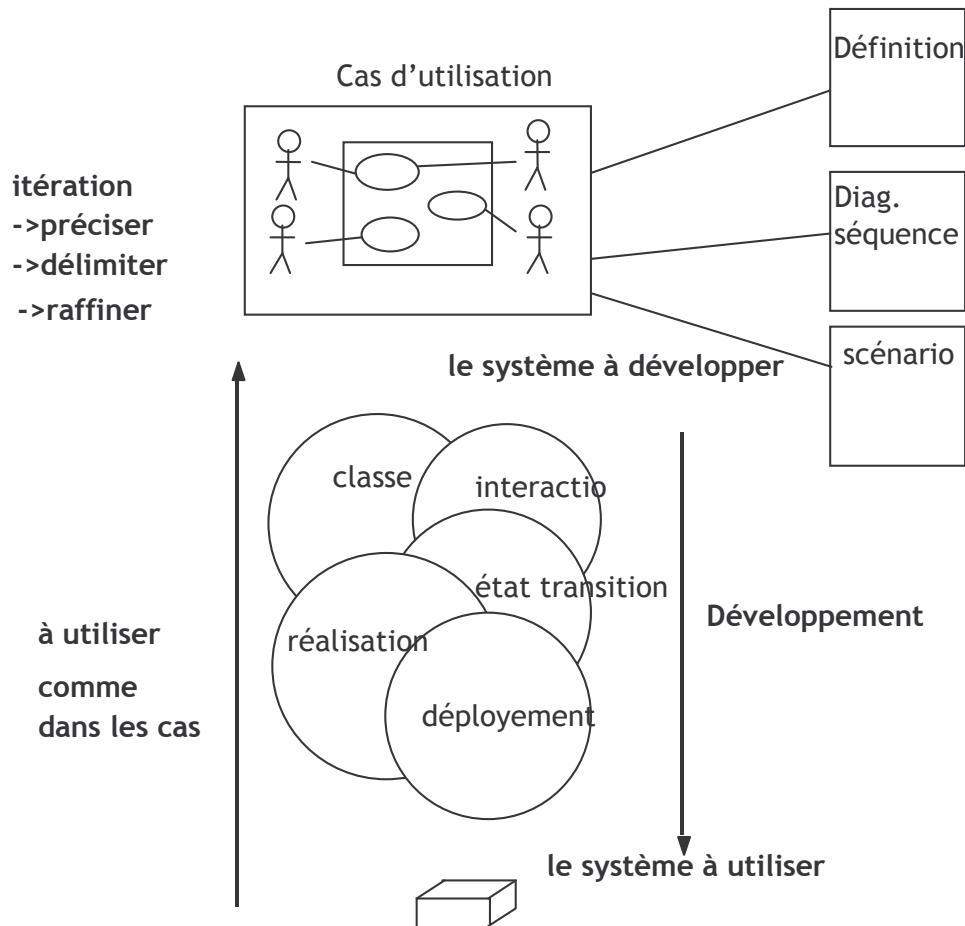


Figure 3-2 : processus de spécification des besoins.

Stéréotypes des cas d'utilisation

Les cas d'utilisation sont définis avec un formalisme simple (ils doivent être facilement appris par les utilisateurs). Il existe trois concepts :

- Le système : Il s'agit du système dont on doit spécifier le comportement ; il établit la frontière entre les acteurs externes au système et les cas d'utilisation interne au système ;
- L'acteur : c'est l'idéalisation d'un utilisateur, d'un processus ou d'un autre système entrant en interaction avec le système à définir ;
- Le cas d'utilisation : c'est une action consistante qu'un acteur peut déclencher en vue d'obtenir un service du système. L'exécution du cas d'utilisation est perçue comme une transaction longue. Le cas d'utilisation est généralement décrit en détail par un scénario.



Figure 3-3 : stéréotypes des diagrammes de cas d'utilisation.

Il existe les relations suivantes entre les concepts des cas d'utilisation :

- Un acteur déclenche un cas d'utilisation ;
- Une personne physique peut être associée à plusieurs acteurs (rôles) ;
- Plusieurs personnes physiques peuvent être associées à un acteur (rôle générique) ;
- Les cas entretiennent des relations entre eux. Un cas peut étendre le comportement d'un autre cas. Un cas peut inclure l'exécution d'un autre cas ;
- Les systèmes peuvent utiliser d'autres systèmes.

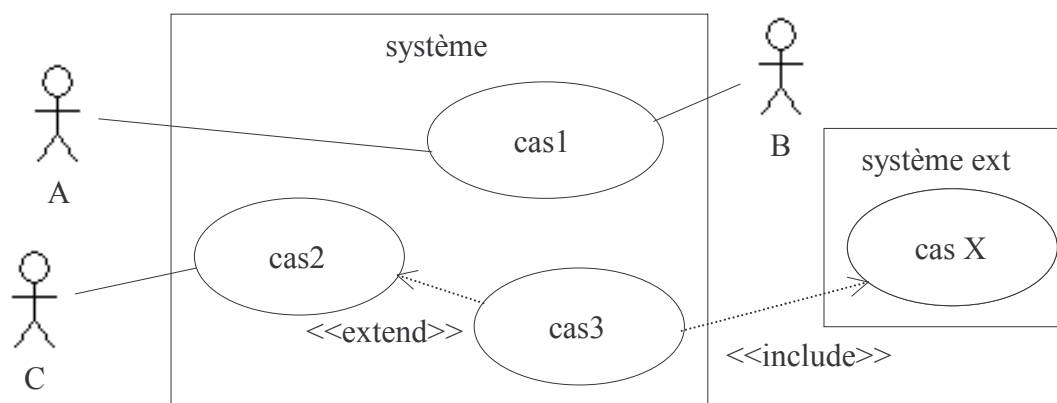


Figure 3-4 : Interaction entre les concepts.

Pour chaque acteur, on donnera une description (texte de quelques lignes) de son rôle générique. On distinguera :

- Les acteurs *principaux* : Ceux qui utilisent les services du système (les clients) ;
- Les acteurs *secondaires* : Ceux qui supportent ou maintiennent le système (par exemple, le postier, l'employé de banque).

On distinguera aussi les composants matériels internes et externes au système :

- Composants internes : Des dispositifs matériels internes au système, qui sont dédiés à l'accomplissement des activités du système (un ordinateur, une caisse enregistreuse, ...)
- Composants externes : Des dispositifs matériels externes au système, qui supportent l'accomplissement des activités du système (un télécopieur, une photocopieuse, un site Web, ...).

L'accomplissement d'une activité nécessite parfois l'utilisation d'un autre système (par exemple, un système de paiement par carte)

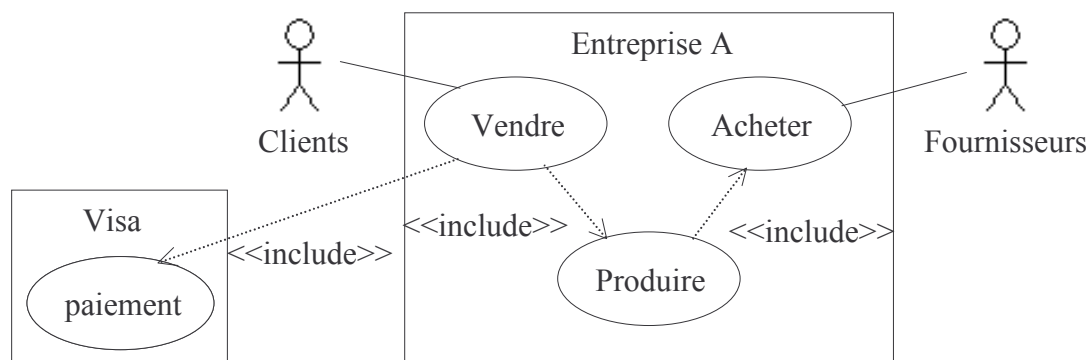


Figure 3-5 : Exemple de diagramme de cas d'utilisation.

On remarquera que, formellement, le cas *Produire* n'est pas une nécessité de l'entreprise A. Lors d'une restructuration de l'entreprise, on peut décider de sous-traiter la production à l'extérieur du système (un nouvel acteur sous-traitant). Cette modification ne serait pas normalement perceptible par le client. En effet, le cas *vendre* demande l'existence d'un produit à vendre, il ne se préoccupe pas de son mode de production.

Cas comparés aux classes

Il nous paraît intéressant de comparer les concepts des cas d'utilisation et ceux des classes.

Un cas d'utilisation est le moule de toutes les interactions entre l'acteur et le système. Un acteur est celui qui déclenche l'exécution du cas d'utilisation et qui interagit avec ce dernier.

En termes de classe, chaque cas est donc une classe ayant une méthode qui correspond au scénario décrit pour le cas d'utilisation. L'acteur est une classe ayant la possibilité d'invoquer les scénarios des cas d'utilisation, de plus il possède des informations et des méthodes lui permettant de répondre aux sollicitations des scénarios.

En termes de cas	En terme de classe
Cas d'utilisation : c'est le moule de toutes les interactions entre l'acteur et le système	Classe (scénario)
Acteur : déclenche le cas et interagit avec le cas	Classe (acteur)
Déclenchement d'un cas (par un acteur)	Objet (instance du scénario)

Interactions avec le système (de l'acteur)	Exécution du scénario (messages échangés)
--	---

Examinons le cas de la Figure 3-6. Le *client* peut déclencher le cas *acheter* auquel est associé un *scénario*. En termes de classes, ceci correspond à une classe *Client* qui peut invoquer une méthode *acheter()* de la classe *casAcheter*. On remarquera que la méthode *acheter()* peut invoquer des méthodes de *Client*, par exemple pour lui demander son numéro de carte de crédit.

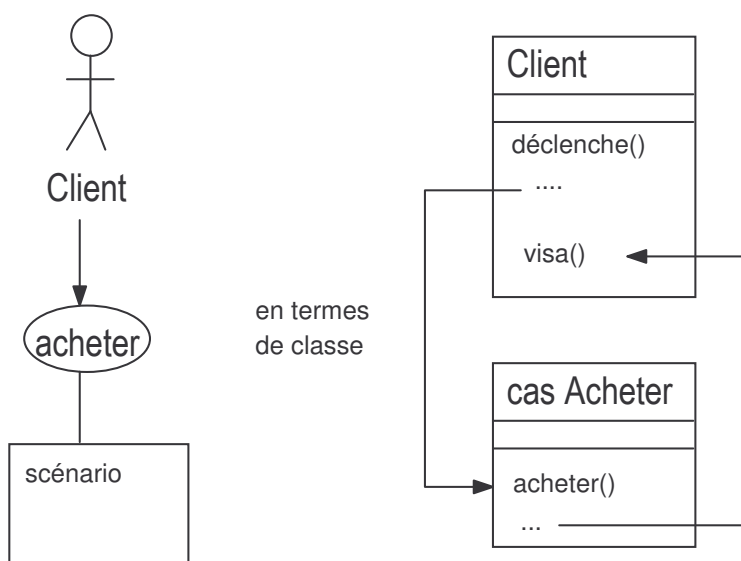


Figure 3-6 : comparaison entre la modélisation des classes et des cas.

En termes d'objets, la trace de l'exécution précédente laissera peut-être les objets de la Figure 3-7. Lors de la modélisation des données, le scénario d'achat produira des données de la commande.

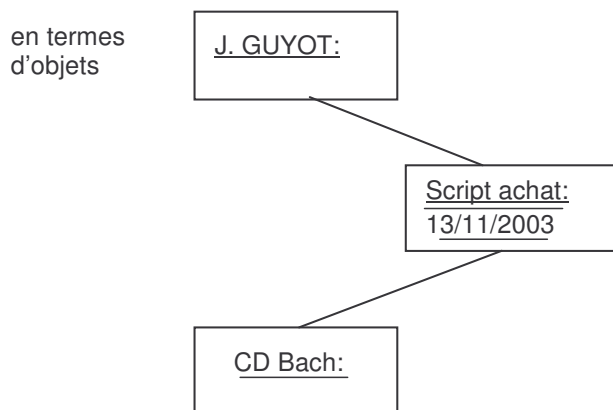


Figure 3-7 : Exemple d'objet restant après l'exécution du scénario.

Ceci nous amène à mieux définir les relations entre nos concepts. Dans la Figure 3-8, on dira que l'acteur A déclenche le cas1. Mais aussi que A crée une instance du scénario cas1. Un cas d'utilisation doit être perçu comme une transaction, c'est-à-dire que son scénario doit être déroulé entièrement jusqu'à une des terminaisons possibles.

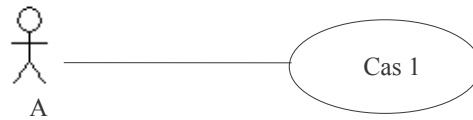


Figure 3-8 : relation de communication.

Un cas peut inclure un autre cas dans son exécution. Dans la Figure 3-9, une instance du cas 1 peut comporter une instance du cas 2. Ceci, nous permet de décomposer des cas en sous-partie pour éviter de les rendre trop complexe ou de partager le scénario d'un cas dans plusieurs autres cas.

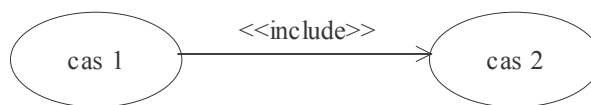


Figure 3-9 : relation d'inclusion.

Un cas peut étendre le comportement d'un autre cas, on dira qu'il le spécialise. Dans la Figure 3-10, une instance du cas 1 étend le comportement du cas 2. Comme dans la généralisation, le cas 1 hérite des comportements du cas 2.

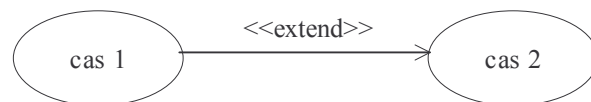


Figure 3-10 : relation d'extension.

Dans l'exemple de la Figure 3-11, nous avons mis en évidence les différentes relations. On peut lire sur le diagramme que :

- Un client déclenche le cas acheter ;
- Le cas acheter peut inclure une vérification d'identité et une demande d'expédition de catalogue ;
- Le cas acheter est étendu en fonction de l'interface utilisé par le client (téléphone, Internet, minitel) ;
- Le client avec téléphone est une extension du client (cette relation n'est pas explicitée sur le diagramme).

Recommandations

Un cas d'utilisation doit procurer une valeur ajoutée à l'acteur qui le déclenche. Si ce n'est pas le cas, pourquoi diable, l'acteur déclencherait-il une activité sans obtenir quelque chose en retour ?

On limitera le nombre d'acteurs déclenchant un cas à un (en créant des rôles génériques si nécessaire).

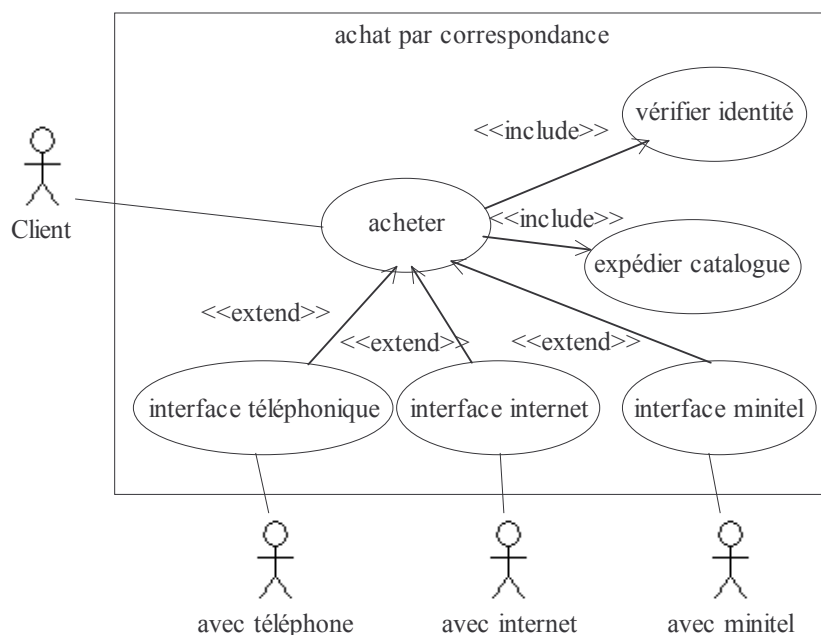


Figure 3-11 : Exemple d'objet restant après l'exécution du scénario.

On limitera le nombre de cas d'utilisation d'un système à 30 (limite proposée par I. Jacobson). Ce nombre peut sembler petit, mais il semble suffisant même pour décrire une multinationale. Sinon on fera usage des relations d'inclusion et d'extension pour rendre plus générique les cas primaires.

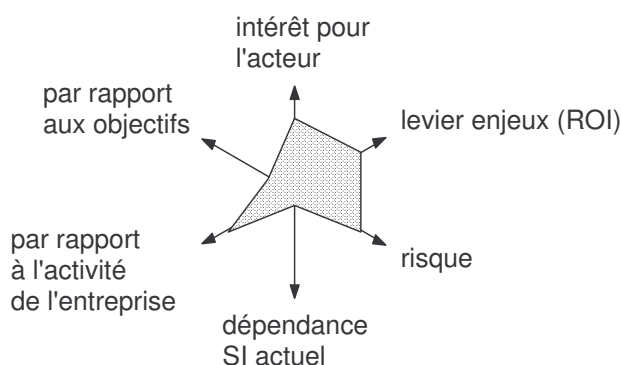


Figure 3-12 : Diagramme en radar des critères de choix.

Dans le cas où l'acteur échangerait des informations avec le système, on définira ce qu'il en fait (créer, sauvegarder, lire, décider, mettre à jour, ..., par exemple mot de passe). On décrira aussi comment on synchronise l'acteur et le système, c'est-à-dire comment :

- L'acteur informe le système (par exemple, un changement d'adresse) ;
- Le système informe l'acteur (par exemple, une limite de crédit).

L'intérêt de la spécification des cas est aussi de pouvoir présenter le système sous différents points de vue :

- Par acteur, tous les cas qu'il déclenche ;
- Par cas, tous les acteurs concernés ;
- Par causalité, enchaînements de séquence d'actions.

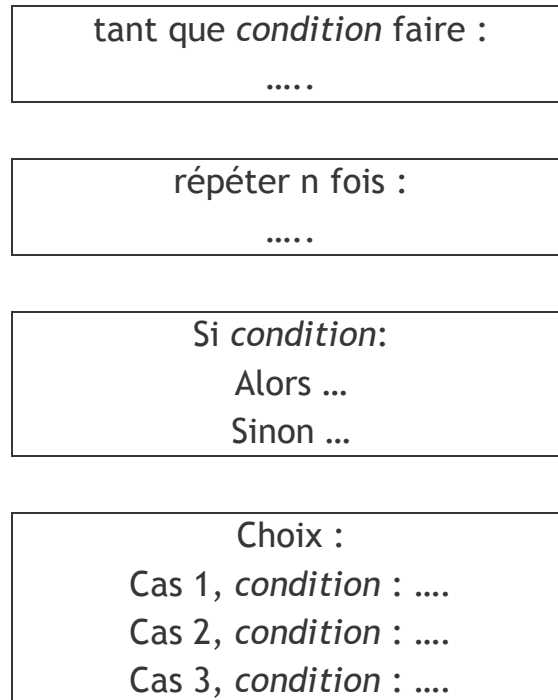
Dans le cas d'une refonte générale du système d'information, les cas peuvent être utiles pour fixer les priorités de mise en œuvre. On peut retenir six critères de choix et établir un diagramme en radar pour représenter chaque cas en fonction de leur importance (Figure 3-12)

Les critères retenus sont :

- L'intérêt pour l'acteur ;
- L'importance et l'intérêt par rapport aux objectifs fixés par l'entreprise ;
- Le levier du retour sur investissement ;
- Le risque (de ne pas le faire, de modifier le cas, ...) ;
- Les enjeux par rapport à l'activité de l'entreprise ;
- La dépendance avec le système d'information actuel (quelles seront les interfaces entre le nouveau et l'ancien)

Voici quelques règles pour les scénarios :

- **Le scénario décrit l'activité** : On définit « Quoi est fait » , on ne précise pas le « comment », Par exemple : la téléphoniste vérifie l'identité du client, on ne dit pas comment ;
- **Rester simple** : les scénarios doivent rester simples, pas trop long. Pour éviter la complexité, décomposer avec les relations <<utilise>> et <<étend>> ;
- **Autonomie** : pas de mélange partiel entre les cas (en dehors des relations <<utilise>> et <<étend>)
- **Style direct impératif** : ne pas laisser d'ambiguïté, pas d'approximation, pas de flou dans les textes de description. Eliminer les très, assez, beaucoup, peu , souvent, en général, exceptionnellement, ... car il cache quelque chose de non spécifié.
- **Le cas est une transaction longue** : il a un début, une fin et on déroule entièrement le scénario.
- **Le cas délimite la frontière du système** : ce qui est attendu du système par les acteurs et les informations externes possédées par les acteurs, ce qui est interne au système, comment il rend les services et donne les informations internes mémorisées par le système.
- **utiliser des itérateurs linguistiques** : dans la description des scénarios, on peut utiliser les constructions suivantes :



Ne pas oublier que c'est une activité des UTILISATEURS ! Les cas ne sont pas parfaits du premier coup : il faut réitérer, raffiner, éliminer les flous et les ambiguïtés. Pendant le processus de spécification, l'utilisateur exprime et découvre ses besoins et ses processus de travail, des choses dont il n'est pas toujours conscient. L'informaticien peut assister l'utilisateur dans la mise en forme du cas. Il peut le forcer à rester générique (abstraction), à vérifier que cette généralité couvre les cas concrets et les exemples mis à disposition. On veillera aussi à rester équilibré dans le niveau de précision, c'est-à-dire de ne pas entrer dans certains détails alors que l'on était pas très précis précédemment. L'utilisateur doit être expert de son domaine. Dans les modélisations importantes, il sera nécessaire de recourir à plusieurs utilisateurs. On aidera l'utilisateur à rester dans le QUOI, en reformulant « Comment il fait » en « Ce qu'il fait ».

Dans les processus itératifs, il y a toujours la question « Quand peut-on s'arrêter ? ». Nous proposons de remplir les critères d'arrêt suivants :

- Complet (par rapport aux besoins) ;
- Compréhensible (pour les développeurs) ;
- Couvrant (explore toutes les alternatives des scénarios) ;
- Intégral (du point de vue des utilisateurs).

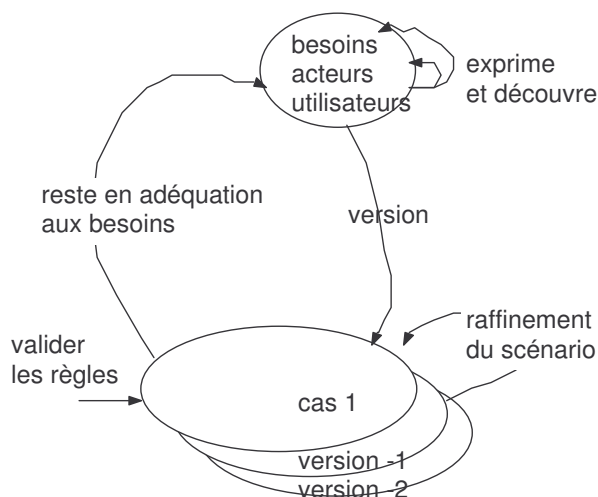


Figure 3-13 : Processus de spécification des cas d'utilisation.

Exemple : le restaurant

Prenons le restaurant comme système. Les acteurs gravitant autour de ce système sont les clients et les fournisseurs. Les clients se font servir un repas et les fournisseurs vendent des ingrédients.

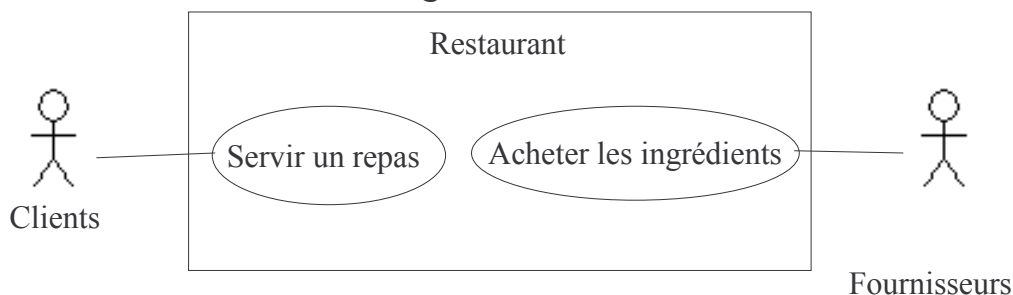


Figure 3-14 : Le restaurant.

Scénario de servir un repas :

1. Le client entre, laisse son manteau au vestiaire.
2. Le client est accueilli par le chef de rang, qui après examen du plan de la salle, le place.
3. Le serveur donne la carte au client.
4. Le client examine la carte et commande ses plats au serveur ainsi que les boissons.
5. Le serveur transmet la commande au cuisinier.
6. Le cuisinier prépare les plats avec les produits de base.
7. Les plats préparés sont apportés par le serveur.
8. Les boissons sont préparées et apportées par le serveur.
9. Le client mange et boit.
10. Le client demande l'addition, le chef de rang la prépare.
11. Le serveur apporte l'addition et encaisse le montant.

12. Le chargé du vestiaire redonne son manteau au client.

13. Le client quitte le restaurant.

En examinant le diagramme de la Figure 3-14, on perçoit l'absence de liens entre les activités des fournisseurs et celles des clients. Dans la figure suivante, nous avons corrigé ceci en ajoutant deux cas :

- C'est le cas « Préparer les menus » qui produira la carte des menus et qui déclenchera la commandes d'ingrédients ;
- Préparer les repas qui est déclencher à chaque occurrence de repas.

On constate que la préparation des menus régule le fonctionnement du système. En effet, la carte des menus doit correspondre aux ingrédients achetés. Cette carte est celle qui sera proposée aux clients. La préparation d'un plats concernera donc un plat de la carte et il se fera avec les ingrédients commandés.

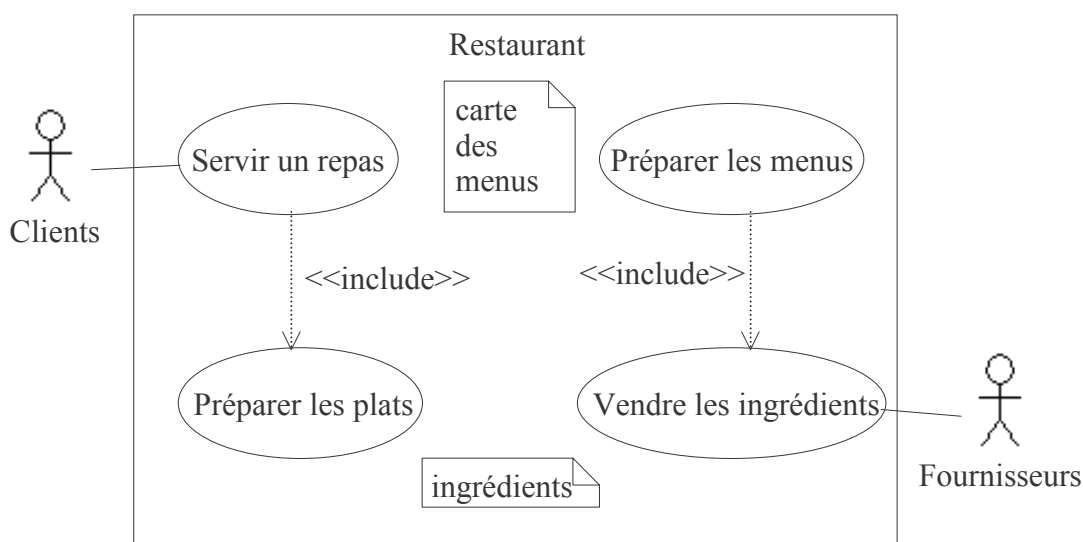






Figure 3-15 : Le restaurant : plus de détail

Extension pour les processus

Pour spécifier certains processus, il est parfois plus pratique de représenter graphiquement le flux. Ceci est possible avec les stéréotypes suivant :

	Acteur (client)
	Interface (mise au vestiaire, prise de commande)
	Entité (carte, plat, boisson, addition)
	Contrôleur (préparation des plats, préparation de l'addition)

L'acteur est un agent externe au système. Il interagit avec le système à travers les interfaces. Les entités sont produites et consommées par les interfaces et les contrôleurs. Les entités sont des objets statiques physiques ou abstraits. Les contrôleurs sont processus qui transforment les entités. En résumé :

- **Interface** : Afficher, demander, collecter
- **Entité** : Information, matière
- **Contrôleur** : Transformer, contrôler

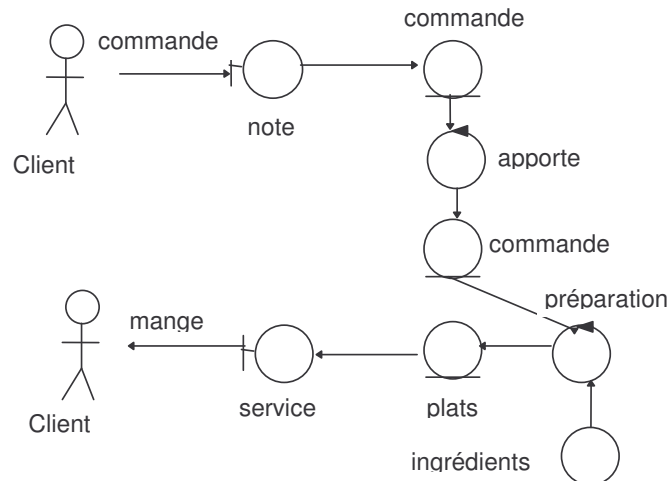


Figure 3-16 : correction partiel du scénario du restaurant

En savoir plus sur UML

Nous avons vu les deux premiers modèles d'UML, et nous complétons notre présentation d'UML par un survol des autres.

UML c'est six modèles:

- **Modèle des cas d'utilisation** : exprimer les besoins des utilisateurs ;
- **Modèle des classes** : exprimer la structure statique des objets ;
- **Modèle des états** : exprimer la structure dynamique des objets ;
- **Modèle des interactions** : exprimer les interactions entre les acteurs et le système, entre les objets ;
- **Modèle de réalisation** : exprimer le regroupement et les unités logiques de réalisation ;
- **Modèle de déploiement** : exprimer la répartition physique des éléments du système.

Un modèle est la description complète depuis un point de vue particulier (celui des objets, des interactions, etc.). Pour exprimer un modèle, on utilise des diagrammes :

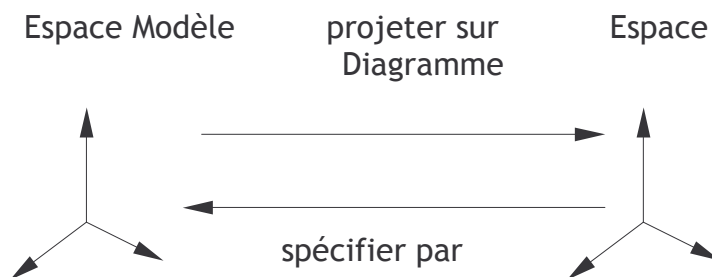


Figure 3-17 : relations entre diagrammes et modèles en UML

Diag. Modèles	Cas d'utilisation	objets	Collaboration	séquences	Classes	état transition	Activités	Composants	déploiement
Cas d'utilisation									
interaction									
classes									
état transition									
réalisation									
déploiement									

UML propose 9 types de diagrammes :

- **de cas d'utilisation** : Acteurs, activité-service rendu par le système ;
- **de classes** : structure statique, classe, association, méthode, héritage ;
- **d'objets** : exemples d'objet, instance, valeur, lien ;
- **de séquences** : ordonner messages entre acteurs / objets ;
- **de collaboration** : échanger messages entre acteurs / objets ;
- **d'états-transitions** : modification des états (classe/objet) par les messages ;
- **d'activité** : structuration des actions (alternative, séquence, parallélisme, ...) ;
- **de composants** : regroupement module, package ;
- **de déploiement** : décrire les objets physiques .

Un modèle peut être projeté sur plusieurs diagrammes. Par exemple, le modèle des cas d'utilisation s'exprime avec un diagramme particulier qui lui est propre, mais on peut aussi utiliser un diagramme de séquence pour clarifier les interactions entre les acteurs.

les éléments d'UML

Les éléments sont les atomes de base des modèles. Une modélisation est un ensemble d'éléments regroupés dans un paquetage. Un paquetage peut

contenir d'autres paquetages. La complexité est donc gérée par un système d'arborescence de paquetage. Un élément a deux représentations, l'une destinée à sa modélisation et l'autre à sa visualisation.

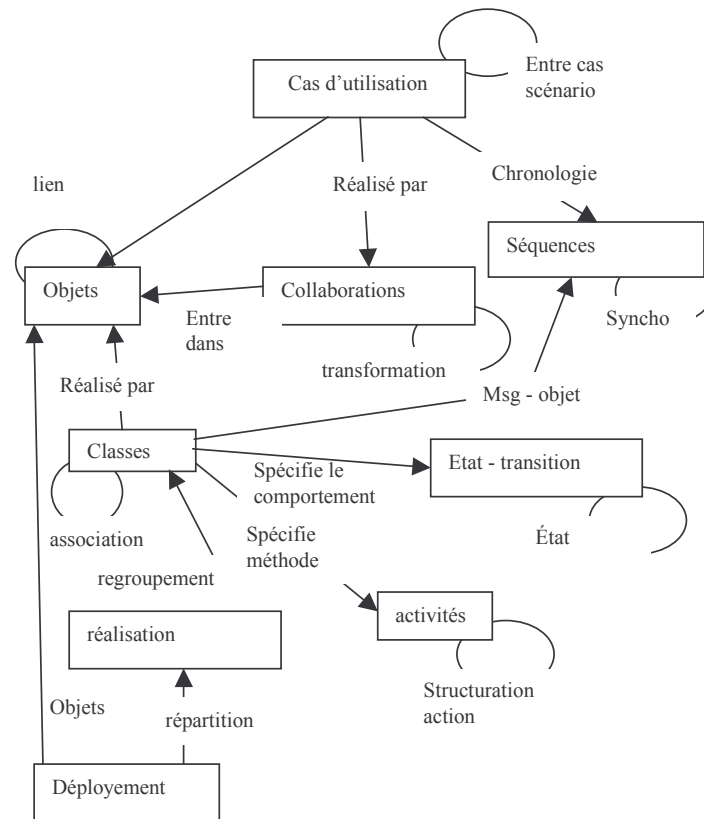


Figure 3-18 :relations entre diagrammes

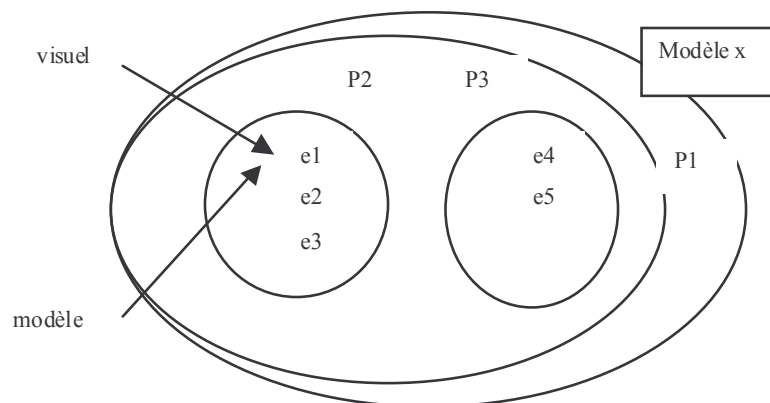


Figure 3-19 :Un modèle est un arbre de paquetages.

Tous les éléments d'UML ont la forme de la Figure 3-20. C'est à dire que chaque élément a un nom (par exemple Article), chaque élément est associé à un stéréotype (par exemple classe), chaque élément peut avoir une liste d'attributs (par exemple nombre d'objets = 1000) chaque élément peut être associé avec d'autres éléments (par exemple Article peut être en relation avec la classe LigneDeCommande). Chaque élément peut avoir une note (un

commentaire sur l'objet) et finalement une expression en OCL (Object Constraint Language) peut être associée à l'élément.

Cette construction systématique des éléments dans UML en fait un système ouvert. Il est donc possible de créer de nouveaux stéréotypes.

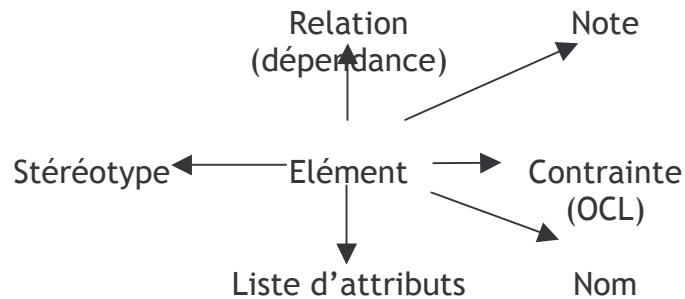


Figure 3-20 : Structure de base d'un élément

UML : un système ouvert

UML s'impose comme un standard de l'informatique et de la communication. UML est avant tout une notation et non une méthode. Il ne dicte donc pas le processus d'élaboration de la modélisation. Il est donc ouvert aux sociétés de service, éditeurs de logiciel, méthodologues qui peuvent y ajouter une valeur propre. Plus généralement, on peut ajouter simplement des stéréotypes, mais il est aussi envisageable d'ajouter des nouveaux types de diagrammes ou même des modèles.

La description du système d'information d'une entreprise peut être mémorisée dans un dictionnaire structuré avec UML. Et autour de ce dictionnaire, on trouvera les différents outils : de modélisation graphique, de conduite de projets, de génération automatique de code, etc.

Exercice

- 1) Décrire complètement le flux des processus du restaurant.
- 2) Prendre un autre type de lieu de vente de nourriture tel que :
 - la cafétéria de l'université ;
 - un Mac Donalds ;
 - un Kebab.

Décrire le cas d'utilisation « prendre un repas », le flux des processus et montrer le *reengineering* du processus en termes d'économie par rapport au scénario classique du restaurant. (Dans ces études de cas, une pile de plateau, une poubelle peuvent être des interfaces au système !)

4. Sortir de la confusion

Modéliser, c'est accepter de prendre le risque de montrer en tant que débutant une certaine confusion. Il n'est pas aussi évident qu'il n'y paraît de décider si quelque chose est une classe, un attribut, une méthode ou une association. Pire, dans un contexte on choisira un attribut et dans l'autre une classe. A priori, il n'y a pas de modélisation fautive, il y a seulement des modélisations qui ne reflètent pas la réalité. La seule façon de « sortir de la confusion » c'est de modéliser, de prendre un crayon, une feuille et d'accepter les règles d'un jeu qui consiste à représenter la réalité dans des boîtes reliées par des lignes.

Exercices

La compagnie TT³ est le champ d'application qui nous servira de fil conducteur pour les exercices. Dans certains cas, nous ajouterons des exercices spécifiques. Dans les annexes, vous trouverez d'autres champs d'application, dont certains avec des corrigés.

Prendre les domaines suivants et décrire les classes et les associations:

- CD-Musique
- K7-Vidéo
- Composant (d'une voiture)
- Université
- Bibliothèque
- Agence de voyage
- ...

Avant de commencer, nous allons faire un mini-cas d'utilisation pour fixer un contexte de modélisation!

Exercice sur RECETTE

Faire le diagramme de classes d'un système qui devrait permettre de mémoriser les ingrédients nécessaires à une recette de cuisine.

Exercice sur TT³

Faire le diagramme de classes de TT³.

Enoncé de TT³

La compagnie de transport TT³ (Tout Transport, Tout Type, Tout Temps) a choisi de se diversifier. Son domaine d'activité principal est lié aux taxis et

aux transports de groupes. Cette diversification récente a entraîné des problèmes de gestion et la direction a compris que les difficultés étaient en partie incriminables à l'obsolescence de son système d'information. La décision d'informatiser a été prise. Le PDG de la compagnie, sur les conseils d'un ami, a décidé de procéder à une modélisation du champ d'application avant d'acheter les ordinateurs personnels que lui réclament les gestionnaires de la compagnie (proverbe du routier: "Ne pas mettre la charrue avant les bœufs").

Le champ d'application couvert par les impératifs de la gestion de cette compagnie de transport concerne le parc des véhicules, son entretien, l'administration des chauffeurs et de leur emploi du temps, la gestion des appels des clients à la centrale téléphonique.

Le texte qui suit est une description du champ d'application tel qu'il apparaît à la suite d'une réunion avec les différents cadres de la direction (les mots en style gras sont les constituants qui sont retenus dans la modélisation).

Compte rendu de la réunion:

Pour le chef mécanicien, un véhicule est identifié par un numéro de châssis **noChassis**. Chaque véhicule possède un numéro de plaque **noPlaque** ainsi qu'une date de mise en service **miseEnService**. Le parc de véhicules est divisé en plusieurs types. Un type est connu par le modèle **modèle**, par exemple: Mercedes 300. Un véhicule ne peut bien entendu appartenir qu'à un seul modèle. La description d'un modèle permet de connaître le nombre de personnes pouvant prendre place dans les véhicules **nbPlaces** de ce modèle; le type de carburant **typeCarburant** consommé; la catégorie de permis **catégorie** que doit posséder le chauffeur; le type de boîte à vitesses **automatique** et le poids du modèle **poids**.

Un des objectifs de ce système d'information est de surveiller la consommation journalière en carburant des véhicules. Ainsi une augmentation de cette consommation sera le signe que le moteur nécessite un réglage. A chaque fois que le chauffeur fait le plein, il remplit une fiche indiquant le numéro de plaque, la date **noJour**, le nombre de kilomètres roulés depuis le dernier plein **kilometrage**, la quantité **litres** et le type de carburant mis dans le réservoir.

L'équipe des mécaniciens s'occupe de l'entretien. Ce qui est mémorisé sur l'entretien des véhicules est donné par une description **description** associée à une date et un numéro de châssis.

Le responsable de la planification a besoin des informations suivantes pour établir l'emploi du temps de ces chauffeurs. Il dispose jusqu'à maintenant de fiches sur les chauffeurs où l'on trouve le numéro du chauffeur **noChauffeur**, son nom **nom**, son prénom **prénom**, son adresse **adresse**. La fiche contient aussi un emplacement où sont notées les catégories de permis que possède le chauffeur.

L'emploi du temps des chauffeurs et des véhicules est défini sur un grand tableau qui occupe une paroi entière de son bureau. Les numéros des châssis des véhicules se trouvent sur les entêtes de ligne. Les entêtes de colonne sont des dates subdivisées en trois tranches horaire **trancheHoraire**. Un seul numéro de chauffeur est inscrit dans une case du tableau.

Le responsable de planification doit faire attention à ce que les modèles conduits par les chauffeurs soient compatibles avec les permis qu'ils possèdent.

Pour gérer les appels téléphoniques, le central téléphonique est équipé d'un système qui permet aux chauffeurs de donner leur position en indiquant la zone **noZone** dans laquelle ils sont inoccupés avec un véhicule. Lorsqu'un client demande un taxi, il suffit de lui assigner un véhicule dans sa zone de prise en charge. Si aucun taxi ne se trouve dans la zone, il faut trouver le plus proche véhicule. Afin d'effectuer cette recherche d'une manière optimale, il existe des tabelles, qui, pour chaque heure de la journée **heure**, indique le temps de parcours **tempsParcours** d'une zone **zoneDe** à une autre **zoneA**. On encode ainsi la variation de fluidité du trafic dans la ville au cours de la journée.

La compagnie TT³ est répartie dans la ville en stations où sont garés les véhicules. Une station possède un numéro de station **noStation**. La station se trouve dans une zone. Un véhicule est associé à une seule station. Un chauffeur est aussi assigné à une et une seule station. C'est à cette station qu'il vient prendre et rendre son véhicule.

Correction RECETTE

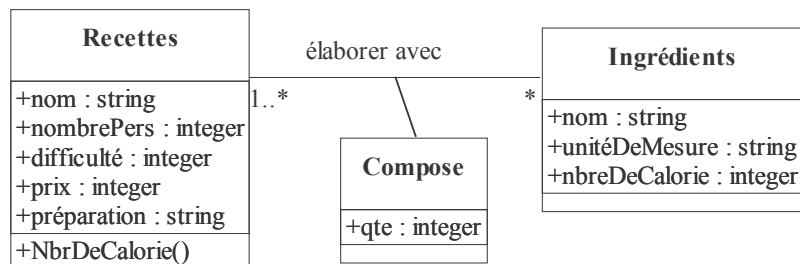


Figure 4-1 : Diagramme de classes (recettes et ingrédients)

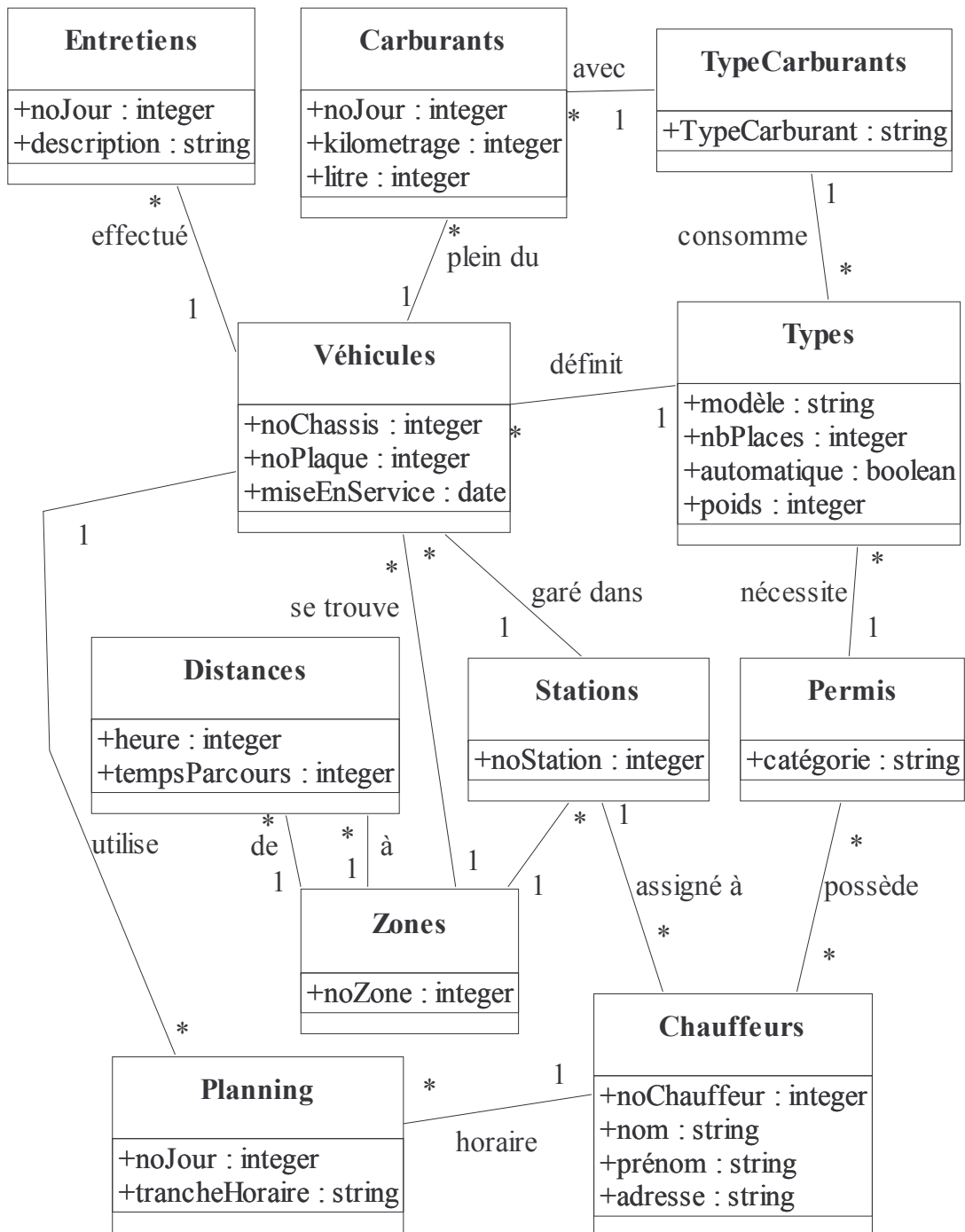
Correction TT³

Figure 4-2 : Modélisation des classes de TT3

5. Encore une introduction

"le pluralisme des théories et des conceptions métaphysiques n'est pas seulement important pour la méthodologie, c'est aussi un élément essentiel dans une perspective humaniste." (Paul Feyerabend - Contre la méthode)

Dans la première partie, nous avons vu comment spécifier un système. Il existe plusieurs possibilité pour implémenter le système. Nous nous restreignons ici au cas où les informations devraient être mémorisées dans une base de données. Les bases de données relationnelles sont au centre des systèmes d'information modernes. La standardisation du langage SQL en 1987 et la mise en réseaux des postes de travail mettent à disposition de tous les données de l'entreprise, pour être analysées, mises en page, médiatisées. Les tableurs et des outils graphiques offrent des facilités de connexions aux bases de données à travers ODBC (Open Data Base Connectivity). Les sites Web sur Internet sont généralement construits autour de bases de données.

Pour effectuer ce passage du fichier à la relation, du programme à la requête, deux éléments sont indispensables: les concepts de base du modèle relationnel et sa mise en application avec le langage SQL.

Une modélisation relationnelle est la concrétisation d'une modélisation de classe. Le modèle des classes est un outil pour la conception. Le modèle relationnel est un outil pour la réalisation du système. La théorie présentée est la clé de lecture de cette modélisation, elle permet d'en évaluer les adéquations et les limites.

SQL est le langage de gestion de bases de données relationnelles, il permet l'interrogation et la manipulation de ces données. Son universalité permet d'accéder aux systèmes de gestion de bases de données (SGBD) de la plupart des constructeurs, de l'ordinateur central d'une multinationale au portable d'un représentant.

Seule la compréhension du modèle relationnel est garante d'une interprétation sémantique correcte des données mémorisées dans un SGBD.

Malgré que la plupart des SGBD propose des interfaces graphiques pour l'interrogation, la connaissance syntaxique de SQL est nécessaire pour une exploitation efficace du SGBD. Ici, toutes les règles du langage SQL sont représentées par des diagrammes syntaxiques, une forme graphique de spécification des langages.

Cette partie de l'ouvrage est élaboré autour des trois problématiques principales du modèle relationnel: l'interrogation - comment interroger une

modélisation pour avoir une réponse fidèle au champ d'application; les modifications - comment modifier le contenu de la base pour continuer à refléter la réalité modélisée; les règles d'intégrité - comment intégrer dans la modélisation les contraintes de gestion perçues dans la réalité. Ces problématiques sont abordées séparément, puis simultanément, dans le cadre des formes normales.

Finalement, nous abordons le thème concernant l'optimisation des requêtes, la sécurité des données et un exemple informatique de représentation physique des relations.

Des exemples divers illustrent la théorie et la pratique de SQL. Une étude de cas corrigée, présentée comme exercice, accompagne le lecteur, à la fin des chapitres.

Des fichiers aux relations

La première application d'un traitement automatisé de l'information est celle du recensement américain de 1890 [GOL72], Herman Hollerith met au point une machine électromécanique capable de trier et de compter des cartes en fonction des trous qui y sont présents. Déjà la standardisation est présente, la carte est au format du billet d'un dollar (la fabrication des cartes est donc assurée). 288 trous peuvent être effectués. Lors du recensement, les critères retenus sont traduits avec des trous sur la zone réservée de la carte correspondant à ce critère pour un individu. Chaque critère demandera un milliard de trous, pour décrire les 63'000'000 d'habitants. Mais pour la première fois, il est possible de répondre à des questions telles que:

- Le nombre d'enfants nés ;
- Le nombre d'enfants vivants ;
- Le nombre de familles parlant l'anglais.

La méthode et les machines de Hollerith furent un succès. On y retrouve les ingrédients d'une base de données actuelle, les entités, les propriétés pertinentes à retenir, les modèles physiques, les requêtes.

Pendant un demi-siècle la technique du traitement des cartes perforées (la mécanographie) sera l'instrument dominant du traitement de l'information. Durant cette période, l'ordinateur sera exclusivement un outil de calcul. En 1950, dans "The federal computing machine program", [REE50] Mima Rees parle d'une utilisation possible d'un ordinateur ayant pour spécificité d'accepter un grand nombre de données en entrée, d'effectuer peu d'opérations et de produire une grande quantité de résultats. C'est le début d'une ère où l'ordinateur n'est plus un outil destiné aux scientifiques et aux militaires mais comme un outil pouvant être utilisé dans la gestion de l'information. L'UNIVAC I (1951) sera la machine civile type de gestion, équipée d'imprimantes à haut débit et supportant jusqu'à dix lecteurs de bandes magnétiques. Durant la même décennie, on trouve la BIZMAC, conçue par RCA, équipée de 200 lecteurs de bandes qui préfigure ce que sera une

grande base de données (pour un exposé complet lire "une histoire de l'informatique " de P. Breton [BRE90]).

Parallèlement au développement des machines, nous avons celui des langages informatiques. Avec le langage FLOW-MATIC (1955) de la compagnie Univac, naît le premier langage destiné à l'informatique de gestion. Ce langage demandait déjà une description séparée des données et des instructions. Ce concept fut ensuite repris lors de la définition du langage COBOL (1960).

La création d'applications programmées en COBOL utilisant des fichiers de données mettra en évidence deux difficultés dues à la structure physique des données qui doit être nécessairement connue par le programmeur lors de l'écriture du programme. La première difficulté résidait dans le rapport de dépendance entre la représentation de l'information et le support physique des informations (les bandes, les tambours et ensuite les disques). Celle-ci rendait difficile le transport des données d'une installation à une autre. La deuxième difficulté était due à la redondance des fichiers ayant des structures différentes mais des informations communes, ce qui empêchait toute centralisation et partage des informations. Chaque application possédait ses propres fichiers et ses propres programmes. Les incompatibilités de structure limitaient la concertation, le partage des données et le travail en équipe qui sont nécessaires pour réaliser les différentes applications travaillant sur les données communes à une entreprise.

Ces faits furent déterminants dans la recherche de concepts favorisant l'indépendance des traitements par rapport aux données. Les bases de données apportent une solution à ces problèmes en proposant un langage de description des données et un langage de manipulation des données; les programmes peuvent être alors écrits indépendamment de la structure physique des données. Trois modèles furent largement utilisés:

- Le modèle hiérarchique ;
- Le modèle réseau ;
- Le modèle relationnel.

Dans un fichier (1950-..), les données d'un même objet sont définies par un enregistrement physique, l'ensemble des enregistrements physiques constitue le fichier. La description de l'enregistrement est implicite et elle est codée dans les programmes qui utilisent le fichier. Si l'on modifie la structure du fichier, on est donc obligé de modifier les programmes. Les systèmes de base de données contournent cet inconvénient majeur en **rendant explicite la structure des données**, rendant ainsi indépendants les programmes de la représentation physique. Les SGBD possèdent donc tous une description explicite de la structure de donnée, mais il existe plusieurs façons de décrire les liens existant entre les objets du champ d'application; on parle alors de modèle de données.

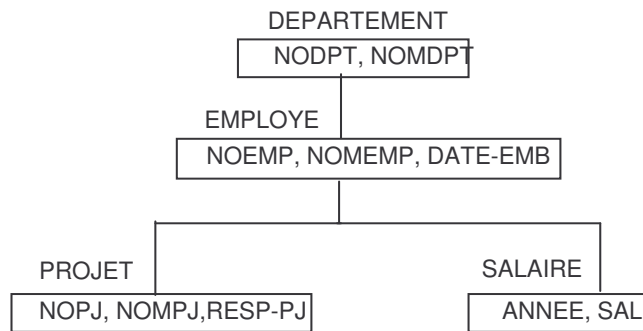


Figure 5-1 : Exemple avec le modèle hiérarchique

Dans le modèle hiérarchique (1965-...), les liens existant entre les objets sont strictement arborescents. Un département contient des employés; un employé travaille dans des projets; un historique des salaires correspond à un employé.

Dans l'exemple, on remarquera que si une feuille de l'arbre doit être utilisée dans une autre arborescence il faudra la dupliquer. Par exemple, pour trouver tous les employés d'un projet, il faut soit parcourir toutes les feuilles de l'arbre ou bien créer une autre arborescence dont la racine est PROJET. Dans le premier cas, la recherche est longue car exhaustive et séquentielle. Dans le second, la seconde arborescence crée des redondances d'information qui seront coûteuses en programmation. IMS est l'exemple typique de SGBD hiérarchique.

Le modèle réseau (1965-..) est une extension du modèle précédent, les liens entre objets peuvent exister sans restriction.

Pour retrouver une donnée dans une telle modélisation, il faut connaître le chemin d'accès (les liens), ceci rend encore les programmes dépendants de la structure de données. Le noeud EMP-PROJET permet de trouver les projets d'un employé et les employés d'un projet. Ceci représente une amélioration car cela supprime la redondance.

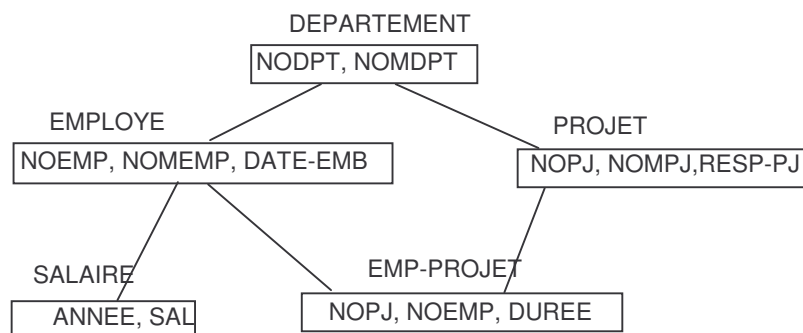


Figure 5-2 : Exemple avec le modèle réseau

Cependant s'il n'existe pas le chemin (lien) entre deux noeuds, nous retombons dans le parcours séquentiel exhaustif. Par exemple, pour la question "quel est le salaire moyen des chefs de projets" il manque le lien entre les chefs de projets et les employés. Les logiciels IDMS, TOTAL, MDBS-III étaient des SGBD de ce type.

Le modèle relationnel (1970-..) est basé sur la notion de relation. Une relation est un ensemble de n-uplets (n est fixe) qui correspondent chacun à une propriété de l'objet à décrire.

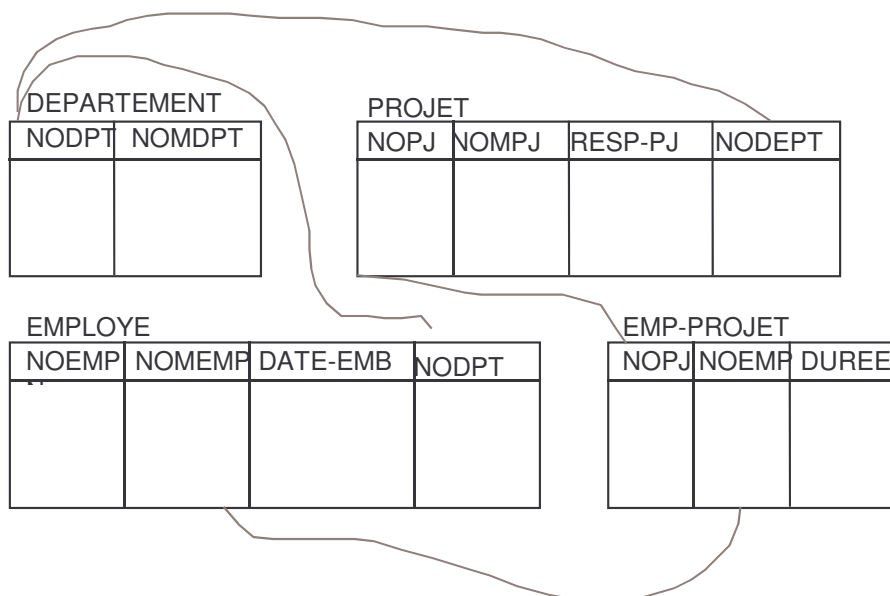


Figure 5-3 : Exemple avec le modèle relationnel

DEPARTEMENT, PROJET, EMPLOYE, EMP-PROJET sont des relations. Les lignes dessinées en pointillé sont les liens sémantiques entre les relations, mais ici il n'est plus nécessaire de décrire explicitement les liens, les chemins d'accès sont indépendants de la modélisation.

On trouvera un historique complet des modèles hiérarchiques et réseaux dans [FRY76] et dans le livre de C. Delobel et M. Adiba [DEL82], un exemple est traité dans les trois modèles.

L'histoire des bases de données est essentiellement une histoire de l'indépendance des données et des traitements, elle se poursuit actuellement avec celle des bases distribuées où le demandeur d'information se trouve devant une base de données virtuelle dont la localisation physique (processeurs) est distribuée sur un réseau.

Problématique des bases de données

"Une base de données est un ensemble structuré de données enregistrées sur des supports accessibles par l'ordinateur pour satisfaire simultanément plusieurs utilisateurs de façon sélective et en un temps opportun" [DEL82]

Cette définition articule les différents éléments que nous allons traiter :

Les **données** de la BD représentent des faits, des activités ou des événements de l'entreprise. La BD doit être considérée comme la mémoire de l'entreprise. De ce fait, le contenu de la BD doit être :

- pertinent (données utiles) ;
- fiable (données cohérentes et justes dans le sens de vrai par rapport au champ d'application) ;
- utilisable (accessible aux traitements).

Les objets mémorisés dans la BD possèdent des propriétés communes, permettant ainsi de les regrouper par type d'objet. La structure de la BD est le *plan* qui permettra d'interpréter les données stockées. La gestion de la base de données se fait par rapport à cette **structure**.

La base de données peut comporter quelques milliers de caractères pour une petite base sur micro-ordinateur, donc elle peut être stockée sur disquette ou bien elle est constituée de plusieurs milliards de caractères et elle doit être stockée sur des unités de disques d'un ordinateur. Malgré les différences de tailles, les techniques et les concepts utilisés sont similaires. La base de données est indépendante de son **support**.

Les données mémorisées sont appelées à être utilisées par différents services de l'entreprise, avec des **utilisateurs** appartenant principalement à trois catégories:

- Les informaticiens; gérant la BD, concevant les nouvelles applications ;
- Les utilisateurs *avertis*; sachant faire des requêtes d'interrogation pour leurs propres besoins qui ne sont pas spécifiables (les gestionnaires) ;
- Les utilisateurs *naïfs*; dont la tâche est entièrement spécifiable (répétitive), saisie de l'information .

La BD est surtout utilisée en interrogation, le langage d'interrogation est donc un élément essentiel du système, il doit être:

- facile à apprendre (pour les utilisateurs avertis) ;
- masquer la structure physique de la base de données (Index, paramètres, ...) ;
- avoir une sémantique claire (comprendre le sens de la question et de la réponse).

Par temps opportun, on entend que si l'information existe dans la BD, alors on peut l'obtenir dans un délai raisonnable (court si l'on travaille de manière interactive (guichet de banque) ou à temps (pour prendre une décision).

Les objectifs de l'organisation

L'utilisation des bases de données peut aussi se considérer sous l'angle organisationnel. La conception d'une application BD est une opération demandant des ressources financières (achat des ordinateurs, logiciel de gestion de BD, ...) ainsi que des ressources humaines (concepteur, programmeur, opératrices de saisie, ...), il est donc important que l'organisation examine les avantages qu'elle doit en retirer. Les arguments suivants peuvent motiver l'organisation:

- simplifier une tâche de l'entreprise ;
- augmenter la qualité d'un service ;
- permettre une meilleure prise de décision ;
- rentabiliser les ressources matérielles et humaines.

En résumé, la BD doit conserver les données stratégiques de l'entreprise pour que l'on puisse les utiliser d'une manière optimale. Les objectifs de l'entreprise peuvent s'échelonner en plusieurs étapes ou bien évoluer dans le

temps, d'où l'importance d'une conception et d'un système de gestion de base de données (SGBD) autorisant les évolutions et les modifications.

Les objectifs de l'organisation délimitent un **champ d'application** dans la réalité dont la BD est le reflet. L'établissement d'un schéma directeur [HOU88] ou les cas d'utilisation [JAC94] permettent de cerner les objectifs et le champ d'application. Ses éléments finaux sont:

- les traitements à effectuer ;
- les requêtes d'interrogation à exécuter ;
- les données nécessaires à mémoriser ;
- les règles d'intégrité à respecter.

Dans l'exemple HOTEL, qui nous accompagnera dans cet ouvrage, plusieurs domaines d'application sont possibles et interdépendants. La détermination de l'appartenance d'un objet à un domaine se fait en examinant les objectifs fixés par l'organisation.

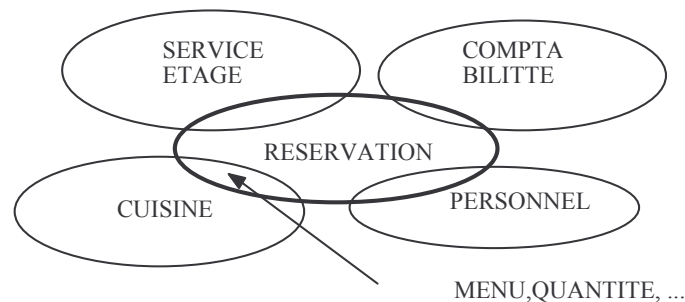


Figure 5-4 : Domaines de notre exemple Hôtel

Les **traitements** de l'application sont définis par toutes les modifications envisagées sur les données de la BD. Trois types d'actions primitives sont possibles:

- La **création**; un "objet" nouveau apparaît dans la réalité et celui-ci est dans le champ d'application, donc il doit être enregistré dans la base de données (un nouveau client) ;
- La **mise à jour**; un "objet" déjà enregistré dans la BD se modifie et ceci doit être reporté dans la BD (changement dans la quantité stockée d'un article) ;
- La **destruction**; un "objet" enregistré dans la BD sort du champ d'application et doit donc être éliminé de la BD (changement d'année comptable, un salarié quitte l'entreprise).

Les traitements permettent de modifier la BD pour tenir compte des changements intervenant dans la réalité du champ d'application.

Pour les **interrogations**, il s'agit d'identifier les besoins de chaque utilisateur devant accéder la BD, en se posant les questions suivantes:

- Quelles sont les informations de la BD nécessaires à l'accomplissement de la tâche de cet utilisateur (le magasinier, la réceptionniste de l'hôtel)?

- Quelle est la fréquence de ces questions, le temps de réponse exigé? Le couple (100 req/jour, 15 secondes) sera examiné différemment de celui (1 req/mois, dans la matinée).
- Qui peut examiner et modifier les informations? Ce point concerne le degré de confidentialité et de sécurité de chaque information.

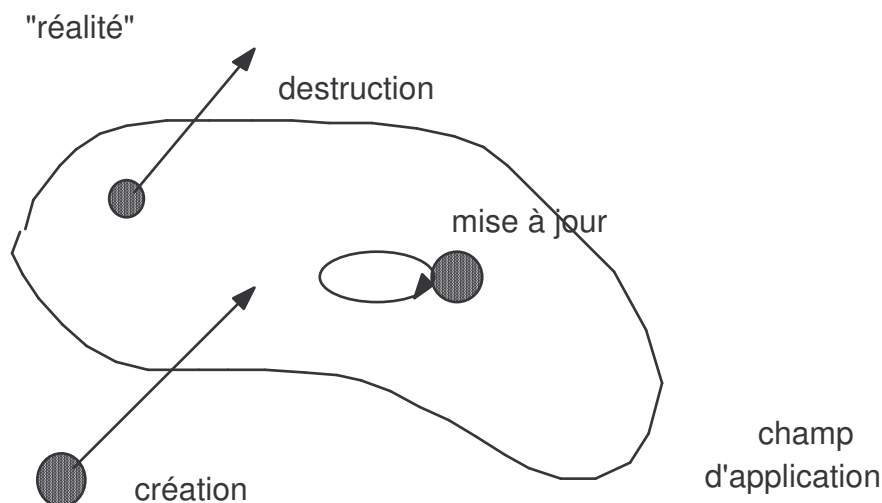


Figure 5-5 : Modification par rapport à la frontière du champ d'application

Les **données** à mémoriser dans la BD sont celles définies par le champ d'application. Les traitements les créent, les mettent à jour et les détruisent. Les requêtes d'interrogation les utilisent en lecture pour répondre aux utilisateurs. Nous avons vu que c'est dans le cadre de la définition du champ d'application que la sélection des données s'effectue. Le choix des propriétés à enregistrer dans la BD doit être nécessaire et suffisant pour exécuter les traitements et répondre aux requêtes d'interrogation:

- **nécessaire:** à court terme, pour être aussi efficace que le système remplacé et à moyen terme, pour répondre à de nouvelles questions (que l'on évite de se poser car dans un système manuel, elles sont trop onéreuses).
- **suffisant:** pour éviter de mémoriser des informations qui seront peu ou pas utilisées.

Pour une personne, nous pouvons la définir par exemple: "nom, prénom, taille, profession, adresse, numéro de téléphone, revenu, poids, appartenance politique, sports pratiqués, état civil, nombre d'enfants, ...". Chacune de ces propriétés a un sens dans un contexte bien défini, par contre elles sont inutiles dans un autre.

Chaque information (comptée en caractères) a un prix calculé avec les coûts suivants:

- coût de saisie (opératrice, poste de saisie)
- coût de stockage (disques, bandes d'archivage)
- coût de manipulation (taille ordinateur ...)

Les **règles d'intégrité** reflètent les règlements de l'organisation, le "bon sens" de la réalité. On peut les exprimer:

- **sur les données;** le fait que la BD respecte les règles d'intégrité permet d'assurer une certaine cohérence des données, donc assure aux utilisateurs des informations de qualité (Une chambre n'est réservée qu'une fois, les quantités du stock sont positives, les clients ont plus de 18 ans, ...).
- **sur les traitements;** ici il s'agit d'exprimer l'ordre dans lequel doivent s'effectuer les modifications de la BD.

Nous pouvons constater que les principaux éléments cités sont interdépendants. De plus, dans une approche classique [BOE76], chaque élément doit traverser plusieurs étapes dans le processus de conception:

- Analyse des besoins: par rapport aux objectifs de l'organisation
- Spécification: une description précise de chaque élément
- Conception Informatique: une description de l'ensemble du système en termes informatiques
- Codage: chaque élément est "codé" dans le langage supportant la gestion de la BD
- Test
- Maintenance

Pour assister l'équipe de conception, il existe des méthodologies et des outils informatiques. Ceux-ci permettent de guider la conception, de construire des prototypes et même de générer le code. Leur utilisation est délicate et elles demandent que le concepteur soit conscient des impasses et de leurs limites. Cependant elles peuvent assurer un rôle de communication au sein d'une équipe.

SQL - Langage des bases de données relationnelles

SQL est l'acronyme de "Structured Query Language". A l'origine, il n'était destiné qu'à l'interrogation des bases de données, mais il fut étendu à la spécification des données, des primitives de modification et la spécification des règles d'intégrité d'une base de données.

Historiquement, c'est l'article de E.F. Codd [COD70] du laboratoire de IBM à San Jose qui fonda le modèle relationnel en y exposant, la simplicité de la représentation de la relation, une forme normale pour décomposer une relation afin d'éviter des redondances, et les principaux opérateurs de l'algèbre relationnelle.

Plusieurs classes de langages furent explorées dans les années qui suivirent, dans les universités et les laboratoires privés. En 1974, D.D. Chamberlin proposa une définition d'un langage nommé SEQUEL qui fut implanté chez IBM dans un projet nommé SEQUEL-XRM. Il fut étendu dans un projet plus vaste touchant l'ensemble de l'architecture d'un SGBD, le System R (1977). Ce système fut jusqu'à la fin des années 70 une plate-forme

d'expérimentation pour IBM. Pendant la même période pour des raisons légales SEQUEL/2 fut renommé SQL.

Le début des années 80, verra l'apparition de plusieurs SGBD relationnels définis autour de SQL (ORACLE de Oracle Inc., SQL/DS et DB2 d'IBM, INGRES de Relational Technology Inc., SYBASE de Sybase Inc., ...) et ainsi SQL devient une norme de fait, mais chaque SGBD possède un dialecte de SQL avec ses propres particularités.

A partir de 1982, l'institut national américain de normalisation (ANSI) sera en charge de définir une norme qui sera finalisée en 1986, la même version est acceptée par l'organisation internationale des standards (ISO-9075) en 1987. Une extension portant sur les règles d'intégrité est éditée par l'ISO en 1989. En 1992, une nouvelle version du standard est éditée par l'ISO, SQL2. En 1999, une version SQL99 correspond à la version la plus récente.

Les exemples de cet ouvrage sont écrits avec le SQL du SGBD ORACLE. Dans sa dernière version, Oracle est conforme à la norme ISO de 1989. Nous avons rarement utilisé les extensions à la norme.

Trois discours

Les bases de données relationnelles prennent racine dans trois univers distincts. La notion de relation est établie formellement dans les mathématiques. Le modèle relationnel est un outil permettant la représentation des objets du champ d'application. Et les systèmes de gestion de bases de données relationnelles sont des logiciels utilisant les technologies de l'informatique. Nous avons trois mondes et par conséquent trois discours. Il convient donc d'être particulièrement attentif pour éviter la confusion entre le formel, le modèle et la technique.

Chaque monde se développe à son propre rythme; les notions mathématiques pouvant être considérées comme stables, les propriétés fondamentales de la modélisation relationnelle évoluent lentement et les technologies de l'informatique se succèdent les unes aux autres rapidement autour du langage SQL standardisé. La maîtrise de ces discours est le passage obligé pour comprendre l'évolution des systèmes de gestion de bases de données relationnelles, ces derniers tendent, dans leurs versions successives, d'incorporer les acquis du formel et les propriétés de la modélisation relationnelle. Ce souci de lier ces trois discours se fera dans le sens du concret et restera directement lié aux technologies de l'informatique.

Nous allons illustrer ces propos par un exemple (ne faites pas attention aux notations, elles seront complètement définies ultérieurement). Dans le champ d'application considéré, il s'agit de définir la distance à vol d'oiseau entre deux villes.

La relation mathématique se réduit au prédicat suivant:

$||\text{Distance}(x,y,z)|| = "z \text{ est la distance entre } x \text{ et } y"$
où $z \in \mathbb{N}$, $x,y \in \text{Ville}$

La modélisation relationnelle enrichit la représentation par rapport au champ d'application, en limitant la source des informations à l'atlas et en précisant l'unité de mesure. On cherche aussi à expliciter les liens existants entre les informations élémentaires. On peut dire par exemple que `Dist_en_Km` dépend fonctionnellement de `Départ` et d'`Arrivée`.

```
Distance(Départ,Arrivée,Dist_en_Km)
Domaine(Départ)= Nom_de_Ville_de_mon_Atlas
Domaine(Arrivée)= Nom_de_Ville_de_mon_Atlas
Domaine(Dist_en_Km)= Numérique entier [0..20'000]
||Distance(Départ,Arrivée,Dist_en_Km)||="Dist_en_Km est la distance
en Km entre la ville de Départ et la ville d'Arrivée"
```

L'ordre de création de la table dans un système de gestion de base de données relationnelles ne se préoccupe que des questions de représentation informatique et de la validation sur l'intervalle de la distance. Les dépendances fonctionnelles définissent alors la clé de la relation.

```
CREATE TABLE Distance(
  Depart      char(12) not null,
  Arrivee     char(12) not null,
  Dist_en_Km  number(5)
  Check (Dist_en_Km between 0 AND 20000),
  primary key (Depart, Arrivee))
```

Pour représenter quelques données, il est possible d'utiliser la notion de table: Distance

Départ	Arrivée	Dist [km]
Genève	Paris	420
Paris	Pékin	8220
Pékin	Vancouver	8510
...

En mathématique, on parle d'ensembles et de prédicats; dans la modélisation, on s'exprime avec des relations, des constituants, des domaines et des dépendances fonctionnelles. Dans la technologie informatique, les termes sont ceux de tables, colonnes, caractères, nombres, clés de table. Cette cascade de transformations, nous fait passer des abstractions pures des mathématiques à une concrétisation sous forme d'un système informatique. Poursuivons notre exemple avec une propriété: La distance à parcourir entre l'arrivée et le départ est identique à celle du départ à l'arrivée.

En mathématique, on parlera de commutativité que l'on notera:

$$\forall x, \forall y \text{ Distance}(x,y,z) \leftrightarrow \text{Distance}(y,x,z)$$

Dans la modélisation, cette commutativité sera perçue comme une règle d'intégrité que devront valider les données:

$$\begin{aligned} \text{ri1) } & \forall r \in \text{Distance}, \forall r' \in \text{Distance} \\ & \text{si } (r.\text{Arrivée}=r'.\text{Départ}) \wedge (r.\text{Départ}=r'.\text{Arrivée}) \\ & \text{alors } r.\text{Dist_en_Km}=r'.\text{Dist_en_Km} \end{aligned}$$

L'implantation informatique de cette règle peut être envisagée de différentes manières; il faudra maintenir automatiquement la distance des couples "aller et retour" à chaque modification de la distance ou bien n'enregistrer que la moitié de la table et permettre à l'utilisateur une interrogation sur la totalité.

Notre parcours suivra un itinéraire entre la théorie, les concepts et la mise en oeuvre, explicitant les liens autant que possible. Ceci permettra au lecteur de se rendre relativement indépendant des futures évolutions des technologies de l'information qui ne toucheront pas son capital de connaissances conceptuelles.

6. Base du modèle relationnel

"2.18 Ce que chaque tableau, de quelque forme que ce soit doit avoir de commun avec la réalité, pour absolument pouvoir la représenter - justement ou faussement - c'est la forme logique, c'est à dire la forme de la réalité" (Ludwig Wittgenstein - Tractatus logico-philosophicus)

L'objectif de ce chapitre est d'élucider la définition de la relation (donnée en termes mathématiques). Nous n'introduirons que les notions mathématiques nécessaires à la compréhension cette dernière.

Définition:

Une *relation n-aire* R est définie sur le produit cartésien des domaines de n constituants formant l'ensemble R^+ et d'un prédicat noté $||R||$ dont les variables libres correspondent aux constituants de R^+ et prennent leurs valeur dans les domaines de ces constituants.

Cette définition est liée aux ensembles, aux propositions logiques et aux prédicats logiques dont nous allons revoir les définitions fondamentales. Ensuite, au fur et à mesure, nous disséquerons la définition de la relation. Les ouvrages de référence pour ce chapitre sont [CHE74] et [HOT89].

Domaines

Un domaine désigne un ensemble de valeurs. Il est similaire à la notion de type que l'on trouve dans les langages de programmation. Ces valeurs seront prises par les données de notre champ d'application.

Définition:

Un *domaine* est un ensemble D non vide, fini ou dénombrable¹

On dira que a est une *valeur* de D si $a \in D$

Exemples de domaine :

Domaine_des_couleurs = {vert, jaune, bleu, rouge}

Domaine_des_nombres_entiers = {1,2,3,...,n,...}

Domaine_des_états_des_portes = {ouvertes, fermées}

Domaine_des_pays = {Suisse, France, Panama, ...}

Domaine_des_états_logiques = {vrai, faux}

¹ Un ensemble est dénombrable si l'on peut compter à l'aide des entiers naturels N . Notons que R les nombres réels ne sont pas dénombrables, mais leur représentation dans les ordinateurs étant finie, ils deviennent dénombrables. On pourra donc les utiliser comme domaine!

```

Domaine_des_titres_de_livre = {"comprendre la logique moderne", "what
is the name of this book", "Le parfum", ...}
Domaine_des_types_de_plante = {arbre, fleur, cactée, ...}
Domaine_des_fleurs = {rose, camélia, marguerite, ...}
Domaine_des_produits = {parfum, maquillage, crème solaire, ...}
Domaine_des_dates = {1-avril-1992, 24-dec-2000, ....)

```

Les domaines sont entièrement dépendants du champ d'application dans lequel on travaille. L'effort de modélisation commence par une élaboration complète des domaines. Dans [LEO88], on trouve une typologie des domaines. A chaque type de domaine est associé les opérations qui sont autorisées. Nous donnons cette typologie dans la Figure 6-1, sous forme d'un arbre où chaque noeud hérite des opérations du noeud précédant.

Le type *texte* définit les domaines où aucune opération n'est possible, il s'agit donc d'un domaine purement informatif. Le domaine des adresses peut être considéré comme appartenant au type texte. Il sera difficile de répondre sûrement à la question "combien de personnes habitent une avenue". En effet, le texte étant totalement libre, la saisie aura permis des abréviations et aussi des erreurs. Le libellé d'une écriture comptable est aussi du type texte, sa valeur nous permet de comprendre l'objet de l'écriture mais ne peut pas être utilisée pour connaître la somme des factures payées aux fournisseurs. Chaque fois que l'on associe ce type à un domaine, on renonce à utiliser ce domaine pour des opérations de sélection.

Le type *mot* définit les domaines où les opérations d'égalité ou d'inégalité sont possibles. Les domaines des noms, des prénoms des personnes sont associés à ce type. Il s'agit donc de domaines où il existe un critère qui permet d'établir si une valeur appartient ou non à ce domaine. Pour répondre à la question sur les "paiements d'un fournisseur", on peut associer à chaque écriture comptable un genre d'écriture qui prendra ses valeurs dans `Domaine_des_genres = { Payement_Fournisseur, Frais_généraux, ...}`. Pour passer du type texte au type mot, il faut donc expliciter l'information contenue implicitement dans le type texte.

Le type *mot ordonné* est un type mot sur lequel une relation d'ordre est définie. Si les valeurs sont représentées avec des caractères, il existe par défaut un ordre lexicographique qui permettra un tri de l'information (ou de chercher les valeurs plus petites, plus grandes, avant, après, ...). Pour certains types mot cet ordre doit être donné explicitement; par exemple pour ordonner le `Domaine_des_couleurs`, on pourra choisir la longueur d'onde associée à chacune.

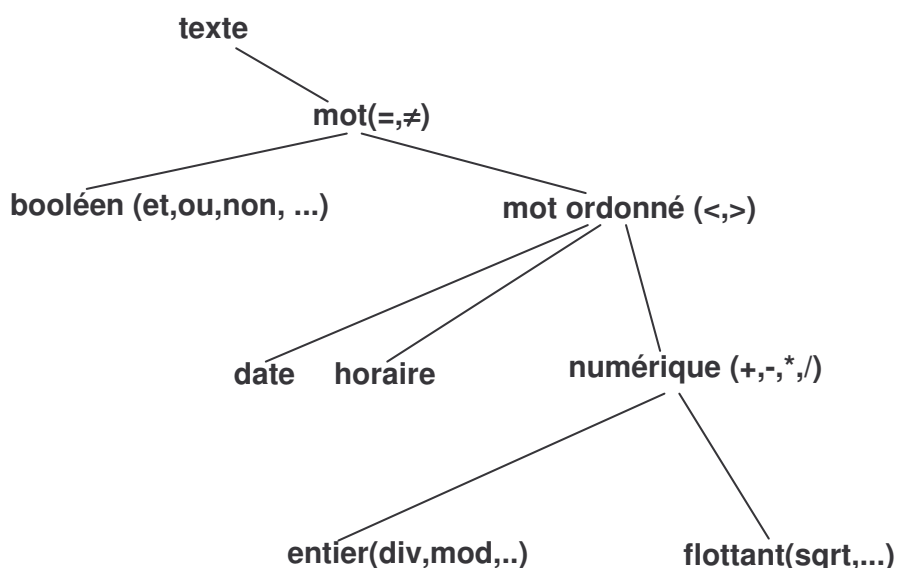


Figure 6-1: Héritage des opérations dans la typologie des domaines.

Le type *numérique* autorise les opérations arithmétiques de l'addition, de la soustraction, de la division et de la multiplication. Il se divise en deux sous types: les entiers et les réels, chacun possédant ses opérations propres. Le type *date* et *horaire* sont des domaines propres à la gestion. Le type *booléen* est un type sur lequel sont définis les opérations logiques (et, ou, non, ...).

Le choix du type de domaine est essentiel car il détermine les opérations de comparaison et de manipulation des données qui seront associées à ce type. Ce choix se fait au moment de la modélisation, car au moment de la mise en oeuvre informatique, les SGBD associent tous ces types à deux catégories: les chaînes de caractères et les nombres. On aura donc recours à d'autres mécanismes pour assurer la validité des données par rapport à leurs domaines.

Constituants

Un *constituant* est un ensemble de données ayant une propriété et un comportement homogène dans le champ d'application. La signification des données réside dans l'appartenance à cette classe. La donnée "ROSE" sera interprétée comme un prénom si elle est une valeur du constituant PRENOM, comme une espèce de fleur si elle est une valeur du constituant FLEUR, comme une couleur si elle est une valeur du constituant COULEUR.

Le constituant est associé à un identificateur (PRENOM, NOM, FLEUR, COULEUR, DATE_ECRITURE, DATE_NAISSANCE).

Le constituant est associé à un et un seul domaine qui définit les valeurs que peut prendre cette propriété.

La fonction *dom* associe à un constituant son domaine

C : ensemble des constituants

D : ensemble des domaines

dom : C \rightarrow D
 dom(C)=D

On a par exemple:

dom(DATE_NAISSANCE) = Domaine_du_calendrier.

Pour interpréter une valeur, il est indispensable de connaître son constituant, sa valeur seule ne suffit pas. Le "14-novembre-1980", est-ce une date de naissance ou une date d'écriture comptable ? Le constituant donne son sens à la valeur, le domaine établit les valeurs autorisées. Il serait possible de modifier tous les domaines d'une modélisation, par exemple, en les traduisant dans une autre langue, sans affecter l'interprétation donnée par les constituants. Les propriétés d'une modélisation seront établies ultérieurement par les liens sémantiques existant entre les constituants.

Rappels sur la logique des propositions

Dans [HOT89], on peut lire: "la logique ne considère comme propositions que des phrases susceptibles d'être **vraies ou fausses**; ceci écarte d'emblée toutes les formes (modales) de la prière, de l'interrogation, du commandement, du souhait ... Seules, en principe, sont recevable les propositions **déclaratives, descriptives, empiriques**, c'est-à-dire, d'une façon générale, les propositions qui **disent quelque chose au sujet de la réalité** et dont l'affirmation est, au moins théoriquement, vérifiable²."

- (p) "Un domaine est un ensemble" est une proposition vraie
- (q) "Les autruches volent" est une proposition fausse
- (r) "Vous lisez cette phrase" est une proposition vraie
- (s) "Paris est dans l'hémisphère sud" est une proposition fausse

La *proposition simple*, habituellement notée par les lettres p, q, r, s ... ne se préoccupe pas de l'analyse du contenu de la phrase, sa totalité est évaluée à vrai ou à faux.

La *proposition composée* est formée de propositions simples connectées par des opérateurs logiques (et éventuellement des parenthèses pour ôter les ambiguïtés). Les opérateurs couramment utilisés sont:

La *conjonction* (et logique notée \wedge): $p \wedge q$ sera évaluée à vrai si p est vrai et si q est vrai.

La *disjonction* (ou logique inclusif notée \vee) $p \vee q$ sera évaluée à vrai dans trois cas: p vrai et q faux, p vrai et q vrai, p faux et q vrai.

Le *conditionnel* ("Si p alors q" noté \rightarrow) $p \rightarrow q$ sera évaluée à vrai dans trois cas: p vrai et q vrai, p faux et q vrai, p faux et q faux.

Le *biconditionnel* ("Si et seulement si" notée \leftrightarrow) $p \leftrightarrow q$ sera évaluée à vrai dans deux cas: p vrai et q vrai, p faux et q faux.

La *négation* (notée \neg) $\neg p$ sera évaluée à vrai si p est faux.

²Mis en évidence par l'auteur

Si p et q correspondent aux propositions définies précédemment, nous aurons l'expression, $p \wedge q$, évaluée à faux ("Un domaine est un ensemble" et "Les autruches volent").

L'évaluation des propositions composées peut être obtenue en construisant une table de vérité, pour les opérateurs $\wedge, \vee, \rightarrow, \leftrightarrow, \neg$ nous obtenons la table de vérité de la Figure 6-2.

p	q	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$	$\neg p$
V	V	V	V	V	V	F
V	F	F	V	F	F	F
F	V	F	V	V	F	V
F	F	F	F	V	V	V

Figure 6-2: table de vérité des opérateurs logiques courants

Une tautologie est une expression qui est toujours évaluée à vrai. Par exemple: $p \vee \neg p$, le principe du tiers exclu est une tautologie. Une tautologie est une expression qui comporte que des vrais dans sa table de vérité.

p	$\neg p$	$p \vee \neg p$
V	F	V
F	V	V

Figure 6-3: table de vérité $p \vee \neg p$

Les logiciens recherchent ces expressions car elles sont des théorèmes ou des axiomes.

Rappels sur la logique des prédicats

La logique des prédicats se propose d'analyser les propositions p, q, r considérées comme inanalysées dont on savait précédemment seulement si elles sont vraies ou fausses. L'objectif principal des prédicats est donc l'analyse des propositions.

En logique des propositions, on a l'ensemble suivant de propositions

les autruches volent : faux
 les moineaux volent : vrai
 les pies volent : vrai
 les pingouins volent : faux
 etc

Le prédicat vole(x) qui associe pour chaque x pris dans l'ensemble Oiseaux la valeur vrai ou faux permet d'analyser les propositions précédentes.

vole : Oiseaux \rightarrow {vrai, faux}

Un prédicat ou formule ouverte est une expression dont la valeur de vérité reste indéterminée car elle contient des variables. Dès que toutes ces variables sont remplacées par une valeur alors elle devient vraie ou fausse. En effet, à cet instant le prédicat s'est transformé en proposition.

exemple:

Si $I(x)$ signifie « x entier non nul est impair »

$I(1)$ et $I(3)$ sont vrais.

$I(2)$ et $I(4002)$ sont faux.

Si $P(x)$ signifie « x entier non nul est pair »

$P(1)$ et $P(3)$ sont faux.

$P(2)$ et $P(4002)$ sont vrais.

L'autre possibilité de transformer un prédicat en proposition est de quantifier ses variables. Il existe deux possibilités.

Par la *quantification universelle*, qui substitue à la variable tous les éléments de l'univers considéré ou domaine d'interprétation, à ce prédicat. Soit:

$\forall x, f(x)$ qui se lit: pour tout x , on a $f(x)$

On a par exemple, $\forall x, I(x)$ est une proposition fautive car tous les nombres ne sont pas impairs. Le domaine d'interprétation est ici l'ensemble des entiers positifs (pour simplifier la notation, nous considérerons le domaine d'interprétation comme implicitement défini) Le quantificateur universel est équivalent à écrire la conjonction de toutes les propositions obtenues avec un prédicat, soit dans notre exemple:

$\forall x, I(x) \equiv I(1) \wedge I(2) \wedge I(3) \wedge I(4) \wedge \dots$

Par la *quantification existentielle*, qui substitue à la variable au moins un des éléments de l'univers associé à ce prédicat. Soit:

$\exists x, f(x)$ qui se lit: il y a un x , tel que $f(x)$

On a par exemple, $\exists x, P(x)$ est une proposition vraie, car 3224 est un nombre pair. Le quantificateur existentiel est équivalent à écrire la disjonction de toutes les propositions obtenues avec un prédicat, soit dans notre exemple:

$\exists x, P(x) \equiv P(1) \vee P(2) \vee P(3) \vee P(4) \vee \dots$

Le calcul des prédicats permet donc de manipuler des expressions propositionnelles de longueur infinie (dénombrable).

On peut étendre les prédicats en utilisant les opérateurs logiques vus précédemment. On aura par exemple les expressions:

$\forall x, (I(x) \vee P(x))$ qui est une proposition vraie (tout x est pair ou impair)

$\neg \exists x, (I(x) \wedge P(x))$ qui est une proposition vraie (il n'existe pas de x impair et pair)

On peut étendre les prédicats en utilisant des expressions avec plusieurs variables. Par exemple:

Distance(x, y, z) où " z est la distance entre x et y "

La transformation d'un prédicat en fonction est définie de la façon suivante:

Un prédicat n'ayant plus aucune *variable libre* devient une proposition.

Une variable est *libre* si elle n'est pas liée. Elle est *liée* par:

- une substitution avec une valeur
- par une quantification.

Ensembles, propositions et prédicats

Nous avons déjà vu que les prédicats dont toutes les variables sont liées deviennent des propositions

Entre les ensembles et la logique des propositions, il existe un isomorphisme (limité) qui mécaniquement permet de transformer une propriété du calcul des classes en un théorème du calcul des propositions et vice-versa. On applique les substitutions suivantes sur les variables et les opérateurs:

$$\begin{array}{ll} A \leftrightarrow p & \cup \leftrightarrow \vee \\ B \leftrightarrow q & \cap \leftrightarrow \wedge \\ C \leftrightarrow r & - \leftrightarrow \neg \\ \emptyset \leftrightarrow \text{Faux} & \\ U \leftrightarrow \text{Vrai} & \end{array}$$

Cet isomorphisme signifie: c'est vrai pour les ensembles si et seulement si c'est vrai pour les propositions.

On a par exemple:

La distributivité de \cap sur \cup

$$\text{Soit: } A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$$

devient la distributivité de \wedge sur \vee , pour le calcul des propositions Soit:

$$p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r).$$

Le principe du tiers exclu $p \vee \neg p = \text{vrai}$ (car une tautologie) devient la formule suivante $A \cup (-A) = U$.

Cet isomorphisme est limité; le calcul des propositions est plus puissant que le calcul sur les ensembles, par contre, la notion d'ensemble introduit des états autres que vrai et faux (tout ou rien).

Les ensembles et les prédicats sont liés par le principe d'abstraction qui spécifie que pour tout prédicat est associé au moins un ensemble. Le prédicat devient la fonction propositionnelle de l'ensemble. On a donc:

Si A est l'ensemble déterminé par le prédicat $f(x)$, il est équivalent pour tout x de dire que "x satisfait la fonction $f(x)$ " ou que "x est élément de A". Soit:

$$\forall x (f(x) \equiv x \in A)$$

Ceci est vrai pour les prédicats à une variable, la définition du produit cartésien va nous permettre de la rendre vraie pour un prédicat à n variables.

Définition:

Soit A et B deux ensembles, $A \times B$ est le *produit cartésien* de A par B et définit le nouvel ensemble suivant:

$$A \times B = \{(a,b) \mid (a \in A) \wedge (b \in B)\}$$

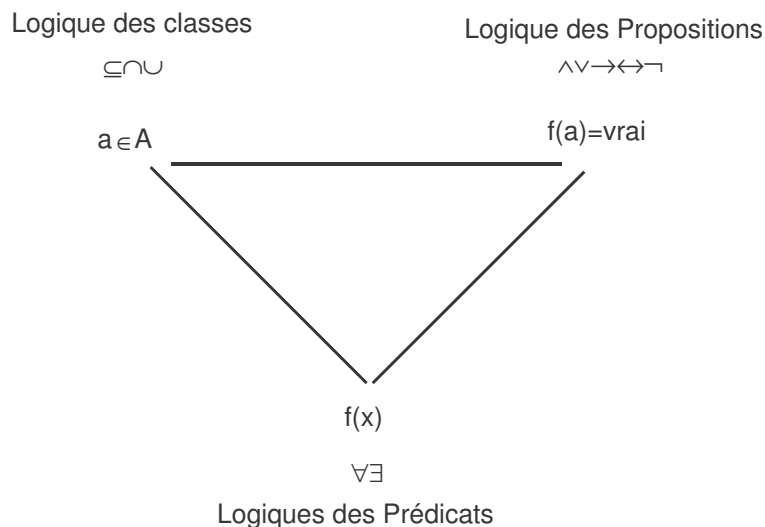


Figure 6-4: Ensembles, propositions et prédicats

Une feuille d'agenda est le produit cartésien de Jour et Heure.

Agenda = Jour \times Heure
 où Jour = {lundi, mardi, ..., dimanche}
 et Heure = {7h, 8h, ..., 19h}

Un échiquier est un produit cartésien:

Echiquier = {a,b,c,...,h} \times {1,2,...,8}

Et bien-sûr les coordonnées cartésiennes:

$3D = \mathfrak{R} \times \mathfrak{R} \times \mathfrak{R}$

On peut étendre la notion de produit cartésien à n ensembles:

$A_1 \times A_2 \times A_3 \times \dots \times A_n$
 $= \prod A_i, i=1..n$
 $= \{(a_1, a_2, \dots, a_n) \mid (\forall a_i, a_i \in A_i) i=1..n\}$

Ainsi un prédicat portant sur plusieurs variables devient la fonction propositionnelle d'un ensemble contenu dans le produit cartésien de l'univers de chaque variable.

La Figure 6-4 résume les liens existant entre les trois formalismes:

Retour à la définition de relation

Nous avons donc maintenant tous les éléments pour comprendre la définition de la relation.

Une relation (ou schéma de relation) R est composée

- d'un ensemble R^+ de constituants et
- d'une formule $||R||$, appelée prédicat de la relation, dont les variables libres sont exactement les constituants de R.

Une relation R est une relation portant sur n variables; dans le modèle relationnel, nous préférons parler en termes de constituants plutôt que de variables³.

$R^+ = \{C_1, C_2, \dots, C_n\}$ définit l'ensemble de constituants de la relation

$\prod \text{dom}(C_i), C_i \in R^+, i=1..n$ définit le produit cartésien des domaines. Il définit donc la **structure** de la relation.

$||R(C_1, C_2, \dots, C_n)||$ définit le prédicat de la relation. Il définit la **sémantique** de la relation.

Illustrons cela avec les exemples suivants:

La relation SOMME

$SOMME^+ = \{GAUCHE, DROITE, TOTAL\}$

$||SOMME|| = (GAUCHE + DROITE = TOTAL)$

la relation ETUDIANT

$ETUDIANT^+ = \{Nom, Prenom, Sexe, Origine, Date_Naissance\}$

$\text{dom}(Nom) = \text{mot} \{Dupont, Durant, \dots\}$

$\text{dom}(Prenom) = \text{mot} \{Jean, Marie, \dots\}$

$\text{dom}(Sexe) = \text{mot} \{Homme, Femme\}$

$\text{dom}(Origine) = \text{mot} \{Suisse, France, Suède, \dots\}$

$\text{dom}(Date_Naissance) = \text{date}$

et nous avons le prédicat suivant:

$||ETUDIANT(n, p, s, o, d)||$: "La personne portant le nom n et le prénom p est de sexe s , d'origine o et elle est née à la date d et elle est actuellement un étudiant de l'Université de Genève"

Dans ce dernier exemple le prédicat se réduit à une expression atomique, c'est à dire non décomposable, comprenant un symbole de prédicat exprimé en langue naturelle "La personne portant le nom n et le prénom p est de sexe s ..." et des variables. Cette formule n'est pas évaluable formellement mais décrit bien l'intention de la relation, elle fournit la sémantique de cette relation. Si l'on attribue des valeurs aux variables, une personne connaissant le domaine d'application pourra évaluer cette formule à vrai ou faux. La formule définissant une relation se réduit la plupart du temps à cette forme

³La relation en mathématique est positionnelle (la première, seconde, etc variable du prédicat). En nommant les variables, il est possible de l'affranchir de la position de la variable, ainsi dans le modèle relationnel la position d'un constituant est sans importance. Les deux relations suivantes sont équivalentes:

$Distance(Départ, Arrivée, Dist_en_Km) \equiv Distance(Dist_en_Km, Arrivée, Départ)$

prédicative de base, sauf pour des relations telles SOMME qui sont "calculables", c'est pourquoi on a choisi de l'appeler prédicat. Nous verrons qu'avec l'algèbre relationnelle on peut obtenir des relations dont le prédicat sera composé à partir des prédicats des relations de départ.

Pour une relation, il n'existe qu'un seul prédicat, donc qu'une seule interprétation possible des données, à un instant fixé. Le domaine d'interprétation est souvent implicite, ici les étudiants de l'université de Genève.

On note xR (ou $\text{type}(R)$) le produit cartésien des domaines des constituants de R , soit $xR = \prod \text{dom}(C_i)$, $C_i \in R^+$, $i=1..n$. Il définit donc la **structure** de la relation.

Définition:

Un *n-uplet* r de R est un élément du produit cartésien, soit :

$$r \in \prod \text{dom}(C_i), C_i \in R^+$$

Une *entité* r de R est un n-uplet de R tel que $\|R\|(r)$ est vrai

Soit R une relation, $X \subseteq R^+$ et r une entité de R , alors on note $r.X$ la valeur que r prend pour les constituants de X

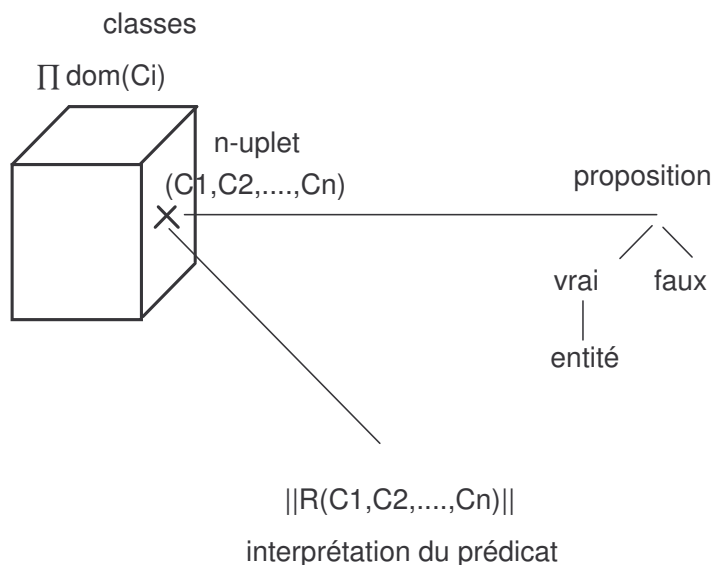


Figure 6-5 : interprétation de la relation

Le n-uplet est un élément d'une relation ayant des valeurs prises dans les domaines des constituants mais dont on ne dit rien sur le prédicat. Il appartient uniquement au produit cartésien. Par contre l'entité vérifie le prédicat de la relation. Nous avons par exemple:

Le n-uplet ("Dupont", "Paul", "Homme", "France", "16-8-63") pour la relation ETUDIANT et si "La personne portant le nom **Dupont** est le prénom **Paul** est de sexe **Homme**, d'origine "**France**" et elle est née à la date **16-8-63** et elle est actuellement un étudiant de l'Université de Genève" alors c'est une entité de la relation.

Si r est l'entité précédente et $X=\{\text{Prenom, Nom}\}$ alors $r.X = (\text{"Dupond"}, \text{"Paul"})$

Chaque n -uplet est donc une proposition évaluée à vrai ou à faux. Seules les entités, nous intéressent, elles formeront l'instance de la relation.

Définition:

Une *instance de R* est l'ensemble des entités de cette relation (notée iR): $iR = \{r_1, r_2, \dots, r_n\}$

Nous distinguons donc la relation qui est définie par sa structure et son prédicat, de l'instance qui est une interprétation du prédicat pour les valeurs autorisées par la structure. Cette distinction nous conduit aux définitions suivantes:

Une modélisation Md est un ensemble de relations soit:

$$Md = \{R_1, R_2, \dots, R_n\}$$

Une base de données iMd correspond à une instance de modélisation Md , soit l'ensemble des instances des relations de cette modélisation, d soit:

$$iMd = \{iR_1, iR_2, \dots, iR_n\}$$

iMd correspond à une interprétation pour un domaine d'interprétation donné (les étudiants de Boston), à un instant fixé (hier à 12h34).

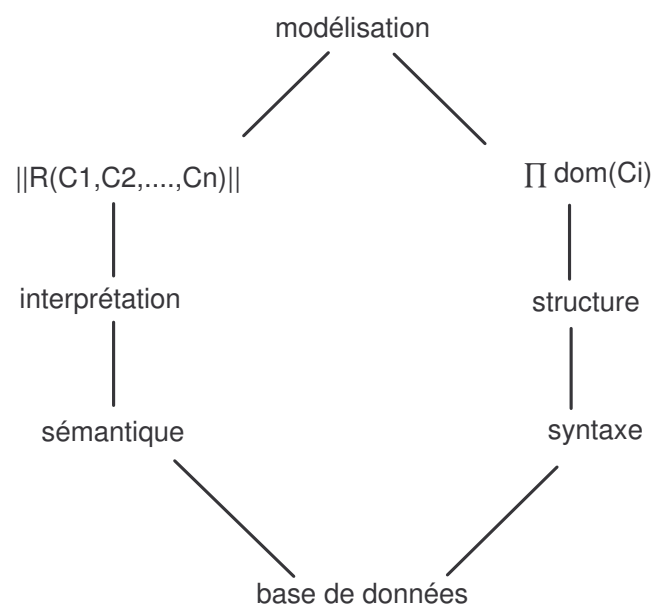


Figure 6-6 : Du modèle à l'instance

La modélisation possède deux aspects. D'une part nous avons la structure décrite en termes de produits cartésiens de constituants définis par leur domaine et d'autre part la dimension sémantique définie par les prédicats des relations. L'instance d'une modélisation est la base de données, c'est-à-dire, l'ensemble des entités appartenant à la structure et vérifiant les prédicats. Les systèmes de gestion de base données sont essentiellement structurels.

Dans les chapitres suivants, nous verrons comment nous pouvons établir des modélisations afin que leur mise en oeuvre avec des systèmes informatiques soit la plus fidèle possible.

Exercices

Questions sur TT³

Trouvez les domaines et le type des domaines de chaque constituant.

Ecrivez un prédicat pour les relations suivantes dont l'interprétation correspond à l'énoncé.

Véhicule(noChassis, noPlaque, miseEnService, modèle, noStation)

Type(modèle,nbPlaces,catégorie,typeCarburant,automatique,poids)

Chauffeur(noChauffeur, nom, prénom, adresse, noStation)

Réponses sur TT³

Domaine des constituants:

adresse	texte
automatique	booléen
catégorie	mot (A,B1,B2,C ...)
description	texte
heure	entier [0..23]
kilometrage	entier positif
litres	réel [0..500]
miseEnService	date
modèle	mot (mercedes300,audi200,car80pl,...)
nbPlaces	entier [4..80]
noChassis	entier positif
noChauffeur	entier positif
noJour	entier [1..366]
nom	mot (Dupont, Durant, ...)
noPlaque	entier [1..9999]
noStation	entier [1..6]
noZone	entier [1..99]
poids	entier [500..15000]
prénom	mot (Jean, Marie, ...)
tempsParcours	entier positif (en minutes)
trancheHoraire	mot (A,B,C)
typeCarburant	mot (super,normal,sansplomb,diesel)
zoneA	entier [1..99]

zoneDe entier [1..99]

Prédicat des relations

Véhicule(noChassis, noPlaque, miseEnService, modèle, noStation)

||Véhicule(nc,np,s,m,ns)|| : "Le véhicule portant le numéro de Chassis **nc** et immatriculé **np** a été mis en service le **s**, il est du modèle **m** et appartient à la station **ns**"

Type(modèle,nbPlaces,catégorie,typeCarburant,automatique,pois)

||Type(m,s,c,tc,a,p)|| : "Le modèle **m** de véhicule peut contenir **s** personnes, pèse **p** [kg], consomme du carburant **tc**, la boîte à vitesse est **a**, le permis **c** est nécessaire pour le conduire"

Chauffeur(noChauffeur, nom, prénom, adresse, noStation)

||Chauffeur(nch,n,p,a,ns)|| : "Le chauffeur portant le nom **n** et le prénom **p** habite **a**. il est identifié par le numéro **nch** et il est assigné à la station **ns**"

On remarque que le *noStation* et le *modèle* se trouvent dans deux relations et que leur sens dépend du contexte, c'est-à-dire, de la relation dans laquelle ils se situent.

7. Modélisation et définition des données

"Il n'y a pas de pensée sans langage symbolique. Les syntaxes et les grammaires nous imposent, certes, une médiation qui est autant une traduction qu'une déformation. Mais ce biais et cet éloignement du réel offrent des contreparties. Il y a en effet une fécondité surprenante de l'activité symbolique." (Philippe Quéau - Eloge de la simulation)

Nous désirons exprimer notre modélisation indépendamment du système de gestion de base de données que nous allons utiliser. A cette fin, nous définirons un langage de description de modélisation (LDM). Cette description est spécifiée à l'aide de diagrammes syntaxiques. Nous donnerons deux exemples de modélisation d'un même champ d'application. Enfin, nous définirons la syntaxe SQL du langage de description des données (LDD) qui est le moyen de traduire, pour un système de gestion de base de données, la modélisation.

Diagrammes syntaxiques

Les diagrammes syntaxiques sont un moyen simple et graphique pour représenter de manière formelle la grammaire d'un langage. Ils sont une alternative à description BNF, Backus-Naur-Form; du nom des deux auteurs ayant défini ALGOL 60 [NAU63]. Jensen K. et N. Wirth ont rendu les diagrammes syntaxiques populaires en les utilisant pour la définition du PASCAL [JEN75].

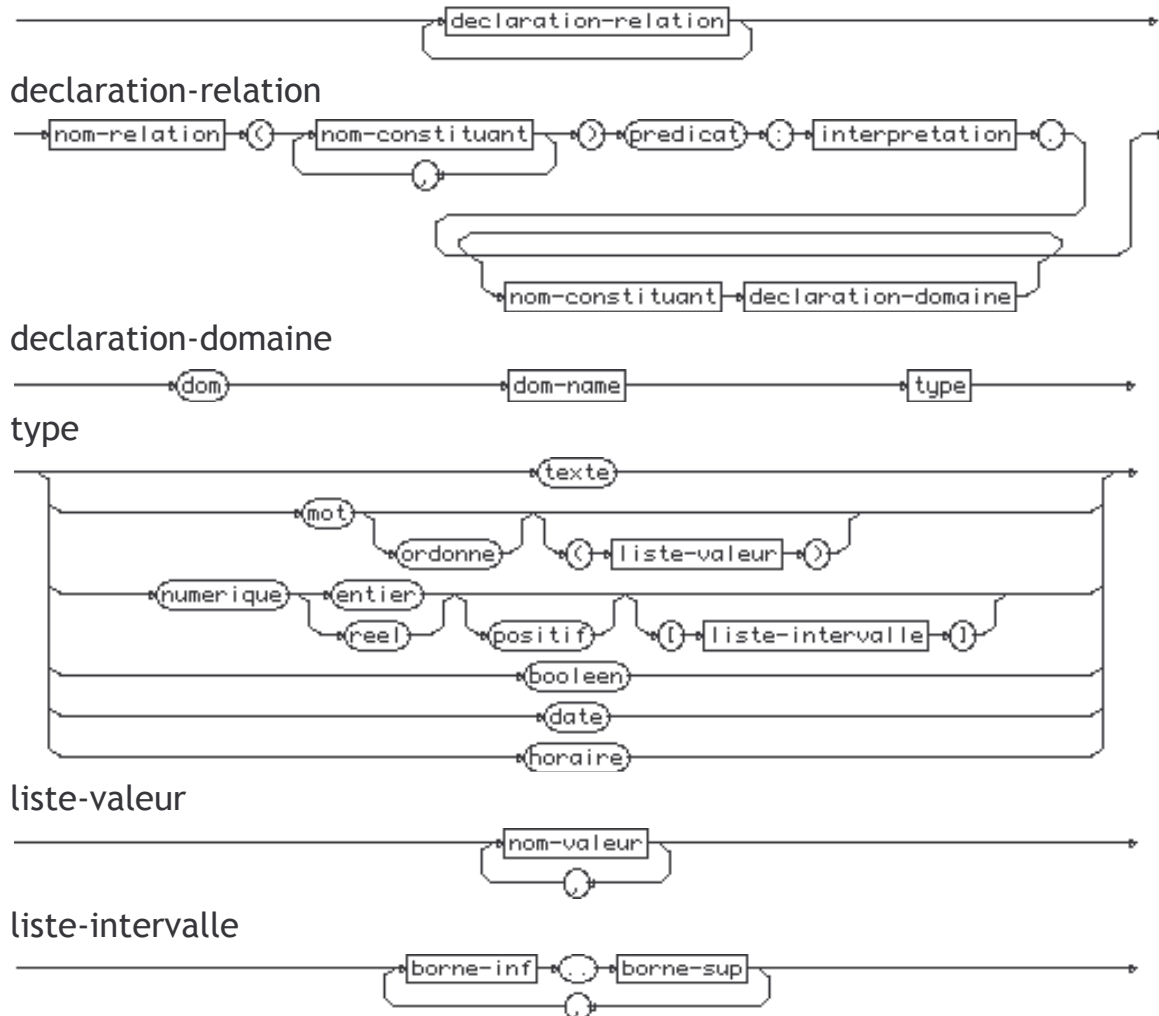
Chaque règle de grammaire est définie par un diagramme. Un diagramme est composé des symboles terminaux contenus dans des boîtes aux coins ronds et de symboles non-terminaux contenus dans des boîtes rectangulaires. Les symboles terminaux déterminent des chaînes de caractères qui apparaissent dans une phrase du langage et les symboles non-terminaux renvoient à une autre règle. Certains symboles lexicaux comme les identificateurs et les nombres sont considérés comme des non-terminaux. Dans ce cas, ils renvoient à la règle lexicale.

Langage de description de modélisation

Ce langage permet de définir une modélisation, c'est-à-dire, un ensemble de relations. Une relation est définie par son nom, sa liste de constituants, son

prédicat. Un constituant est défini par son domaine. Les types de domaines sont ceux vus précédemment.

modélisation



Nom-relation, nom-constituant sont des identificateurs, Interpretation et nom-valeur sont des chaînes caractères, borne-inf et borne-sup sont des nombres.

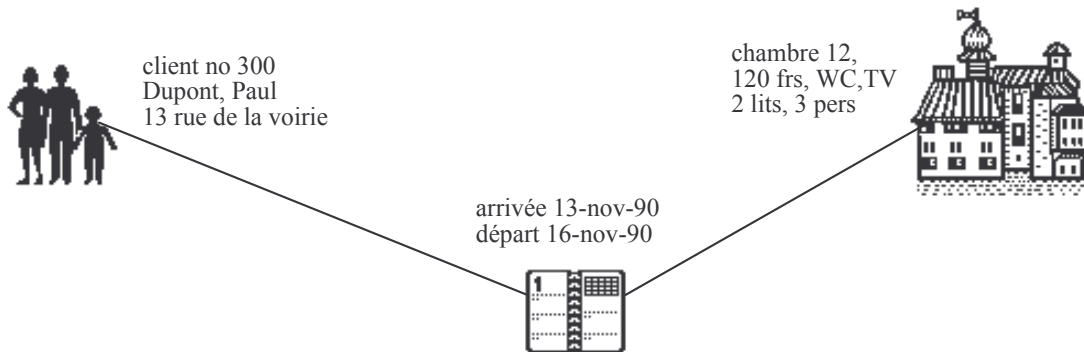


Figure 7-1: Une réservation dans un hôtel

Utilisons ce langage pour modéliser un système de réservation des chambres dans un hôtel. Notre hôtel est constitué de chambres réservées par des

clients pour une période donnée. La figure 1a montre les différentes caractéristiques d'une réservation.

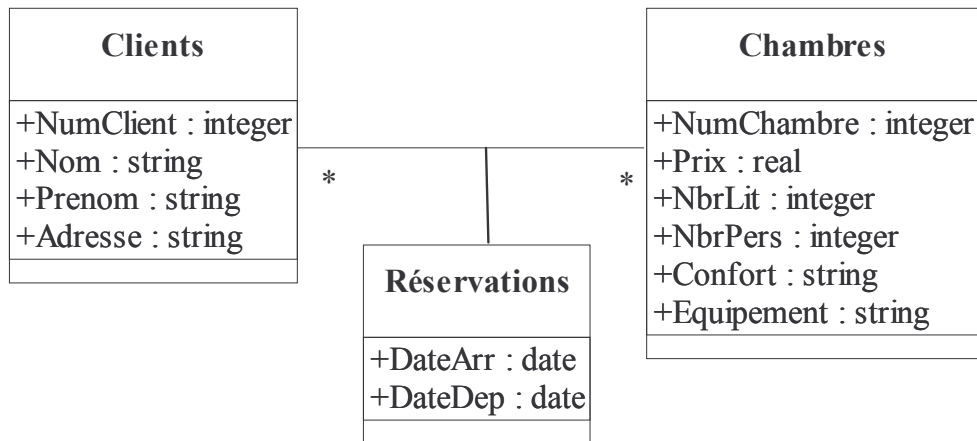


Figure 7-2: Diagramme de classes pour Hotel

Une première modélisation de Hôtel (version 1)

Hôtel(NumChambre, NumClient, Nom, Prenom, Adresse, Prix, NbrLit, NbrPers, DateArr, DateDep, Confort, Equipement)

predicat: "||Hôtel(a,b,c,d,e,f,g,h,i,j,k,l)|| Le client, portant le numéro b nommé d,c habitant e arrivant le i et partant le j, a réservé la chambre portant le numéro a qui contient g lits et peut loger h personnes, ayant pour confort k et équipement l et coûte f francs par nuit"

NumChambre dom numérique entier [1..50]
 NumClient dom numérique entier [1..999999]
 Nom dom mot (Dupont, Durand, ...)
 Prenom dom mot (Jean, Marie, ...)
 Adresse dom texte
 Prix dom numérique réel [100..1000]
 NbrLit dom numérique entier [1..3]
 NbrPers dom numérique entier [1..3]
 DateArr dom date
 DateDep dom date
 Confort dom mot ordonné (wc,douche,bain)
 Equipement dom mot (sans,TV)

Cette entité (12, 300, Dupont, Paul, "13 rue de la voirie", 120, 2, 3, 13-Nov-92, 16-Nov-92, WC, TV) est l'interprétation du fait : "Le client, portant le numéro 300 nommé Dupont, Paul habitant "13 rue de la voirie" arrivant le

13-Nov-92 et partant le 16-Nov-92, a réservé la chambre portant le numéro **12** qui contient **2** lits et peut loger **3** personnes, ayant pour confort **WC** et équipement **TV** et coûte **120** francs par nuit".

Nous allons voir une deuxième modélisation du même champ d'application (Version 2):

```
Chambres(NumChambre, Prix, NbrLit, NbrPers, Confort,
Equipement)
predicat:"||Chambres(a,b,c,d,e,f)|| La chambre portant
le numéro a qui contient c lits et peut loger d
personnes a pour confort e et équipement f et coûte b
francs par nuit"

Clients(NumClient, Nom, Prenom, Adresse)
predicat:"||Clients(a,b,c,d)|| Le client portant le
numéro a se nomme c,b et habite d"

Réservation(NumChambre, NumClient, DateArr, DateDep)
predicat:"||Réservation(a,b,c,d)|| Le client, portant
le numéro b arrivant le c et partant le d, a réservé
la chambre portant le numéro a"

NumChambre      dom numérique entier [1..50]
NumClient       dom numérique entier [1..999999]
Nom             dom mot (Dupont, Durand, ...)
Prenom         dom mot (Jean, Marie, ...)
Adresse        dom texte
Prix           dom numérique réel [100..1000]
NbrLit         dom numérique entier [1..3]
NbrPers        dom numérique entier [1..3]
DateArr        dom date
DateDep        dom date
Confort        dom mot ordonné (wc,douche,bain)
Equipement     dom mot (sans,TV)
```

Le fait exprimé précédemment devient ici les trois entités suivantes: Une entité de Clients (300, Dupont, Paul, "13 rue de la voirie") exprimant le fait: "Le client portant le numéro **300** se nomme **Dupont, Paul** et habite **13 rue de la voirie**". Une entité de Chambres (12, 120, 2, 3, WC, TV) exprimant le fait: "La chambre portant le numéro **12** qui contient **2** lits et peut loger **3** personnes a pour confort **WC** et équipement **TV** et coûte **120** francs par nuit". Et une entité de Réservation (12, 300, 13-Nov-92, 16-Nov-92) exprimant le fait : "Le client, portant le numéro **300** arrivant le **13-Nov-92** et partant le **16-Nov-92**, a réservé la chambre portant le numéro **12**".

Les deux modélisations semblent dire la même chose (elles sont basées sur les mêmes constituants) et pourtant elles sont structurellement différentes, l'une décompose l'ensemble des constituants en une relation, l'autre en trois. Ces problèmes sont centraux lors du processus de modélisation. Dans le chapitre suivant, nous verrons comment il est possible de traduire le diagramme des classes en un schéma de relation. Pour l'instant, examinons,

comment il est possible de traduire une modélisation en description de données pour un SGBD.

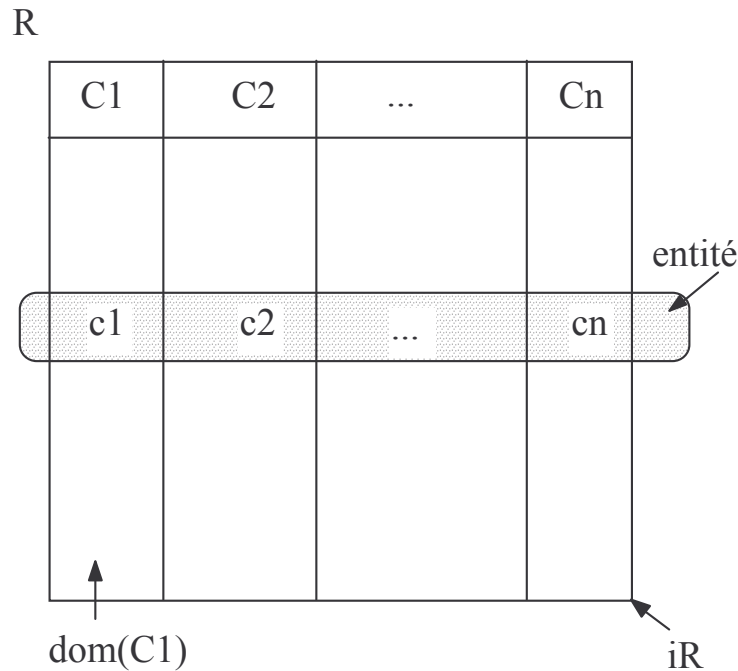


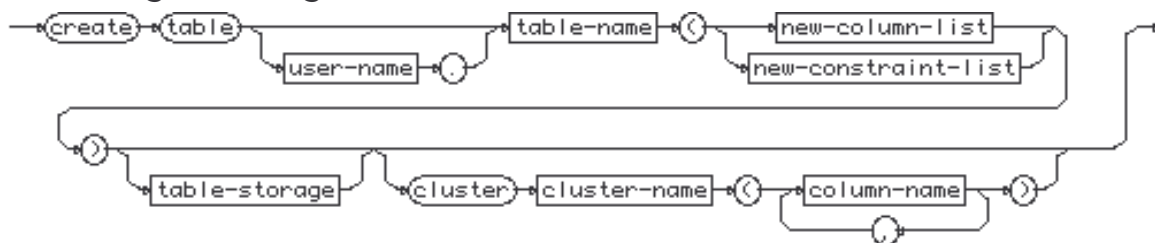
Figure 7-3: Equivalence entre une table et une instance de relation R

Langage de description des données (SQL)

Pour différencier les relations de la modélisation de celles manipulées par un SGBD, nous appelons ces dernières tables. La structure de table est équivalente pour une relation R au produit cartésien du domaine de ses constituants. Chaque colonne de la table sera un constituant. Les valeurs possibles dans une colonne sont celles du domaine de ce constituant. Le contenu de la table est équivalent à une instance de R. Une ligne de la table est équivalent à une entité de R.

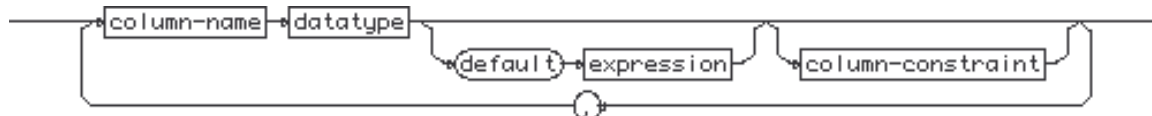
Nous voyons ici les premiers diagrammes syntaxiques de SQL pour la création des tables. Nous indiquons les différences principales avec le standard dans les notes de bas de page.

Création de structure de table vide⁴: on indique le nom de la table (éventuellement précédé du nom de l'utilisateur) et la liste des descriptions de colonnes. Nous reviendrons sur les notions de contraintes dans le chapitre sur les règles d'intégrité.

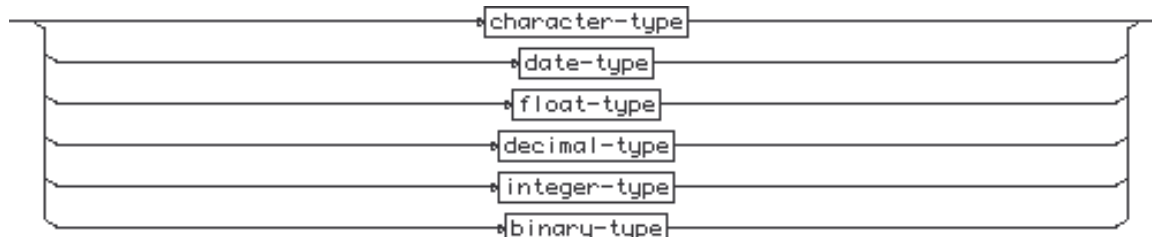


⁴ Les clauses (table storage et cluster) concernant la façon de mémoriser les tables ne sont pas standardisées.

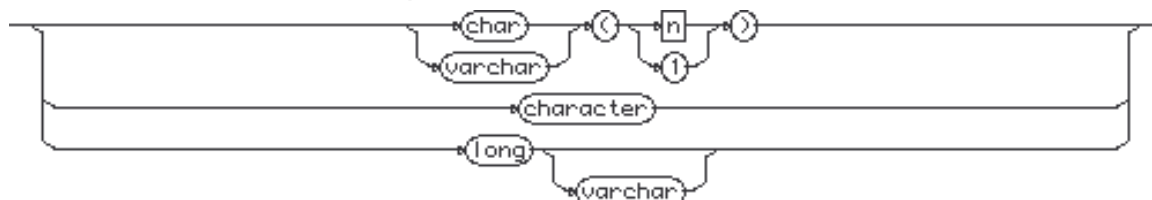
New-column-list⁵: une colonne est spécifiée par son nom et son type et éventuellement par une valeur par défaut donnée à la création d'une entité



datatype⁶: les types de données sont très classiquement ceux que l'on trouve dans les langages de programmation avec une redondance (sans doute due à la normalisation) qui permet de définir de plusieurs manières le même type.



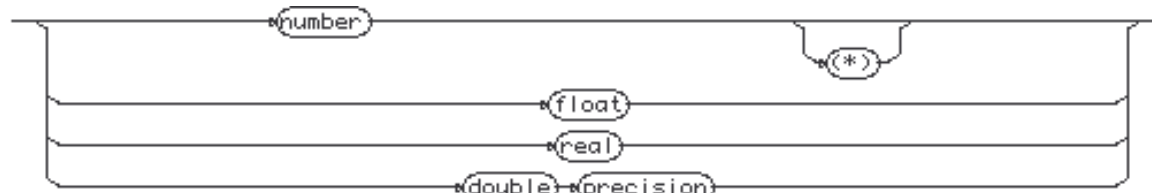
character-type: permet de définir un caractère (character), une chaîne de caractères (limitée à 255 pour char)



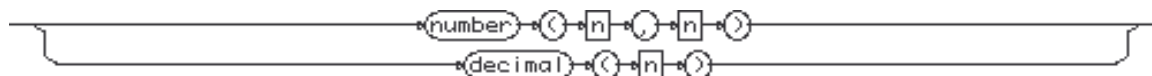
date-type: une date définie à la seconde près



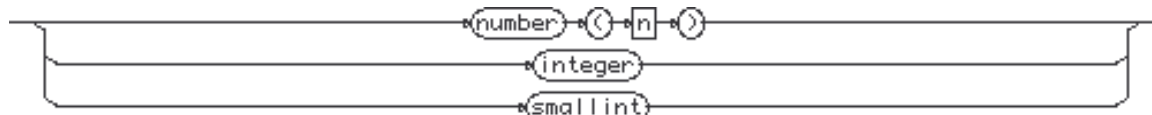
float-type: des nombres en virgule flottante (notation scientifique)



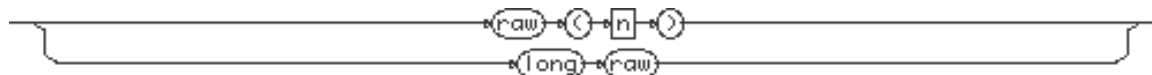
decimal-type: des nombres à virgule fixe (en comptabilité)



integer-type: des nombres entiers



binary-type: des informations binaires (son, image, ...)



⁵ Le standard ne mentionne pas de valeur par défaut

⁶ VARCHAR, LONG, RAW ne sont pas des types standardisés, leur utilisation limite les possibilités de manipulation des relations qui les contiennent.

En se restreignant à la clause "number", il est possible de déclarer tous les types numériques:

number: un nombre réel

number(m): un nombre entier de m chiffres

number(m,n): un nombre décimal de m chiffres avec n décimales où $m \geq n$

Le choix des longueurs des chaînes de caractères dépend entièrement du champs d'application, la longueur maximum des symboles de chaque domaine est une borne acceptable.

Reprenons notre deuxième modélisation est traduisons-la:

```
CREATE TABLE CLIENTS (
    NUM_CLIENT NUMBER (6) not null ,
    NOM CHAR (20),
    PRENOM CHAR (20),
    ADRESSE CHAR (40))
```

```
CREATE TABLE CHAMBRES (
    NUM_CHAMBRE NUMBER (2),
    PRIX NUMBER (8,2),
    NBR_LITS NUMBER (1),
    NBR_PERS NUMBER (1),
    CONFORT CHAR(6),
    EQUIPEMENT CHAR(3))
```

```
CREATE TABLE RESERVATIONS (
    NUM_CLIENT NUMBER (6),
    NUM_CHAMBRE NUMBER (2),
    DATE_ARR DATE,
    DATE_DEP DATE )
```

En soumettant ces trois déclarations à un système de gestion de bases de données, trois structures de tables vides seront créées. Dans chacune, il sera possible d'insérer des entités correspondant à la réalité du champ d'application. La description de données ne concerne que la partie structurale de la modélisation. Une table est le produit cartésien des domaines de chaque constituant, rien n'est spécifié sur le prédicat des relations. En effet, le prédicat concerne uniquement la sémantique du champ d'application modélisé. Si l'on trouve dans un système de gestion de base de données la table suivante:

```
CREATE TABLE PERSONNES (
    NUM_PERSONNE NUMBER (6) not null ,
    NOM CHAR (20),
    PRENOM CHAR (20),
    ADRESSE CHAR (40))
```

S'agit-il des personnes que je connais, de celles qui sont employées par une entreprise, de celles qui sont surveillées par la CIA, ... ? Seule la modélisation et plus précisément le prédicat de la relation PERSONNE peut nous donner cette indication. On trouvera par exemple:

```
Personnes(Num_Personne, Nom, Prenom, Adresse)
predicat: "||Personnes(a,b,c,d)|| Le membre du club de
Football de Rouen portant le numéro a se nomme c, b et
habite d"
```

Et que dire de la table suivante:

```
CREATE TABLE R1 (
  C1 NUMBER (6) not null ,
  C2 CHAR (20),
  C3 CHAR (20),
  C4 CHAR (40))
```

Ici les noms des constituants ne nous guident en rien. Cela peut être une table de personnes, une nomenclature de pièces détachée, un herbier, ... Il est indispensable de connaître la modélisation pour pouvoir interpréter une table. Car à une même structure de table peut correspondre une multitude de prédicats possibles. De plus, il est fortement recommandé de donner des noms clairs aux constituants des tables, les requêtes aux SGBD deviennent plus longues à écrire mais elles gagnent surtout en lisibilité.

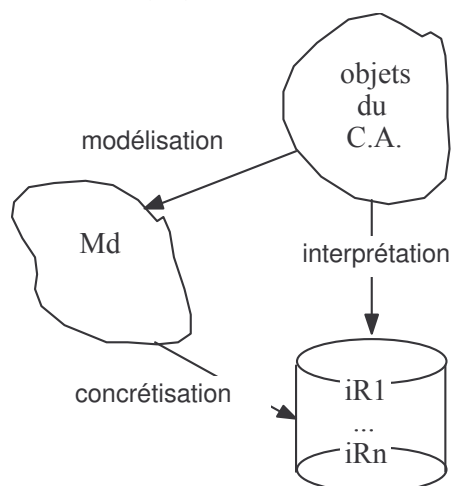


Figure 7-4: les trois étapes pour une base de données

Nous avons résumé dans la Figure 7-4, les concepts manipulés par l'élaboration d'une base de données. A partir des objets du champ d'application et par le processus de modélisation (cas d'utilisation, diagramme de classes, traduction en relations) nous obtenons le modèle Md. Ce modèle peut se concrétiser dans un ensemble d'instances (la base de données). Ces instances sont obtenues en interprétant les objets du champ d'application par rapport au prédicat de la modélisation.

Exercices

Nous reportons les exercices concernant ce chapitre au chapitre suivant !

8. Des classes aux relations

« Il joue de mieux en mieux car l'inquiétude est venue nuancer son caractère trop ingénu et optimiste »
L'empreinte de l'ange – Nancy Huston

Ce chapitre est le point de jonction entre les deux modèles. Pour conceptualiser, nous utilisons le modèle objet, pour stocker, nous utilisons le modèle relationnel. Dans ce chapitre, nous allons donner un ensemble de règles simples pour passer d'un modèle à l'autre.

Le passage de l'objet au relationnel est effectué lors de l'implémentation d'un champ d'application. Le chemin inverse du relationnel à l'objet s'effectue sur des systèmes d'information n'ayant pas été conçus avec une approche objet. Il s'agira dans ce cas de retrouver une interprétation ou de documenter cette application.

Rappelons les concepts des modèles utilisés.

Le modèle relationnel, notre cible, utilise structurellement trois concepts :

- La relation qui regroupe un ensemble d'attributs ;
- L'attribut qui est défini sur un domaine ;
- Le domaine qui est défini sur ensemble de valeurs.

Nous laissons de côté le prédicat de la relation. La notion de domaine est identique à celle de type pour les diagrammes de classes. Nous utiliserons donc des domaines identiques.

Notre traduction du modèle objet vers le modèle relationnel doit donc tenir compte de deux concepts la relation et les attributs.

Chaque classe devient une relation

La règle est assez simple, « chaque classe devient une relation ». Chaque attribut de la classe devient un attribut de la relation. Dans l'exemple suivant, pour la classe *Rectangles*, nous aurons une relation *Rectangles* avec deux attributs largeur et hauteur. En termes SQL, nous avons :

```
Create Table Rectangle (  
    largeur    number,  
    hauteur    number)
```

Rectangles
+largeur : real +hauteur : real
+périmètre() +surface() +agrandir() +diagonal()

Comment identifier les objets dans une relation ? Dans le monde des objets, il existe par définition un Oid (Object identifier) pour tout objet. Ce n'est pas le cas pour les relations. Soit il existe une clé naturelle, par exemple un numéro d'employé, (ou un groupe d'attributs), alors on choisira ces attributs comme clé de la relation. Sinon, on ajoute une clé artificiel à la relation.

Dans notre exemple de la classe *Rectangle*, il n'existe pas de clé (on peut bien sûr choisir largeur et hauteur comme clé). Il est donc nécessaire d'ajouter un attribut supplémentaire à la relation. On en profite pour indiquer gestionnaire de la base de données que cet attribut est la clé primaire en ajoutant derrière la déclaration de type les mots *primary key*. Nous obtenons donc la déclaration suivante :

```
Create Table Rectangle (
    Id_rectangle integer primary key,
    largeur      number,
    hauteur      number)
```

Que faire avec les méthodes ?

Il n'existe pas de solution unique pour les méthodes. L'avantage de la programmation objet est de rapprocher les données des traitements. Dans le cas des SGBD relationnels, les traitements doivent être décrit dans un langage procédural tel que Visual Basic, PL/SQL ou Java. Les traitements peuvent être extérieurs à la base de données ou stockés dans le serveur de la base de données.

Nous donnons ici trois possibilités pour traiter des méthodes sans faire appel à des traitements procéduraux.

Mémoriser les attributs calculés

Pour les méthodes qui ne modifient pas l'état de l'objet et dont but est de retourner une valeur sur l'état de l'objet, il est possible de rendre statique ces méthodes en leur substituant un attribut qui correspond à la valeur de l'objet. Pour notre exemple, nous obtenons la déclaration suivante :

```
Create Table Rectangle1 (
```

```

    Id_rectangle integer primary key,
    largeur      number,
    hauteur      number)
    surface      number,
    perimetre    number,
    diagonal     number)

```

L'inconvénient de ce choix réside dans l'aspect statique. Lors de modification des attributs dont dépendent ces méthodes, il faut mettre à jour les attributs correspondants aux méthodes. Dans notre cas, si l'on modifie la largeur d'un rectangle, il faut mettre à jour les attributs *surface*, *perimetre*, *diagonal*.

Utiliser des vues

Nous consacrons un chapitre complet aux vues. Mais dès maintenant, nous pouvons souligner que les vues peuvent être une solution dynamique aux méthodes qui ne modifient pas l'état de l'objet et dont but est de retourner une valeur sur l'état de l'objet. On peut dire que la vue va simuler une table semblable à celle de *Rectangle1*. Mais les attributs *surface*, *perimetre*, *diagonal* seront calculés à chaque fois que la table sera requise dans une sélection.

```

Create vue Rectangle2 as
  Select
    Id_rectangle,
    largeur,
    hauteur
    largeur*hauteur surface,
    2*(largeur+hauteur) perimetre
    sqrt(largeur*largeur+hauteur*hauteur) diagonal)
from Rectangle

```

Utiliser les méthodes de mise à jour

Nous consacrons aussi un chapitre à la mise à jour des relations. Et cette technique peut être parfois utilisée pour écrire des scripts de mise à jour correspondant à une méthode modifiant l'état de l'objet. Sans entrer dans les détails, nous pouvons transcrire la méthode *doubler* de *Rectangle* dans la requête SQL suivante :

```

Update Rectangle
  Set largeur=2*largeur, hauteur=2*hauteur
  Where num_rectangle= ...

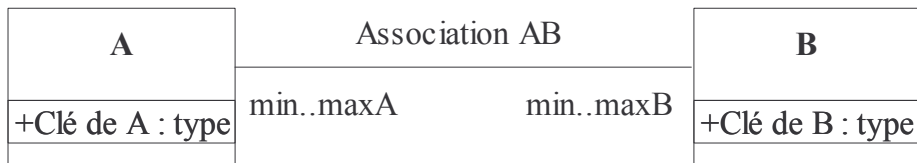
```

Nous verrons qu'il existe pratiquement toujours une solution à la transcription des méthodes que l'utilisation d'un langage procédural permet

de traiter tous les cas. Mais ceci ne remplace pas complètement les méthodes surtout dans les aspects d'héritage !

Traduire les associations

L'objectif est de mémoriser les liens entre les objets de A et de B. La solution à cette mémorisation dépend de la cardinalité de l'association qui lie A et B, plus exactement de ses maximums.



Le tableau suivant examine les cas possibles concernant le maximum des cardinalités de A et de B. Si un des deux maximums est égal à 1, alors il est possible de simplement ajouter une des clés dans une des relations. Si les deux maximums sont plus grands que 1, alors il faut créer une nouvelle relation.

MaxA--> MaxB y	1	>1
1	si la <i>clé de A</i> = la <i>clé de B</i> , ne rien faire Ajouter la <i>clé de B</i> dans la relation de A comme attribut ou Ajouter la <i>clé de A</i> dans la relation de B comme attribut	Ajouter la <i>clé de B</i> dans la relation de A comme attribut
>1	Ajouter la <i>clé de A</i> dans la relation de B comme attribut	Créer une relation AB ayant comme attribut la clé de A et la clé de B

Examinons des exemples de ces cas

Un des maximum est égal à un

Dans l'exemple suivant une ville est rattachée à un canton au plus, par contre un canton peut posséder plusieurs villes.

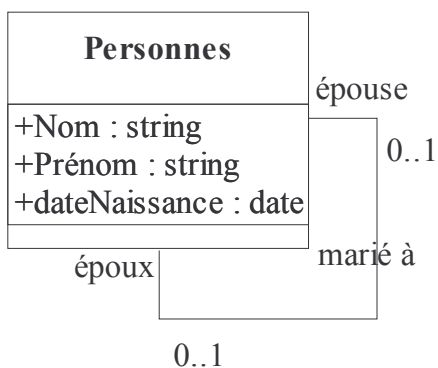


Nous créons d'abord les relations *Cantons* et *Villes* directement à partir des classes. *NomCanton* et *NomVille* sont les clés des classes. Comme la cardinalité est de 1 au maximum, nous devons ajouter, à la relation *Villes*, l'attribut *Rattache*. Nous pouvons ajouter que l'attribut *Rattache* doit toujours faire référence à une valeur dans la relation *Cantons* (Cet aspect sera plus amplement discuté dans le chapitre sur les règles d'intégrité).

```
Create Table Cantons(
    NomCanton varchar(20) primary key,
    Surface number)

Create Table Villes(
    NomVille varchar(20) primary key,
    Population number
    Rattache varchar(20)
        references Cantons(NomCanton) )
```

Dans l'exemple suivant, nous avons le cas d'une association qui lie des objets de la même classe. La classe *Personnes* n'a pas de clé (*Nom* et *Prénom* ne forment pas une clé sûre car il existe toujours la possibilité de doublons). Nous ajoutons donc l'attribut *Id_personne* supplémentaire à la Relation *Personnes*.

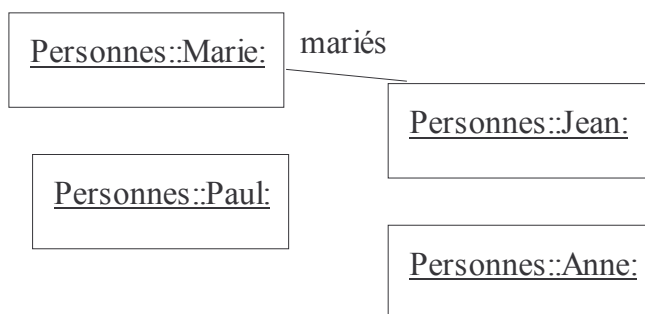


L'association *marié à* sera transcrite par l'ajout d'un attribut *Conjoint* faisant référence à une personne existante. Nous obtenons finalement le schéma suivant :

```
Create Table Personne(
    Id_personne number primary key, ,
    Nom varchar(20) ,
    Prenom varchar(20) ,
    DateNaiss date,
```

```
Conjoint number references Personne(Id_personne)
```

Les objets et les liens de notre exemple, se représenteront dans la table sous la forme suivante

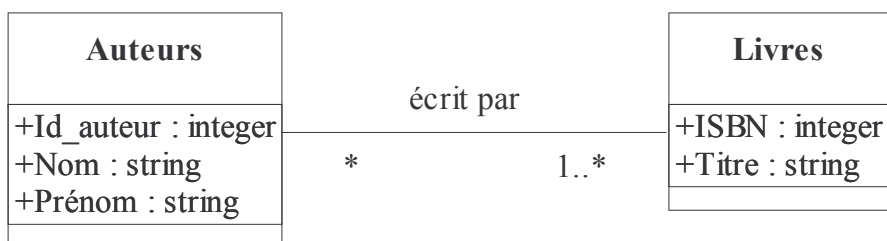


Id_personne	Nom	Prenom	DateNaiss	Conjoint
11		Marie		13
12		Paul		
13		Jean		11
14		Anne		

On remarquera que la maintenance des informations Conjoint demande une attention accrue, car cette association requiert la modification de deux lignes simultanément.

Les deux maximum sont plus grand que un

Dans l'exemple suivant un livre peut être associé à plusieurs auteurs et un auteur peut avoir écrit plusieurs livres. Manifestement pour traiter cette multiplicité de liens, il n'est pas possible de simplement ajouter un attribut dans une relation ou l'autre. Il faut ajouter une relation pour mémoriser ces liens.



Nous créons d'abord les relations *Auteurs* et *Livres* avec leur clé respective.

```

Create Table Auteurs(
  Id_auteur number primary key,
  Nom varchar(20),

```

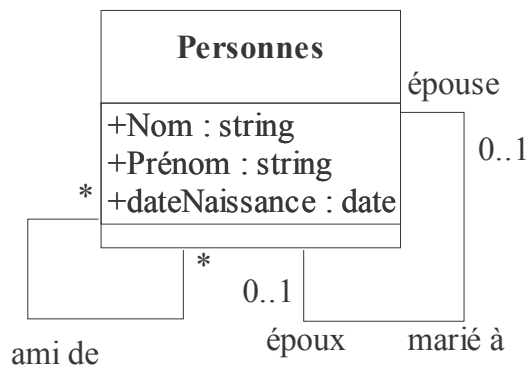
```
Prenom varchar(20))
```

```
Create Table Livres(
  ISBN number primary key,,
  Titre varchar(20))
```

Ensuite, nous créons la relation *EcritPar* pour transcrire l'association. Remarquons que la clé primaire de cette relation est formée par l'ensemble de ces attributs.

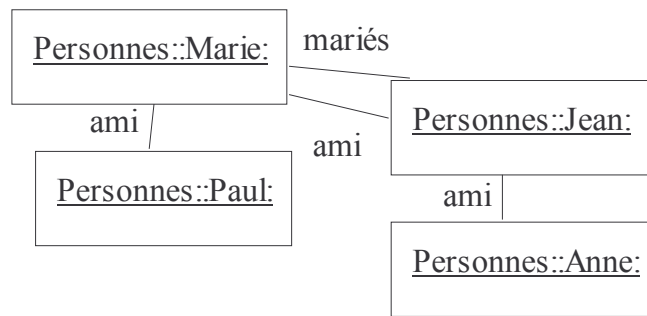
```
Create Table EcritPar(
  Id_auteur number references Auteurs,
  ISBN number references Livres,
  primary key(Id_auteur, ISBN))
```

Reprenons l'exemple de la classe *Personnes*, ajoutons lui une association *ami de*. Ici aussi, l'ajout d'un attribut ne suffit pas, il faut une nouvelle relation que nous nommons *Ami_de*.



```
Create Table Ami_de(
  Id_personne number references Personne,
  Id_ami number references Personne(id_personne),
  primary key(Id_personne, Id_ami))
```

Etendons l'exemple d'objets et de liens. Nous montrons dans la table comment nous devons tenir compte des liens d'amitiés.



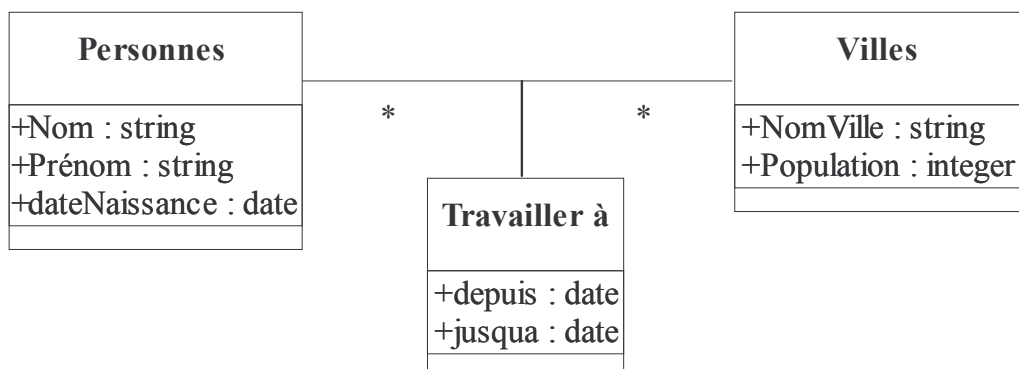
Id_personne	Id_ami
12	11
11	12
12	13
13	12
13	14
14	13

A nouveau, nous attirons votre attention sur la difficulté à maintenir la symétrie et éventuellement la transitivité (ce que nous ne faisons pas).

Association attribuée

L'association attribuée est traitée comme l'association à laquelle elle est attachée. Dans le cas où il a suffi d'ajouter un attribut clé dans une des deux relations, il suffit aussi d'ajouter les attributs de l'association attribuée dans cette même relation. Dans le cas où il a fallu créer une nouvelle relation, il faut ajouter les attributs de l'association attribuée dans cette relation.

Examinons l'exemple suivant :



Nous créons les relations Personnes et Villes. Pour tenir compte de l'association, nous créons la relation *Travailler_a* avec les attributs clés des

relations *Personnes* et *Villes*. Finalement, nous ajoutons les attributs *Depuis* et *Jusqua*.

```

Create Table Personnes (
    Id_personne number primary key,
    Nom varchar(20),
    Prenom varchar(20),
    DateNaiss date)

Create Table Villes (
    NomVille varchar(20) primary key,
    Population number)

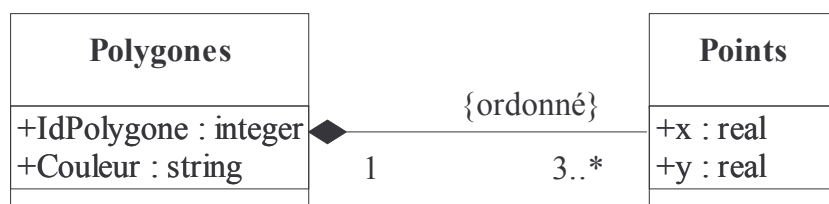
Create Table Travailler_a (
    Id_personne number references Personnes,
    NomVille varchar(20) references Villes,
    Depuis date,
    Jusqua date,
    primary key(Id_personne, NomVille))

```

Agrégation et composition

L'agrégation et la composition se traitent comme les associations. (La dépendance entre les classes est souvent une indication pour l'utilisation du *delete cascade*, pour plus de détail voir le chapitre sur les règles d'intégrité).

Dans l'exemple suivant, les polygones sont composés de points et en plus il existe une contrainte d'ordre sur les points le premier, second, etc. point du polygones. Cette contrainte d'ordre est traduite par un attribut *num_ordre* qui permet d'ordonner les points. On remarquera aussi que la relation Points ne possède pas de clé propre, la clé est formée par les attributs. *IdPolygone* et *num_ordre*.



```

Create Table Polygones (
    IdPolygone number primary key,
    Couleur varchar(20))

Create Table Points (
    IdPolygone number references Polygones,
    num_ordre number,
    x number,
    y number,
    primary key (IdPolygone, num_ordre))

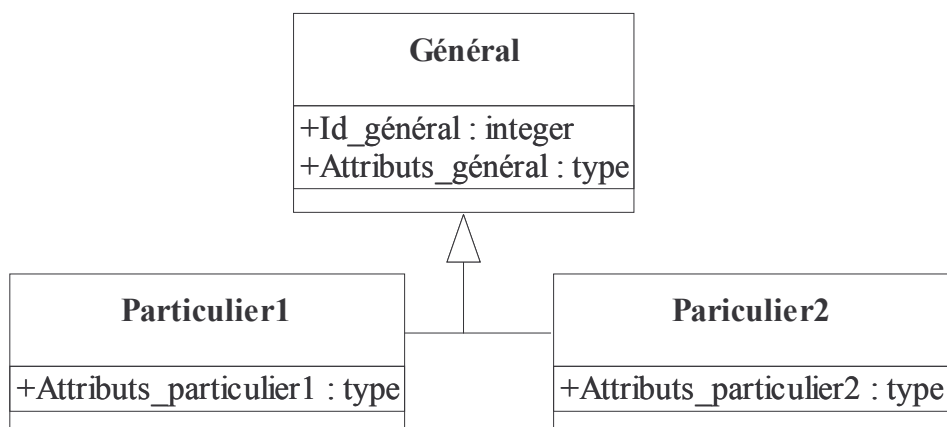
```

Généralisation

La généralisation ou l'héritage est un concept qui n'a pas d'équivalent dans le modèle relationnel. Il faut donc faire un choix entre regrouper toutes les particularité dans une seule relation ou conserver chaque particularité dans une relation propre. Examinons ces deux solutions

Tout dans un

Cette solution consiste à mettre tous les attributs dans la même table, à ajouter un constituant donnant le genre de l'objet.



```

create table Général (
    Id_général number primary key,
    genre varchar(20), -- valeurs possibles G, P1, P2
    attributs_de_général ...,
    ...
    attributs_de_particulier1...,

```

```
...  
attributs_de_particulier2...,  
...)
```

Les attributs non utilisés sont laissés à *null*. On peut créer une vue pour chaque classe pour retrouver exactement la description d'une classe particulière.

Il est important de voir que cette solution correspond conceptuellement au fait que l'on considère les classes *Général*, *Particulier1* et *Particulier2* comme une seule classe. Cette agrégation fait perdre de la finesse à la modélisation, surtout si les sous-classes entretenaient des associations qui leurs étaient propres.

Chacun à sa place

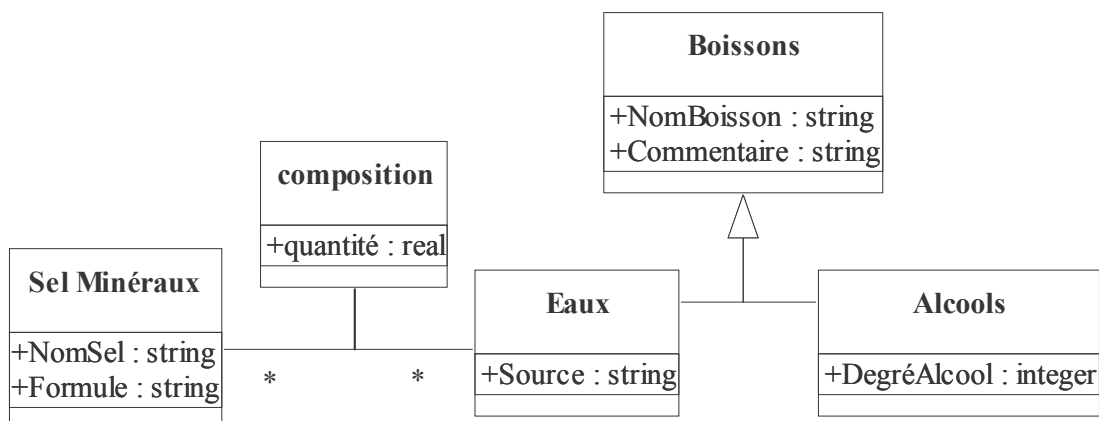
Dans cette solution, chaque classe devient une relation. On ajoute éventuellement un attribut de genre dans la relation *Général*.

```
create table Général(  
    id_général number primary key,  
    genre varchar(20), -- valeurs possibles G, P1, P2  
    attribut_de_général ...)  
  
create table Particulier1(  
    id_général number references Général,  
    attribut_de_particulier2 ...)  
  
create table Particulier2(  
    id_général number references Général,  
    attribut_de_particulier2 ...)
```

Ici le problème devient celui de l'éclatement de l'information. On peut aussi créer une vue pour chaque classe pour retrouver exactement la description d'une classe particulière.

Il est important de voir que cette solution correspond conceptuellement au fait que l'on considère les classes *Général*, *Particulier1* et *Particulier2* liées par des associations. Cette substitution fait perdre de la finesse à la modélisation, mais si les sous-classes entretenaient des associations qui leurs étaient propres, celles-ci restent distinctes.

Examinons l'exemple suivant avec les deux solutions



Avec la solution tout dans un, nous retrouvons tous les attributs dans la relation *Boissons*. L'association *composition* est liée maintenant directement à *Boisson*, il serait donc possible de donner la composition d'un alcool. Pour se préserver de ceci, il faudra ajouter des contraintes.

```

Create table Boissons (
  Nom_boisson varchar(20) primary key,
  Commentaire varchar(1000),
  Genre char(7), -- Boisson, Eau, Alcool
  Source varchar(20),
  Teneur_en_alcool number)
  
```

```

Create table Sel_mineral (
  Nom_sel varchar(20) primary key,
  Formule varchar(100))
  
```

```

Create table Composition (
  Nom_boisson varchar(20) references Boissons,
  Nom_sel varchar(20) references Sel_mineral,
  Quantite_mg number,
  primary key (Nom_boisson, Nom_sel))
  
```

Nous anticipons un peu sur le chapitre des vues et nous donnons un exemple de vue qui permet de reconstituer le concept de classe *Eaux*:

```

Create view Eaux as
  Select Nom_boisson, Commentaire, Source
  From Boisson
  Where genre = 'Eau'
  
```

La solution chacun à sa place, nous apporte une relation pour chaque classe. L'association *composition* est liée maintenant directement à Eaux, il n'est donc plus possible de donner la composition d'un alcool. Par contre, une boisson doit se gérer dans plusieurs relations ce qui complexifie la gestion des modifications de la base de données.

```
Create table Boisson(  
    Nom_boisson varchar(20) primary key,  
    Commentaire varchar(1000),  
    Genre char(7)) -- Boisson, Eau, Alcool
```

```
Create table Eaux(  
    Nom_boisson varchar(20) references Boisson,  
    Source varchar(20),  
    primary key Nom_boisson)
```

```
Create table Alcools (  
    Nom_boisson varchar(20) references Boisson,  
    Teneur_en_alcool number,  
    primary key Nom_boisson)
```

```
Create table Sel_mineral(  
    Nom_sel varchar(20) primary key,  
    Formule varchar(100))
```

```
Create table Composition(  
    Nom_boisson varchar(20) references Eaux,  
    Nom_sel varchar(20) references Sel_mineral,  
    Quantite_mg number  
    primary key (Nom_boisson, Nom_sel))
```

Exemple de vue:

```
Create view Mineral as  
    Select m.Nom_boisson, b.Commentaire, m.Source  
    From Boisson b, Mineral m  
    Where m.nom_boisson=b.nom_boisson
```

Tableau résumant les traductions

Modèle Objet	Modèle Relationnel
Classe	Relation
Attribut	Attribut
Méthode	(vue, fonction, attribut, procédure)
Association	
0..1, 1..1	Attribut
0..n, n..m	Relation
Association attribuée	(voir la cardinalité de l'association)
Agrégation	
0..1, 1..1	Attribut (+valeur null)
0..n, n..m	Relation
Généralisation	
Tout dans un	Attribut (+valeur null)
Chacun à sa place	Relation

Nous sommes maintenant en mesure de lire et d'interpréter une description de base de données accompagnée de sa modélisation. La prochaine étape est l'interrogation d'une base de données.

Exercices

Questions sur TT³

1. En utilisant les règles de traductions passer du diagramme des classes à celui des relations.
2. A partir de l'ensemble des relations de TT³, En utilisant le langage de modélisation, Donnez les prédicats des relations
3. Remplir des tables correspondant aux instances des relations afin d'exprimer le texte qui suit: "Tout se passe le premier de l'an (NoJour=1), dans la tranche horaire A. Le véhicule immatriculé No 11 portant le numéro de chasis 2493 est conduit par le chauffeur Marcel Dupont. Il est le chauffeur numéro 7 et va chercher son taxi à la station No 4. Il possède les permis B1 et C. Il fait le plein du véhicule avec 30 litres de super pour 320 Kilomètres roulés. Une consommation modérée pour un modèle BUICK automatique de 5 places"
4. Exprimer à l'aide du langage de description de données SQL, les relations précédentes.

Réponses sur TT³

Relations de TT³

Véhicule(noChassis, noPlaque, miseEnService, modèle, noStation)

Type(modèle,nbPlaces,catégorie,typeCarburant, automatique, poids)

Chauffeur(noChauffeur,nom, prénom, adresse, noStation)

Carburant(noPlaque, noJour, kilometrage, litres, typeCarburant)

Entretien(noChassis, noJour, description)

Permis(noChauffeur, catégorie)

Planning(noChauffeur, noChassis, noJour, trancheHoraire)

Station(noZone, noStation)

Distance(heure, zoneDe, zoneA, tempsParcours)

Situation(noChassis, noZone)

Prédicats de TT³

Véhicule(noChassis, noPlaque, miseEnService, modèle, noStation)

predicat: "|| Véhicule(nc,np,s,m,ns)|| Le véhicule portant le numéro de Chassis *nc* et immatriculé *np* a été mis en service le *s*, il est du modèle *m* et appartient à la station *ns*"

Type(modèle,nbPlaces,catégorie,typeCarburant, automatique, poids)

predicat: "|| Type(m,s,c,tc,a,p)|| Le modèle *m* de véhicule peut contenir *s* personnes, pèse *p* [kg], consomme du carburant *tc*, la boîte à vitesse est *a*, le permis *c* est nécessaire pour le conduire"

Chauffeur(noChauffeur, nom, prénom, adresse, noStation)

predicat: "|| Chauffeur(nch,n,p,a,ns)|| Le chauffeur portant le nom *n* est le prénom *p* habite *a*. il est identifié par le numéro *nch* et il est assigné à la station *ns*"

Carburant(noPlaque, noJour, kilometrage, litres, typeCarburant)

predicat: "|| Carburant(np,j,k,l,tc)|| Le jour *j*, le véhicule immatriculé *np* a effectué un plein de *l* litres de *tc* carburant après avoir roulé *k* kilomètres"

Entretien(noChassis, noJour, description)

predicat: "|| Entretien(nc, j, d)|| Le jour *j*, le véhicule avec numéro de chassis *nc* a subi l'entretien *d*"

Permis(noChauffeur, catégorie)

predicat: "|| Permis(nch,c)|| Le chauffeur portant le numéro *nch* possède un permis *c*"

Planning(noChauffeur, noChassis, noJour, trancheHoraire)

predicat: " | | Planning(nch,nc,j,h) | | Le chauffeur portant le numéro nch doit conduire le véhicule portant le numéro de chassis nc le jour j durant la tranche horaire h"

Station(noZone, noStation)

predicat: " | | Station(nZ,nS) | | La station nS se trouve dans la zone nZ"

Distance(heure, zoneDe, zoneA, tempsParcours)

predicat: " | | Distance(h,zd,za,t) | | A l'heure h, pour aller de zd à za, on estime qu'il faut t minutes"

Situation(noChassis, noZone)

predicat: " | | Situation(nc,nz) | | Actuellement, le véhicule portant le numéro de chassis nc attend dans la zone nz"

Sémantique : Entités du champ d'application

Véhicule(noChassis	noPlaque	miseEnService	modèle	noStation)
	2493	11		Buick	4

Type(modèle	nbPlaces	catégorie	typeCarburant	automatique	poids)
	Buick	5			oui	

Carburant(noPlaque	noJour	kilometrage	litres	typeCarburant)
	11	1	320	30	super

Entretien(noChassis	noJour	description)

Chauffeur(noChauffeur	nom	prénom	adresse	noStation)
	7	Dupont	Marcel		4

Permis(noChauffeur	catégorie)
	7	b1
	7	c

Planning(noChauffeur	noChassis	noJour	trancheHoraire)
	7	2493	1	a

Station(noZone	noStation)

On constate que certaines entités sont incomplètes par rapport à l'énoncé.

Création de table

On commence par donner un domaine "SQL" à chaque constituant:

adresse	texte	varchar(80)
automatique	booléen	char(1)
catégorie	mot (A,B1,B2,C ...)	char(2)
description	texte	varchar(240)
heure	entier [0..23]	number(2)
kilometrage	entier positif	number
litres	réel [0..500]	number(3)
miseEnService	date	date
modèle	mot (mercedes300,...)	varchar(12)
nbPlaces	entier [4..80]	number(2)
noChassis	entier positif	number
noChauffeur	entier positif	number
noJour	entier [1..366]	number(3)
nom	mot (Dupont, ...)	varchar(24)
noPlaque	entier [1..9999]	number
noStation	entier [1..6]	number(1)
noZone	entier [1..99]	number(2)
poids	entier [500..15000]	number(5)
prénom	mot (Jean, Marie, ...)	varchar(24)
tempsParcours	entier positif	number(3)
trancheHoraire	mot (A,B,C)	char(1)
typeCarburant	mot (super,)	varchar(12)
zoneA	entier [1..99]	number(2)
zoneDe	entier [1..99]	number(2)

```
CREATE TABLE Vehicule (
    noChassis    number,
    noPlaque     number,
    miseEnService date,
    modele       varchar(12),
    noStation    number(1))
```

```
CREATE TABLE Type (
    modele       varchar(12),
    nbPlaces     number(2),
    categorie     char(2),
    typeCarburant char(12),
    automatique  char(1),
    poids        number(5))
```

```
CREATE TABLE Carburant (
    noJour          number(3),
    kilometrage    number(3),
    litres         number(3),
    typeCarburant  varchar(12))

CREATE TABLE Entretien(
    noJour          number(3),
    description     varchar(240))

CREATE TABLE Chauffeur(
    nom             varchar(24),
    prenom         varchar(24),
    adresse        varchar(80),
    noStation      number(1))

CREATE TABLE Permis(
    noChauffeur    number,
    categorie       char(2))

CREATE TABLE Planning(
    noChassis      number,
    noJour         number(3),
    trancheHoraire char(1))

CREATE TABLE Station(
    noZone         number(2),
    noStation      number(1))

CREATE TABLE Distance(
    zoneDe         number(2),
    zoneA          number(2),
    tempsParcours number(3))

CREATE TABLE Situation(
    noChassis      number,
    noZone         number(2))
```


9. Algèbre relationnelle

"Le processus de la pensée dans son ensemble nous est encore relativement mystérieux, mais je crois que toutes les tentatives de création de machines pensantes nous seront d'une grande aide pour découvrir comment nous pensons nous-mêmes" Alan Turing Cité par Andrew Hodges dans *Alan Turing ou l'énigme de l'intelligence*

Avant d'examiner, l'interrogation en SQL d'une base de données, nous allons définir l'algèbre relationnelle d'une part, car elle est le fondement des mécanismes mis en oeuvre dans l'interrogation SQL. D'autre part car elle nous permettra d'explicitier simplement des règles d'intégrité lorsque nous aborderons ce chapitre.

L'union, la différence, la projection, le produit cartésien et la sélection sont les cinq opérations de base sur les relations qui définissent l'algèbre relationnelle. Une des propriétés d'une algèbre est que si l'on utilise une opération de celle-ci sur des éléments pour laquelle elle est définie, le résultat est aussi un élément de cet univers. Dans notre cas, cela signifie que le résultat d'une expression de l'algèbre relationnelle définie sur des relations est aussi une relation. L'algèbre relationnelle, nous permet donc de manipuler les relations et d'obtenir de nouvelles relations. L'intérêt de l'algèbre relationnelle réside dans le fait que les expressions algébriques peuvent être interprétées comme la spécification d'interrogation sur une base de données.

Opérations algébriques

Nous utilisons, pour les exemples, les deux instances de relation suivantes:

R		
A	B	C
a	b	c
d	a	f
c	b	d

S		
D	E	F
b	g	a
d	a	f

Les opérations algébriques ont un résultat qui est une relation. Ils sont donc définis selon les deux dimensions, l'une structurale qui indique le produit cartésien, les constituants et les domaines du résultat et l'autre sémantique qui spécifie le prédicat du résultat. Ces définitions sont basées sur les relations impliquées dans l'opération.

En mathématique, il n'y a pas de distinction entre la relation et son instance. La définition des opérations algébriques est positionnelle par rapport aux constituants, C_{R2} définit le 2^{ième} constituant de la relation R. L'arité d'une relation R correspond à la cardinalité de R^+ soit : $|R^+|$. Dans notre cas, l'arité de R et de S est égale à 3. La cardinalité de la relation correspond au nombre d'entités qu'elle contient (notée $|R|$). La cardinalité de R est 3 et celle de S est 2.

L'union

Définition:

Soit R et S des relations telles que $|R^+| = |S^+|$ (même arité)

Alors $T = R \cup S$ dénote l'union de R et S

où

1) $C_{Tj} \in T^+$ tel que $\text{dom}(C_{Tj}) = \text{dom}(C_{Rj}) \cup \text{dom}(C_{Sj})$

2) $||T|| = ||R|| \vee ||S||$, en renommant les constituants s'ils sont différents dans R et S.

Exemple:

T=R ∪ S		
a	b	c
d	a	f
c	b	d
b	g	a

La cardinalité du résultat est $|iT| \leq |iR| + |iS|$.

Au niveau des instances on aura :

Un n-uplet t sera une entité de $R \cup S$ si et seulement si $(||R|| \vee ||S||)(t) = \text{vrai}$, c-à-d si et seulement si $||R||(t) = \text{vrai} \vee ||S||(t) = \text{vrai}$ donc si et seulement si t est une entité de R ou une entité de S. En d'autres termes : on obtiendra l'instance de $R \cup S$ correspondante en faisant $iR \cup iS$.

Les domaines des constituants du résultat sont formés par l'union des domaines des constituants des opérandes (en respectant leur position). Le prédicat du résultat spécifie que les entités sont des entités appartenant à R, plus celles appartenant à S.

L'union permet de définir de nouvelles relations qui ont pour prédicat le *ou* logique des prédicats des relations opérandes. Par exemple, les personnes absentes sont les personnes malades ou les personnes en vacances, soit:

PERS_ABSENTES = PERS_MALADES \cup PERS_EN_VACANCES

La différence

Définition:

Soit R et S des relations telles que $|R^+| = |S^+|$ (même arité)

Alors $T=R - S$ dénote la *différence de S sur R*

où

$$1) T^+ = R^+$$

$$2) ||T|| = ||R|| \wedge \neg ||S||$$

Exemple:

T=R - S		
A	B	C
a	b	c
c	b	d

La cardinalité du résultat est $|iT| \leq |iR|$

Pour les instances, on a: $iT = iR - iS$

Les domaines des constituants du résultat sont les domaines des constituants du premier opérande. Le prédicat du résultat spécifie que les entités sont des entités appartenant à R et n'appartenant pas à S.

La différence permet de définir de nouvelles relations qui vérifient le prédicat logique du premier opérande et qui ne vérifient pas celui du deuxième. Par exemple, les plantes non comestibles sont des plantes sauf celles qui sont comestibles, soit:

$$\text{NON_COMESTIBLE} = \text{PLANTES} - \text{PLANTES_COMESTIBLES}$$

La projection

Définition:

Soit R une relation, $X = \{X_1, \dots, X_k\}$ et $X \subseteq R^+$

Alors $T=R[X]$ dénote la *projection de R sur X*⁷

où

$$1) T^+ = X$$

$$2) ||T|| = \exists X_1, \dots, X_k. ||R||$$

Exemple:

T=R[A,C]	
A	C
a	c
d	f
c	d

Remarques:

La cardinalité du résultat est $|iT| \leq |iR|$

Pour les instances, on a: $iT = \{t \in \times T \mid \exists r \in iR. r.X = t\}$.

⁷ autre notation courante: $R[X_1, X_2, \dots, X_n] = \pi_{X_1, X_2, \dots, X_n}(R)$

Les domaines des constituants du résultat sont équivalents à ceux de la relation opérande. Le prédicat du résultat est restreint aux constituants de la projection.

La projection permet de conserver les constituants "intéressants". Par exemple, "les couleurs des plantes comestibles" est défini par la projection des plantes comestibles sur le constituant COULEUR(en supposant qu'il existe), soit

COULEUR_DES_COMESTIBLE= PLANTES_COMESTIBLES[COULEUR]

Par extension, on peut utiliser l'opérateur de projection sur une entité.

$r.[X]$ est équivalent à $r.X$.

Le produit

Définition:

Soit R et S des relations telles que $R^+ \cap S^+ = \emptyset$

Alors $T=R \times S$ dénote le *produit (cartésien) de S et R*

où

1) $T^+ = R^+ \cup S^+$

2) $||T|| = ||R|| \wedge ||S||$

Exemple:

T=R × S					
A	B	C	D	E	F
a	b	c	b	g	a
a	b	c	d	a	f
d	a	f	b	g	a
d	a	f	d	a	f
c	b	d	b	g	a
c	b	d	d	a	f

La cardinalité du résultat est $|iT| = |iR| * |iS|$

L'instance iT de $T = R \times S$ correspondant aux instance iR et iS de R et S est $\{t \in iT \mid t.R^+ \in iR \wedge t.S^+ \in iS\}$.

Les constituants du résultat sont formés par l'union des constituants des opérandes. Le prédicat du résultat spécifie que les entités projetées sur R^+ sont des entités appartenant à R et projetées sur S^+ sont des entités appartenant à S .

Le produit permet de définir des nouvelles relations qui sont le produit cartésien de deux autres relations. Par exemple, un agenda sera le produit cartésien des jours de la semaine et des heures ouvrables, soit:

AGENDA = JOURS_SEMAINE × HEURES_OUVRABLES

La sélection

Définition:

Soit R une relation et F une formule comprenant:

- i) des opérandes (des noms de constituants de R^+)
- ii) des opérateurs de comparaison (<, >, =, ...)
- iii) des connecteurs logiques (\wedge, \vee)

Alors $T = (*F) R$ dénote la *sélection de R par F*⁸

où

$$1) T^+ = R^+$$

$$2) ||T|| = ||R||_{\wedge F}$$

Exemple: ici la formule sélectionne les entités dont la projection sur B est égal à "b"

(* B='b') R		
A	B	C
a	b	c
c	b	d

La cardinalité du résultat est $|iT| \leq |iR|$.

On a pour l'instance iT, $iT = \{t \in iR. F(t)\}$

Les domaines des constituants du résultat sont équivalents à ceux de la relation opérande. Le prédicat du résultat est celui de la relation opérande restreint à la fonction de sélection.

La sélection permet de définir de nouvelles relations qui restreignent les entités de la relation opérande par une fonction de sélection sur ses constituants. Par exemple, les plantes vertes sont des plantes dont la couleur est verte, soit:

PLANTES_VERTES = (*COULEUR=verte) PLANTES

Jointures

Nous avons vu les cinq opérations de base de l'algèbre relationnelle, nous allons examiner d'autres opérations dérivées de celles-ci. L'opération de *jointure* est l'opération la plus fondamentale dans la manipulation des relations, elle permet d'associer deux relations par des constituants ayant une même sémantique. Dans les exemples qui suivent nous utiliserons, la relation R et S suivantes (les constituants sont indicés par le nom de la relation):

R			S	
A _R	B _R	C _R	C _S	D _S
a	b	1	1	d

⁸autre notation courante : $(*F) R = \sigma_F(R)$

d	a	6
c	b	3

3	e
1	b

La Θ jointure (la théta-jointure)

Définition:

Soit R et S des relations telles que $R^+ \cap S^+ = \emptyset$

Alors $T=R \bowtie_{C_R \Theta C_S} S$ dénote la Θ jointure de S et R .

Θ est un opérateur logique de comparaison (=, <, >, ...)

où

$$1) T^+ = R^+ \cup S^+$$

$$2) ||T|| = ||R|| \wedge ||S|| \wedge C_R \Theta C_S$$

Exemple: les C_R sont strictement plus petits que les C_S

$$T=R \bowtie_{C_R \Theta C_S} S$$

A	B	C_R	C_S	D
a	b	1	3	e

La cardinalité du résultat est $|iT| \leq |iR| * |iS|$

Les constituants du résultat sont formés par l'union des constituants des opérands. Le prédicat du résultat spécifie que les entités projetées sur R^+ sont des entités appartenant à R et projetées sur S^+ sont des entités appartenant à S, et de plus que les entités vérifient la condition Θ pour les constituants spécifiés.

La Θ jointure permet de définir une nouvelle relation dont les entités respectent une condition sur les constituants de la jointure. Par exemple, supposons que le constituant SOL soit commun aux relations PLANTES et REGIONS, alors la Θ jointure de REGIONS et PLANTES par rapport à l'égalité de SOL dans chacune aura pour résultat toutes les plantes compatibles avec le sol d'une région, soit:

PLANTES_REGIONNAL = PLANTES $\bowtie_{SOL_{PLANTES}=SOL_{REGIONS}}$ REGIONS

On peut exprimer la Θ jointure, à l'aide des opérations de base:

$$R \bowtie_{C_R \Theta C_S} S = (*C_R \Theta C_S) (R \times S)$$

On peut étendre la condition à plusieurs constituants. Les entités du résultat doivent alors vérifier toutes les conditions:

$$R \bowtie_{(C_{R1} \Theta C_{S1}) \wedge (C_{R2} \Theta C_{S2}) \wedge \dots (C_{Rn} \Theta C_{Sn})} S$$

Variations sur la jointure

Il existe plusieurs variations de la jointure. Sur la base de l'inventaire effectué par P. Mishra et M.H. Eich [MISH92], nous en donnerons une

définition équivalente en terme des opérateurs de base de l'algèbre relationnelle.

Equi-jointure

L'opérateur Θ le plus couramment utilisé est celui de l'égalité; l'opération est alors appelée *équi-jointure*.

équi-jointure : $T=R \bowtie_{C_R=C_S} S$

A	B	C _R	C _S	D
a	b	1	1	d
a	b	1	1	b
c	b	3	3	e

L'équi-jointure est équivalente à l'expression algébrique:

$$R \bowtie_{C_R=C_S} S = (*_{C_R=C_S}) (R \times S)$$

Jointure naturelle

Les colonnes jointes ont toujours un contenu identique, on peut donc en supprimer une (celle de la deuxième relation). Dans ce cas, l'opération est appelée *jointure naturelle*, que l'on note R^*S , il faut cependant que les constituants à joindre portent le même nom.

jointure naturelle : $T=R^*S$

A	B	C	D
a	b	1	d
a	b	1	b
c	b	3	e

On appelle *charnière de R et S*, l'intersection de R^+ et S^+ notée RS^+

Soit les constituants (indités) S appartenant à la charnière :

$$SRS^+ = \{C_{S1}, C_{S2}, \dots, C_{Sn}\} \text{ et ceux de R à } RRS^+ = \{C_{R1}, C_{R2}, \dots, C_{Rn}\}$$

La jointure naturelle est équivalente à l'expression algébrique:

$$R^*S = (*_{(C_{R1}=C_{S1}) \wedge (C_{R2}=C_{S2}) \wedge \dots \wedge (C_{Rn}=C_{Sn})} (R \times S)) [(R^+ \cup S^+) - SRS^+]$$

Semi-jointure

Dans la jointure, les constituants de R et de S sont présents dans le résultat, dans la semi-jointure seuls ceux de R sont conservés, elle est notée :

$$R \ltimes_{C_R=C_S} S$$

semi-jointure : $T = R \ltimes_{C_R=C_S} S$

A	B	C
---	---	---

a	b	1
a	b	1
c	b	3

La semi-jointure est équivalente à l'expression algébrique:

$$R \bowtie_{C_R \Theta C_S} S = (*C_R \Theta C_S)(R \times S)[R^+]$$

Jointure externe

Dans la jointure, un certain nombre d'entités ne sont pas prises en considération. La jointure externe permet de rajouter au résultat les entités d'une des deux relations (ceci permet de travailler avec des données incomplètes ayant des valeurs nulles). On distinguera la *jointure externe gauche*, notée $R \ltimes S$ qui ajoute les entités de R complétées par des valeurs nulles pour les constituants de S; la *jointure externe droite*, notée $R \rtimes S$ qui ajoute les entités de S complétées par des valeurs nulles pour les constituants de R et *jointure externe complète*, notée $R \bowtie S$ qui ajoute les entités de R et de S complétées par des valeurs nulles pour les constituants respectifs non-définis.

semi-jointure gauche : $T = R \ltimes S$

A	B	C_R	C_S	D
a	b	1	1	d
a	b	1	1	b
c	b	3	3	e
d	a	6	\perp	\perp

La jointure externe gauche est équivalente à l'expression algébrique⁹:

$$R \ltimes S = ((*C_R = C_S)(R \times S)) \cup R \perp S$$

Expression algébrique

Nous avons vu isolément chaque opération, voyons maintenant comment nous pouvons former des expressions. Nous donnons premièrement le vocabulaire de nos expressions et ensuite la définition d'expressions algébriques bien formées.

Vocabulaire: trois catégories de symboles apparaissent dans les expressions algébriques

- i) les noms des relations appartenant à la modélisation
 $\{R_1, R_2, \dots, R_m\} = Md.$

⁹ Pour définir formellement cette opération, il faut étendre la définition de l'union au traitement des valeurs nulles. $R \perp S$ dénote la relation R étendue aux constituants de S avec des valeurs nulles.

- ii) les noms des constituants contenus dans $\cup R_i^+$, $i=1..m$
- iii) les formule F telle que:
 - 1) les constantes et les noms de constituants sont des atomes
 - 2) la forme "(atome θ_a atome)" est une formule
où $\theta_a \in \{<, >, =, \neq, \dots\}$
 - 3) la forme "(formule θ_l formule)" est une formule
où $\theta_l \in \{\wedge, \vee\}$
 - 4) la forme "(\neg formule)" est une formule
 - 5) rien d'autre n'est une formule

F^+ désigne l'ensemble des constituants utilisés dans une formule F

On dit qu'une expression (algébrique) est bien formée (ebf) si elle est obtenue par les règles suivantes:

- i) un nom de relation est une ebf.
- ii) si E_1 et E_2 sont des ebf alors
 - 1) $(E_1 \cup E_2)$, $(E_1 \cap E_2)$ et $(E_1 - E_2)$ sont des ebf si $|E_1^+| = |E_2^+|$
 - 2) $(E_1 \times E_2)$ est une ebf si $E_1^+ \cap E_2^+ = \emptyset$
 - 3) $(E_1 \bowtie (*F) E_2)$ est une ebf si $F^+ \subseteq E_1^+ \cup E_2^+$ et $E_1^+ \cap E_2^+ = \emptyset$
 - 4) $(E_1[X])$ est une ebf si $X \subseteq E_1^+$
 - 5) $((*F)E_1)$ est une ebf si $F^+ \subseteq E_1^+$
- iii) rien d'autre n'est une ebf

Remarques:

On peut supprimer le test de l'intersection vide entre deux relations si l'on indice le nom de tous les constituants d'une relation.

On peut supprimer des parenthèses si la sémantique de l'ebf est sans ambiguïté

Langage algébrique, un langage d'interrogation

Nous avons suggéré que les expressions algébriques sont la représentation de questions que l'on peut formuler sur un champ d'application. Examinons le processus d'interrogation. Dans l'idéal nous dirigerions directement vers le champ d'application notre interrogation en vue d'obtenir une réponse, mais la réalité est relativement "muette". Nous avons modélisé le champ d'application, en vue d'y effectuer des calculs symboliques, il faut donc transformer notre interrogation portant sur le champ d'application en une interrogation portant sur la modélisation. Cette interrogation correspond à une expression algébrique. Il suffit alors de concrétiser cette expression en la calculant avec les valeurs des instances de la base de données. le résultat est une instance qui doit être interprétée comme la réponse de la question initiale sur le champ d'application.

La qualité de cette réponse dépend des trois étapes suivantes:

- la modélisation est fidèle dans le contexte de la question

- les instances des relations sont des interprétations correctes du champ d'application
- la formulation de l'interrogation est sémantiquement correcte. En effet une expression algébrique bien formée correspond à une interrogation, mais pas forcément l'interrogation initiale.

Exemple d'interrogation

Pour faciliter la formulation des interrogations, nous pouvons représenter notre modélisation graphiquement. Deux types de nœuds sont représentés: les constituants et les relations. Les arêtes non orientées lient les constituants à la relation dont ils font partie. Pour le champ d'application Hôtel, nous obtenons le graphe de la Figure 9-2.

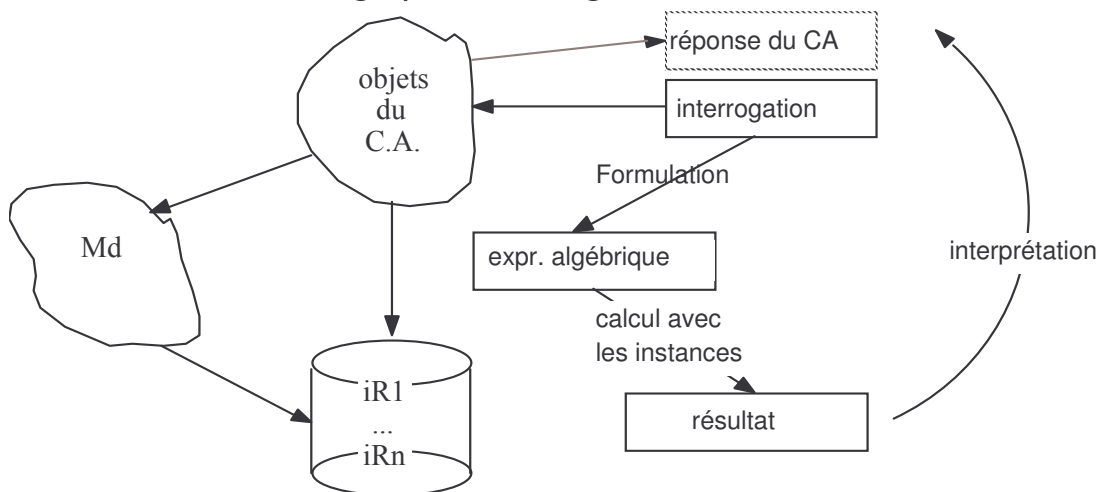


Figure 9-1 : interrogation avec une expression algébrique

Le processus de modélisation de l'interrogation suit le schéma suivant:

- spécifier les constituants de la sélection
- spécifier les constituants du résultat (projection)
- les étapes a) et b) définissent les relations impliquées dans l'expression algébrique.
- définir les jointures entre les relations

1) Les chambres avec bain et TV ?

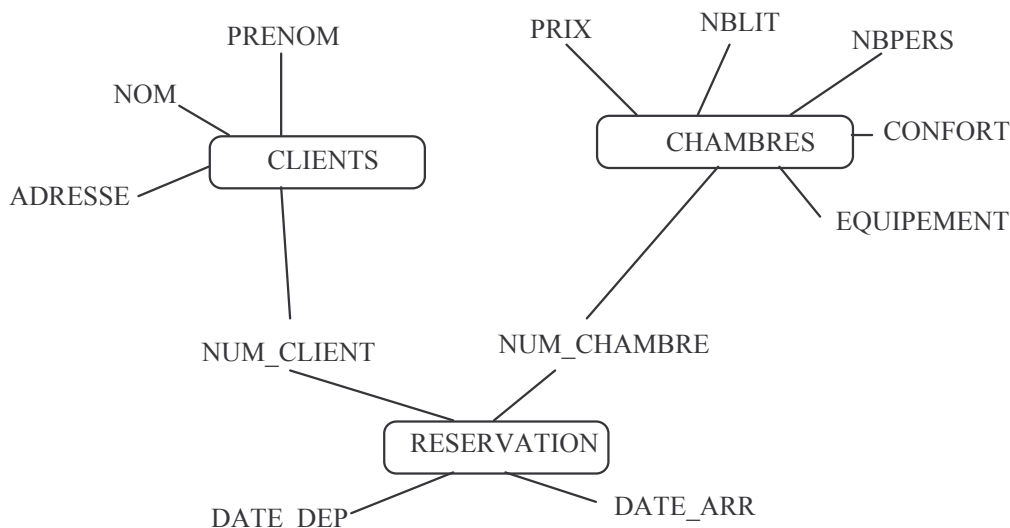


Figure 9-2 :modélisation graphique d'HOTEL

Les informations bain et TV appartiennent respectivement aux domaines des constituants Confort et Equipement, le résultat est défini par les constituants de la relation Chambres. Seule la relation chambre est impliquée dans cette interrogation, donc nous ne spécifions pas de jointure. Nous obtenons donc l'expression suivante:

(* ((Confort=bain)^(Equipement=TV))) Chambres

2) Les numéros des chambres et leur capacité ?

Seules les informations sur les constituants Numchambre et NbrPers, nous intéressent. Nous obtenons donc l'expression suivante:

(Chambres [NumChambre, NbrPers])

3) Les noms des clients ayant réservés une chambre pour le 25-12-89 ?

La sélection porte sur les constituants DateArr et DateDep de la relation Réservation. Le constituant du résultat Nom appartient à la relation Clients. Il faut donc définir le lien entre les relations Client et Réservation, nous utilisons ici la jointure naturelle portant sur le constituant NumClient commun aux deux relations.

(*((DateArr <= 25-12-89)^(DateDep >25-12-89))(Clients*Réservation))[Nom]

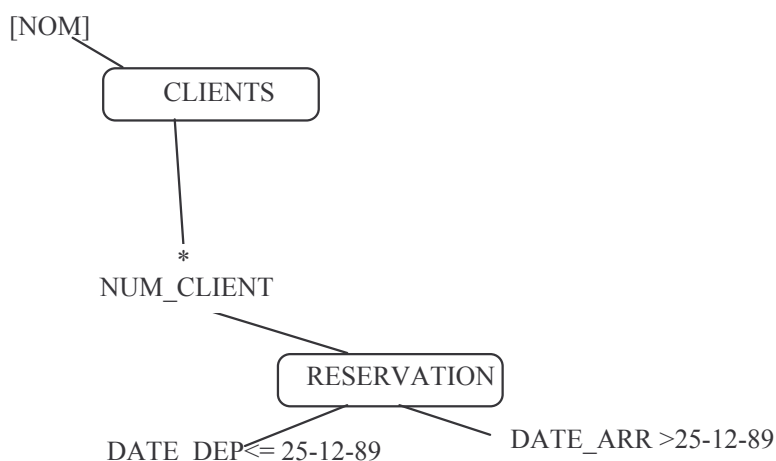


Figure 9-3 :modélisation graphique de la question: Les noms des clients ayant réservés une chambre pour le 25-12-89 ?

4) Le nom des clients et le confort des chambres qu'ils ont réservés?

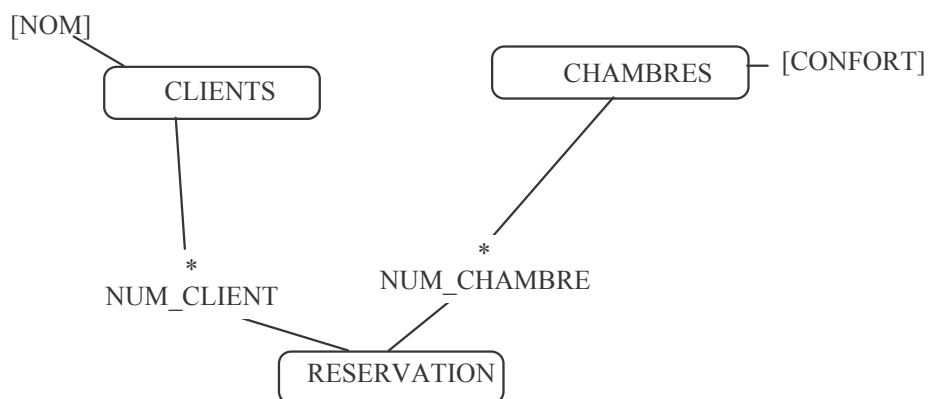


Figure 9-4 :modélisation graphique de la question: Le nom des clients et le confort des chambres qu'ils ont réservés?

Le constituant du résultat Nom appartient à la relation Clients et le constituant Confort appartient à Chambres. Il faut donc définir le lien entre les relations Clients et Chambres, pour cela il faut aussi impliquer la relation Réservation. Nous utilisons ici la jointure naturelle portant sur le constituant NumClient commun aux relations Clients et Réservation, ensuite, nous utilisons ici la jointure naturelle portant sur le constituant NumChambre commun à l'expression précédemment calculée et à la relation Chambres.

$((\text{Clients} * \text{Réservation}) * \text{Chambres})[\text{Nom}, \text{Confort}]$

5) La capacité théorique d'accueil de l'hôtel ?

Pour répondre à cette question, il est nécessaire d'effectuer la somme de toutes les valeurs de NbrPers définie pour chaque chambre. Notre langage algébrique, ne peut pas exprimer une telle expression. L'expression de cette interrogation est donc remise au chapitre suivant.

Exercices

Questions sur TT³

- 1) Etablir le graphe de relations de la modélisation de TT³
- 2) Exprimer les interrogations suivantes avec une expression algébrique, en suivant le schéma définit plus-haut.
 - La zone où est situé le véhicule ayant le no de chassis 324.
 - Le planning du chauffeur nommé "Dupont".
 - La liste des entretiens effectués sur le véhicule ayant le no de chassis 324.
 - La liste des véhicules dont le plein a été effectué avec un type de carburant qui n'était pas celui spécifié par le modèle du véhicule.
 - La liste des chauffeurs pouvant conduire le véhicule ayant le no de chassis 324.
 - La liste des chauffeurs pouvant conduire le véhicule ayant le no de chassis 324 et qui appartiennent à la même station que ce véhicule.

Réponses sur TT³

graphe de relations de la modélisation de TT³

Nous avons regroupé NoZone, ZoneDe, ZoneA car il s'agit de synonymes exprimant la même notion. Par contre, nous avons considéré que NoJour qui apparaissait dans plusieurs relations recouvrait des notions différentes; NoJour_du_plein, NoJour_de_l_entretien, etc et que des jointures naturelles sur un tel constituant ne donnaient que des informations faiblement associées. En fait, Entretien * Carburant permutent tous les entretiens et tous les pleins effectués le même jour.

Expression algébrique

a) La zone où est situé le véhicule ayant le no de chassis 324.

a1) (((*Nochassis=324)Véhicule) * Situation)[NoZone]

Il est intéressant de voir que par les propriétés de l'algèbre relationnelle, la sélection peut être placée avant la jointure ou après (a2) sans modifier le résultat

a2) ((*Nochassis=324)(Véhicule * Situation))[NoZone]

Une autre jointure (a3) permet aussi d'associer les NoZone et les NoChassis, mais le NoZone a un sens complètement différent, il s'agit de la zone du véhicule quand il est garé à sa station.

a3) ((*Nochassis=324)(Véhicule * Station))[NoZone]

b) Le planning du chauffeur nommé "Dupont".

b1) ((*Nom='Dupont')(Chauffeur * Planning))[Planning⁺]

S'il existe plusieurs Dupont dans la compagnie, nous allons trouver les planning des Dupont.

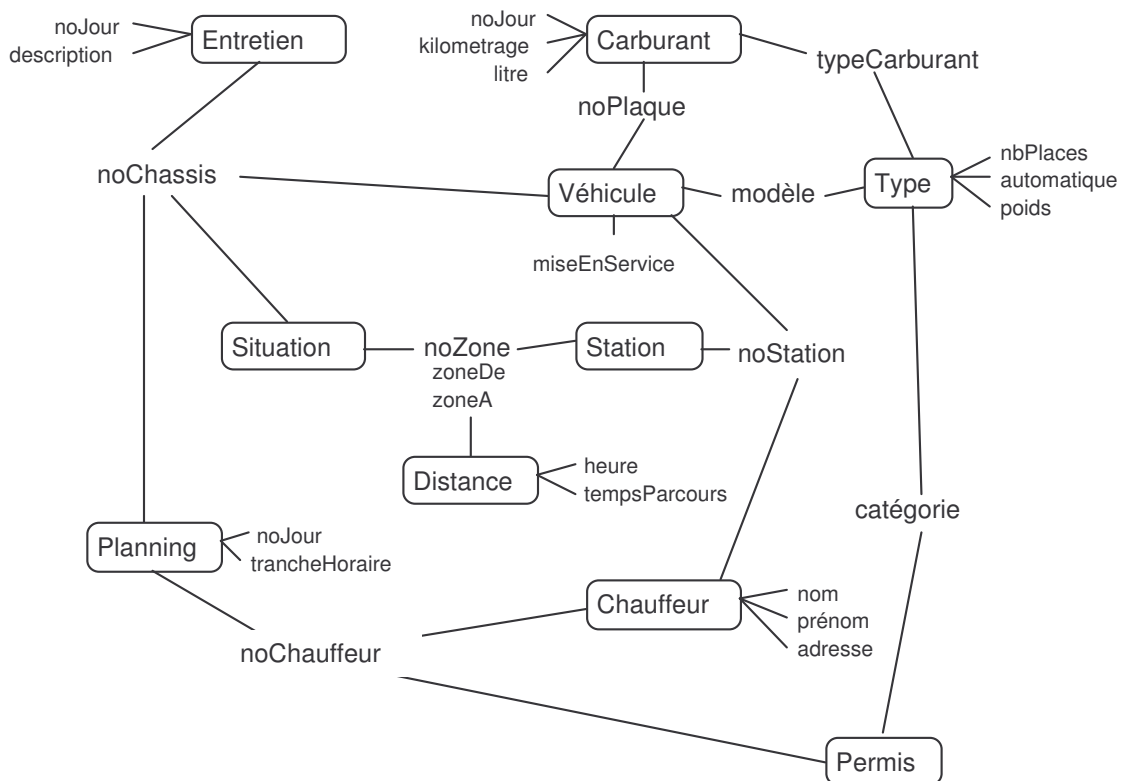


Figure 9-5 :modélisation TT3

c) La liste des véhicules dont le plein a été effectué avec un type de carburant qui n'était pas celui spécifié par le modèle du véhicule.

c1) ((*typecarburantCarburant≠typecarburantVéhicule)
((Carburant * Véhicule) * Type))[Carburant⁺]

d) La liste des chauffeurs pouvant conduire le véhicule ayant le no de chassis 324

d1) ((*Nochassis=324)
(((Véhicule*Type)* permis)* Chauffeur))[Chauffeur⁺]

e) La liste des chauffeurs pouvant conduire le véhicule ayant le no de chassis 324 et qui appartiennent à la même station que ce véhicule

Pour répondre à cette question, il faut spécifier une condition supplémentaire sur d1)

d1) ((*Nochassis=324)^(NoStationVéhicule=NoStationChauffeur)
(((Véhicule*Type)* permis)* Chauffeur))[Chauffeur⁺]

10. Interrogation en SQL

"Le jour l'aurore
 les arbres tremblent
 comme un délire
 le langage le monde
 ne nous appartient pas"
 Patrick Laupin - Le jour l'aurore

Nous allons établir la correspondance entre le langage algébrique et les déclarations de sélection en SQL. Ceci nous donnera un premier modèle d'une machine SQL qui calculerait le résultat de la sélection à partir des tables. Ensuite, nous examinerons la syntaxe complète de la sélection en SQL et au fur et à mesure, nous étendrons le modèle de la machine SQL aux différentes clauses. La machine SQL est un modèle qui nous permet de déterminer de façon sûre le résultat d'une requête d'interrogation.

Du langage algébrique à SQL

La clause de la requête SQL est formée principalement de trois mots clés **SELECT ... FROM ... WHERE ...**. Le premier détermine les constituants de la table résultat. Le second détermine les tables impliquées dans la requête. Le dernier impose une condition sur les entités. Nous pouvons donc établir les correspondances suivantes entre les expressions algébriques et les requêtes SQL.

La relation R

$$R \equiv \text{SELECT } * \text{ FROM } R$$

La projection $R[X]$ où $X = \{C_1, C_2, \dots, C_n\}$ $R[X] \equiv \text{SELECT } C_1, C_2, \dots, C_n \text{ FROM } R$

La sélection $(*F)R$ $(*F)R \equiv \text{SELECT } * \text{ FROM } R \text{ WHERE } F$

Le produit $R ** S$ $R ** S \equiv \text{SELECT } R.*, S.* \text{ FROM } R, S$

La Θ jointure de R et S $R \bowtie_{C_R \Theta C_S} S \equiv \text{SELECT } R.*, S.* \text{ FROM } R, S \text{ WHERE } R.C_1 \Theta S.C_1$

L'union (intersection, différence) $R \cup S$ $R \cup S = \text{SELECT } * \text{ FROM } R \text{ union } \text{SELECT } * \text{ FROM } S$

D'une manière générale, la requête SQL suivante:

```
SELECT C1, C2, ..., Cm
FROM R1, R2, ..., Rn
WHERE F
```

est équivalente à l'expression algébrique suivante: $(*F) R_1 \times R_2 \times \dots \times R_n [C_1, C_2, \dots, C_m]$

Ce résultat nous donne le modèle I de la machine SQL (Figure 10-1):

- faire le produit cartésien de toutes les relations citées après le *FROM*
- sélectionner les entités satisfaisant la condition décrite après *WHERE*
- projeter sur les constituants cités après le *SELECT*

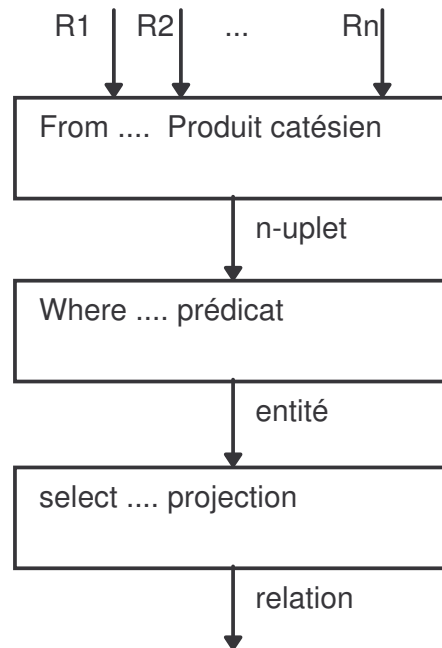


Figure 10-1 :Modèle I, modèle simplifié de la machine SQL

Pour nos exemples, nous utiliserons l'exemple "Hôtel" avec les instance suivantes des relations, que nous obtenons avec les requêtes suivantes:

```
SELECT * FROM CHAMBRES
```

NUM_CHAMBRE	PRIX	NBR_LITS	NBR_PERS	CONFOR	EQU
10	80	1	2	WC	NON
20	80	1	2	WC	NON
30	80	1	2	WC	NON
40	80	1	2	WC	NON
11	90	2	2	WC	NON
21	90	2	2	WC	NON
31	90	2	2	WC	NON
41	90	2	2	WC	NON
12	100	2	2	DOUCHE	NON
22	100	2	2	DOUCHE	NON
32	100	2	2	DOUCHE	NON
42	100	2	2	DOUCHE	NON
13	120	1	2	BAIN	NON
23	120	1	2	BAIN	NON
33	120	1	2	BAIN	NON
43	120	1	2	BAIN	NON
14	140	2	2	BAIN	TV
24	140	2	2	BAIN	TV

34	140	2	2 BAIN	TV
44	140	2	2 BAIN	TV
15	180	3	4 BAIN	TV
25	180	3	4 BAIN	TV
35	180	3	4 BAIN	TV
45	180	3	4 BAIN	TV

```
SELECT * FROM CLIENTS
```

NUM_CLIENT	NOM	PRENOM	ADRESSE
1000	GASCON	GASTON	12 av. du Général 1239 ICI
1001	DUPONT	PIERRE	12 ch. des hirondelles 1238 LABAS
1002	DUFOUR	JEAN	10 av. de la gar 1300 AILLEURS
1003	ZORO	DIEGO	10 ch des voleurs Los Angeles
1004	EINSTEIN	ALBERT	10 rt la relativité 1004 PLUS-LOIN
1005	DUMAS	ALEXANDRE	10 route du moulins LE-SUD
1007	NOBODY	FRANCOISE	403 route de l inconnu 75000 Paris
1006	ROMULUS	BERNADETTE	241 route de rome 1409 Lion
1009	AGDA	BRUNO	10 route de l impossible 1508 TEXAS
1008	CHADOK	AMELIE	25 rue de la rame 1456 Tombouctou

```
SELECT * FROM RESERVATIONS
```

NUM_CLIENT	NUM_CHAMBRE	DATE_ARR	DATE_DEP
1000	11	11-JAN-90	15-JAN-90
1001	21	10-JAN-90	
1002	34	20-DEC-89	27-DEC-89
1003	44	24-DEC-89	27-DEC-89
1005	45	23-DEC-89	28-DEC-89
1006	14	01-DEC-89	28-DEC-89
1007	23	01-DEC-89	02-DEC-89
1007	23	08-DEC-89	09-DEC-89
1007	23	15-DEC-89	16-DEC-89
1007	23	22-DEC-89	23-DEC-89
1007	23	29-DEC-89	30-DEC-89

Reprenons les exemples du chapitre précédent:

1) Les chambres avec bain et TV ?

(* ((Confort=bain)^(Equipement=TV)) Chambres
devient en SQL

```
SELECT *
FROM CHAMBRES
WHERE confort='BAIN' AND equipement='TV'
```

NUM_CHAMBRE	PRIX	NBR_LITS	NBR_PERS	CONFOR	EQU
14	140	2	2	BAIN	TV
24	140	2	2	BAIN	TV
34	140	2	2	BAIN	TV
44	140	2	2	BAIN	TV
15	180	3	4	BAIN	TV
25	180	3	4	BAIN	TV
35	180	3	4	BAIN	TV
45	180	3	4	BAIN	TV

2) Les numéros des chambres et leur capacité ?(Chambres [NumChambre,

NbrPers])
devient en SQL

```
SELECT Num_chambre, nbr_pers
FROM CHAMBRES
```

NUM_CHAMBRE	NBR_PERS
10	2
20	2
30	2
40	2
11	2
21	2
31	2
41	2
12	2
22	2
32	2
42	2
13	2
23	2
33	2
43	2
14	2
24	2
34	2
44	2
15	4
25	4
35	4
45	4

3) Les noms des clients ayant réservés une chambre pour le 25-12-89 ?

((DateArr <= 25-12-89)^(DateDep >25-12-89))(Clients*Réservation))[Nom]
devient en SQL

```
SELECT Nom
FROM CLIENTS, RESERVATIONS
WHERE Date_Arr<=to_date('25-dec-89')
AND Date_Dep> to_date('25-dec-89')
AND Clients.num_client=Reservations.num_client
```

NOM

```
-----
DUFOUR
ZORO
DUMAS
ROMULUS
```

On remarque que la jointure naturelle de l'expression est explicitée dans la requête SQL.

4) Le nom des clients et le confort des chambres qu'ils ont réservées?

((Clients * Réservation) * Chambres)[Nom, Confort]
devient en SQL

```
SELECT Nom, Confort
      FROM CHAMBRES, CLIENTS, RESERVATIONS
      WHERE Clients.num_client=Reservations.num_client
      AND Chambres.num_chambre=Reservations.num_chambre
```

NOM	CONFOR
-----	-----
GASCON	WC
ROMULUS	BAIN
DUPONT	WC
NOBODY	BAIN
NOBODY	BAIN
NOBODY	BAIN
NOBODY	BAIN
NOBODY	BAIN
DUFOUR	BAIN
ZORO	BAIN
DUMAS	BAIN

5) La capacité théorique d'accueil de l'hôtel ?

Cette question, non-formulable avec les expressions algébriques trouve la spécification suivante:

```
SELECT sum(nbr_pers)
      FROM CHAMBRES
```

```
SUM(NBR_PERS)
```

```
-----
                    56
```

Syntaxe des requêtes SQL

Nous avons examiné la traduction des expressions algébriques en SQL. Nous allons examiner maintenant systématiquement toutes les règles de l'expression de sélection SQL.

Cette première règle est un peu comme une table des matières du "SELECT" en SQL, nous y trouvons les parties suivantes:

- **displayed column** : permet de spécifier la nature du résultat (la projection et les expressions calculées)
- **selected-table** : permet de spécifier les relations impliquées dans la requête (le produit cartésien)
- **Condition** : permet de spécifier la condition de sélection
- **Connect-Clause** : permet de spécifier un parcours et une condition arborescents
- **Group-Clause** : permet de spécifier les regroupements, pour le calculs des fonctions agrégatives
- **Set-Clause**: permet de spécifier les opérations ensemblistes
- **Order-Clause** : permet de spécifier les critères de tri sur le résultat
- **Update-Clause**: permet de spécifier les critères de verrouillage pour la concurrence

SELECT-command


```
SELECT distinct confort, équipement
      FROM CHAMBRES
```

```
CONFOR EQU
```

```
----- ---
```

```
BAIN     NON
```

```
BAIN     TV
```

```
DOUCHE  NON
```

```
WC       NON
```

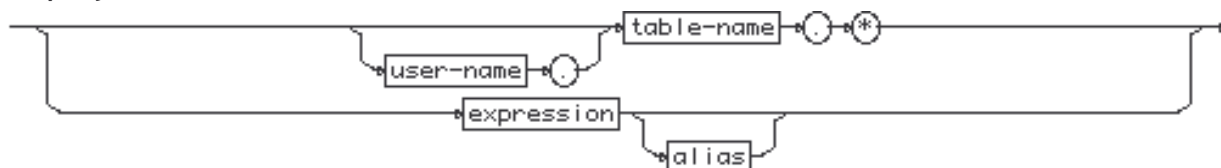
Le mot réservé **all** spécifie que l'on désire toutes les rangées du résultat, par défaut on obtient toutes les entités

En spécifiant *****, on obtient toutes les colonnes de toutes les tables de la clause FROM

La projection

Dans cette clause, nous spécifions la nature des colonnes du résultat. Ce résultat peut être la totalité des colonnes d'une table, un constituant particulier, le résultat d'une expression arithmétique ou le résultat d'une fonction d'agrégation (regroupement)

displayed column



Nous pouvons obtenir toutes les colonnes d'une table en spécifiant **table-name.***. Lors de la création d'une table, le SGBD mémorise avec la table, l'identification du propriétaire de la table. Nous verrons ultérieurement comment ces informations sont utilisées dans les mécanismes de sécurité. En préfixant une table de l'identificateur du propriétaire, on indique au SGBD que l'on désire la table créée par ce propriétaire. Il est ainsi possible que plusieurs utilisateurs possèdent des tables ayant des noms identiques. Dans notre exemple, le propriétaire est associé au nom du projet "HOTEL". Les trois requêtes suivantes sont donc équivalentes

```
SELECT * FROM CHAMBRES
```

```
SELECT CHAMBRES.* FROM CHAMBRES
```

```
SELECT HOTEL.CHAMBRES.* FROM CHAMBRES
```

L'**alias** permet de donner un titre à une colonne, dans le cas des expressions ou simplement de renommer une colonne.

Exemple:

Le prix par personne des chambres ayant une TV ?

```
SELECT num_chambre "Numéro de Chambre", prix/nbr_pers "prix
par pers"
```

```
      FROM Chambres
```

```
      WHERE équipement='TV'
```

```
Numéro de Chambre prix par pers
```

14	70
24	70
34	70
44	70
15	45
25	45
35	45
45	45

Expression

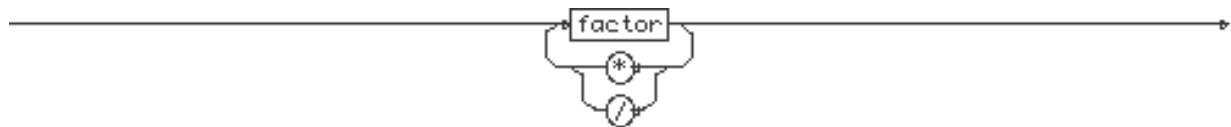
Expression



Une expression est un terme ou une suite de termes connectés par les opérateurs arithmétiques + ou -.

Un terme est un facteur ou une suite de facteurs connectés par les opérateurs arithmétiques * ou /.

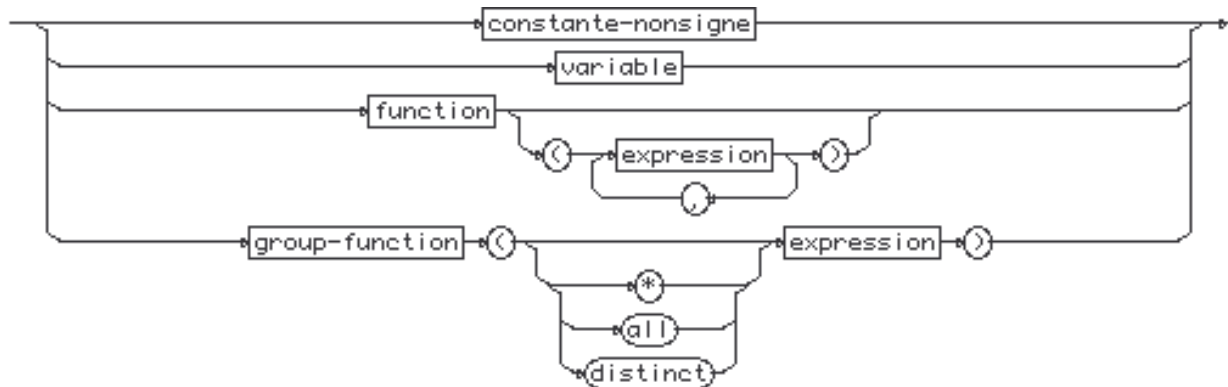
Terme



Un facteur appartient aux catégories suivantes:

- la constante non signée qui est un nombre (12.43, 324, 1.56E-23) ou une chaîne de caractères ('bain'). Les chaînes de caractères peuvent être concaténées avec l'opérateur "||" ('bain' || ' et WC')
- la variable qui est un nom de colonne. Eventuellement préfixée par le nom de la table, dans le cas où ce nom appartient à plusieurs tables manipulées dans une même requête (numchambre, Chambres.numchambre).
- une fonction avec ses paramètres d'appel: `power(taux_annuel, nombre_annees)`. Les fonctions disponibles dépendent du SGBD utilisé. Nous donnons en annexe un aperçu de l'étendue de celles-ci.
- une fonction de regroupement avec ses paramètres d'appel: `count(*)`, `avg(prix/nbpers)`. Nous donnons une explication détaillée dans la clause de regroupement.
- une expression entre parenthèses afin d'ôter les ambiguïtés: $(a+b) * (c+d)$

Factor

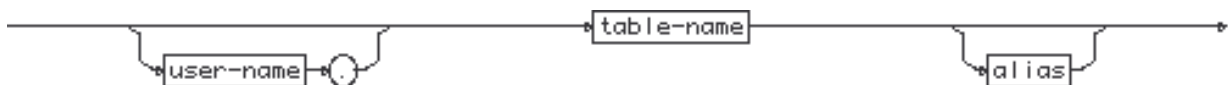


La syntaxe permet de construire des expressions sans limitation de complexité : $(-a * \text{sum}(\text{power}(a, b) + c) * d)$. Plus loin, dans la syntaxe, nous utilisons à nouveau la notion d'expression. Nous faisons référence alors à expression telle que nous venons de la définir.

Les relations du produit cartésien

Les relations nommées dans la clause `FROM` sont celles qui seront impliquées dans le calcul de la requête. Les noms des colonnes apparaissant dans les expressions doivent appartenir à ces tables.

selected-table



La table est simplement nommée, elle est éventuellement préfixée par le nom de son propriétaire. L'alias permet de renommer une table afin de lui donner un nom plus court ou bien un autre nom dans le cas où une table est utilisée plusieurs fois dans une même requête. Ceci permet l'autojointure, par exemple "les chambres ayant le même confort et le même équipement qu'une autre chambre mais ayant un prix inférieur de 10% à celle-ci"

```
SELECT distinct c1.num_chambre
      FROM CHAMBRES c1, CHAMBRES c2
      WHERE c1.confort=c2.confort
            AND c1.equipement=c2.equipement
            AND c1.prix*1.1<c2.prix
```

NUM_CHAMBRE

```
-----
      10
      14
      20
      24
      30
      34
      40
      44
```

Dans ce cas, le SGBD agit comme s'il existait deux instances identiques de CHAMBRE.

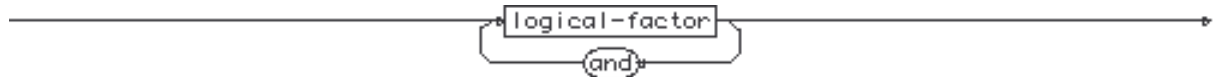
Condition

La clause `WHERE` est associée à une condition qui exprime un prédicat sur les rangées du produit cartésien. Chaque élément de cette condition est évalué à vrai ou à faux. Un terme peut être nié par le mot clé `not`. Les termes peuvent être connectés par le "ou logique inclusif" à l'aide du mot clé `OR`.

Condition



Un terme logique est un facteur ou des facteurs connectés par le "et logique" à l'aide du mot clé `AND`. Les opérateurs logiques sont identiques à ceux décrits dans le chapitre des rappels sur la logique.



exemple:

Les chambres coûtants au max. 80 francs ou ayant un bain et valant au max. 120

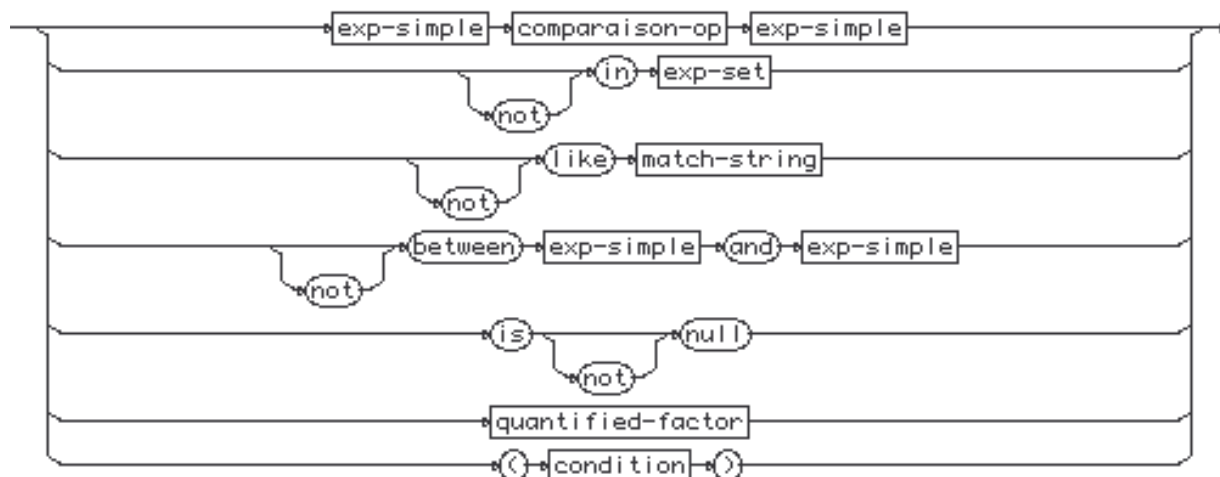
```
SELECT num_chambre, prix, confort
FROM CHAMBRES
WHERE (prix<=80)
      OR ((confort='BAIN') AND (prix<=120))
```

NUM_CHAMBRE	PRIX	CONFOR
10	80	WC
20	80	WC
30	80	WC
40	80	WC
13	120	BAIN
23	120	BAIN
33	120	BAIN
43	120	BAIN

Un facteur logique appartient aux catégories suivantes:

- une comparaison entre deux expressions (`=`, `>`, `<`, ...)
- un test d'appartenance à un ensemble (`in`)
- un test d'appartenance à une chaîne de caractères (`like`)
- un test d'appartenance à un intervalle (`between`).
- une comparaison avec la valeur `null`
- un prédicat quantifié (`exists`, `all`, `any`)
- une condition entre parenthèses afin d'ôter les ambiguïtés.

logical-factor



Comparaison entre deux expressions

Les opérateurs de comparaison sont:

- $\text{exp1} = \text{exp2}$; l'égalité du résultat exp1 et exp2
- $\text{exp1} < \text{exp2}$; exp1 est strictement inférieur à exp2
- $\text{exp1} \leq \text{exp2}$; exp1 est inférieur ou égal à exp2
- $\text{exp1} > \text{exp2}$; exp1 est strictement supérieur à exp2
- $\text{exp1} \geq \text{exp2}$; exp1 est supérieur ou égal à exp2
- $\text{exp1} <> \text{exp2}$; exp1 est différent de exp2 (aussi noté \neq ou \wedge)

Les expressions sont du type de celles vues précédemment, à l'exclusion des fonctions d'agrégation (qui font l'objet d'une clause spéciale *having*). Le résultat d'une expression est un nombre, une chaîne de caractères ou une date. La relation d'ordre utilisée est définie en fonction du résultat. Dans le cas où les résultats appartiennent à des types différents, il faut utiliser les fonctions de conversion de type.

Si une des deux expressions est évaluée à la valeur null, alors le résultat de la comparaison est toujours faux. ($\text{null}=\text{null}$ est aussi évalué à faux). Pour l'égalité, nous avons le tableau suivant:

exp1/exp2	$\neq \text{null}$	$= \text{null}$
$\neq \text{null}$	vrai ou faux	faux
$= \text{null}$	faux	faux

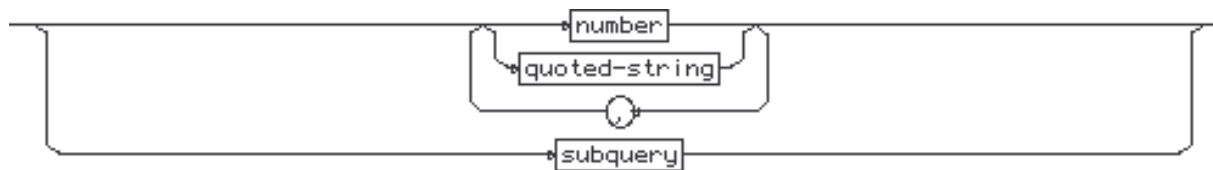
Appartenance à un ensemble

Il existe deux possibilités pour définir l'ensemble avec lequel on doit tester l'appartenance d'un élément. L'une consiste à énumérer explicitement les éléments de l'ensemble; on constitue ainsi une liste de ces éléments. L'autre consiste à définir les éléments par une sous-requête. Si l'ensemble est $\{c_1, c_2, c_3, \dots, c_n\}$ alors les conditions suivantes sont équivalentes:

$\text{Exp1 in } (c_1, c_2, c_3, \dots, c_n)$

$(\text{Exp1} = c_1) \text{ AND } (\text{Exp1} = c_2) \text{ AND } \dots (\text{Exp1} = c_n)$

exp-set



Exemples:

Chambres avec un moyen de se laver

```
SELECT num_chambre FROM CHAMBRES
       WHERE confort in ('BAIN', 'DOUCHE')
```

NUM_CHAMBRE

```
-----
      12
      22
      32
      42
      13
      23
      33
      43
      14
      24
      34
      44
      15
      25
      35
      45
```

Recette du 25-décembre-1989

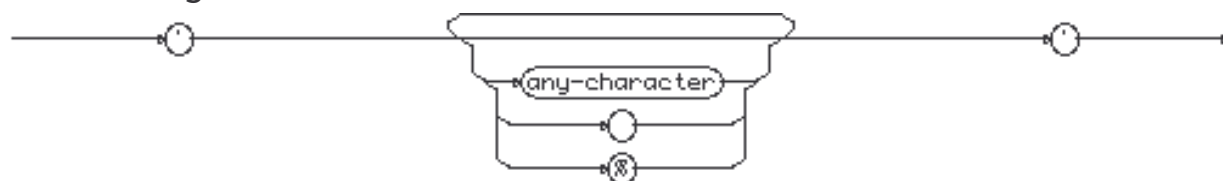
```
SELECT sum(prix)
       FROM CHAMBRES Ch
       WHERE num_chambre in
             SELECT num_chambre
                FROM Reservations R
                WHERE date_arr<='25-dec-89'
                   AND date_dep>'25-dec-89'
```

```
SUM(PRIX)
-----
      600
```

Appartenance à une chaîne de caractères

On test ici si une chaîne de caractères ressemble à un *patron*. Ceci est particulièrement utile si l'on doit effectuer des recherches sur un constituant dont le domaine de modélisation est du type texte. En effet, il permet d'effectuer une sélection à l'intérieur même des chaînes. Deux patrons sont disponibles. Le % qui est substituable à n'importe quelle chaîne de caractères y compris la chaîne vide. Le _ (souligné) qui est substituable à un seul caractère.

match-string



Exemples:

Nom du client commençant par "DU"

```
SELECT nom
      FROM CLIENTS
      WHERE Nom like 'DU%'
```

NOM

DUPONT
DUFOUR
DUMAS

Nom du client ayant un "0" pour quatrième lettre

```
SELECT nom
      FROM CLIENTS
      WHERE Nom like '___0%'
```

NOM

DUPONT
DUFOUR
ZORO
NOBODY

Appartenance à un intervalle

La clause `between` est une facilité d'écriture. Les deux conditions suivantes sont équivalentes:

```
exp1 between exp2 AND exp3
(exp2 <= exp1) AND (exp1 <= exp3)
```

Exemple:

nombre de chambres dont le prix est entre 85 et 120 francs

```
SELECT count(num_chambre)
      FROM CHAMBRES
      WHERE prix between 85 AND 120
```

COUNT(NUM_CHAMBRE)

12

Comparaison avec la valeur null

La clause `is null` permet de tester si une expression est indéfinie. On avait vu précédemment que l'opérateur d'égalité ramenait toujours la valeur fausse. Le résultat de l'évaluation est donné dans le tableau suivant:

<code>expl</code>	<code>expl is null</code>
<code>≠null</code>	faux
<code>=null</code>	vrai

Les deux conditions suivantes sont équivalentes:

```
not (expl is null)
```

```
expl is not null
```

Exemple:

Clients n'ayant pas fixé leur date de départ

```
SELECT Nom
      FROM CLIENTS, RESERVATIONS
      WHERE CLIENTS.num_client=RESERVATIONS.num_client
            AND date_dep is null
```

```
NOM
```

```
-----
```

```
DUPONT
```

Prédicat quantifié

La clause `exists` permet de tester s'il existe au moins une rangée correspondant au prédicat de la sous-requête. Comme il s'agit de tester uniquement l'existence de certaines rangées, les valeurs des colonnes de ces dernières sont sans importance, on peut donc spécifier une constante comme résultat, soit:

```
Exists (SELECT 'Vrai' FROM .... WHERE ...)
```

La clause `all` permet de tester si toutes les rangées, correspondant au prédicat de la sous-requête, vérifient une certaine condition. Par exemple, si la sous-requête `(SELECT col1 FROM ... WHERE ...)` donne pour résultat `{c1, c2, c3, ... cn}` alors les conditions suivantes sont équivalentes (où Θ est un opérateur de comparaison, `=, <, >, ...`):

```
Expl  $\Theta$  all (SELECT col1 FROM ... WHERE ...)
(Expl  $\Theta$  c1) AND (Expl  $\Theta$  c2) AND ... (Expl  $\Theta$  cn)
```

La clause `any` permet de tester si au moins une rangée, correspondant au prédicat de la sous-requête, vérifie une certaine condition. Par exemple, si la sous-requête `(SELECT col1 FROM ... WHERE ...)` donne pour résultat `{c1, c2, c3, ... cn}` alors les conditions suivantes sont équivalentes (où Θ est un opérateur de comparaison, `=, <, >, ...`):

```
Expl  $\Theta$  any (SELECT col1 FROM ... WHERE ...)
(Expl  $\Theta$  c1) OR (Expl  $\Theta$  c2) OR ... (Expl  $\Theta$  cn)
```

La clause `in` et `any` sont équivalentes pour l'opérateur de comparaison `=`. Nous avons l'équivalence suivante:

```
Exp1 in (SELECT col1 FROM ... WHERE ...)
```

```
Exp1 = any (SELECT col1 FROM ... WHERE ...)
```

Rappelons qu'il est possible de passer du quantificateur universel à l'existentiel par une double négation:

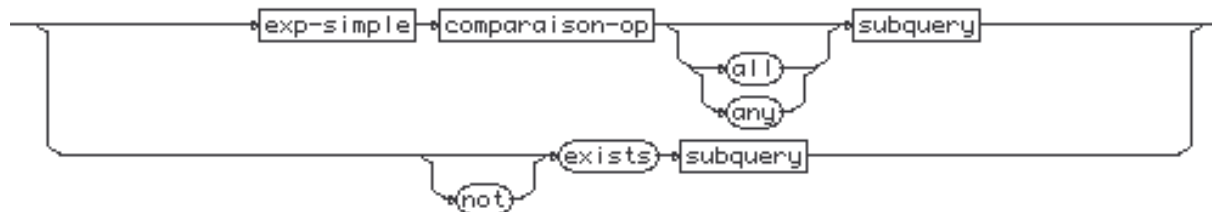
$$\forall x, (P(x)) \equiv \neg \exists x, (\neg P(x))$$

Cette équivalence est aussi applicable aux conditions de SQL. Soit:

```
Exp1  $\Theta$  all (SELECT col1 FROM ...
              WHERE ...)
```

```
not Exists (SELECT 'Vrai' FROM ....
            WHERE ... AND not (exp1  $\Theta$  col1))
```

quantified-factor



Jointure externe

En postfixant les colonnes d'une table par (+), on spécifie une jointure externe avec cette table. Si aucune rangée de la table ne satisfait les conditions de sélection, alors le SGBD met des valeurs null pour les colonnes de cette table.

exemple: afficher tous les numéros de chambre et le numéro du client ayant réservé la chambre pour le 25-décembre-1989.

```
SELECT Ch.num_chambre, num_client
      FROM Chambres Ch,Reservations R
      WHERE Ch.num_chambre=R.num_chambre
            AND date_arr<='25-dec-89'
            AND date_dep>'25-dec-89'
```

```
NUM_CHAMBRE NUM_CLIENT
-----
14          1006
34          1002
44          1003
45          1005
```

```
SELECT Ch.num_chambre, num_client
      FROM Chambres Ch,Reservations R
      WHERE Ch.num_chambre=R.num_chambre(+)
            AND date_arr(+)<='25-dec-89'
            AND date_dep(+)>'25-dec-89'
```

```
NUM_CHAMBRE NUM_CLIENT
```

```

-----
10
11
12
13
14      1006
15
20
21
22
23
24
25
30
31
32
33
34      1002
35
40
41
42
43
44      1003
45      1005

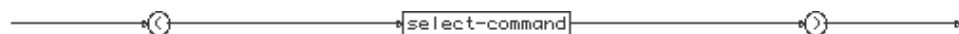
```

S'il n'existe pas de réservation pour une chambre, elle sera composée avec une entité "fictive" de Réservation n'ayant que des valeurs nulles.

Sous-requêtes

Une sous-requête permet de définir un ensemble d'entités. Elle permet d'exprimer une condition ou une expression par rapport au contenu des tables. Elle sera aussi utilisée plus loin pour insérer des rangées dans une table, pour initialiser une table (Insert, CREATE) et pour définir l'étendue d'une modification (Update, Delete)

Subquery



exemple: prix moyen des chambres ayant le même confort que celle du client No 1006 ?

```

SELECT avg(prix) FROM Chambres
      WHERE Confort =
            (SELECT Confort FROM Chambres, Reservations
              WHERE Chambres.num_chambre
=Reservations.num_chambre
              AND num_client=1006)

      AVG (PRIX)
-----
146.666667

```

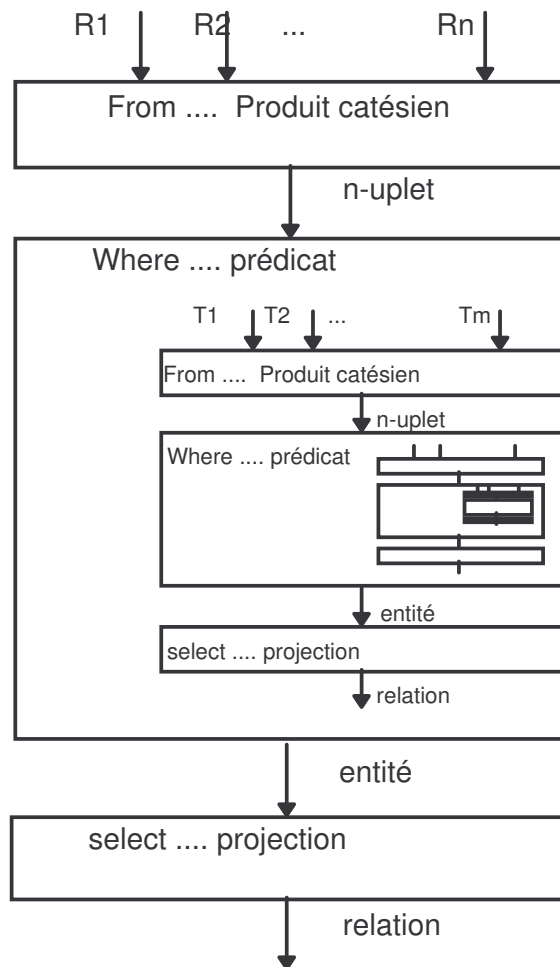


Figure 10-2 :Modèle II, modèle avec les sous-requêtes de la machine SQL

La sous-requête est imbriquée dans une autre requête. Dans une sous-requête, on peut faire référence à des variables définies à un niveau supérieur. Il faut alors les considérer comme des paramètres qui sont modifiés pour chaque rangée du produit cartésien du niveau supérieur.

exemple: chambre ayant un prix 10% inférieur à la moyenne pour une même catégorie de confort

```
SELECT Num_chambre, confort FROM Chambres a
       WHERE prix < (SELECT avg(prix)*0.90 FROM Chambres b
                    WHERE b.confort=a.confort)
```

```
NUM_CHAMBRE  CONFOR
-----
13 BAIN
23 BAIN
33 BAIN
43 BAIN
```

La sous-requête calcule le prix moyen d'une chambre ayant le confort fixé par la valeur `a.confort` prise par la rangée de la requête du niveau supérieur.

Nous devons donc modifier notre modèle de machine SQL pour tenir compte des sous-requêtes. Celles-ci s'insèrent dans le module de la clause WHERE. Elles y introduisent la récursivité car chaque clause WHERE peut contenir des sous-requêtes (Figure 10-2)

Regroupements

Cette clause permet de spécifier le regroupement d'un certain nombre d'entités selon un critère de partitionnement afin d'évaluer une fonction agrégative. Le résultat ne donne donc qu'une seule entité pour chaque regroupement. Les principales fonctions d'agrégation sont:

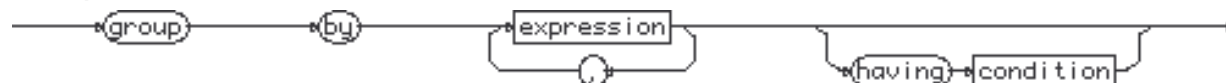
- `avg` : calculer la moyenne d'une liste
- `count` : dénombrer les éléments d'une liste
- `min` : déterminer l'élément minimum d'une liste
- `max` : déterminer l'élément maximum d'une liste
- `sum` : calculer la somme d'une liste

Les paramètres de la fonction sont une expression qui peut être préfixée par les mots clé `all` et `distinct`. Par défaut `all` est utilisé; dans ce cas toutes les valeurs de la liste sont prises en considération. Avec `distinct`, seules les valeurs différentes sont considérées. Les valeurs null, ne sont pas sélectionnées. Cependant `Count(*)` permet de dénombrer une liste en tenant compte de ces dernières.

L'expression de la clause `GROUP BY` détermine le partitionnement. Celui-ci est effectué sur les rangées qui sont sélectionnées par la clause `WHERE`. Les rangées éliminées par la condition de sélection ne participent donc pas à l'agrégation. L'expression de regroupement doit être équivalente à toutes les expressions comportant des variables apparaissant dans le résultat.

Le processus de regroupement intervenant après la sélection des rangées, il est possible d'exprimer une condition de sélection sur les valeurs agrégées dans la condition de la clause `having`. La clause `having` permet de spécifier le critère sur les entités regroupées La clause `WHERE` permet de spécifier le critère sur les entités intervenant dans le regroupement

Group-Clause



exemples:

Prix moyen des chambres par type de confort:

```
SELECT Confort,AVG(Prix) "prix moyen"
      FROM Chambres
      GROUP BY Confort
```

```
CONFOR prix moyen
-----
BAIN      146.666667
DOUCHE    100
```

WC

85

Prix minimum et maximum des chambres par type de confort:

```
SELECT Confort,Min(Prix),Max(Prix)
      FROM Chambres
      GROUP BY Confort
```

CONFOR	MIN(PRIX)	MAX(PRIX)
BAIN	120	180
DOUCHE	100	100
WC	80	90

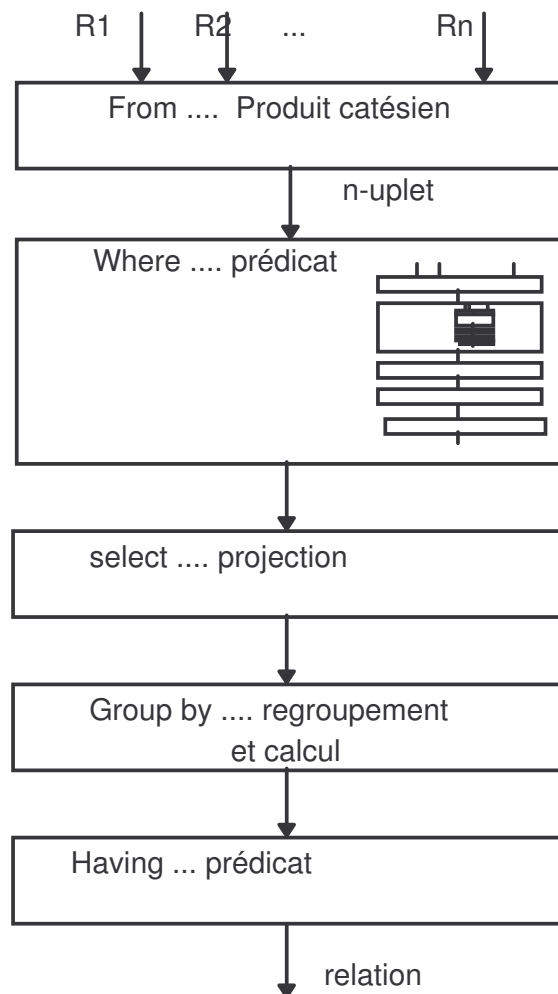


Figure 10-3 :Modèle III, modèle avec les regroupements de la machine SQL
 Prix minimum et maximum des chambres par type de confort, mais dont le prix Min est plus petit que 100:

```
SELECT Confort,Min(Prix),Max(Prix)
      FROM Chambres
      GROUP BY Confort Having Min(Prix)<100
```

CONFOR	MIN(PRIX)	MAX(PRIX)
WC	80	90

Nous devons donc modifier notre modèle de machine SQL pour tenir compte des regroupements. Ceux-ci se placent après le test du prédicat: un premier module permet les regroupements et le calcul des fonctions agrégatives ensuite on effectue une sélection sur les valeurs prises par les fonctions agrégatives.

Ensembles

Cette clause permet d'effectuer des requêtes en utilisant les opérateurs ensemblistes: `union`¹⁰, `intersect`, `minus` (Différence) ont le sens habituel. Les opérandes, le résultat de sélection, de ces opérateurs doivent avoir la même arité (nombre de constituants) et les constituants doivent être égaux en type, pas forcément en taille. L'utilisation de ces opérateurs implique implicitement la clause `DISTINCT`, donc l'élimination des "doublons". Si les colonnes des différentes sélections sont identiques alors il est possible d'effectuer une seule sélection en utilisant les opérateurs logiques.

Set-Clause



Exemple:

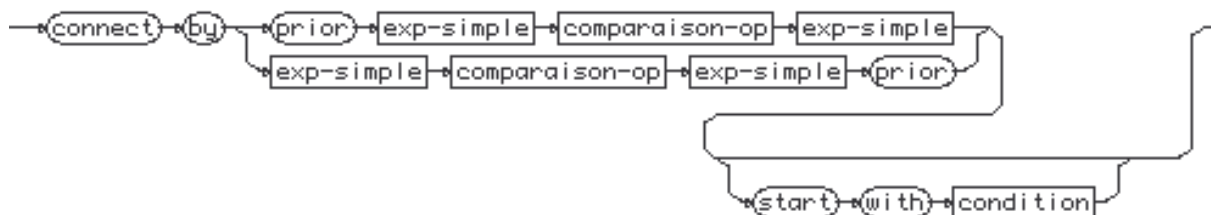
Les termes employés pour une chambre

```
SELECT confort "Termes"
      FROM CHAMBRES
union
SELECT equipement
      FROM CHAMBRES
```

```
Termes
-----
BAIN
DOUCHE
NON
TV
WC
```

Arborescences

Connect-Clause



Cette clause permet de parcourir une relation en fixant un ordre hiérarchique. La clause `prior` doit être utilisée pour fixer la relation

¹⁰UNION ALL permet de faire l'union par concaténation des résultats.

parent-enfant pour le parcours de l'arbre (la profondeur maximum de l'arbre est limitée à 256). Le côté gauche définit le parent et le côté droite l'enfant si la clause prior précède l'expression, et l'inverse si elle la succède. La clause `start with` définit la racine de l'arbre (ou les racines)

exemple: Soit la relation

Emp(Nom,Num_Emp,Num_Manager,Salaire):

"||EMP(a,b,c,d)|| La personne, portant le numéro b et se nommant a a un manager portant le numéro c et perçoit le salaire d"

avec l'instance suivante:

NOM	NUM_EMP	NUM_MANAGER	SALAIRE
DOMINIQUE	1		10000
MARIE	2	1	6000
JEAN	3	1	5000
PAUL	4	2	5000
MARTINE	5	2	5000
VINCENT	6	4	5500
HENRI	7	4	4000
MADELEINE	8	3	4000
ANNE	9	3	3000

Etablir une liste des noms de personne par ordre hiérarchique.

```
SELECT lpad('-',3*level)||Nom "Hiérarchie"
FROM EMP
Connect by prior Num_Emp=Num_Manager
Start with Num_Manager is NULL
```

Hiérarchie

```
-----
-DOMINIQUE
  -MARIE
    -PAUL
      -VINCENT
      -HENRI
    -MARTINE
  -JEAN
    -MADELEINE
    -ANNE
```

La fonction `level` indique le niveau d'un noeud dans l'arborescence. La fonction `lpad` permet d'insérer des blancs à gauche (cadrer à droite)

autres exemples avec les fonctions prédicatives:

Qui est responsable d'employés ?

```
SELECT Nom
FROM EMP E1
WHERE exists (SELECT * FROM EMP E2
              WHERE E1.num_emp=E2.num_manager)
```

NOM

```
-----
DOMINIQUE
MARIE
```


CHADOK	25 rue de la rame 1456 Tombouctou
DUFOUR	10 av. de la gar 1300 AILLEURS
DUMAS	10 route du moulins LE-SUD
DUPONT	12 ch. des hirondelles 1238 LABAS
EINSTEIN	10 route de la relativité 1004 PLUS-LOIN
GASCON	12 av. du Général 1239 ICI
NOBODY	403 route de l inconnu 75000 Paris
ROMULUS	241 route de rome 1409 Lion
ZORO	10 ch des voleurs Los Angeles

Modèle de la machine SQL

Finalement, nous obtenons le modèle de la Figure 10-4 pour la machine SQL. Les relations d'une opération ensembliste sont calculées indépendamment. Le tri est effectué en dernier sur le résultat final.

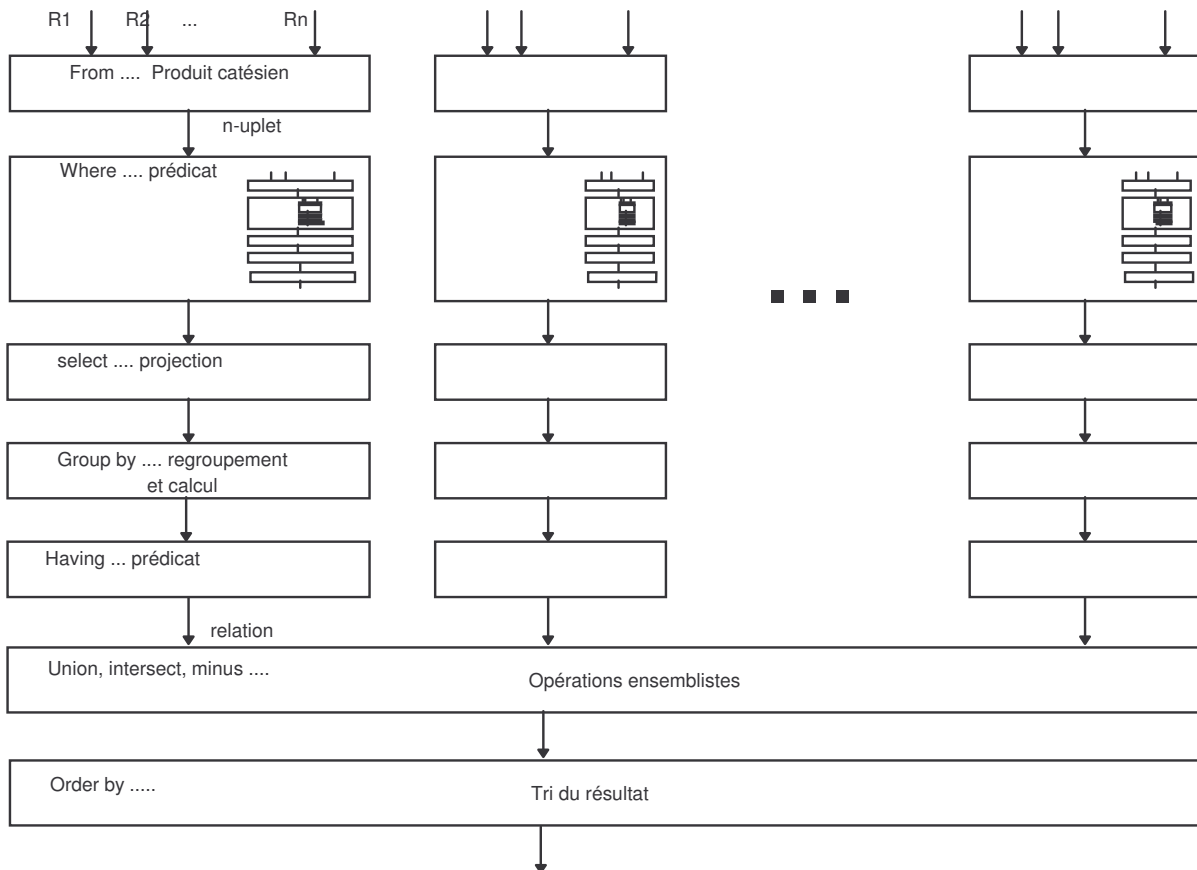


Figure 10-4 :modèle complet de la machine SQL

Ce modèle n'a d'utilité que de déterminer le résultat d'une requête SQL. Lors de l'exécution réelle d'une requête, le SGBD va optimiser la requête afin d'éviter de travailler sur les n-uplets du produit cartésien. Cette optimisation (traitée plus loin) utilise d'une part les propriétés de l'algèbre relationnelle et d'autre part les structures de données accompagnant les tables mémorisées.

Expression graphique d'une requête

Les SGBD sont pratiquement tous munis d'une interface graphique pour l'interrogation des données. Pour l'utilisateur occasionnel, cette solution est très efficace car intuitivement, il peut spécifier une requête sans se préoccuper de la syntaxe SQL. Cependant pour des requêtes complexes, cette approche graphique peut devenir plus complexe que la spécification textuelle de la requête.

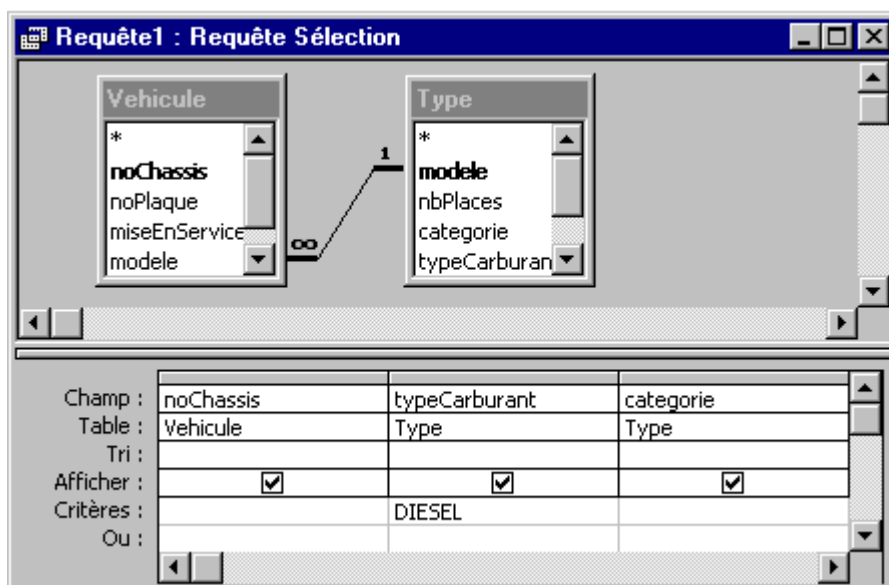


Figure 10-5 :modèle complet de la machine SQL

Dans la Figure 10-5, nous montrons une requête graphique effectuée avec Access de Microsoft. Nous voulons une liste des véhicules « diesel » avec la catégorie de permis. Conceptuellement, il faut procéder de la même manière, c'est-à-dire, déterminer les relations du contexte de la requête. Dans notre cas il s'agit de *Vehicule* et de *Type*, après les avoir sélectionnées dans une liste, le système les présente sur la surface de travail. En utilisant les informations sur les attributs et les cardinalités, le système propose de composer, les deux relations par l'attribut *modele*.

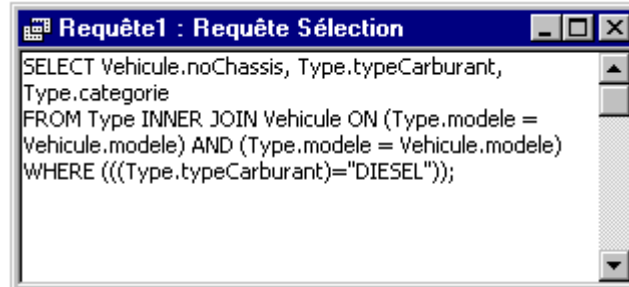
En cliquant sur les champs Nochassis, typeCarburant et categorie, nous les faisons apparaître dans le tableau inférieur. Dans ce même table, nous allons préciser les critères de sélection, de projection, de tri et éventuellement de sélection

	noChassis	typeCarburant	categorie
▶	1000001	DIESEL	PL
	1000005	DIESEL	PL
	1000003	DIESEL	V

Enr : 1 sur 3

Figure 10-6 :modèle complet de la machine SQL

A tout moment, il est possible d'obtenir d'autres vues de la requête. Celle de l'exécution (Figure 10-6) et celle de sa traduction en SQL (Figure 10-7).



```

SELECT Vehicule.noChassis, Type.typeCarburant,
Type.categorie
FROM Type INNER JOIN Vehicule ON (Type.modele =
Vehicule.modele) AND (Type.modele = Vehicule.modele)
WHERE (((Type.typeCarburant)="DIESEL"));

```

Figure 10-7 : modèle complet de la machine SQL

Exercices

Questions sur TT³

Nous avons les instances suivantes des relations:

```
SELECT * FROM Vehicule
```

NOCHASSIS	NOPLAQUE	MISENSER	MODELE	NOSTATION
100001	121	11-DEC-91	TAXI1	1
100002	122	24-JAN-92	TAXI2	1
100003	123	14-DEC-91	TAXI1	2
100005	125	20-DEC-91	TAXI2	2
100004	124	13-OCT-91	BUS	2

```
SELECT * FROM Type
```

MODELE	NBPLACES	CA	TYPECARBURAN	A	POIDS
TAXI1	5	V	ESSENCE	N	850
TAXI2	7	V	DIESEL	Y	1200
BUS	35	PL	DIESEL	N	6500

```
SELECT * FROM Carburant
```

NOPLAQUE	NOJOUR	KILOMETRAGE	LITRES	TYPECARBURAN
121	1	300	30	ESSENCE
122	1	300	20	DIESEL
123	1	300	30	ESSENCE
124	1	300	60	DIESEL
125	1	300	20	DIESEL
121	2	310	30	ESSENCE
122	2	320	20	DIESEL
123	2	300	31	ESSENCE
124	2	300	62	DIESEL
125	2	300	21	DIESEL
121	3	310	31	ESSENCE

122	3	320	21	DIESEL
123	3	300	32	ESSENCE
124	3	300	58	DIESEL
125	3	300	22	DIESEL
125	4	10	1	ESSENCE

```
SELECT * FROM Entretien
```

NOCHASSIS	NOJOUR	DESCRIPTION
100001	1	VIDANGE
100001	1	BOITE A VITESSE
100002	2	VIDANGE
100003	3	VIDANGE
100004	1	VIDANGE

```
SELECT * FROM Chauffeur
```

NOCHAUFFEUR	NOM	PRENOM	ADRESSE	NOSTATION
1	DUPONT	JEAN	ici	1
2	MAX	MAXIM	ici1	1
3	BOL	PAUL	labas	2
4	PASBOL	PAUL	labas2	2

```
SELECT * FROM Permis
```

NOCHAUFFEUR	CA
1	V
2	V
3	V
4	V
1	PL
3	PL

```
SELECT * FROM Planning
```

NOCHAUFFEUR	NOCHASSIS	NOJOUR	T
1	100001	1	A
2	100002	1	A
3	100003	2	A
4	100005	1	A
1	100001	1	B
4	100003	2	A
1	100004	3	A

```
SELECT * FROM Station
```

NOZONE	NOSTATION
10	1
12	2

```
SELECT * FROM Distance
```

HEURE	ZONEDE	ZONEA	TEMPSPARCOURS
10	10	20	5
11	10	20	6
12	10	20	10
13	10	20	5

10	20	10	5
11	20	10	6
12	20	10	10
13	20	10	5
10	10	10	0
11	10	10	0
12	10	10	0
13	10	10	0
10	20	20	0
11	20	20	0
12	20	20	0
13	20	20	0
13	20	20	0

```
SELECT * FROM Situation
```

NOCHASSIS	NOZONE
-----	-----
1000001	10
1000002	20
1000003	20
1000005	20

Donnez une requête SQL pour chaque point ci-dessous, utilisez le graphe de relation pour effectuer les jointures:

- 1) liste des véhicules par numéro de châssis croissant
- 2) liste des véhicules et leur catégorie de permis
- 3) liste des véhicules de plus de 20 places
- 4) liste des véhicules ayant entre 5 et 8 places
- 5) liste du poids total des véhicules (poids du véhicule + 65 kg par personne)
- 6) liste alphabétique des chauffeurs et de leur permis
- 7) liste des prénoms partagés par plusieurs chauffeurs en utilisant count()
- 8) liste des prénoms partagés par plusieurs chauffeurs en utilisant EXISTS
- 9) liste des véhicules ayant un numéro de plaque commençant par un 1 et un 3 en troisième position
- 10) liste des véhicules ayant un entretien dont le libellé comporte successivement les mots BOITE et VITESSE
- 11) liste des véhicules que peut conduire le chauffeur No 4 (avec le IN)
- 12) liste des véhicules que peut conduire le chauffeur No 4 (avec des jointures seulement)
- 13) déterminer si un véhicule a été rempli avec un carburant ne correspondant pas à son modèle
- 14) consommation totale de chaque véhicule, triée par type de carburant
- 15) consommation totale par jour, par station, par type de carburant
- 16) consommation moyenne de chaque véhicule pour 100 km
- 17) consommation moyenne journalière de chaque véhicule pour 100 km
- 18) liste des pleins anormaux (20 % de plus que la consommation moyenne du véhicule)
- 19) le planning du chauffeur No 1 trié par jour et tranche horaire
- 20) pour un client appelant depuis la zone 10 à 13 heure, donnez la liste des véhicules inoccupés par ordre de proximité.

Réponses sur TT³

- 1) liste des véhicules par numéro de châssis croissant

```
SELECT *
  FROM VEHICULE
  ORDER BY nochassis
```

NOCHASSIS	NOPLAQUE	MISEENSER	MODELE	NOSTATION
100001	121	11-DEC-91	TAXI1	1
100002	122	24-JAN-92	TAXI2	1
100003	123	14-DEC-91	TAXI1	2
100004	124	13-OCT-91	BUS	2
100005	125	20-DEC-91	TAXI2	2

- 2) liste des véhicules et leur catégorie de permis

```
SELECT nochassis,v.modele
      FROM VEHICULE v, TYPE t
      WHERE v.modele=t.modele
```

```
NOCHASSIS MODELE
```

```
-----
100004 BUS
100001 TAXI1
100003 TAXI1
100002 TAXI2
100005 TAXI2
```

3) liste des véhicules de plus de 20 places

```
SELECT nochassis,nbplaces
      FROM VEHICULE v, TYPE t
      WHERE v.modele=t.modele
            AND nbplaces>20
```

```
NOCHASSIS  NBPLACES
```

```
-----
100004      35
```

4) liste des véhicules ayant entre 5 et 8 places

```
SELECT nochassis,nbplaces
      FROM VEHICULE v, TYPE t
      WHERE v.modele=t.modele
            AND nbplaces between 5 AND 8
```

```
NOCHASSIS  NBPLACES
```

```
-----
100001      5
100003      5
100002      7
100005      7
```

5) liste du poids des véhicules (65 kg par personne)

```
SELECT nochassis,poids+65*nbplaces "Poids total"
      FROM VEHICULE v, TYPE t
      WHERE v.modele=t.modele
```

```
NOCHASSIS Poids total
```

```
-----
100004      8775
100001      1175
100003      1175
100002      1655
100005      1655
```

6) liste alphabétique des chauffeurs et de leur permis

```
SELECT ch.nochauffeur,nom,prenom,categorie
      FROM CHAUFFEUR ch, Permis pe
      WHERE ch.nochauffeur=pe.nochauffeur
      ORDER BY nom,prenom
```

NOCHAUFFEUR	NOM	PRENOM	CA
3	BOL	PAUL	V
3	BOL	PAUL	PL
1	DUPONT	JEAN	V
1	DUPONT	JEAN	PL
2	MAX	MAXIM	V
4	PASBOL	PAUL	V

7) liste des prénoms partagés par plusieurs chauffeurs en utilisant count()

```
SELECT distinct prenom
      FROM CHAUFFEUR
      GROUP BY prenom having count(prenom)>=2
```

PRENOM

PAUL

8) liste des prénoms partagés par plusieurs chauffeurs en utilisant EXISTS

```
SELECT distinct prenom
      FROM CHAUFFEUR ch1
      WHERE exists(SELECT 'vrai'
                  FROM CHAUFFEUR ch2
                  WHERE ch1.nochauffeur<>ch2.nochauffeur
                  AND ch1.prenom=ch2.prenom)
```

PRENOM

PAUL

9) liste des véhicules ayant un numéro de plaque commençant par un 1 et un 3 en troisième position

```
SELECT nochassis,noplaque
      FROM VEHICULE
      WHERE noplaque like '1_3%'
```

NOCHASSIS NOPLAQUE

100003 123

10) liste des véhicules ayant un entretien dont le libellé comporte successivement les mots BOITE et VITESSE

```
SELECT nochassis,description
      FROM ENTRETIEN
      WHERE description like '%BOITE%VITESSE%'
```

NOCHASSIS DESCRIPTION

100001 BOITE A VITESSE

11) liste des véhicules que peut conduire le chauffeur No 4 (avec le IN)

```

SELECT nochassis,v.modele
  FROM VEHICULE v, TYPE t
  WHERE v.modele=t.modele
        AND categorie in (SELECT pe.categorie
                          FROM CHAUFFEUR ch, Permis pe
                          WHERE ch.nochauffeur=pe.nochauffeur
                             AND ch.nochauffeur=4)

```

NOCHASSIS MODELE

```

-----
100001 TAXI1
100005 TAXI2
100002 TAXI2
100003 TAXI1

```

12) liste des véhicules que peut conduire le chauffeur No 4 (avec des jointures seulement)

```

SELECT nochassis,v.modele
  FROM VEHICULE v, TYPE t,CHAUFFEUR ch, Permis pe
  WHERE v.modele=t.modele
        AND ch.nochauffeur=pe.nochauffeur
        AND pe.categorie=t.categorie
        AND ch.nochauffeur=4

```

NOCHASSIS MODELE

```

-----
100001 TAXI1
100003 TAXI1
100002 TAXI2
100005 TAXI2

```

13) déterminer si un véhicule a été rempli avec un carburant ne correspondant pas à son modele

```

SELECT nochassis,v.modele
  FROM VEHICULE v, TYPE t
  WHERE v.modele=t.modele
        AND typecarburant <> any (SELECT typecarburant
                                  FROM carburant tc
                                  WHERE v.noplaque=tc.noplaque)

```

NOCHASSIS MODELE

```

-----
100005 TAXI2

```

14) consommation totale de chaque véhicule, triée par type de carburant

```

SELECT typecarburant,noplaque,sum(litres)
  FROM CARBURANT
  GROUP BY typecarburant,noplaque
  ORDER BY typecarburant

```

TYPECARBURAN	NOPLAQUE	SUM(LITRES)
DIESEL	122	61
DIESEL	124	180
DIESEL	125	63

```

ESSENCE          121          91
ESSENCE          123          93
ESSENCE          125           1

```

15) consommation totale par jour, par station, par type de carburant

```

SELECT nojour, nostation ,typecarburant ,sum(litres)
  FROM CARBURANT c, VEHICULE v
  WHERE c.noplaque=v.noplaque
  GROUP BY nojour, nostation ,typecarburant

```

```

NOJOUR  NOSTATION  TYPECARBURAN  SUM(LITRES)
-----  -
1        1  DIESEL        20
1        1  ESSENCE       30
1        2  DIESEL        80
1        2  ESSENCE       30
2        1  DIESEL        20
2        1  ESSENCE       30
2        2  DIESEL        83
2        2  ESSENCE       31
3        1  DIESEL        21
3        1  ESSENCE       31
3        2  DIESEL        80
3        2  ESSENCE       32
4        2  ESSENCE        1

```

16) consommation moyenne de chaque vehicule pour 100 km

```

SELECT noplaque ,
       (sum(litres)/sum(kilometrage))*100 "consommation moyenne
globale"
  FROM CARBURANT c
  GROUP BY noplaque

```

```

NOPLAQUE  consommation moyenne globale
-----  -
121                9.89130435
122                6.4893617
123               10.3333333
124                 20
125               7.03296703

```

17) consommation moyenne journalière de chaque véhicule pour 100 km

```

SELECT noplaque,nojour ,
       (sum(litres)/sum(kilometrage))*100 "moyenne journaliere"
  FROM CARBURANT c
  GROUP BY noplaque,nojour

```

```

NOPLAQUE  NOJOUR  moyenne journaliere
-----  -
121        1          10
121        2      9.67741935
121        3          10
122        1      6.66666667
122        2          6.25
122        3      6.5625
123        1          10

```

123	2	10.3333333
123	3	10.6666667
124	1	20
124	2	20.6666667
124	3	19.3333333
125	1	6.6666667
125	2	7
125	3	7.3333333
125	4	10

18) liste des pleins anormaux (20 % de plus que la consommation moyenne du véhicule)

```
SELECT noplaque, nojour, litres*100/kilometrage
  FROM CARBURANT c1
  WHERE litres*100/kilometrage
        > (SELECT
1.2*(sum(litres)*100/sum(kilometrage))
        FROM CARBURANT c2
        WHERE c2.noplaque=c1.noplaque)
```

NOPLAQUE	NOJOUR	LITRES*100/KILOMETRAGE
125	4	10

19) le planning du chauffeur No 1 trié par jour et tranche horaire

```
SELECT nochauffeur, nojour, tranchehoraire, nochassis
  FROM PLANNING c1
  WHERE nochauffeur=1
  ORDER BY nojour, tranchehoraire
```

NOCHAUFFEUR	NOJOUR	T	NOCHASSIS
1	1	A	100001
1	1	B	100001
1	3	A	100004

20) pour un client appelant depuis la zone 10 à 13 heure, donnez la liste des véhicules inoccupés par ordre de proximité.

```
SELECT s.nochassis, s.nozone, tempsparcours
  FROM SITUATION S, DISTANCE D
  WHERE s.nozone=d.zonede
        AND d.zonea=10
        AND d.heure=13
  ORDER BY tempsparcours
```

NOCHASSIS	NOZONE	TEMPSPARCOURS
1000001	10	0
1000002	20	5
1000003	20	5
1000005	20	5

11. Modification des données

"Un matin, au sortir d'un rêve agité, Grégoire Samsa s'éveilla transformé dans son lit en une véritable vermine"
(Franz Kafka - La métamorphose)

Les champs d'application qui ne se transforment pas au cours du temps sont extrêmement peu fréquent (dans le domaine de l'analyse statistique d'enquête). La modification des objets du champ d'application est nécessaire dans la majorité des cas. Ces modifications touchent l'instance de la base de données. En poursuivant l'idée que l'instance de la base de données est une interprétation du champ d'application à l'instant t , pour obtenir l'instance de la base à l'instant t' , il suffit en théorie de réinterpréter le contenu du champ à ce moment là (Figure 11-1). La modélisation et sa concrétisation, fixées lors de la conception initiale, sont considérées comme des invariants par rapport aux modifications des objets du champs d'application.

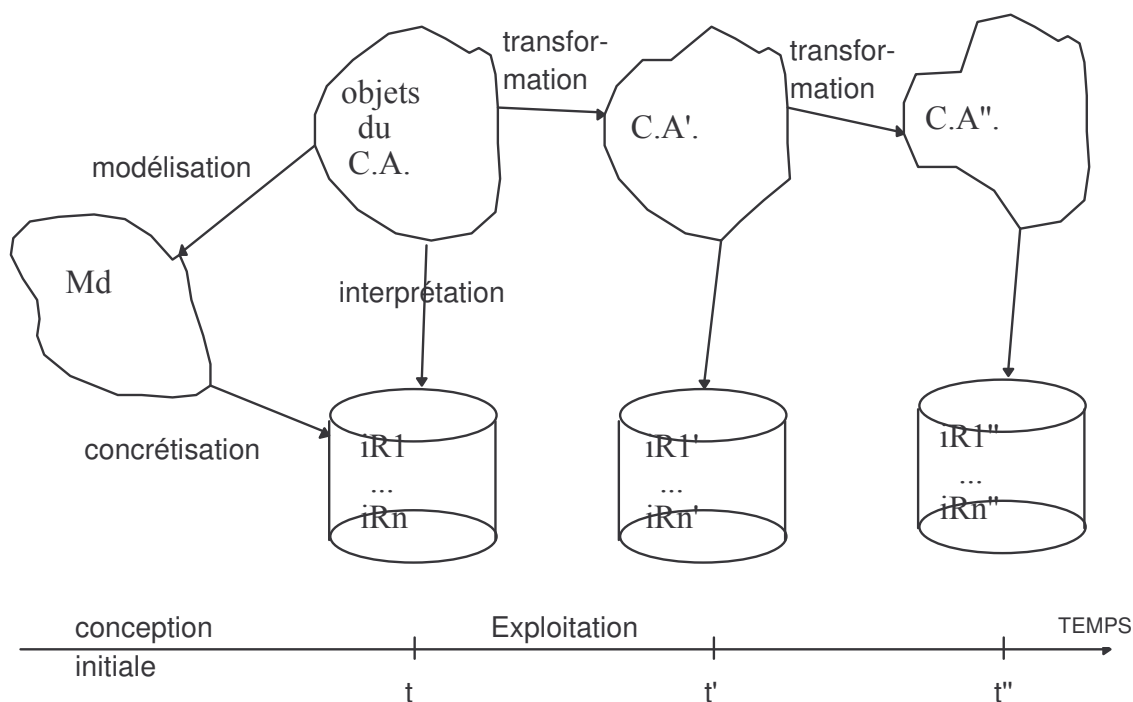


Figure 11-1 : Vision de la modification à travers les prédicats.

L'exploitation d'une base de données par une réinterprétation continue du champ d'application n'est pas possible. L'initialisation de la première instance peut déjà être longue dans la durée. De plus, il serait difficile de fixer l'intervalle de temps entre deux réinterprétations. Cette vision de la modification à travers les prédicats ne doit être donc conservée que comme

référence pour valider d'autres mises en oeuvre des modifications. Cependant, une telle vision nous aurait permis de faire l'économie de l'analyse des transformations du champ d'application.

Notre modèle des modifications demande l'introduction d'un nouveau concept, celui d'événement. L'occurrence d'un événement appliqué au champ d'application modifie ce dernier. Par exemple, nous avons l'événement: "Un client réserve une chambre" et son occurrence: "le client No 1004 réserve la chambre 24 du 1-jan-90 au 4-jan-90". L'occurrence d'un événement modifie les objets du champ d'application. En effet un événement sans conséquence sur les données ne pourrait pas laisser de trace dans le champ d'application. La recherche des événements se situe au niveau du champ d'application, les événements sont donc significatifs pour les utilisateurs.

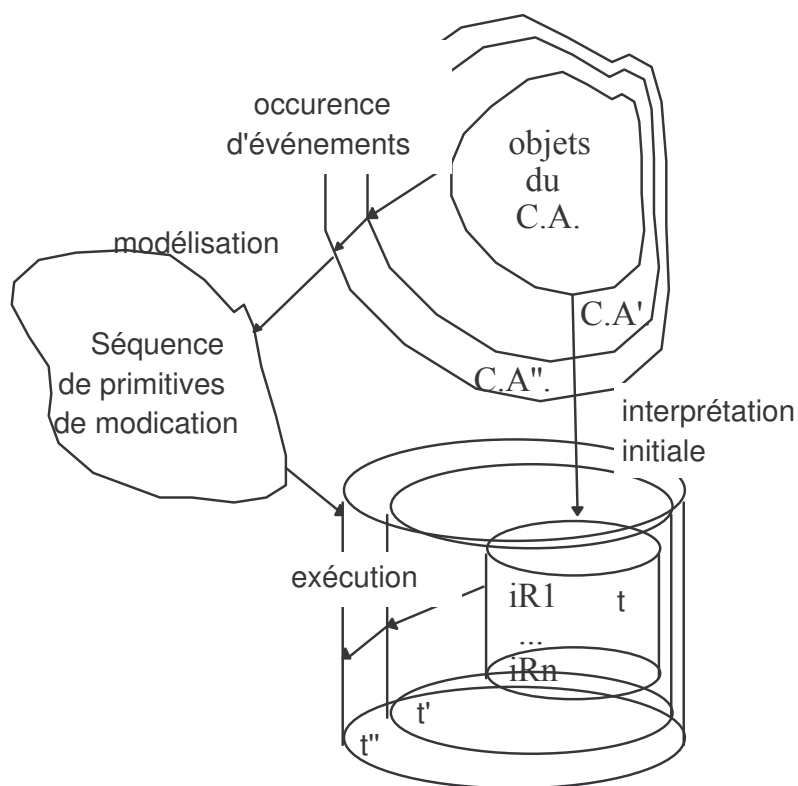


Figure 11-2 : Vision de la modification à travers les événements.

L'événement est traduit par une séquence de primitives de modification dans le modèle relationnel; on appelle cette séquence transaction. Finalement, l'exécution de ces primitives sur la base de données modifie les instances des relations. Pour l'exemple précédant, l'événement de la réservation correspond à la création d'une nouvelle entité de la relation Réservation. L'occurrence de l'événement correspond à l'insertion de la rangée (1004, 24, 1-jan-90, 4-jan-90) dans la table Réservation.

Ce modèle est donc basé sur le fait que de "petites" modifications du champ d'application sont reportées comme de "petites" modifications de la base de données (Figure 11-2). Les objets du champ d'application sont interprétés initialement au temps t comme l'instance BD_t . Les occurrences des

événements de t à t' sont reportés en exécutant les primitives correspondantes donnant l'instance $BD_{t'}$. Et ainsi de suite ...

Ce modèle amène les remarques suivantes. La base de données contient le reflet du champ d'application pour un instant déterminé dans le temps. La base de données ne mémorise pas l'histoire de ses modifications car l'occurrence des événements entraînent immédiatement son actualisation. Si l'on veut conserver cette histoire, il faut la modéliser explicitement (ou travailler avec un SGBD historique). La base de données au temps t est le reflet de l'ensemble des événements depuis l'initialisation et devrait être égale à une instance initialisée au temps t . La validité de cette égalité est soumise à des hypothèses et des contraintes que l'on doit valider sur le modèle. Nous résumons le modèle des modifications par les 5 équations de la Figure 11-3.

1. Le champ d'application actuel est égal à la séquence des occurrences des événements appliqués au champ d'application initial.
2. La base de données actuelle est égale à la séquence des primitives de modification exécutées sur la base de données initiale.
3. L'instance de la base de données initiale est une interprétation correcte du champ d'application initial.
4. La séquence d'occurrence des événements est équivalente à la séquence d'exécution des primitives de modification
5. L'instance de la base de données actuelle est une interprétation correcte du champ d'application actuel.

Ces équations sont des hypothèses fortes, examinons les possibles faiblesses de ce système.

L'équation 1 est vraie si la totalité des événements peuvent être enregistrés (ou perçus). Il est donc essentiel que tous les événements soient mis en évidence et que leurs sources d'occurrence soient identifiées. Sinon, les transformations du champ d'application apparaissent comme spontanées, sans cause. Et ceci rend impossible la mise à jour de la base de données par l'exécution des primitives. Dans le cas où de tels événements sont inhérents au champ d'application, il est indispensable de réinitialiser périodiquement la base de données afin de réduire l'écart entre le champ d'application et son instantiation. Par exemple, dans un magasin par rapport à la gestion de son stock, les vols de marchandises sont inévitables et ne peuvent être pris en compte par le système d'information (par définition le vol doit passer inaperçu). Un inventaire périodique permet la réinitialisation de la base de données.

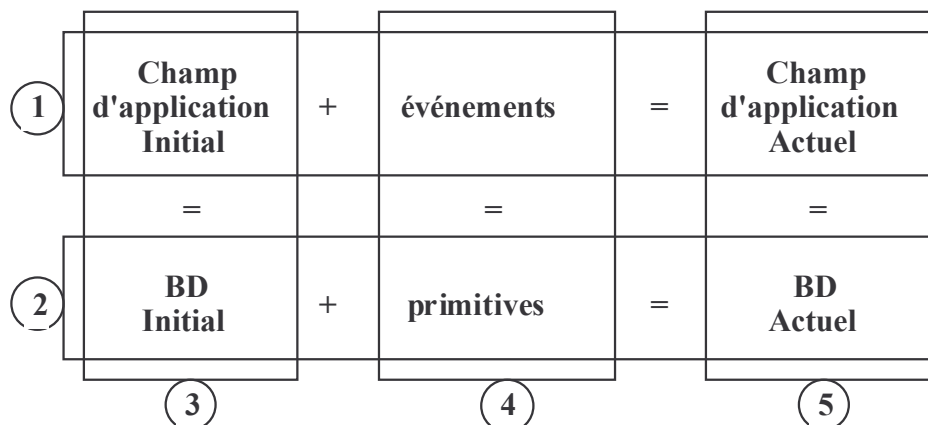


Figure 11-3 : Equations sous jacentes du modèle de modification.

L'équation 2 est vraie par définition, elle constitue la spécification même des primitives de modification d'un SGBD. Cependant, sa validité dépend de mécanismes tels que la gestion des transactions dans le SGBD, d'une gestion correcte de la concurrence de l'exécution des transactions, de la sécurité des données par rapport aux défaillances du matériel.

L'équation 3 est vraie si d'une part la modélisation est fidèle et d'autre part si l'instance initiale est une interprétation correcte du champ d'application. La fidélité de l'interprétation dépend de la justesse de l'analyse des objets du champ d'application et de leur modélisation. La qualité de l'instance initiale doit être contrôlée minutieusement, par exemple par des requêtes dont les résultats sont déjà connus dans le champ d'application.

L'équation 4 est vraie si l'on restreint la conséquence des événements aux objets inclus dans la modélisation. Cette restriction est importante car elle ampute l'événement d'une partie de son information concernant son adéquation à la modélisation, au modèle, au paradigme de gestion utilisé. Pour nous, l'événement est porteur de modification sur ces autres niveaux et l'organisation doit être attentive à celles-ci. Par exemple, la transformation des habitudes de paiement des clients par l'utilisation de carte de crédit doit être prise en compte dans la modélisation, en ajoutant les constituants nécessaires. Ceci est encore plus manifeste si la structure du champ d'application n'est pas bien établie. Par exemple dans le domaine de la cognition, la compréhension de ce livre n'est pas réductible à l'insertion des mots dans une hypothétique table contenue dans la tête du lecteur. Au fur et à mesure de sa progression dans la lecture, mais chaque événement modifie peu à peu son modèle et finalement établit un paradigme intériorisé des systèmes d'information. Les événements des champs d'applications bien structurés entraînent principalement des modifications sur les objets et plus rarement des modifications sur la structure expliquant le succès des SGBD dans ces domaines.

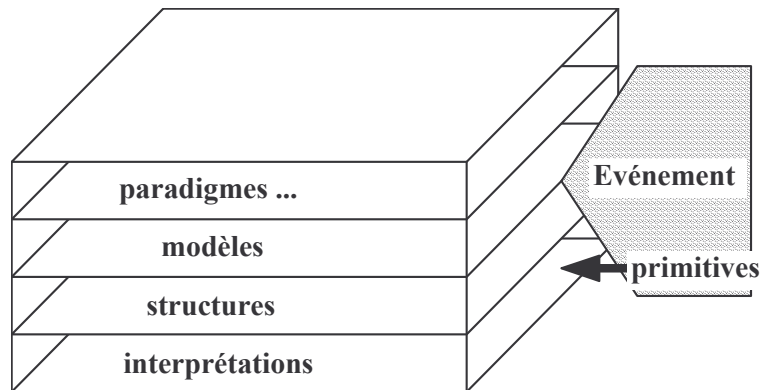


Figure 11-4 : Spectre des modifications dont l'événement est porteur.

L'équation 5 est vraie pour autant que les équations précédentes soient vérifiées. A terme, un écart inévitable se crée entre le champ d'application et la base de données, obligeant le concepteur et l'organisation (par ordre décroissant de fréquence):

- à réinitialiser la base de données
- à restructurer la modélisation des données et des événements
- à changer de modèle (informatique)
- à changer de paradigme de gestion

Le défi actuel des systèmes d'information réside dans l'assurance d'une continuité par rapport à ces différents changements (qui sont imprévisibles)

Primitives de modification

Les primitives sont le reflet de trois événements élémentaires du champ d'application:

- La **création**; un objet¹¹ nouveau apparaît dans le champ d'application, donc il doit être enregistré dans la base de données (un nouveau client de l'hôtel)
- La **mise à jour**; un objet déjà présent dans le champ d'application se modifie et ceci doit être reporté dans la BD (un client modifie sa date de départ)
- La **suppression**; un objet enregistré dans la BD sort du champ d'application et doit donc être éliminé de la BD (Une chambre est aménagée en salon)

Une primitive ne modifie qu'une seule relation et une seule entité. Pour s'exécuter, elle doit vérifier certaines pré-conditions et après son exécution on peut toujours vérifier certaines post-conditions. On notera par: {pré-conditions} primitive {post-conditions}, l'effet de la primitive.

Création

Soit $R(C_1, C_2, \dots, C_n)$ et iR une instance de R , r un tuple de R

¹¹ Un objet du champ d'application peut être modélisé par plusieurs relations.

Alors la création de r dans R notée $c(R,r)$ est l'opération qui modifie l'instance iR de la manière suivante:

$$\{r \notin iR\} c(R,r) \{r \in iR\}$$

On déclenche cette primitive pour le tuple r lorsque le prédicat de R devient valide pour les valeurs de r .

exemple: un nouveau client

$$c(\text{Clients}, (\text{Dumas}, \text{Alexandre}, \text{"le vieux moulin..."}))$$

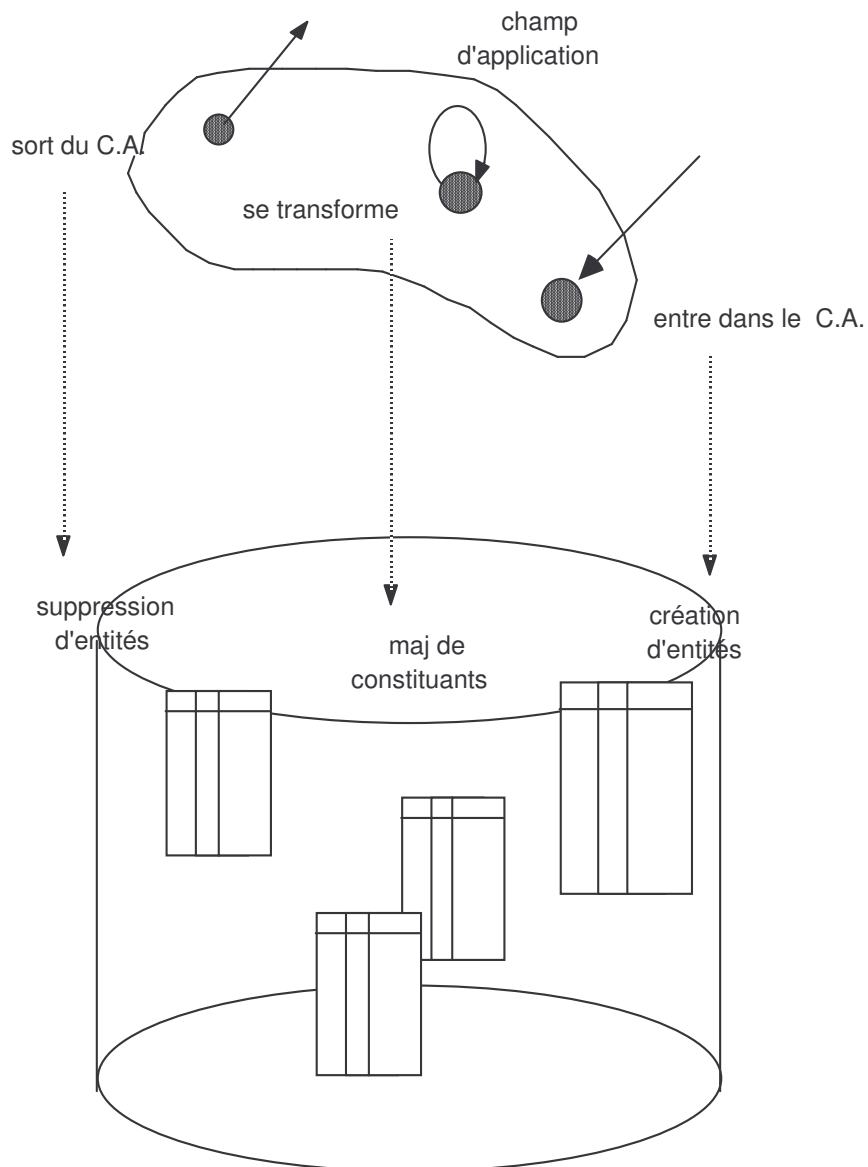


Figure 11-5 : Association des événements élémentaires aux primitives.

Suppression

Soit $R(C_1, C_2, \dots, C_n)$ et iR une instance de R , r un tuple de R

Alors la suppression de r dans R notée $s(R,r)$ est l'opération qui modifie l'instance iR de la manière suivante:

$$\{r \in iR\} s(R,r) \{r \notin iR\}$$

On déclenche cette primitive pour le tuple r , lorsque le prédicat de R devient invalide pour les valeurs de r et que r n'est pas remplacé par un autre ayant d'autres valeurs.

exemple: transformation d'une chambre en salon

$s(\text{Chambres}, (23, 120, 1, 2, \text{BAIN}, \text{NON}))$

Mise à jour

Soit $R(C_1, C_2, \dots, C_n)$ et iR une instance de R , r un tuple de R

Alors la mise à jour de r dans R pour le constituant C par la valeur v est l'opération notée $m(R, r, C, v)$ est l'opération qui modifie l'instance iR de la manière suivante:

$$\{r \in iR\} m(R, r, C, v) \{r[C]=v\}$$

On déclenche cette primitive pour le tuple r lorsque le prédicat de R devient invalide pour r , mais reste valide si r prend la valeur v pour le constituant C .

Exemple: modification de la date de départ d'une réservation

$m(\text{Réservation}, (1006, 14, 01\text{-DEC-89}, 28\text{-DEC-89}), \text{Datedep}, 30\text{-DEC-89})$.

Transaction

Une transaction est définie par une séquence de primitives de modification.

$$T = \langle p_1, p_2, \dots, p_n \rangle$$

De plus l'exécution de la transaction doit vérifier les propriétés suivantes [GRA81]:

- La **consistance**; la transaction doit obéir aux protocoles légaux. Les protocoles légaux apparaissent comme un ensemble de règles à respecter pour que la concurrence des accès soit cohérentes. Cet ensemble dépend de la technique choisie dans l'architecture du SGBD. Néanmoins, tous les protocoles exigent que la transaction accédant à un état consistant de la base de données transforme celui-ci en un état consistant si on l'exécute isolément (ceci rapport à la définition des règles d'intégrité que l'on verra ultérieurement)
- L'**atomicité**; L'exécution des transactions doit être vue comme un tout, il n'existe pas de transaction inachevée. Les transformations de la base sont conservées si la transaction s'achève, donc si toutes les primitives de modifications se sont exécutées normalement. Si la transaction est inachevée, les transformations sont effacées et l'on restitue l'état précédent.
- La **durabilité**; une fois qu'une transaction est confirmée, on ne peut plus l'annuler (sauf par une nouvelle transaction)

La transaction munie de ces concepts est l'unité de transformation de la base de données et aussi l'unité de récupération des erreurs. Un ensemble de techniques ont été développées pour assurer ces propriétés. Elles

permettent la modification d'une base de données par plusieurs utilisateurs en toute sécurité. Et aussi elles suppléent aux éventuelles défaillances du matériel.

Les protocoles de gestion de la concurrence sont régis par le principe d'équivalence des ordonnancements des transactions [PAP79]. Imaginons que nous ayons l'ensemble $\{T_1, T_2, \dots, T_n\}$ de transactions à exécuter de manière concurrente. Si nous avons une machine infiniment rapide, la durée d'exécution des transactions serait ramenée à zéro et le résultat de la modification correspondrait à l'exécution séquentielle d'une des permutations possibles de l'ensemble de départ. Nous ne possédons pas une telle machine par contre, qui fixe l'ensemble des permutations exécutées séquentiellement comme référence admissible. Les protocoles de gestion (en utilisant des verrous sur les données) de la concurrence vont permettre un entrelacement entre les primitives des transactions mais assurer que le résultat final soit équivalent à une des permutations. Autrement dit ces protocoles ne permettent pas aux transactions de communiquer entre-elles. Une transaction peut modifier la base de données et cette modification ne doit pas être perçue par d'autres transactions tant qu'elle n'est pas confirmée (achevée).

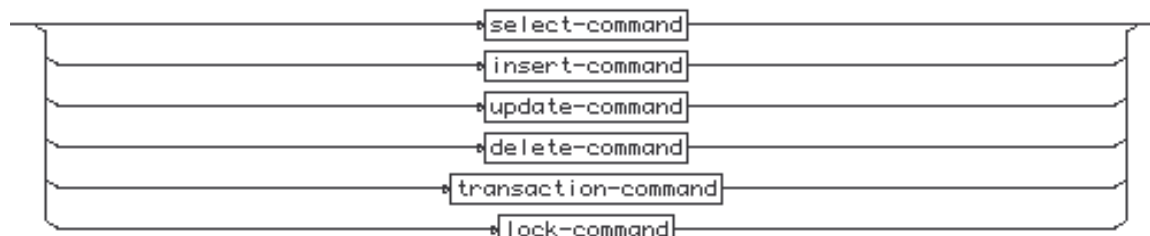
Dans les SGBD, il en va de même, une transaction peut modifier le contenu complet de la base de données, ces modifications ne seront perçues qu'au moment de la confirmation.

L'atomicité de la transaction doit prévenir la base de données de modifications incomplètes. Celles-ci peuvent être dues à une transaction ne pouvant s'achever (mauvais paramètres, limite des ressources), à une transaction erronée (les conditions erronées sont prévues et la transaction provoque elle-même l'abandon), ou plus simplement à une défaillance matérielle durant l'exécution. Cette dernière peut provoquer une perte du contenu de la mémoire vive ou d'un média de la mémoire secondaire. Quelqu'en soit la raison, l'atomicité exige que les effets des transactions en cours soient éliminés et que ceux des transactions confirmées soient restitués (si détruits). La tenue d'un journal (stocké indépendamment de la base de données) des modifications entreprises par les transactions constitue généralement la technique employée pour assurer l'atomicité. En cas d'interruption, il suffit de parcourir le journal, pour refaire les transactions confirmées et pour défaire celles qui ne le sont pas. Cette reprise elle-même sujette à des défaillances doit pouvoir être ré-exécuter autant de fois qu'il le faut.

Une part importante du code d'un SGBD est consacrée à la résolution de ces problèmes et l'on peut considérer qu'un logiciel qui négligerait ces concepts ne peut pas être exploité raisonnablement par une entreprise.

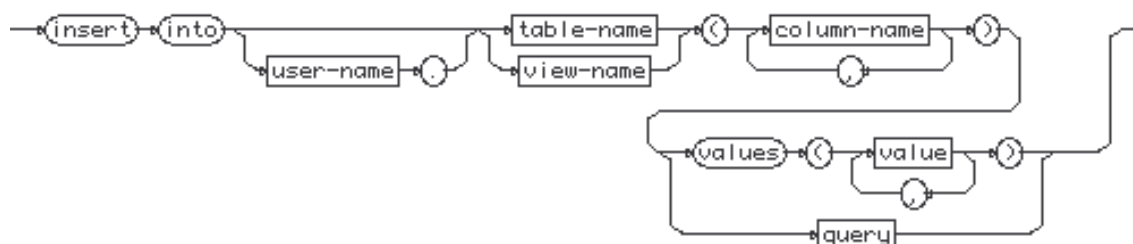
Langage de manipulation de données en SQL

Le langage SQL offre des primitives semblables. Elles définissent un sous-ensemble qui permet de manipuler les données des tables du SGBD. Aux trois primitives de base sont ajoutées les commandes permettant de spécifier des transactions et celles permettant définir les verrous pour la gestion de la concurrence.



Cette clause permet d'insérer soit une nouvelle rangée définie explicitement ou bien d'insérer un ensemble de rangées définies par le résultat d'une requête. Les colonnes non définies lors de l'insertion prennent la valeur null pour cette rangée. La liste des colonnes peut être évitée, dans ce cas les valeurs se sont associées aux colonnes dans l'ordre de la déclaration de la table (CREATE ...)

insert-command



Exemple:

création d'une nouvelle réservation pour un nouveau client

```
insert into clients
  (num_client, nom, prenom, adresse)
  values(2000, 'New', 'Man', '12 rue de la découverte')

insert into reservations
  (num_client, num_chambre, date_arr, date_dep)
  values(2000, 12, '12-jan-91', '14-jan-91')
```

La première modification est équivalente à :

```
insert into clients
  values(2000, 'New', 'Man', '12 rue de la découverte')
```

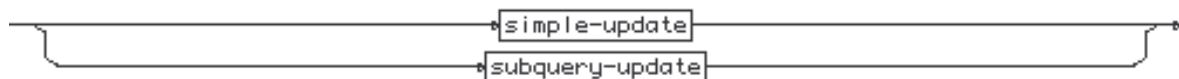
Imaginons que nous ayons créé une table CLIENTS_RECENTS qui doit contenir les clients ayant réservé à partir de l'année 1990. Nous avons l'ordre de création suivant et l'initialisation de la table peut s'effectuer avec une insertion accompagnée d'une sous-requête.

```
CREATE TABLE CLIENTS_RECENTS (
    NUM_CLIENT NUMBER (6) not null ,
    NOM CHAR (20),
    PRENOM CHAR (20),
    ADRESSE CHAR (40))

insert into CLIENTS_RECENTS
    (num_client, nom, prenom, adresse)
SELECT c.num_client, nom, prenom, adresse
    FROM CLIENTS c, RESERVATIONS r
    WHERE c.num_client=r.num_client
        AND date_arr>='1-jan-90'
```

Il est intéressant de noter que ce type d'insertion conduit à créer des informations redondantes. En effet, les informations qui sont insérées sont déjà par définition dans la base de données.

Update-command



Cette clause permet de mettre à jour une colonne ou un groupe de colonne. L'expression de mise à jour est soit une expression simple ou bien une sous-requête. La condition de la clause `WHERE` permet de spécifier les rangées qui doivent être mise à jour.

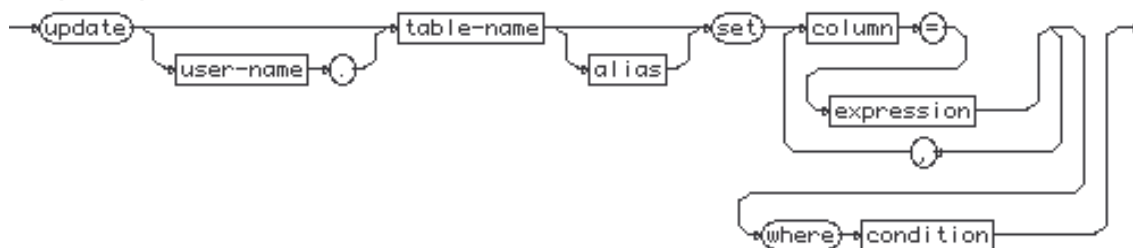
Exemple:

modification de la date de départ du client 1006

```
update reservations
    set date_dep='30-DEC-89'
    WHERE num_client=2000 AND date_dep='28-DEC-89';
```

augmentation de tous les prix de 10%

simple-update



```
update chambres
    set prix=prix*1.10
```

augmentation des prix de 10% pour les chambres avec bain

```
update chambres
    set prix=prix*1.10
    WHERE confort='BAIN';
```

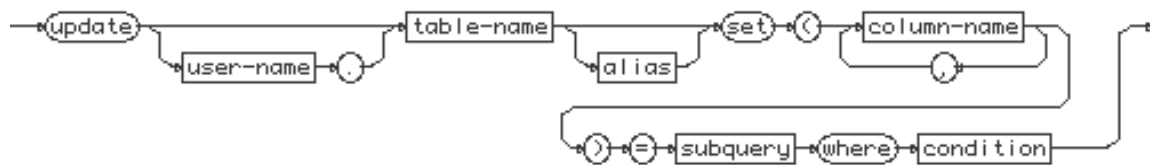
augmentation des prix de 10% + 5 Francs par personne pouvant occuper la chambre;

```
update chambres
    set prix=prix*1.10 + 5*nr_pers
```

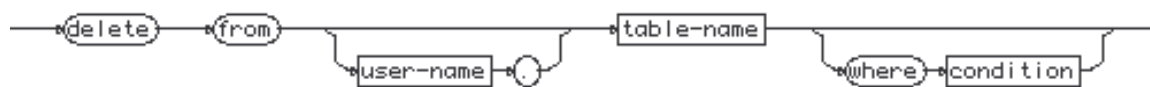
augmentation des prix au prix moyen en fonction du confort (ici les rangées de la table à modifier sont utilisées comme variable dans la sous-requête;

```
update chambres c1
  set (prix)=(SELECT avg(prix)
              FROM chambres c2
              WHERE c1.confort=c2.confort)
```

subquery-update



delete-command



Exemple:

Suppression de la chambre 23

```
delete FROM chambres
  WHERE num_chambre=23
```

Suppression de toutes les chambres

```
delete FROM chambres
```

transaction-command



Cette clause permet de spécifier les transactions. Le début de la transaction est implicite, il est situé directement après le dernier point de confirmation ou au début de la session de travail.

La commande `commit` définit la fin de la transaction. Les commandes de manipulation de la modélisation (`CREATE`, `drop`, ...) provoquent immédiatement après leur exécution un `commit` implicite.

La commande `set transaction` permet de définir explicitement le début d'une transaction dans le cas où l'on veut exécuter plusieurs requêtes consistantes par rapport à la lecture. L'option `read only` définit que cette transaction ne fera pas de modification.

La commande `savepoint ...` permet de définir des points de retour à l'intérieur même de la transaction. Il sera possible de revenir explicitement à l'état défini en ces points.

La commande `rollback ...` permet de retourner à l'état défini au début de la transaction. En spécifiant un savepoint, on revient à l'état défini à ce point.

exemple:

Une transaction confirmée

```
update chambres
  set prix=prix*1.10
commit
```

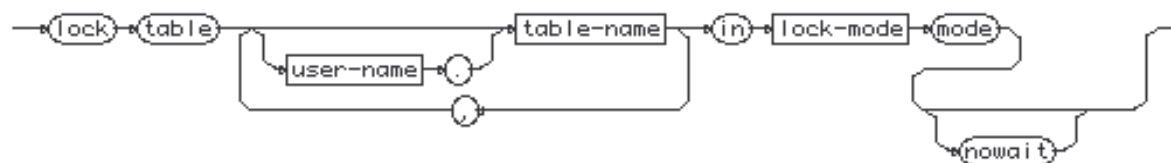
Après réception de ce message, les effets de la transactions sont visibles pour d'autres processus interrogeant la base de données

Une transaction annulée

```
update chambres
  set prix=prix*1.10
rollback
```

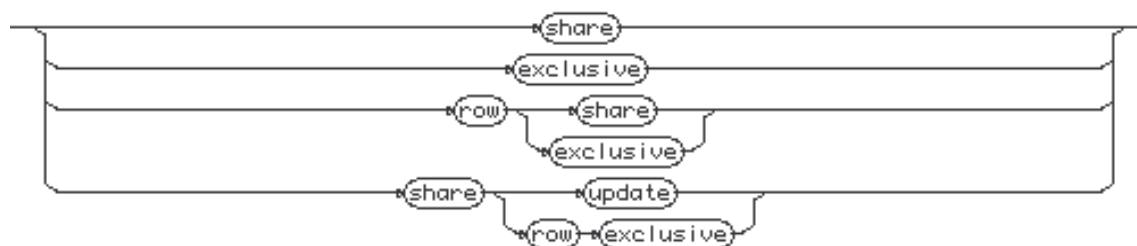
Les effets de la transaction sont annulés

lock-command



La gestion des verrous est, par défaut, assurée par le SGBD. Cependant, il est possible de spécifier pour une transaction les verrous demandés. Ceci sont posés au niveau de la table ou au niveau de la rangée. Dans le cas où un verrou est déjà posé par un autre processus, l'option `nowait` spécifie que la transaction ne doit pas attendre mais retourne le contrôle à l'utilisateur.

Lock-mode

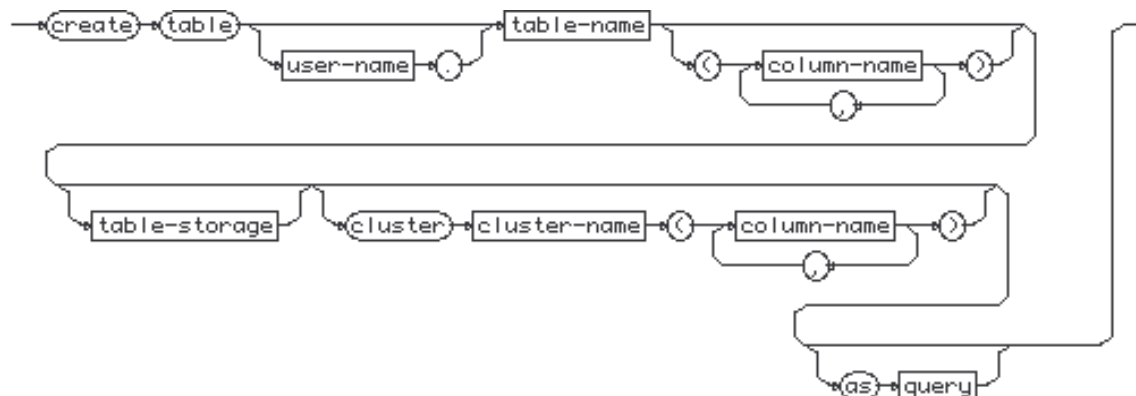


Les différents mode de verouillage sont les suivants:

- `share`; au niveau de la table, mode partagé en lecture, sans mise à jour possible
- `exclusive`; au niveau de la table, mode exclusif en lecture, ne permettant aucune modification
- `row share` (ou `row update`); au niveau des rangées, mode partagé en lecture, interdit un accès exclusif de la table.
- `row exclusive`; au niveau des rangées, mode partagé en lecture, interdit un accès partagé de la table.

- `share row exclusive;` au niveau des rangées, mode partagé en lecture, interdit un accès partagé de la table et la modification des rangées.

subquery-CREATE



Nous ajoutons cette clause ici car elle cumule deux effets, la création d'une table et l'insertion immédiate de données. Nous pouvons donc exprimer les deux requêtes précédentes pour `CLIENTS_RECENTS` en une seule:

```
CREATE TABLE CLIENTS_RECENTS AS
  SELECT c.num_client, nom, prenom, adresse
     FROM CLIENTS c, RESERVATIONS r
     WHERE c.num_client=r.num_client
           AND date_arr>='1-jan-90'
```

Par défaut, les noms et les types des colonnes sont pris dans le résultat de la requête.

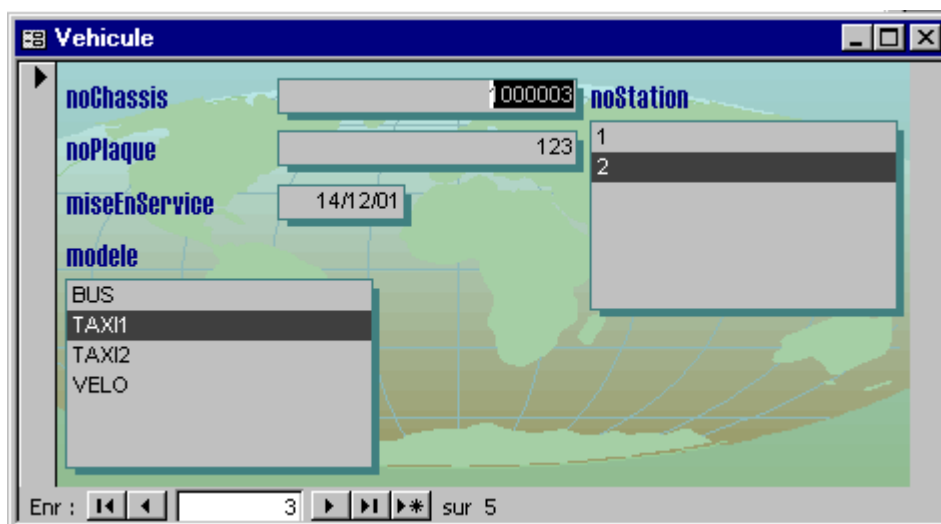


Figure 11-6 : Masque de modification pour les clients.

Interfaces utilisateurs

Il est évident que les commandes SQL vues plus haut ne peuvent pas être utilisées directement par l'utilisateur. Ces dernières doivent être insérées dans une interface permettant le dialogue entre l'utilisateur et le SGBD. Cette interface peut être réalisée avec un générateur d'écran ou avec un

langage de programmation (C++, Java, ...) Cette interface présentera à l'utilisateur un masque de saisie classique; dans chaque champ, il pourra entrer ses données. L'interface fera, elle l'appel au SGBD avec la commande SQL. Dans un même masque (Figure 11-6), il est possible de créer, sélectionner, mettre à jour et supprimer des données.

On peut effectuer certaines vérifications à la saisie, mais nous allons voir, dans le chapitre suivant, qu'il est aussi nécessaire de compléter la modélisation avec des règles d'intégrité portant sur l'ensemble des instances.

Exercices

Questions sur TT³

Donnez une requête SQL pour chaque point ci-dessous:

1. insérer l'entretien "peinture carrosserie" du véhicule 100004 au jour No 24
2. mettre à jour l'adresse du chauffeur No 1 qui est actuellement "23 chemin de l'avenir"
3. supprimer les informations sur le planning concernant les jours inférieurs à 100

Réponses sur TT³

1) `insert into Entretien (Nochassis, nojour, description)
values(100004, 24, 'peinture carrosserie')`

2) On remarque les deux apostrophes pour en insérer un !

```
update Chauffeur
  set adresse='23 chemin de l''avenir'
  WHERE nochauffeur=1
```

3)

```
delete FROM planning
  WHERE nojour<100;
```

On supprime l'effet de ces transactions avec le Rollback

```
Rollback
```

12. Les Règles d'intégrité

"Chaque pièce occupe une case et chaque case ne peut contenir qu'une seule pièce." (Camil Seneca - Les échecs)

Nous avons vu comment insérer, modifier et supprimer des entités dans une instance. Ces modifications doivent être le reflet d'événements appartenant au champ d'application. Il est évident qu'il est très simple de commettre des erreurs grossières lors de ces opérations. Cependant, le champ d'application est soumis à des invariants appelés règles d'intégrité qui permettent de réduire les risques de produire des instances inconsistantes. Par exemple, on ne peut pas réserver une chambre qui n'existe pas à un client inconnu.

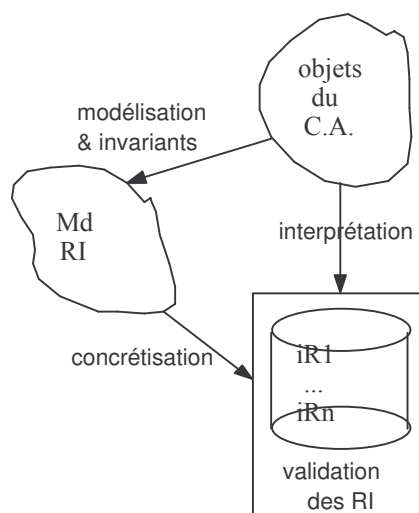


Figure 12-1 : Intégration des règles d'intégrité dans le processus de modélisation

Les règles d'intégrité sont des prédicats qui sont toujours évalués à vrai. À énoncer, elles semblent être des lapalissades : " la date d'arrivée précède la date de départ". Mais elles constituent un puissant moyen de vérification et d'élimination des instances qui ne peuvent pas exister dans un champ d'application. La modélisation du champ d'application s'accompagne donc d'une phase de recherche des invariants que l'on exprime sous forme de règles d'intégrité. Lors de l'étape de concrétisation, elles sont transformées en un ensemble de mécanismes qui valident ces règles.

Ces mécanismes sont variés ils peuvent être intégrés aux masques de saisie ou faire partie du noyau du SGBD et être spécifiés lors de la création d'une table ou plus simplement être des requêtes qui détectent la présence d'anomalies dans la base de données.

Définitions et propriétés

IR est l'ensemble des instances de R

$IR = \{iR \mid iR \text{ est une instance de } R\}$

IBD est l'ensemble des instances de la base de données

$IBD = \{iBD \mid iBD \in \prod IR_j, j=1..n\}$

Une règle d'intégrité ri est un prédicat ayant IBD pour domaine

$ri : IBD \rightarrow \{\text{vrai}, \text{faux}\}$

On dira que $iBD \in IBD$ *valide* la règle d'intégrité ri si et seulement si $ri(iBD) = \text{vrai}$

IBD/ri est l'ensemble des instances de la base de données validant ri

RI désigne l'ensemble des ri d'une modélisation

$RI = \{ri_1, \dots, ri_n\}$

On dira que $iBD \in IBD$ *valide* l'ensemble RI (ou est consistant par rapport à RI) si et seulement si iBD valide chaque ri_j de RI .

$RI : IBD \rightarrow \{\text{vrai}, \text{faux}\}$

$RI(iBD) = \prod ri_j(iBD), j=1..n$

$= ri_1(iBD) \wedge ri_2(iBD) \wedge \dots \wedge ri_n(iBD)$

IBD/RI est l'ensemble des instances de la base de données validant RI

$IBD/RI = \{iBD \mid RI(iBD)\}$

On remarque que

$IBD/RI = IBD/ri_1 \cap IBD/ri_2 \cap \dots \cap IBD/ri_n$

IBD définit un espace dont chaque dimension correspond à une relation. Chaque point de cet espace est associé à une instance de la base de données. IBD/ri définit le sous-ensemble des instances de la base de données validant la règle ri . IBD/RI est l'intersection de tous ces sous-ensembles.

Il s'agit donc d'établir des règles que valide le champ d'application indépendamment des valeurs des objets de ce dernier.

Dans le travail de N.H. Le Pham [PHA90] sur les règles d'intégrité, on trouve les propriétés suivantes:

Un ensemble RI est *trivial* si et seulement si toutes les instances valident RI , soit:

RI est trivial $\Leftrightarrow IBD/RI = IBD$

Par exemple, $ri: (Date_Arr \leq Date_Dep) \vee (Date_Arr \geq Date_Dep)$ est une ri triviale, n'importe quel couple de dates satisfait cette règle.

Un ensemble RI est *satisfaisable* si et seulement s'il existe une instance validant RI , soit:

RI est satisfaisable $\Leftrightarrow IBD/RI \neq \emptyset$

Dans le cas contraire on dit que RI est *insatisfaisable*, soit

RI est satisfaisable \Leftrightarrow IBD/RI = \emptyset

Par exemple, ri: (Date_Arr < Date_Dep) \wedge (Date_Arr > Date_Dep) est une ri insatisfaisable, aucun couple de dates satisfait cette règle.

La notion de satisfaisabilité est à rapprocher de celle de consistance en logique. Nous nous intéressons donc qu'aux ensembles de règles satisfaisables.

La règle ri est *impliquée* par RI (notée $RI \Rightarrow ri$, ou est la conséquence logique de RI) si et seulement si toutes les instances validant RI valident aussi ri, soit

$$(RI \Rightarrow ri) \Leftrightarrow (IBD/RI \subseteq IBD/ri)$$

Par exemple, ri2: (Date_Arr \leq Date_Dep + n jours) où n est positif, est impliquée par la ri1 : (Date_Arr \leq Date_Dep).

L'ensemble RI est *équivalent* à RI' si et seulement si toutes les instances validant RI valident aussi RI' et toutes les instances validant RI' valident aussi RI, soit

$$(RI \equiv RI') \Leftrightarrow (IBD/RI = IBD/RI')$$

Par exemple, ri1: (Date_Arr \leq Date_Dep) \wedge (Date_Arr \geq Date_Dep) est équivalente à ri2: (Date_Arr = Date_Dep).

L'ensemble RI est *plus contraignant* que RI' si et seulement si toutes les instances validant RI valident aussi RI', soit

$$(RI \leq RI') \Leftrightarrow (IBD/RI \subseteq IBD/RI')$$

Par exemple, ri1: (Date_Arr > 1-jan-90) est plus contraignante que ri2: (Date_Arr > 1-jan-89).

La *fermeture* de RI est l'ensemble des contraintes impliquées par RI.

$$RI^{++} = \{ri \mid RI \Rightarrow ri\}$$

RI est *irredondant* si et seulement s'il n'existe aucune règle ri \in RI telle que $(RI - \{ri\}) \Rightarrow ri$. Aucune ri ne peut donc se déduire des autres.

Les notions de fermeture et d'irredondance seront particulièrement utiles lorsque nous aborderons l'étude des dépendances fonctionnelles, un groupe particulier de règles d'intégrité. Elles permettront de rechercher l'ensemble minimal de ri.

La consistance d'une transaction

Nous sommes maintenant en mesure de comprendre la propriété de la consistance d'une transaction. Une transaction T s'exécute sur une instance consistante iBD et transforme celle-ci en une autre instance consistante iBD' par rapport à un ensemble de règles d'intégrité prévalant sur la base de données. En termes de pré et post-conditions, nous pouvons noter:

$$\{iBD \in IBD/RI\} T \{iBD' \in IBD/RI\}$$

Pendant la durée de la transaction, l'état de la base de données peut être inconsistant. Mais ces états ne sont perçus que par la transaction T.

L'atomicité et les protocoles de gestion de la concurrence assurent que seul l'état iBD' sera visible (ou iBD si on annule la transaction).

Dans la figure 2, les primitives de modification définissent les sauts minimaux d'un état à l'autre de la base de données. La transaction définit un parcours commençant et se terminant par un état consistant.

Ces règles d'intégrité sont *statiques* Car l'état initial et final ne sont pas liés, par opposition aux règles *dynamiques* où ces états sont liés. Classiquement l'état civil d'une personne est un exemple d'une règle dynamique, en effet à l'état marié ne peut succéder que l'état veuf ou divorcé (si l'on exclut les annulations de mariage, très rares !)

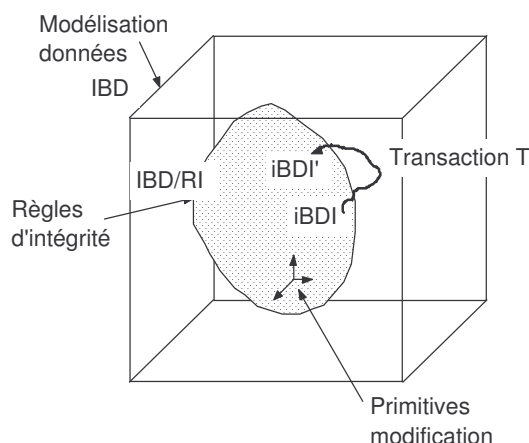


Figure 12-2 : Règles d'intégrité et transactions

Portée de la règle d'intégrité

Examinons les trois règles d'intégrité suivantes:

ri1: Le nombre des membres de l'Oulipo est un nombre premier et un nombre de Queneau.¹²

ri2: "La date de départ d'une réservation doit être postérieure à la date d'arrivée", soit

$$ri2: \forall r \in \text{Réservations} \quad (r.\text{datedep} > r.\text{datearr})$$

ri3: "Une même chambre ne peut être réservée par deux clients différents le même jour"

$$ri3: \forall r \in \text{Réservations}, \neg \exists r' \in \text{Réservations} \text{ tel que:}$$

$$\begin{aligned} & r.\text{Numclient} \neq r'.\text{Numclient} \\ \text{et} & \quad r.\text{Numchambre} = r'.\text{Numchambre} \\ \text{et} & \quad r.\text{Datearr} \leq r'.\text{Datedep} \\ \text{et} & \quad r.\text{Datedep} > r'.\text{Datearr} \end{aligned}$$

ri4: "Les numéros de clients dans Réservation sont définis dans Clients"

$$ri4: \text{Réservation}[\text{Num_client}] \subseteq \text{Clients}[\text{Num_client}]$$

¹² Point 8 des "Indications liminaires" de Jacques Roubaud in La bibliothèque Oulipienne, Editions Ramsay, 1987

ri5: "Les numéros de clients sont uniques"

ri5: $\forall r \in \text{Clients}, \neg \exists r' \in \text{Clients}$ tel que:
 $r[\text{Nom}, \text{Prenom}, \text{Adresse}] \neq r'[\text{Nom}, \text{Prenom}, \text{Adresse}]$
 et $r.\text{NumClient} = r'.\text{NumClient}$

On peut remarquer que la validation des ri précédentes demande des connaissances diverses sur les instances de la base de données.

ri1 peut être validée directement sur le constituant `nombre_de_membres` (si l'on constituait une base de données historique pour l'Oulipo). Cette règle est de type *domaine*, il suffit de connaître la valeur prise par le constituant pour valider la règle. Normalement, ces règles sont définies au niveau du domaine du constituant, cependant la complexité de certaines dépasse les capacités de modélisation des domaines (type, intervalle, ...)

ri2 peut être validée directement sur l'entité de `Réservation`. Cette règle est de type *intra-tuple* il suffit de connaître la valeur prise par l'entité pour valider la règle.

ri3 peut être validée directement sur l'instance de la relation `Réservation`. Cette règle est de type *intra-relation*, il suffit de connaître la valeur prise par plusieurs des entités d'une instance pour valider la règle.

ri4 demande pour être validée les instances de plusieurs relations différentes. Cette règle est de type *inter-relation*.

ri5 peut être validée directement sur l'instance de la relation `Clients`. Cette règle est de type *intra-relation*. On dit aussi que `Numclient` est une clé de la relation `Clients`. Une clé est un ensemble de constituants (minimum) qui détermine de manière unique tous les autres constituants de la relation (On reviendra sur cette notion dans le chapitre sur les dépendances fonctionnelles).

La figure 3 résume ces différents types. La complexité de la validation du type dépend directement de l'étendue de la règle. Un des objectifs lors de la conception de la base de données est de minimiser les coûts de validation de ces règles d'intégrité.

Directement liée à la validation d'une règle d'intégrité, nous avons la notion de portée d'une règle d'intégrité. Une primitive de modification appartient à la portée d'une règle ri si elle peut rendre invalide l'instance de la BD. C'est-à-dire qu'elle peut influencer la consistance de la base de données par rapport à ri

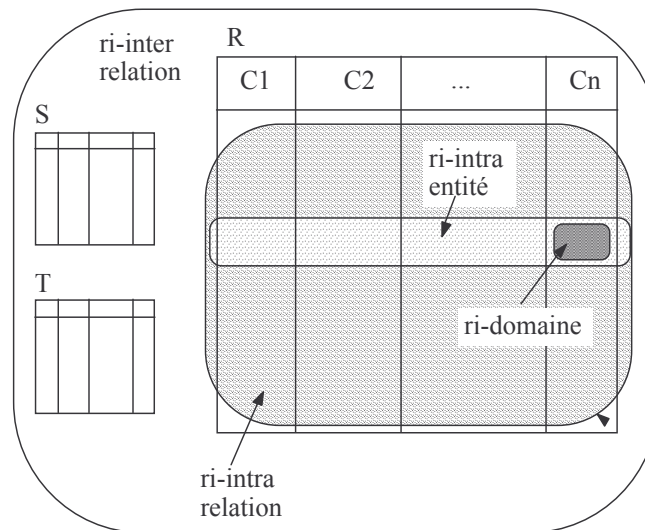


Figure 12-3 : Étendue des différents types de règles d'intégrité

Définition

Soit P l'ensemble des primitives applicables au schéma $S=\{R1, R2, \dots, Rn\}$ la *portée d'une règle* ri est l'ensemble des primitives de modification qui peuvent rendre invalide une instance iBD , soit:

$$P_p(ri) = \{p \in P \mid \{iBD \in IBD/ri\} p \{iBD' \notin IBD/ri\}\}$$

Pour les ri précédentes, nous avons (C =création, M =mise à jour, S =suppression, les $_$ correspondent aux paramètres non spécifiés des primitives):

$$P_p(ri1) = \{C(\text{Oulipo}, _), \\ M(\text{Oulipo}, _, \text{nombre_de_membres}, _)\}$$

$$P_p(ri2) = \{C(\text{Réservations}, _), \\ M(\text{Réservations}, _, \text{date_Arr}, _), \\ M(\text{Réservations}, _, \text{date_Dep}, _)\}$$

$$P_p(ri3) = \{C(\text{Réservations}, _), \\ M(\text{Réservations}, _, \text{num_client}, _), \\ M(\text{Réservations}, _, \text{num_chambre}, _), \\ M(\text{Réservations}, _, \text{date_Arr}, _), \\ M(\text{Réservations}, _, \text{date_Dep}, _)\}$$

$$P_p(ri4) = \{C(\text{Réservations}, _), \\ M(\text{Réservations}, _, \text{num_client}, _), \\ S(\text{Clients}, _), \\ M(\text{Clients}, _, \text{num_client}, _)\}$$

$$P_p(ri5) = \{C(\text{Clients}, _), \\ M(\text{Clients}, _, \text{num_client}, _)\}$$

On peut constater que la portée est en rapport avec les types de règle. La portée est utile pour déterminer les contrôles suffisants et nécessaires pour la BD par rapport aux transactions et aux interfaces utilisateurs.

D'une manière inverse, on peut définir la *sensibilité* d'une primitive par l'ensemble des ri dont elle est élément de la portée.

$$S_p(p) = \{ri \in RI \mid p \in P_p(ri)\}$$

Avec les ri ci-dessus, nous avons:

$$S_p(C(\text{Réservations}, _)) = \{ri2, ri3, ri4\}$$

$$S_p(S(\text{Clients}, _)) = \{ri4\}$$

On peut étendre les notions de portée et de sensibilité au niveau de la transaction.

Soit A une application définie comme l'ensemble des transactions $\{T_1, T_2, \dots, T_n\}$ où chaque T_i est définie par un ensemble de primitives de modification $T_i = \{p_{i1}, p_{i2}, \dots, p_{im}\}$.

La portée de la règle ri par rapport à l'application A est alors définie comme l'ensemble des transactions dont une des primitives appartient à la portée de la ri .

$$P_T(ri) = \{T \in A \mid \exists p \in T \text{ où } p \in P_p(ri)\}$$

On peut aussi définir la *sensibilité* d'une transaction par l'ensemble des ri dont elle est élément de la portée.

$$S_T(T) = \{ri \in RI \mid T \in P_T(ri)\}$$

Actuellement, les SGBD permettent de valider certaines règles d'intégrité exprimées lors de la conception du schéma. Ces règles sont restreintes à des catégories bien définies, dont la portée en terme de primitives est implicitement associée aux paramètres de la catégorie. Cela signifie que le SGBD détermine automatiquement la portée de ces règles et valide ces dernières dès qu'une de ces primitives est exécutée. Ces catégories sont:

- une règle *intra-tuple*, exprimée par un prédicat sur les constituants d'une relation. Les primitives de la portée sont la création d'une entité et la mise à jour d'un des constituants du prédicat de la règle
- exemple, ri : $\text{prix} < 50 * \text{nbr_pers}$
 $P_p(ri) = \{ C(\text{Chambres}, _), \\ M(\text{Chambres}, _, \text{prix}, _), \\ M(\text{Chambres}, _, \text{nbr_pers}, _) \}$
- une règle portant sur l'*unicité* des valeurs prises par un ensemble de constituants X d'une relation (on verra plus loin que ceci est une clé). Soit $\forall r \in iR, \forall r' \in iR \ r[X] \neq r'[X]$. Il n'existe donc pas deux entités ayant même valeur pour X . Les primitives de la portée sont la création d'une entité et la mise à jour d'un des constituants de X .
- exemple, ri : Unicité sur Num_chambre dans Chambres
 $P_p(ri) = \{ C(\text{Chambres}, _ , _), \\ M(\text{Chambres}, _, \text{num_chambre}, _) \}$
- une règle portant sur l'inclusion entre deux ensembles d'entités, exprimée par $R[X] \subseteq S[Y]$ (dépendance d'inclusion). Les primitives de la

portée sont la création d'une entité de R, la mise à jour d'un des constituants de X, la suppression d'une entité de S et la mise à jour d'un des constituants de Y,

- exemple déjà cité,
 $ri4: \text{Réservation}[\text{Num_client}] \subseteq \text{Clients}[\text{Num_client}]$
 $P_p(ri4) = \{ \text{C}(\text{Réservations}, _),$
 $\text{M}(\text{Réservations}, _, \text{num_client}, _),$
 $\text{S}(\text{Clients}, _),$
 $\text{M}(\text{Clients}, _, \text{num_client}, _) \}$

En définissant une de ces règles d'intégrité directement dans le SGBD, le concepteur ne doit plus se préoccuper de sa validation lors d'une transaction. Le SGBD la valide automatiquement, cependant il doit récupérer les messages d'erreur afin de les réinterpréter pour l'utilisateur.

Pour les règles d'intégrité n'entrant pas dans ces catégories, le concepteur peut envisager trois types de solutions.

- La *validation locale* de la ri, pendant l'exécution de la transaction; dans le code de la transaction, l'exécution de la primitive "dangereuse" pour l'intégrité de la ri est soumise à des pré-conditions validant la ri. Cette solution permet de minimiser les coûts de la validation. En effet, on connaît au moment de l'exécution de la primitive tous ses paramètres et on peut donc les utiliser pour circonscrire le contexte de la validation. Par exemple, pour assurer qu'une chambre n'est pas louée deux fois le même jour, lors de la transaction de réservation, la validation ne portera que sur la chambre et la période concernée par les paramètres de la transaction. Cette solution a le désavantage de coder dans les transactions les règles. En cas de modifications de ces règles, il faut revoir toutes les transactions appartenant à leur portée.
- La *validation globale* de la ri, après l'exécution de la transaction; avant la confirmation de la transaction, on valide toutes les règles d'intégrité appartenant à la sensibilité de la transaction. Dans cette solution, on perd l'efficacité de la solution précédente, par contre, on gagne en souplesse en cas de modification des règles d'intégrité.
- Le *diagnostic*; aucune validation n'est effectuée pendant l'exécution des transactions. La base de données peut devenir inconsistante. Il existe cependant pour chaque règle, une possibilité de diagnostiquer les anomalies. Ces diagnostics sont exécutés avant d'entreprendre certaines transactions où il est indispensable que la base soit consistante. La stratégie est donc de permettre la mémorisation des données et d'effectuer périodiquement des corrections sur les anomalies.

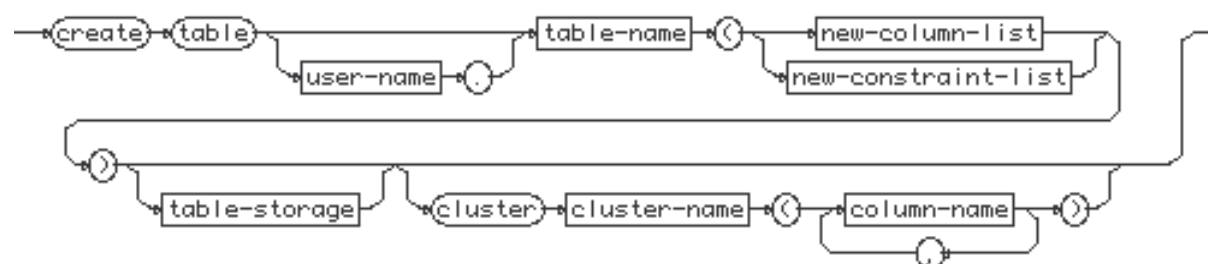
Un système d'information complexe nécessite généralement l'utilisation des trois types de solutions. La validation locale pour son efficacité et pour préserver des temps de réponse acceptables. La validation globale lorsque les règles sont régulièrement modifiées (plus vite que le temps de

reprogrammation des transactions). Le diagnostic quand il est important de saisir au plus vite l'information, en assumant une période de consolidation des données pour éliminer les anomalies.

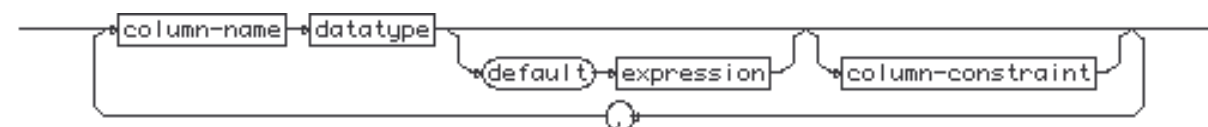
7.4. Les règles d'intégrité en SQL

Examinons la syntaxe du langage SQL pour définir les règles d'intégrité. Ces dernières sont spécifiées soit à la création de la table ou bien lors d'une modification de la structure de la table. Deux types de règles sont admis, l'un portant sur une seule colonne et l'autre portant sur un groupe de colonnes. Un identifiant (nom) peut être associé à chaque règle.

CREATE-TABLE



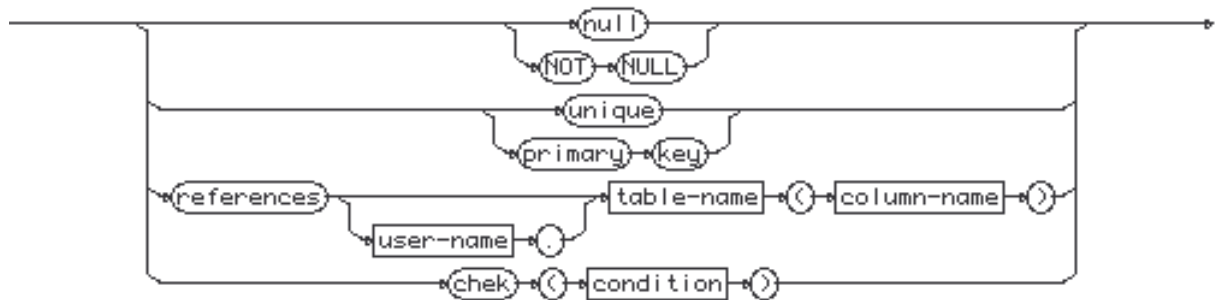
new-column-list



Les règles portant sur une colonne seulement sont déclarées immédiatement après le type de la colonne. On trouve les clauses suivantes :

- `Null/ Not Null` : interdit que la colonne puisse prendre des valeurs nulles (non-définies).
- `Unique` : la colonne est une clé de la table. Les valeurs de cette colonne sont toutes distinctes les unes des autres. (La colonne doit aussi être spécifiée comme `not null`)
- `Primary Key` : identique à `unique`. La clé primaire est la clé que le concepteur privilégie dans les accès et les jointures avec cette table, elle est indexée.
- `References ...` : les valeurs de cette colonne doivent être des éléments de la table projetée sur la colonne de référence. La colonne de référence doit être spécifiée comme `unique` (ou clé primaire)
- `Check(...)` : un prédicat est associé à la colonne, il utilise seulement les noms des constituants de la table et se vérifie localement sur la rangée (ceci exclut les sous requêtes)

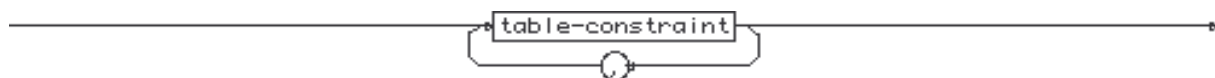
column-constraint



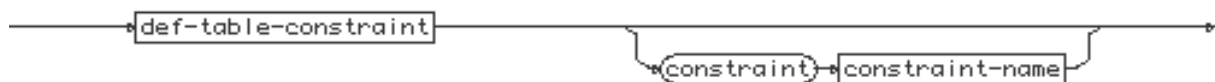
Il est possible de déclarer des règles portant sur plusieurs colonnes. On trouve les clauses suivantes:

- `Unique(...)` : le groupe de colonnes est une clé de la table. Les valeurs de ces colonnes sont toutes distinctes les unes des autres. (Les colonnes doivent aussi être spécifiées comme `not null`)
- `Primary Key(...)` : identique à la clause unique. Une seule clé primaire est autorisée par table.
- `Foreign Key () References ...` : les valeurs de ce groupe de colonnes doivent être des éléments de la table projetée sur le groupe de référence. Les colonnes de référence doivent être spécifiées comme unique (ou clé primaire)
- `Check(...)` : déjà vu plus haut

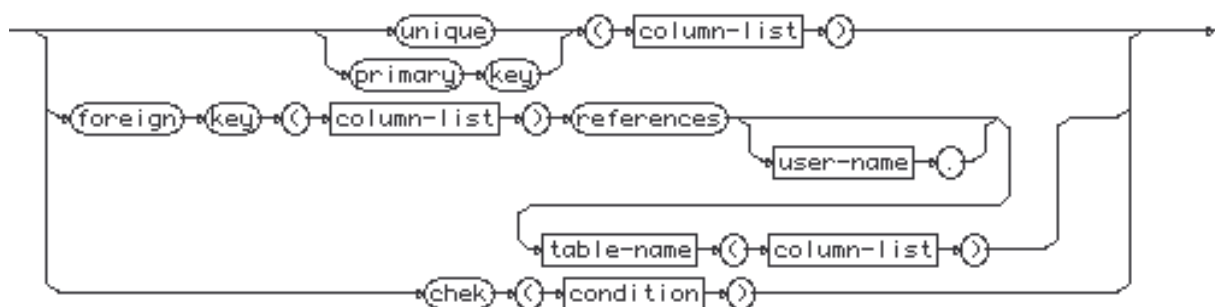
`new-constraint-list`



`table-constraint`



`def-table-constraint`



Exemple:

Reprenons notre modélisation Hôtel et ajoutons les règles d'intégrités lors de la création des tables. Nous obtenons:

```
CREATE TABLE CLIENTS (
    NUM_CLIENT NUMBER (6)    not null ,
    NOM CHAR (20)           not null
    check (NOM=upper(NOM)) constraint CLIENTS_RI1,
    PRENOM CHAR (20),
    ADRESSE CHAR (40),
    primary key (NUM_CLIENT) constraint CLIENTS_RI2)
```

Num_client est la clé. Tous les noms sont en majuscule.

```
CREATE TABLE CHAMBRES (
  NUM_CHAMBRE NUMBER (2)    not null,
  PRIX NUMBER (8,2)        not null,
  NBR_LITS NUMBER (1) not null
  check (NBR_LITS between 1 AND 4) constraint CHAMBRES_RI1,
  NBR_PERS NUMBER (1) not null
  check (NBR_PERS>NBR_LITS)
  constraint CHAMBRES_RI2,
  CONFORT CHAR(6) not null
  check (CONFORT in ('bain','WC','douche'))
  constraint CHAMBRES_RI3,
  EQUIPEMENT CHAR(3) not null
  check (EQUIPEMENT in ('TV','NON'))
  constraint CHAMBRES_RI4,
  primary key (NUM_CHAMBRE)
  constraint CHAMBRES_RI5)
```

Num_chambre est la clé. Le nombre de lits est compris entre 1 et 4. Le nombre de personnes est supérieur au nombre de lits. Le confort et les équipements sont restreints à certaines valeurs.

```
CREATE TABLE RESERVATIONS (
  NUM_CLIENT NUMBER (6) not null,
  NUM_CHAMBRE NUMBER (2) not null,
  DATE_ARR DATE not null
  check (DATE_ARR<DATE_DEP)
  constraint RESERVATIONS_RI1,
  DATE_DEP DATE,
  primary key (NUM_CHAMBRE,DATE_ARR)
  constraint RESERVATIONS_RI2,
  foreign key (NUM_CLIENT)
  references CLIENTS (NUM_CLIENT)
  constraint RESERVATIONS_RI4,
  foreign key (NUM_CHAMBRE)
  references CHAMBRES (NUM_CHAMBRE)
  constraint RESERVATIONS_RI5)
```

La clé primaire est le groupe num_chambre et date_arr. Les chambres et les clients de réservations sont connus dans les relations de référence. La date de départ est postérieure à la date d'arrivée.

Imposer un ensemble de valeurs pour une colonne est délicat du point de vue de la conception, car le jour où se groupe évolue il faut modifier la règle et il semble que les SGBD aient encore quelques difficultés lors de telles modifications. Par exemple, dans le futur, il faudra peut-être ajouter 'sauna' aux valeurs de confort. Ceci est donc à réserver à des groupes très stables (oui/non, par exemple) autrement, il faut définir des tables pour les codes.

Les règles définies avec des références déterminent un ordre de création et de suppression des tables. En effet, la table de référence doit être créée avant celle qui y fait référence et supprimée après cette dernière. Ainsi il faudra supprimer les relations d'hôtel dans l'ordre suivant.

```
CREATE TABLE T2
  (A char(8) not null unique,
  B char(8) not null references T1(B))
alter TABLE T1 add ( foreign key (A) references T2(A))
```

Rappelons que certaines ri ne sont pas exprimables directement dans le SGBD. Par exemple: pas deux réservations le même jour pour la même chambre. On peut alors écrire, des requêtes pour déterminer les anomalies. Dans notre cas, après insertion d'une réservation erronée, on obtient:

```
insert into reservations values(1004,14,'27-DEC-89','30-DEC-89');
```

rem: ne tient pas compte des valeurs nulles de date_dep

```
SELECT *
  FROM Reservations r1, Reservations r2
 WHERE r1.num_chambre=r2.num_chambre
 AND r1.num_client<>r2.num_client
 AND r2.date_arr between r1.date_arr AND r1.date_dep
 AND r2.date_arr<>r1.date_dep;
```

Rem: on défait notre requête pour éliminer les données du test
rollback;

Dans le chapitre suivant, nous allons examiner les dépendances fonctionnelles qui sont un type de règles d'intégrité qui permettent de trouver les clés d'une relation, les dépendances d'inclusion et surtout d'étudier les propriétés d'une décomposition.

Exercice

Questions sur TT3

- 1) Redéfinir le schéma de TT3 pour tenir compte des règles d'intégrité
- 2) Déterminer l'ordre de suppression des tables (inverse de l'ordre de création)
- 3) Définir des ri ne pouvant être directement validées par le SGBD pour TT3 et donner l'ordre SQL pour détecter les anomalies.

Réponses sur TT3

Création du schéma

```
CREATE TABLE Station(  
    noZone    number(2),  
    noStation number(1) not null  
        primary key constraint station_RI1); CREATE TABLE  
Type(  
    modele    char(12) not null  
        primary key  
            constraint Type_RI1,  
    nbPlaces  number(2) not null  
        check (nbPlaces>0)  
            Constraint Type_RI2,  
    categorie char(2) not null  
        check (categorie in ('V','PL'))  
            Constraint Type_RI3,  
    typeCarburant char(12) not null  
        check (typeCarburant in ('ESSENCE','DIESEL'))  
            Constraint Type_RI4,  
    automatique char(1) not null  
        check (automatique in ('Y','N'))  
            Constraint Type_RI5,  
    poids     number(5) not null  
        check (poids>0)  
            Constraint Type_RI6);  
CREATE TABLE Distance(  
    heure     number(2),  
    zoneDe    number(2) not null,  
    zoneA     number(2) not null,  
    tempsParcours number(3) not null  
        check (tempsParcours>0)  
            Constraint Distance_RI1,  
    primary key (zoneDe,zoneA)  
        constraint Distance_RI2);  
CREATE TABLE Vehicule(  
    noChassis number  
        primary key  
            constraint Vehicule_RI1,  
    noPlaque  number not null  
        unique  
            constraint Vehicule_RI2,  
    miseEnService date not null,  
    modele    char(12) not null  
        references Type(modele)  
            constraint Vehicule_RI3,  
    noStation number(1) not null  
        references Station(noStation)  
            constraint Vehicule_RI4);
```

```
CREATE TABLE Chauffeur(  
    noChauffeur    number not null  
        primary key constraint Chauffeur_RI1,  
    nom    char(24) not null,  
    prenom    char(24) not null,  
    adresse    char(80) not null,  
    noStation number(1) not null  
        references Station(noStation)  
        constraint Chauffeur_RI2);  
  
CREATE TABLE Situation(  
    noChassis number not null  
        primary key  
        constraint Situation_RI1  
        references Vehicule(noChassis)  
        constraint Situation_RI2,  
    noZone    number(2) not null);  
  
CREATE TABLE Carburant(  
    noPlaque number not null,  
    noJour number(3) not null,  
    kilometrage number not null  
        check (kilometrage>0)  
        Constraint Carburant_RI1,  
    litres    number(3) not null  
        check (litres>0)  
        Constraint Carburant_RI2,  
    typeCarburant char(12) not null  
        check (typeCarburant in ('ESSENCE', 'DIESEL'))  
        Constraint Carburant_RI3,  
    primary key (noPlaque,noJour,kilometrage)  
        constraint Carburant_RI4);  
  
CREATE TABLE Entretien(  
    noChassis number not null  
        references Vehicule(noChassis)  
        constraint Entretien_RI1,  
    noJour    number(3) not null,  
    description char(240) not null,  
        primary key (noChassis,noJour)  
        constraint Entretien_RI2);  
  
CREATE TABLE Permis(  
    noChauffeur    number not null  
        references Chauffeur(noChauffeur)  
        constraint Permis_RI1,  
    categorie char(2) not null  
        check (categorie in ('V', 'PL'))  
        Constraint Permis_RI2,  
    primary key (noChauffeur,categorie)  
        constraint Permis_RI3);  
  
CREATE TABLE Planning(  
    noChauffeur    number not null  
        references Chauffeur(noChauffeur)
```

```
        constraint Planning_RI1,  
noChassis number not null  
        references Vehicule(noChassis)  
        constraint Planning_RI2,  
noJour    number(3) not null,  
trancheHoraire char(1) not null  
        check (trancheHoraire in ('A','B','C'))  
        Constraint Planning_RI3,  
primary key (noChauffeur,noChassis,noJour)  
        constraint Planning_RI4);
```

Suppression du Schéma

```
drop TABLE Carburant;  
drop TABLE Entretien;  
drop TABLE Planning;  
drop TABLE Permis;  
drop TABLE Situation;  
drop TABLE Vehicule;  
drop TABLE Chauffeur;  
drop TABLE Distance;  
drop TABLE Type;  
drop TABLE Station;
```

Autres règles d'intégrité

rechercher d'autres règles pertinentes

13. Dépendances fonctionnelles

"Dès que l'on pose une question quelle qu'elle soit, on introduit le doute, on cesse de réciter, on va voir. Et des tas de surprises nous attendent, y compris dans ce qui a l'apparence du banal" (Boris Cyrulnik - De la parole comme une molécule)

Depuis son introduction par Codd en 1970, le concept de dépendance fonctionnelle reste un des outils principaux du concepteur de base de données. Il permet :

- d'exprimer des invariants de la modélisation ;
- de définir des modélisations équivalentes (conduisant à des implantations différentes) ;
- d'assurer l'utilisation correcte des charnières lors de la composition ;
- d'être un outil d'analyse lors du processus de conception (ou de restructuration) de la BD.

Les dépendances fonctionnelles sont un outil d'analyse des relations car elles sont fondées sur les propriétés existant entre les constituants eux-mêmes, sans tenir compte des regroupements de ces derniers en relation. Les relations sont donc subordonnées aux dépendances fonctionnelles.

Définition: la dépendance fonctionnelle (df)

Soit R une relation où $X \subseteq R^+$ et $Y \subseteq R^+$

Alors Y dépend fonctionnellement de X (noté: $X \rightarrow Y$) si et seulement si

$\forall r$ et $r' \in iR$ ($r.X=r'.X \rightarrow r.Y=r'.Y$)

On dit aussi que X *détermine* Y . Ceci signifie que si on a deux entités de R ayant des valeurs identiques pour X alors elles ont aussi des valeurs identiques pour Y .

Dans notre exemple Hôtel, nous avons $\text{Num_Client} \rightarrow \text{Nom}$. Nous avons donc pour un numéro client qu'un seul nom. La dépendance fonctionnelle spécifie donc l'unicité entre deux groupes de constituants. Nous ne pouvons pas dire l'inverse $\text{Nom} \rightarrow \text{Num_Client}$, car il peut exister deux personnes s'appelant Dupont associées à des numéros clients différents.

Il ne faut pas confondre les dépendances fonctionnelles avec les clés des relations (bien qu'une clé détermine toujours une df). Les dépendances fonctionnelles permettent de déterminer les clés et non l'inverse.

Les dépendances fonctionnelles sont à considérer comme des règles d'intégrité. Pour l'exemple Hôtel, nous avons l'ensemble de df suivant:

Num_chambre → Prix
 Num_chambre → Confort
 Num_chambre → Nbr_lit
 Num_chambre → Nbr_pers
 Num_chambre → Equipement
 Num_client → Nom
 Num_client → Prénom
 Num_client → Adresse
 Num_chambre,DateArr → Date_Dep
 Num_chambre,DateArr → Num_client

Il existe aussi une représentation graphique des df (utile pour la recherche des clés). Les noeuds sont constituants et les étoiles représentent les df. Les arcs orientés du graphe détermine les groupes de constituants intervenant dans les df.

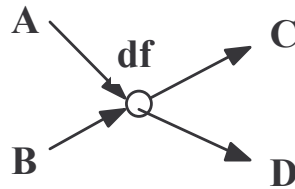


Figure 13-1 : représentation graphique de la df $AB \rightarrow CD$

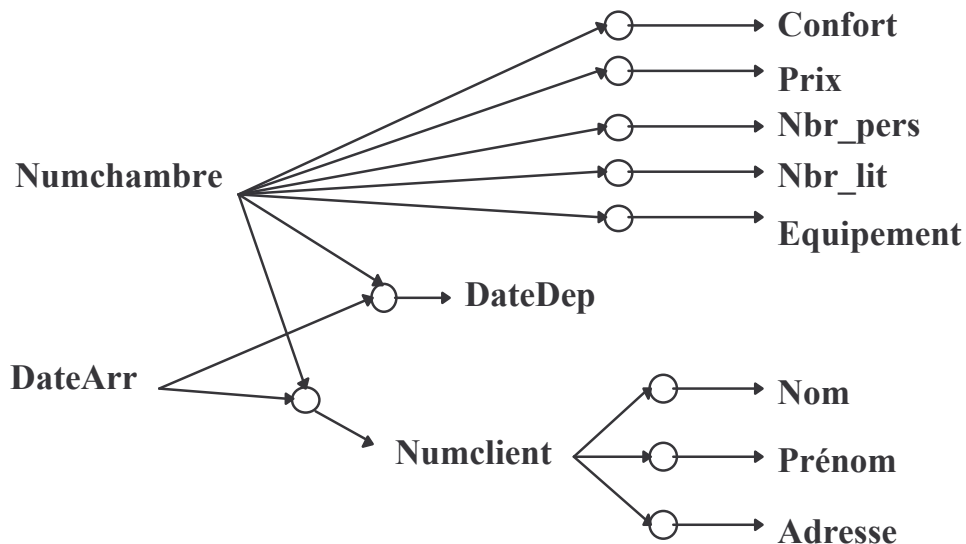


Figure 13-2 : représentation graphique des df d'Hôtel

En regardant la Figure 13-2, on a envie de dire intuitivement: "Si Num_chambre et Date_Arr déterminent Num_client et si Num_client détermine Nom alors Num_chambre et Date_Arr détermine Nom" Soit:

Num_chambre,DateArr → Num_client
 Num_client → Nom
 Num_chambre,DateArr → Nom

A partir des dépendances fonctionnelles, on peut dériver de nouvelles df. Formalisons notre intuition.

Fermeture et conséquence logique de F

Définitions

Le *schéma de relation* est le couple $S=(R,F)$ où R est la modélisation d'une relation et F est un ensemble de dépendances fonctionnelles exprimées sur R

Soit $S=(R,F)$ et f une df sur R , on dira que f est la *conséquence logique* de F si toutes les instances de R valident f . (On notera $F \Rightarrow f$)

Soit $S=(R,F)$, la *fermeture* de F , notée F^{++} est l'ensemble des dépendances fonctionnelles qui sont la conséquence logique de F . Soit:

$$F^{++}=\{X \rightarrow Y \mid F \Rightarrow X \rightarrow Y\}$$

La fermeture de F répond à notre questionnement précédant. Nous avons le droit de déduire une nouvelle df si elle appartient à la fermeture de F , donc si toutes les instances de R valident f . Dans la figure 3, nous explicitons cette notion de fermeture qui apparaît dès que nous avons un ensemble initial de df et un ensemble de règles de manipulation de l'ensemble initial. Il est alors important de déterminer ce que nous pouvons produire à partir de là. Dans les jeux télévisés où les concurrents doivent déterminer un nombre donné à partir d'un ensemble limité de nombres de départ en utilisant les opérations arithmétiques, il s'agit de déterminer si le nombre donné est dans la fermeture de l'ensemble des nombres initiaux. Les livres sont dans la fermeture des mots du dictionnaire, structurés par les règles de grammaire.

La fermeture d'un ensemble de df F doit être déterminée par les instances des relations validant F .

Exemple: soit le schéma $S(R;\{A, B, C\},\{A \rightarrow B, C\})$

Le tableau suivant exprime toutes les df possibles pour le schéma S . Pour le remplir, nous avons utilisé les raisonnements suivant:

oui-identité: par définition $X \rightarrow X$

oui-contenu: par définition $X \rightarrow Y$ est vrai si Y est contenu dans X

non-contre-exemple: l'instance du contre-exemple vérifie la df du schéma mais ne vérifie pas la df du tableau, par exemple

a	b	c
a'	b'	c
a''	b	c''

oui-df: La df du tableau peut être définie à partir de la df du schéma, par exemple $AC \rightarrow B$ car on a déjà $A \rightarrow B$

□	C	B	BC	A	AC	AB	ABC
C	OUI-IDENTITE	NON-CONTREEX	NON-CONTREEX	NON-CONTREEX	NON-CONTREEX	NON-CONTREEX	NON-CONTREEX
B	NON-	OUI-	NON-	NON-	NON-	NON-	NON-

	CONTREEX	IDENTITE	CONTREEX	CONTREEX	CONTREEX	CONTREEX	CONTREEX
BC	OUI-CONTENU	OUI-CONTENU	OUI-IDENTITE	NON-CONTREEX	NON-CONTREEX	NON-CONTREEX	NON-CONTREEX
A	OUI-DF	OUI-DF	OUI-DF	OUI-IDENTITE	OUI-DF	OUI-DF	OUI-DF
AC	OUI-CONTENU	OUI-DF *	OUI-DF	OUI-CONTENU	OUI-IDENTITE	OUI-DF	OUI-DF
AB	OUI-DF	OUI-CONTENU	OUI-DF	OUI-CONTENU	OUI-DF	OUI-IDENTITE	OUI-DF
ABC	OUI-CONTENU	OUI-CONTENU	OUI-CONTENU	OUI-CONTENU	OUI-CONTENU	OUI-CONTENU	OUI-IDENTITE

On constate que le calcul de la fermeture est un travail fastidieux si l'on doit passer par les instances. Comment calculer la fermeture sans devoir passer explicitement par l'ensemble des instances de R soumises à F ? Nous allons voir qu'il existe un système de déduction qui à partir du schéma, nous permet de travailler sur les df sans devoir examiner les instances.

Un système de déduction

Les axiomes de Armstrong [ARM74] constituent un système de déduction. En effet, en les appliquant sur un schéma S, il est possible de déduire de nouvelles dépendances fonctionnelles.

Définition: Règles de déduction

Soit un schéma $s=(R,F)$

DF1 (réflexivité):

Si $X \subseteq Y \subseteq R^+$ alors $Y \rightarrow X$ est une conséquence logique de F

DF2 (augmentation):

Si $X \rightarrow Y$ est une conséquence logique de F et $Z \subseteq R^+$ alors $XZ \rightarrow YZ$ est une conséquence logique de F

DF3 (transitivité):

Si $X \rightarrow Y$ et $Y \rightarrow Z$ sont des conséquences logiques de F alors $X \rightarrow Z$ est une conséquence logique de F

Exemple:

Soit le schéma $S=(R(A,B,C), \{A \rightarrow B, B \rightarrow C\})$

En utilisant DF1, on déduit:

DF1 $\Rightarrow ABC \rightarrow A$, car $A \subseteq ABC$
 $\Rightarrow AB \rightarrow A$, car $A \subseteq AB$
 $\Rightarrow A \rightarrow A$, car $A \subseteq A$
 $\Rightarrow ABC \rightarrow B$, car $B \subseteq ABB$

...

En utilisant DF2, on déduit:

DF2 appliqué à $A \rightarrow B$ et $BC \subseteq R^+$ $\Rightarrow ABC \rightarrow BC$
DF2 appliqué à $A \rightarrow B$ et $C \subseteq R^+$ $\Rightarrow AC \rightarrow BC$

En utilisant DF3, on déduit:

$$\text{DF3 appliqué à } A \rightarrow B \text{ et } B \rightarrow C \quad \Rightarrow A \rightarrow C$$

Les df obtenues par dérivation peuvent à leur tour être utilisées dans un processus de dérivation

$$\text{DF2 appliqué à } A \rightarrow CB \subseteq R^+ \quad \Rightarrow AB \rightarrow BC$$

Ce système de dérivation nous permet donc de dériver mécaniquement de nouvelles df. Avant de l'utiliser, nous devons nous assurer qu'il est valide et complet.

Le système est-il **valide**? C'est à dire: toutes les df déduites sont-elles des conséquences logiques de F donc qui sont validées par les instances de R.

Le système est-il **complet**? C'est à dire: toutes les conséquences logiques de F sont-elles des df déductibles du système.

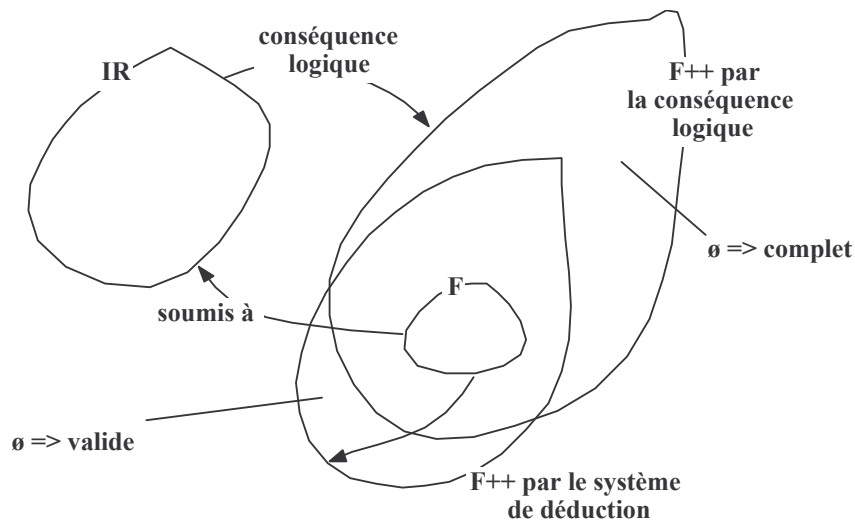


Figure 13-3 : Validité et complétude du système de déduction

On peut calculer la fermeture de F par deux mécanismes. La validité et la complétude du système de déduction se préoccupent de l'égalité de ces deux fermetures.

Validité du système de déduction

Proposition 1

DF1, DF2, DF3 sont valides

Preuve:

DF1 : appliquer la définition de df

$$X \rightarrow Y \text{ ssi } \forall r \text{ et } r' \in iR (r.X=r'.X \rightarrow r.Y=r'.Y)$$

mais $Y \subseteq X$ donc si $r.X=r'.X$ on a aussi $r.Y=r'.Y$

DF2: par contradiction de $XZ \rightarrow YZ$

Par définition, on a $\exists r$ et $r' \in iR$ tel que

$$\text{mais } X \rightarrow Y \quad \begin{array}{ll} 1) & r.XZ=r'.XZ \quad \text{et} \quad 1') \quad r.YZ \neq r'.YZ \\ 2) & r.X=r'.X \quad \text{et} \quad 2') \quad r.Y=r'.Y \end{array}$$

$$1) \text{ et } 2) \text{ impliquent } r.Z=r'.Z$$

1') et 2') impliquent $r.Z \neq r'.Z$

on a une contradiction donc a bien $XZ \rightarrow YZ$ vrai

DF3: par les définitions

1) $X \rightarrow Y \Rightarrow \forall r \text{ et } r' \in iR (r.X=r'.X \rightarrow r.Y=r'.Y)$

2) $Y \rightarrow Z \Rightarrow \forall r \text{ et } r' \in iR (r.Y=r'.Y \rightarrow r.Z=r'.Z)$

mais 1) et 2) $\forall r \text{ et } r' \in iR (r.X=r'.X \rightarrow r.Z=r'.Z)$ soit la définition de $X \rightarrow Z$

Extension du système de déduction

Avec les règles DF1, DF2 et DF3, on peut construire de nouvelles règles utiles dans la manipulation des df.

Proposition 2

Soit $S=(R; F)$

DF4 (Pseudo transitivité) :

si $X \rightarrow Y$ et $YW \rightarrow Z$ sont des conséquences logiques
alors $XW \rightarrow Z$ est une conséquence logique

DF5 (Union) :

si $X \rightarrow Y$ et $X \rightarrow Z$ sont des conséquences logiques
alors $X \rightarrow YZ$ est une conséquence logique

DF6 (Décomposition) :

si $X \rightarrow Y$ est une conséquence logique et $Z \subseteq Y$
alors $X \rightarrow Z$ est une conséquence logique

Preuve:

DF4:

DF2 sur $X \rightarrow Y$ avec $W \Rightarrow XW \rightarrow YW$

DF3 sur $XW \rightarrow YW$ et $YW \rightarrow Z \Rightarrow XW \rightarrow Z$

DF5.

DF2 sur $X \rightarrow Y$ avec $X \Rightarrow 1) X \rightarrow XY$

DF2 sur $X \rightarrow Z$ avec $Y \Rightarrow 2) XY \rightarrow YZ$

DF3 sur 1) et 2) $\Rightarrow X \rightarrow YZ$

DF6

DF1 à $Z \subseteq Y \Rightarrow Y \rightarrow Z$

DF3 sur $X \rightarrow Y$ et $Y \rightarrow Z \Rightarrow X \rightarrow Z$

Les règles DF5 et DF6, nous donnent une équivalence entre

$$X \rightarrow Y_1 Y_2 \dots Y_n$$

et

$$X \rightarrow Y_1$$

$$X \rightarrow Y_2$$

...

$$X \rightarrow Y_n$$

Forme canonique de F et saturation de X

On dira que F est en *forme canonique* si toutes les df de F ont un seul constituant dans la partie droite

Définition

Soit le schéma $S=(R,F)$ et $X \subseteq R^+$ alors on dit que X^{++} est la *saturation de X*

où $X^{++}=\{C \in R^+ \mid X \rightarrow C \in F^{++}\}$

Les constituants de X^{++} sont déterminés par F à partir de X. Les constituants de X^{++} s'interprètent comme les informations que l'on peut déterminer en connaissant seulement X dans un schéma R contraint par F.

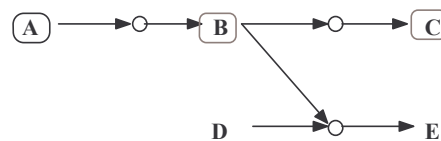
Sur un graphe de df, il est possible de déterminer simplement X^{++} en procédant ainsi:

- 1) mettre tous les noeuds de R^+
- 2) dessiner le graphe de F
- 3) marquer les noeuds de X
- 4) si tous les noeuds de la partie gauche d'une df sont marqués alors marquer tous les noeuds de la partie droite
- 5) si on n'a pas trouvé de df pour appliquer le point 4) alors continuer en 6) sinon continuer en 4)
- 6) les noeuds marqués correspondent à X^{++}

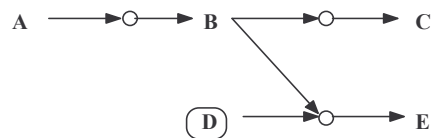
Par analogie, on peut voir les df comme des vannes et les constituants comme des robinets qui doivent être ouverts pour laisser passer le flux. Ce flux se propage de proche en proche. La saturation correspond alors à tous les robinets qui sont ouverts.

Exemple: Soit $(R(A,B,C,D,E); \{A \rightarrow B, B \rightarrow C, BD \rightarrow E\})$

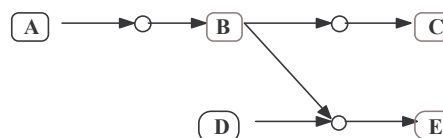
$\{A\}^{++}=\{A,B,C\}$



$\{D\}^{++}=\{D\}$



$\{AD\}^{++}=\{A,B,C,D,E\}$



Proposition 3

Soit $S=(R; F)$ alors $X \rightarrow Y$ est déductible $\Leftrightarrow Y \subseteq X^{++}$

Preuve

$\Rightarrow X \rightarrow Y$ et DF6 (décomposition)
 $\Rightarrow \forall Y_i \in Y, X \rightarrow Y_i$
 \Rightarrow par définition de la saturation $\forall Y_i \in Y, Y_i \in X^{++}$
 $\Leftrightarrow Y \subseteq X^{++}$
 $\Rightarrow \forall Y_i \in Y, Y_i \in X^{++}$ par définition de la saturation $X \rightarrow Y_i$
 $\Rightarrow \forall Y_i \in Y, X \rightarrow Y_i$ et DF5 (union) on a $X \rightarrow Y$

Complétude du système de déduction

Nous sommes maintenant en mesure de démontrer que notre système de déduction est complet

Proposition4

DF1, DF2, DF3 forment un système complet

Preuve:

La complétude signifie $\forall f \in F^{++}$ il existe une déduction de F donnant f.
 idée de la preuve:

par la contraposée: on va montrer que si $X \rightarrow Y$ n'est pas dérivable de F alors, il existe iR telle que toutes les df de F sont vérifiées dans iR (1) mais $X \rightarrow Y$ ne l'est pas ce qui implique que $X \rightarrow Y$ n'est pas une conséquence logique de F (2)

On construit iR avec 2 entités de la façon suivante pour la démonstration:

iR	X^{++}	$R^+ - X^{++}$
	111...111	111...111
	111...111	000...000

montrons (1)

Soit $V \rightarrow W \in F$ supposons $W = \{W_1 .. W_n\}$

Notre instance iR valide $V \rightarrow W$ si et seulement si $V \rightarrow W_i$ est vérifiée pour chaque $i=1..n$

V	X^{++}	$R^+ - X^{++}$	commun à X^{++} et $R^+ - X^{++}$
W_i			
X^{++}	vrai	vrai	vrai
$R^+ - X^{++}$	non mais impossible	vrai	vrai

Le tableau précédent analyse tous les cas possibles de V et W par rapport à X^{++} et $R^+ - X^{++}$. Le seul cas qui est négatif pour iR est impossible car:

$V \subseteq X^{++}$ et la proposition sur la saturation implique $X \rightarrow V$, on a aussi $V \rightarrow W_i$ et DF3 implique $X \rightarrow W_i$, en appliquant à nouveau la proposition sur la saturation on a $W_i \in X^{++}$ (on n'est plus dans la case que l'on analysait). Ceci rend donc impossible le cas négatif.

On vérifie bien $V \rightarrow W$ dans tous les cas.

montrons (2)

$X \rightarrow Y$ n'est pas déductible implique $X \rightarrow Y$ n'est pas une conséquence logique de F
 la proposition sur la saturation donne $\neg(X \rightarrow Y) \Rightarrow \neg(Y \subseteq X^{++})$, il existe r et r' dans iR
 tels que $r.X=r'.X$ et $r.Y \neq r'.Y$. Effectivement $X \subseteq X^{++}$ et $Y \subseteq R^+ \cdot X^{++}$ donc r et r' sont les 2
 entités notre instance iR , nous avons donc que $X \rightarrow Y$ n'est pas une conséquence
 logique de F

Théorème:

Les règles DF1, DF2 et DF3 constituent un système valide et complet

Preuve:

proposition 1 sur la validité et proposition 4 sur la complétude

Equivalence, couverture et irredundance de F

A partir de maintenant, la question de savoir si une df a été obtenue par le concept de dérivation ou par le concept de conséquence logique perd de son importance car ce théorème nous assure de l'équivalence des concepts

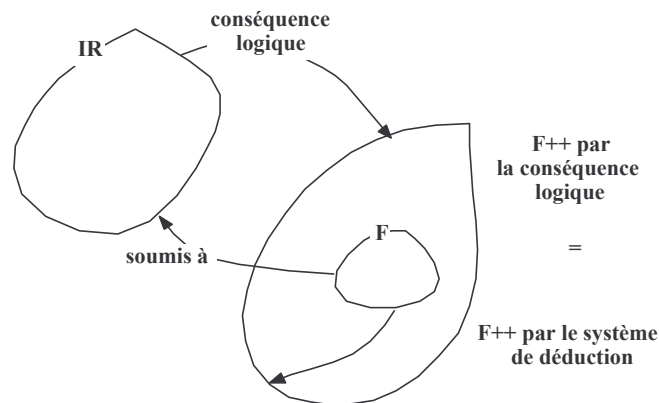


Figure 13-4 : Équivalence entre les deux concepts permettant d'obtenir la fermeture de F

La fermeture de F est un ensemble très "volumineux" qui comporte toutes les df possibles. Certaines de ces df peuvent être éliminées car elles sont redondantes. La redondance dans la partie droite est éliminée par la notion de forme canonique qui assure qu'il n'y a qu'un seul constituant dans la partie droite. La redondance dans la partie gauche est éliminée par la notion de *dépendance élémentaire*.

Définition:

Une dépendance fonctionnelle $X \rightarrow Y$ est élémentaire s'il n'existe pas $X \supset X'$ tel que $X' \rightarrow Y$

Exemple:

Soit $S=(R(A,B,C);\{A \rightarrow B, B \rightarrow C\})$

$F^{++}=\{$
 $A \rightarrow B,$ df élémentaire
 $AB \rightarrow B,$ df non élémentaire
 $ABC \rightarrow B,$ df non élémentaire
 $\dots \}$

Il existe encore une autre façon de créer de la redondance: des df élémentaires peuvent être combinées entre elles avec la règle de déduction transitive et produire de nouvelles dépendances qui sont aussi élémentaires. La notion d'ensemble de df *irredondant* élimine ces dernières

Exemple:

Soit $S=(R(A,B,C);\{A \rightarrow B, B \rightarrow C\})$

$F^{++}=\{$
 $A \rightarrow B,$ df élémentaire
 $B \rightarrow C$ df élémentaire
 $A \rightarrow C,$ df élémentaire (dépendance transitive)
 $\dots \}$

En éliminant des df, il faut faire attention à conserver un équivalent à celui de départ. La notion de *couverture* d'un ensemble de df définit cette équivalence.

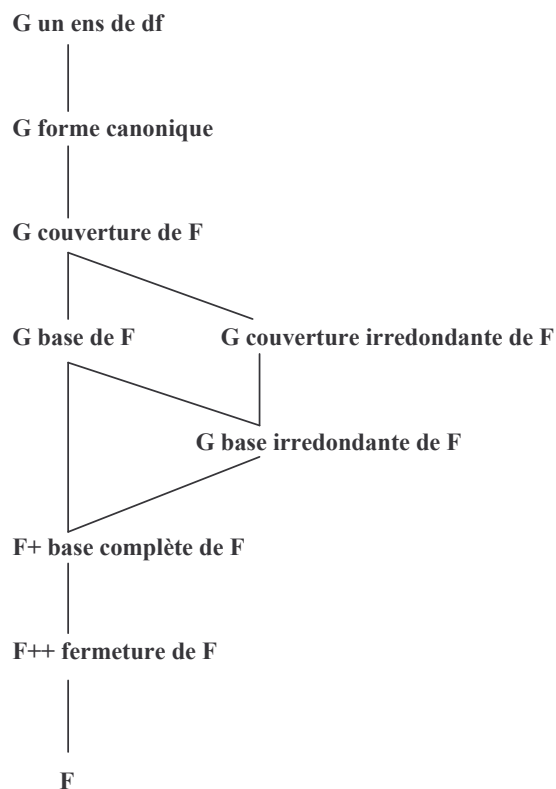


Figure 13-5 : Liens entre les différentes définitions sur les ensembles de df

Définition [LEO88]

G (un ensemble de df) est une couverture de F si $F^{++}=G^{++}$

La base complète de F (notée F^+) est l'ensemble des dépendances élémentaires de F^{++}

G est une base si $G \subseteq F^+$

G (un ensemble de df) est une couverture irredondante de F s'il n'existe pas $f \in G$ tel que $(G-\{f\})^{++}=F^{++}$

G est une base irrédundante de F si G est une base de F et si G est une couverture irrédundante de F.

Dans la Figure 13-5, on voit comment sont reliés les différents concepts portant sur deux ensembles de df. Par rapport à un ensemble F, la recherche et l'analyse de ses bases irrédundantes sont essentielles lors de la conception de la base de données. La base complète de F sera utile lorsque l'on travaille sur un sous-ensemble de constituants du schéma, il sera alors possible de projeter directement les df de la base complète sur ce sous-ensemble et ainsi connaître toutes les df qui concernent ce dernier.

Les questions pragmatiques et pertinentes maintenant sont de savoir comment:

- tester l'équivalence de F et G sans calculer les fermetures
- calculer la base complète de F
- calculer les bases irrédundantes de F

Algorithme lié au df

Tester l'équivalence de F et G sans calculer les fermetures

Idée: pour chaque df de F vérifier si elle peut être déduite de G et pour chaque df de G vérifier si elle peut être déduite de F

Equivalence (F, G)

```

pour chaque g ∈ G faire
    si non (droite(g) ⊆ satX(gauche(g), F))
        alors Equivalence ← faux
        quitter
    finsi
refaire
pour chaque f ∈ F faire
    si non (droite(f) ⊆ satX(gauche(f), G))
        alors Equivalence ← faux
        quitter
    finsi
refaire
Equivalence ← vrai
fin Equivalence

```

satX(X, F) -- saturation de X

```

satX ← X
nouveau ← vrai -- si on a trouvé un cst
tant que nouveau faire
    nouveau ← faux
    pour chaque f ∈ F faire
        si (gauche(f) ⊆ satX) ∧ ¬(droite(f) ⊆ satX)
            alors satX ← satX ∪ droite(f)

```

```

                                nouveau ← vrai
                                fin si
                                refaire
                                refaire
                                X++ ← satX
                                fin X++

```

Calculer la base complète de F

Idée:

pour chaque df de F vérifier si elle est élémentaire (éliminer la redondance dans la partie gauche de la df). Ensuite générer toutes les dépendances obtenues par transitivité.

base_complete(F)

```

                                base_complete ← Transitive_de(Elémentaire_de(F))
                                fin base_complete

```

Elémentaire_de(F)

```

                                Elémentaire ← F
                                pour chaque f ∈ Elémentaire faire
                                    si cardinalité(gauche(f)) > 1
                                        alors pour chaque C ∈ gauche(f) faire
                                            -- tester si l'élimination du cst détermine
                                            -- toujours la même partie droite
                                            f' ← (gauche(f) - C) → droite(f)
                                            si droite(f) ⊆ satX(gauche(g), Elémentaire - f ∪ f')
                                                alors Elémentaire ← Elémentaire - f ∪ f'
                                            fin si
                                        refaire
                                    fin si
                                refaire
                                Elémentaire(F) ← Elémentaire
                                fin Elémentaire_de

```

Transitive_de(F)

```

                                Transitive_de ← ∅
                                nouveau ← vrai -- si on a trouvé un cst
                                tant que nouveau faire
                                    nouveau ← faux
                                    pour chaque f ∈ F faire
                                        alors Transitive_de ← Transitive_de ∪
                                        { (gauche(f)) → satX(gauche(f), F) }
                                        nouveau ← vrai -- si on a trouvé un cst
                                    fin si
                                refaire
                                refaire

```

```

    Transitive_de(F) ← Transitive_de
fin Transitive_de

```

Calculer une base irrédundante de F

idée: pour chaque df de F vérifier si elle est élémentaire (éliminer la redondance dans la partie gauche de la df). Ensuite éliminer les df qui sont redondantes (dans le cas où F est cyclique il existe plusieurs choix possibles pour cette élimination donc plusieurs bases irrédundantes, ici la base irrédundante dépend donc de l'ordre dans lequel on teste les df).

irrédundant_de (F)

```

    irrédundant_de ← Elémentaire_de(F)
    pour chaque f ∈ irrédundant_de faire
        -- tester si l'élimination de f détermine
        -- toujours la même couverture
        si equivalence(irrédundant_de,irrédundant_de-f)
            alors irrédundant_de ← irrédundant_de-f
        fin si
    refaire
    irrédundant_de(F) ← irrédundant_de
fin irrédundant_de

```

Exercices

Questions sur TT³

Les df de TT3 sont les suivantes:

- 1) noChassis -> noPlaque miseEnService modèle noStation
- 2) modèle -> nbPlaces catégorie typeCarburant automatique poids
- 3) noPlaque noJour -> kilometrage litres typeCarburant
- 4) noChassis noJour -> description
- 5) noChauffeur -> nom prénom adresse noStation
- 6) noChassis noJour trancheHoraire -> noChauffeur
- 7) noStation -> noZone
- 8) noChassis -> noZone
- 9) heure zoneDe zoneA -> tempsParcours

a) Reformuler en français ce que signifient ces df

b) Répondre aux questions suivantes en basant vos arguments uniquement sur la modélisation (domaine des constituants, des relations et des dépendances fonctionnelles; voir question 0):

- Un véhicule peut-il avoir plusieurs numéros de plaque ?
 - *Non Car il existe la df noChassis -> noPlaque déductible de la df1 validée par les entités de la relation véhicule*
1. Un même véhicule peut-il être conduit par deux chauffeurs le même jour et la même tranche horaire ?

2. Un chauffeur peut-il conduire deux voitures le même jour et la même tranche horaire ?
3. Un chauffeur peut faire le plein de son véhicule plus d'une fois par jour ?
4. Le type de carburant est-il nécessaire dans le relation carburant ?
5. Un même véhicule peut-il être associé à deux zones différentes à un même instant ?

c) Pour les trois questions suivantes, situez clairement la différence avant et après.

- 1) La dépendance fonctionnelle ci-dessous est non-élémentaire par rapport à celles de l'énoncé. Si on l'acceptait comme élémentaire, qu'impliquerait-elle dans le champ d'application du point de vue organisationnel ?
noPlaque noJour kilometrage -> litres typeCarburant
- 2) Si l'on ajoutait la dépendance fonctionnelle suivante qu'impliquerait-elle dans le champ d'application du point de vue organisationnel ?
 - noChauffeur noJour trancheHoraire -> noChassis
- 3) La dépendance fonctionnelle ci-dessous est contradictoire par rapport à celles de l'énoncé. Si on l'acceptait comme élémentaire qu'impliquerait-elle dans le champ d'application du point de vue organisationnel ?
 - noPlaque -> noJour kilometrage litres typeCarburant

Réponses sur TT³

a) Reformuler en français ce que signifient ces df

- 1) A un noChassis est associé un seul noPlaque, une seule miseEnService, un seul modèle et un seul noStation
- 2) A un modèle est associé un seul nbPlaces, une seule catégorie, un seul typeCarburant, un seul statut automatique et un seul poids
- 3) A un noPlaque et un noJour est associé un seul kilometrage, une seule consommation litres et un seul typeCarburant
- 4) A un noChassis et un noJour est associée une seule description
- 5) A un noChauffeur est associé un seul nom, un seul prénom, une seule adresse et un seul noStation
- 6) A un noChassis , un noJour et une trancheHoraire est associé un seul noChauffeur
- 7) A un noStation est associé un seul noZone
- 8) A un noChassis est associé un seul noZone
- 9) A une heure, une zoneDe et une zoneA est associé un seul tempsParcours

On peut vérifier que ces assertions sont bien exactes dans le champ d'application.

b) Répondre aux questions suivantes en basant vos arguments uniquement sur la modélisation :

- 1) Un même véhicule peut-il être conduit par deux chauffeurs le même jour et la même tranche horaire ?
- *Non car la df 6) indique le contraire*
- 2) Un chauffeur peut-il conduire deux voitures le même jour et la même tranche horaire ?
- *Oui car aucune df tel qui suit ne peut être déduite de l'ensemble actuel*
- *noChauffeur noJour trancheHoraire -> noChassis*
- 3) Un chauffeur peut faire le plein de son véhicule plus d'une fois par jour ?
- *non car la df 3 associe un seul plein à un véhicule (oui si le chauffeur conduit deux véhicules différents)*
- 4) Le type de carburant est-il nécessaire dans le relation carburant ?
- *Oui car il n'existe pas de df noPlaque -> noChassis qui permettrait de déduire noPlaque -> typeCarburant*
- 5) Un même véhicule peut-il être associé à deux zones différentes à un même instant ?
- *Non si l'on s'en tient strictement à la df noChassis -> noZone.*
- *Oui si l'on voit que:*
- *df6 associe un noChassis à un noChauffeur*
- *df5 associe un noChauffeur à un noStation*
- *et df7 noStation à un noZone*
- *donc un véhicule est aussi associé à la zone du chauffeur qui le conduit et donc elle peut être différente de la zone où est stationné le véhicule.*

c) Pour les trois questions suivantes, situez clairement la différence avant et après.

- 1) La dépendance fonctionnelle ci-dessous est non-élémentaire par rapport à celles de l'énoncé. Si on l'acceptait comme élémentaire, qu'impliquerait-elle dans le champ d'application du point de vue organisationnel ?
- *noPlaque noJour kilometrage -> litres typeCarburant*
- *Avant il n'était pas possible de faire plusieurs pleins pour un véhicule le même jour.*
- *Après il sera possible de faire plusieurs pleins pour un véhicule le même jour si les kilométrages sont différents.*
- 2) Si l'on ajoutait la dépendance fonctionnelle suivante qu'impliquerait-elle dans le champ d'application du point de vue organisationnel ?
- *noChauffeur noJour trancheHoraire -> noChassis*
- *Avant un chauffeur pouvait conduire deux véhicules le même jour à la même tranche horaire*

- *Après il ne peut conduire qu'un seul véhicule.*
- 3) La dépendance fonctionnelle ci-dessous est contradictoire par rapport à celles de l'énoncé. Si on l'acceptait comme élémentaire qu'impliquerait-elle dans le champ d'application du point de vue organisationnel ?
 - noPlaque -> noJour kilometrage litres typeCarburant
- *On ne pourrait plus qu'enregistrer un seul et unique plein pour un véhicule.*

14. Clés

"AIL - Mangez en beaucoup. Il rajeunit l'organisme et éloigne les importuns" (Alexandre Vialatte - Almanach des quatre saisons)

Une clé d'une relation est un ensemble de constituants (minimum) qui détermine de manière unique tous les autres constituants de la relation. En exprimant ceci en termes de dépendances fonctionnelles, on peut dire que les constituants d'une relation dépendent fonctionnellement de sa clé (si elle est unique).

Du point de vue de la conception, la connaissance des clés est importante car il suffit de connaître les valeurs des constituants de la clé pour déterminer de façon unique un tuple dans une relation et ainsi déterminer les autres valeurs de ses constituants.

Définition: clé de R

Soit un schéma $S=(R,F)$.

Alors K est une clé de R si et seulement si K satisfait les conditions suivantes:

1) $K \subseteq R^+$

2) $R^+ \subseteq K^{++}$

3) $\neg \exists K'$ strictement contenu dans K tel que $R^+ \subseteq K'^{++}$

Les conditions définissant la clé s'interprètent de la façon suivante:

- 1) une clé est contenue dans les constituants de la relation
- 2) une clé détermine tous les constituants de la relation (les constituants de R sont contenus dans la saturation de la clé)
- 3) une clé est minimale, car tous ses éléments sont essentiels.

Cette définition permet de dépasser la simple intuition qui nous fait dire, par exemple, que {NOM PRENOM} est une clé pour une relation. Nous sommes maintenant en mesure de tester si effectivement un ensemble d'attributs est une clé. Les clés sont entièrement déterminées par les dépendances fonctionnelles validées par le schéma. Si le schéma a aucune dépendance fonctionnelle pour une relation R alors la clé est constituée par R^+ lui-même. On constate donc que les dépendances fonctionnelles doivent être attentivement déterminées pour que les clés soient précises.

Examinons un exemple:

Soit $((A,B,C,D,E); \{A \rightarrow B, B \rightarrow C, BD \rightarrow E\})$

{AD} est clé de R car

- 1) $\{AD\} \subseteq R^+$
 2) $\{AD\}^{++} = \{A, B, C, D, E\}$ ce qui implique $R^+ \subseteq K^{++}$

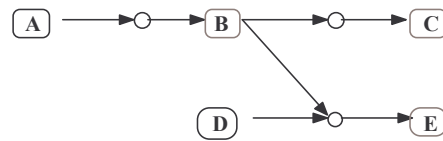


Figure 14-1 : Calcul de la saturation de $\{AD\}$

- 3) $\neg \exists K'$ strictement contenu dans K tel que $R^+ \subseteq K'^{++}$ en effet
 $\{A\}^{++} = \{A, B, C\}$ et $\{D\}^{++} = \{D\}$ ne déterminent par R

Algorithme de recherche de clés

On peut systématiquement tester tous les sous-ensembles possibles de R^+ .
 Nous obtenons alors l'algorithme suivant [LEO88]:

Clés (R : relation, F : ensemble de df)

$C \leftarrow \emptyset$

pour chaque $X \in 2^{R^+}$ faire

 si $X^{++} = R^+$ alors $C \leftarrow C \cup \{X\}$ finsi
 refaire

Clés $\leftarrow \min(C)$

fin Clés

2^{R^+} -- ensemble des parties de R^+

$P \leftarrow \emptyset$

$P \leftarrow P \cup \{\emptyset\}$

pour chaque $A \in R^+$ faire

 pour chaque $E \in P$ faire

$P \leftarrow P \cup \{E \cup \{A\}\}$

 refaire

refaire

$2^{R^+} \leftarrow P$

fin 2^{R^+}

X^{++} -- saturation de X

satX $\leftarrow X$

nouveau \leftarrow vrai -- si on a trouvé un cst
 tant que nouveau faire

 nouveau \leftarrow faux

 pour chaque $f \in F$ faire

 si $(\text{gauche}(f) \subseteq \text{satX}) \wedge \neg(\text{droite}(f) \subseteq \text{satX})$

 alors satX \leftarrow satX \cup droite(f)

 nouveau \leftarrow vrai

 finsi

 refaire

```

    refaire
    X++ ← satX
fin X++
min(XYZ) -- chercher les éléments minimaux
min ← ∅
pour chaque X ∈ XYZ faire
    minOK ← vrai
    pour chaque Y ∈ XYZ-{X} faire
        si Y ⊆ X alors minOK ← faux finsi
    refaire
    si minOK alors min ← min ∪ {X} finsi
refaire
min(XYZ) ← min
fin min

```

Cet algorithme est établi directement à partir de la définition de la clé. On construit de manière explicite tous les sous-ensembles possibles de R^+ . Pour chacun, on teste si la saturation contient R^+ . Et finalement, on ne conserve que les sous-ensembles qui n'en contiennent aucun autre.

Le nombre d'éléments à tester est de $2^{|R^+|}$ où $|R^+|$ est la cardinalité de R . 20 attributs nous amènent un million de clés potentielles à tester.

Attributs puits et source

En utilisant la notion de source et de puits sur un schéma $S=(R;F)$, on diminue le nombre d'éléments de $2^{|R^+|}$ à examiner pour chercher les clés

Définition:

Une *source* C est constituant de R^+ telle que

$$\forall f \in F, C \notin \text{droite}(f)$$

Un *puits* C est un constituant de R^+ tel que

$$\forall f \in F, C \notin \text{gauche}(f)$$

et C n'est pas une source (constituant isolé)

Graphiquement, les sources sont les noeuds où aucune arête aboutit et les puits sont les noeuds où aucune arête part.

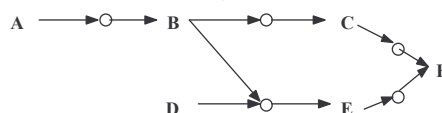


Figure 14-2 : Puits et Source du Graphe de DF

Les sources du schéma précédent sont $\{A,D\}$ et il possède un puits qui est $\{F\}$.

Proposition: sur les clés, puits et sources

Soit K l'ensemble des clés de $S=(R;F)$

1) si C est un puits alors $\forall k \in K \Rightarrow C \notin k$
Un puits n'appartient à aucune clé

2) si C est une source alors $\forall k \in K \Rightarrow C \in k$
Une source appartient à toutes les clés

Preuve:

1) Si C était pas élément d'une clé k , k ne serait pas minimal car $k^{**} = (k - \{C\})^{**}$ étant donné que C est élément d'aucune partie gauche de df , condition pour apporter des éléments nouveaux dans la saturation d'un ensemble

2) Si C n'était élément d'une clé k , k ne serait pas une clé car $C \notin k^{**}$ étant donné que C est élément d'aucune partie droite de df , condition pour que C soit dans la saturation d'un ensemble

Cette proposition nous permet d'effectuer la partition de R^+ en trois parties:

- - Les sources qui appartiennent toujours à une clé.
- - Les puits qui appartiennent à aucune clé.
- - Les autres attributs qui doivent être testés

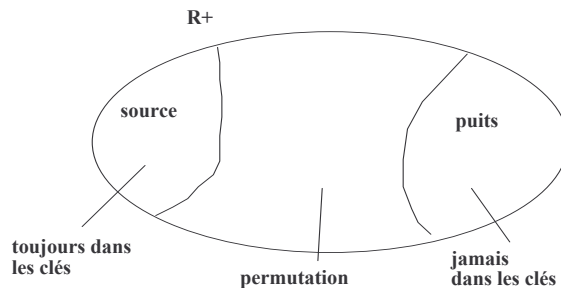


Figure 14-3 : Partition de R^+

Unicité de la clé dans les graphes de df acycliques

Un autre cas intéressant est celui où le graphe de df ne contient pas de cycles car alors la clé est unique et égale aux sources du graphe.

Définition:

Un graphe de df F est acyclique si

$\neg \exists \{f_1, f_2, \dots, f_n\}, n > 1, f_i \in F$ tel que

1) $\forall i \in [1..n-1], droite(f_i) \cap gauche(f_{i+1}) \neq \emptyset$

2) $droite(f_n) \cap gauche(f_1) \neq \emptyset$

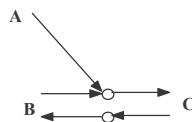


Figure 14-4 : Exemple de graphe cyclique.

Proposition:

Si le graphe de dépendances est acyclique alors il existe une seule clé formée des sources

Preuve:

1) L'ensemble S des sources est une clé, en effet si S n'est pas une clé cela signifie qu'il existe un constituant X élément de la clé et pas élément de S , mais dans ce cas X doit être un noeud appartenant à un cycle du graphe de df ce qui est contraire aux hypothèses du théorème

2) L'unicité de la clé; on ne peut retirer un élément de S car tous sont nécessaires dans une clé (voir proposition sur les clés, puits et sources). On ne peut pas ajouter un élément à la clé à cause de la minimalité d'une clé.

L'exemple de la Figure 14-1 est un graphe acyclique dont la clé $\{AD\}$ constitue aussi les sources du graphe.

L'inverse n'est pas vrai, un graphe cyclique peut avoir une clé unique. Par exemple:

Exemple:

$S = (\{A, B, C, D\}; \{A \rightarrow B, B \rightarrow C, C \rightarrow B, C \rightarrow D\})$

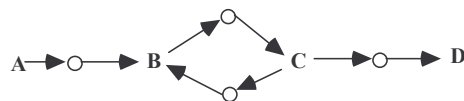


Figure 14-5 : Exemple de graphe cyclique à clé unique

Le graphe de F est cyclique (BC).

Les sources de F sont $\{A\}$

Les puits de F sont $\{D\}$

La clé de R est $\{A\}$

Cet exemple montre que l'on peut éliminer des clés possibles les attributs appartenant à la saturation des sources.

Proposition:

Soit $S = (R; F)$ et $SOURCES$ l'ensemble des sources du schéma.

alors aucun élément de $SOURCES^{**} - SOURCES$ appartient à une clé de R .

Preuve:

L'ensemble S des sources fait partie de toutes les clés, ajouter un élément déjà déterminé par la saturation des sources n'amène pas d'attribut nouveau et de plus rend non minimal la clé.

Heuristiques pour la recherche des clés

Des propositions précédentes, on peut déduire les heuristiques suivantes:

Heuristique 1: déduite de la proposition sur les puits et sources:

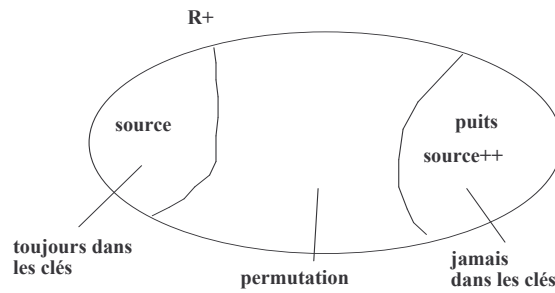
On peut limiter la recherche des clés aux ensembles qui contiennent que toutes les sources et qui ne contiennent aucun puits.

Heuristique 2: déduite de la proposition sur les graphes de df acycliques:

Vérifier si le graphe de df est acyclique; si oui, former la clé avec les sources.

Heuristique 3 : déduite de la proposition sur la saturation des sources:

On peut éliminer des clés les attributs qui appartiennent à la saturation des sources.

Figure 14-6 : Nouvelle partition de R^+

L'application de ces heuristiques permet de rechercher "manuellement" les clés d'un schéma sur le graphe de dépendances fonctionnelles.

Autres exemples:

Exemple 1)

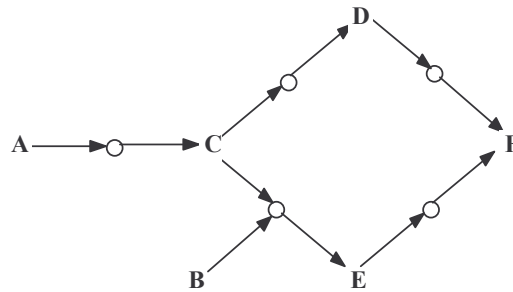


Figure 14-7 : Graphe de l'exemple 1

$S = (\{A, B, C, D, E, F\}; \{A \rightarrow C, C \rightarrow D, D \rightarrow F, CB \rightarrow E, E \rightarrow F\})$

Le graphe de F est acyclique.

Les sources de F sont $\{A, B\}$

La clé unique de R est $\{A, B\}$

Exemple 2)

$S = (\{A, B, C, D, E\}; \{A \rightarrow B, B \rightarrow C, C \rightarrow D, D \rightarrow A\})$

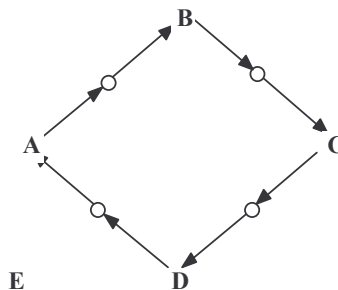


Figure 14-8 : Graphe de l'exemple 2

Le graphe de F est cyclique.

Les sources de F sont $\{E\}$

Les clés de R sont $\{EA\}, \{EB\}, \{EC\}, \{ED\}$

Décomposition d'une relation

Nous avons examiné le cas de la clé d'un schéma - une seule relation avec un ensemble de df- alors que nous devons envisager un ensemble de relations décomposant notre schéma.

Définition:

La décomposition d'une relation R où $R^+ = \{C_1, C_2, \dots, C_n\}$ est un ensemble d_R de relations $d_R = \{R_1, R_2, \dots, R_n\}$ tel que

$$R^+ = R_1^+ \cup R_2^+ \cup \dots \cup R_n^+$$

Les R_i^+ ne sont pas forcément disjoints. Nous examinerons dans le prochain chapitre les propriétés associées à la décomposition. Notre problème actuel est de trouver les clés de chaque élément de la décomposition. Il nous faut adapter les df de F à chaque relation R_i .

Définition:

La projection de F sur un ensemble Z de constituants

$$F[Z] = \{X \rightarrow Y \in F^{++} \mid XY \subseteq Z\}$$

La clé d'une relation de la décomposition est celle du schéma formé par $(R_i^+; F[R_i^+])$.

Algorithme pour calculer une base de $F[Z]$ avec la base complète de F :

```

FZ(F, Z)  -- F[Z]
  FZ ← ∅
  pour chaque f ∈ F+ faire
    si (gauche(f) ∪ droite(f) ⊆ Z)
      alors FZ ← FZ ∪ f
  fin si
  refaire
  FZ(F, Z) ← FZ
fin FZ(F, Z)

```

Exemple:

soit le schéma $S = (\{A, B, C\}; \{A \rightarrow B, B \rightarrow C\})$ et la décomposition $d_R = \{AB, AC\}$

La base complète est $\{A \rightarrow B, B \rightarrow C, A \rightarrow C\}$

donc nous avons les projections suivantes

$F[AB] = \{A \rightarrow B\}$ et $F[AC] = \{A \rightarrow C\}$

la clé de AB est donc A et celle de AC est aussi A.

Exercices

Questions sur les clés

Voici 10 relations et leurs dépendances fonctionnelles. Construire le réseau de noeuds et d'étoiles de chacune d'elles et déterminer leurs clés.

1. R1 (ABCDE)
A -> B
B -> C
C -> D
D -> E
2. R2 (ABCDEF)
A -> B
B -> C
C -> D
D -> E
3. R3 (ABCD)
A -> C
B -> C
C -> D
4. R4 (ABCD)
A B -> C
C -> D
5. R5 (ABCD)
A -> C
B -> C
C -> D
D -> B
6. R6 (ABCDE)
AB -> C
C -> D
D -> B
7. R7 (ABCD)
A -> B
B -> C
C -> A
8. R8 (ABCD)
A -> C
DC -> A
B -> D
AC -> B
9. R9 (ABCD)
ABCD -> A
10. R10 (ABCD)
aucune dépendance fonctionnelle

Questions sur TT³

Les df de TT³ sont les suivantes:

- 1) noChassis -> noPlaque miseEnService modèle noStation
- 2) modèle -> nbPlaces catégorie typeCarburant automatique poids
- 3) noPlaque noJour -> kilometrage litres typeCarburant
- 4) noChassis noJour -> description
- 5) noChauffeur -> nom prénom adresse noStation
- 6) noChassis noJour trancheHoraire -> noChauffeur

7) noStation -> noZone

8) noChassis -> noZone

9) heure zoneDe zoneA -> tempsParcours

Chercher pour les relations de la modélisation les clés

Véhicule(noChassis noPlaque miseEnService modèle noStation)

Type(modèle nbPlaces catégorie typeCarburant automatique poids)

Carburant(noPlaque noJour kilometrage litres typeCarburant)

Entretien(noChassis noJour description)

Chauffeur(noChauffeur nom prénom adresse noStation)

Permis(noChauffeur catégorie)

Planning(noChauffeur noChassis noJour trancheHoraire)

Station(noZone noStation)

Distance(heure zoneDe zoneA tempsParcours)

Situation(noChassis noZone)

Réponses sur les clés

- 1. clé A
- 2. clé AF
- 3. clé AB
- 4. clé AB
- 5. clé A
- 6. clés ACE, ADE, ABE
- 7 clés AD, BD, CD
- 8. clés A, BC, CD
- 9. clé ABCD
- 10. clé ABCD

Réponses sur TT³

Les clés sont données par les constituants soulignés. La projection de F sur les relations de la décomposition donnent une df par relation (sauf pour Permis); la clé est donc la partie gauche de cette df. Pour la relation Permis nous n'avons pas de df donc le tout est la clé.

Véhicule(noChassis noPlaque miseEnService modèle noStation)

Type(modèle nbPlaces catégorie typeCarburant automatique poids)

Carburant(noPlaque noJour kilometrage litres typeCarburant)

Entretien(noChassis noJour description)

Chauffeur(noChauffeur nom prénom adresse noStation)

Permis(noChauffeur catégorie)

Planning(noChauffeur noChassis noJour trancheHoraire)

Station(noZone noStation)

Distance(heure zoneDe zoneA tempsParcours)

Situation(noChassis noZone)

15. Décomposition d'une relation

“Le logogramme dessine le mot ou la chose. Le système logo-syllabique devient syllabique et découpe le mot, parlé maintenant; il devient bientôt consonantique, puis un vrai alphabet, où les syllabes se répartissent en lettres. Dès lors, le dessin, sur la plage, la tablette ou le parchemin, analyse tout autre chose que l'objet qu'il est censé désigner. Il est le signe de signe de signe.” Michel Serres - Les origines de la géométrie.

Rappelons qu'une décomposition d'une relation R est un ensemble d_R de relations $d_R = \{R_1, R_2, \dots, R_n\}$ tel que

$$R^+ = R_1^+ \cup R_2^+ \cup \dots \cup R_n^+$$

Nous voyons qu'il existe donc beaucoup de possibilités pour effectuer celle-ci. Pour $R = \{ABC\}$, nous avons, en excluant la possibilité qu'une relation soit sous-ensemble d'une autre, les décompositions suivantes:

{A, B, C}
 {AB, C}
 {A, BC}
 {AC, B}
 {AB, BC}
 {AC, BC}
 {AB, AC}
 {ABC}

Toutes ne sont pas "équivalentes". Certaines décompositions de R jouissent de "bonnes" propriétés et ceci par rapport aux dépendances fonctionnelles qu'elles doivent valider. Ces propriétés sont exprimées par rapport:

- **l'interrogation:** la conservation de l'information par la décomposition.
- **la modification:** l'élimination des anomalies de mise à jour
- **les règles d'intégrité:** la préservation des dépendances fonctionnelles

En prenant le schéma $S = (ABC, \{A \rightarrow B, B \rightarrow C\})$ on peut dresser le tableau suivant par rapport aux différentes décompositions (une croix indique si elle possède la propriété).

Décomposition	information ok	anomalie ok	ri ok
{A, B, C}		X	
{AB, C}		X	
{A, BC}		X	
{AC, B}		X	

{AB, BC}	X	X	X
{AC, BC}		X	
{ABC}	X		X

On constate qu'une seule satisfait toutes les propriétés et que la décomposition triviale (égale à elle-même) ne pose que des problèmes en terme de mise à jour. Sinon les autres ne conservent pas l'information ni ne préservent les dépendances fonctionnelles.

Dans la suite de ce chapitre nous analysons la décomposition et nous donnons quelques indications sur les processus permettant de concevoir des décompositions ayant de bonnes propriétés.

Conservation de l'information

L'interprétation du champ d'application se faisait par rapport à une relation R et donnait l'instance iR . Si l'on projette iR sur les relations de la décomposition, peut-on encore retrouver iR ?

Nous ne nous intéressons donc qu'aux décompositions *joiniables*, celles où il existe un ordre de composition (celui énoncé dans la donnée de la décomposition) telle que la jointure naturelle donne les constituants de R . Soit:

$$(R_1 * R_2 * \dots * R_n)^+ = R^+$$

Si la décomposition n'est pas joiniable, cela peut-être le signe que deux domaines d'application distincts sont modélisés simultanément (la philatélie et le vignoble du bordelais n'ont pas de raison d'avoir une relation liant des constituants appartenant aux deux domaines!). Dorénavant, nous sous-entendons toujours que la décomposition est joiniable.

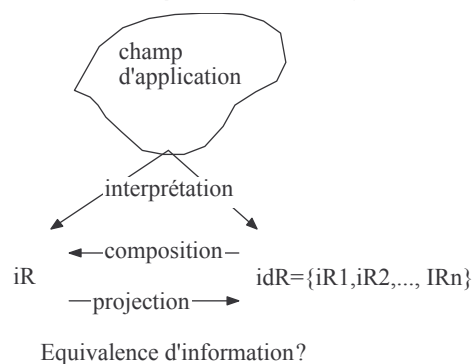


Figure 15-1 : préservation de l'information

Exemple: Hôtel

Hôtel(NumChambre, NumClient, Nom, Prenom, Adresse, Prix, NbrLit, NbrPers, DateArr, DateDep, Confort, Equipement)

$d_{\text{Hôtel}} = \{$

- Chambres(NumChambre, Prix, NbrLit, NbrPers, Confort, Equipement),

- Clients(NumClient, Nom, Prenom, Adresse),
- Réservation(NumChambre, NumClient, DateArr, DateDep)}

A-t-on conservé l'information contenue dans "hôtel" ?

Les instances de la décomposition sont toujours égales à leur projection sur la relation décomposée

$$i_{\text{Hotel}}[\text{Clients}^+] = i_{\text{Clients}}$$

$$i_{\text{Hotel}}[\text{Chambres}^+] = i_{\text{Chambres}}$$

$$i_{\text{Hotel}}[\text{Réservations}^+] = i_{\text{Réservations}}$$

Pour la composition cela n'est pas toujours vrai.

$$i_{\text{Client}} * i_{\text{Réservation}} * i_{\text{Chambre}} = i_{\text{Hôtel}} (?)$$

Examinons une décomposition ne préservant pas l'information

Exemple canonique:

Soit $R(A,B,C)$, $F=\{\}$, $d_R=\{R_1(AB), R_2(AC)\}$

et les instances suivantes

i_R

a1	b1	c1
a1	b2	c1
a1	b1	c2

i_{R_1}

a1	b1
a1	b2

i_{R_2}

a1	c1
a1	c2

$R_1 * R_2 \neq R$ car la composition génère le tuple (a1 b2 c2)

Par contre, on a $R \subseteq R_1 * R_2$ (la décomposition "invente" des tuples)

Propriété de la décomposition

notation: $\prod_{j=1..n} iR[R_j^+] = iR[R_1^+] * iR[R_2^+] * \dots * iR[R_n^+]$

Soit $d_R=\{R_1, R_2, \dots, R_n\}$ une décomposition de R alors:

a) $iR \subseteq \prod_{j=1..n} iR[R_j^+]$

b) $(\prod_{j=1..n} iR[R_j^+])[R_i^+] = iR[R_i^+]$ pour $i=1, \dots, n$

c) $\prod_{l=1..n} (\prod_{j=1..n} iR[R_j^+])[R_l^+] = \prod_{k=1..n} iR[R_k^+]$

Le processus "projection-composition" peut créer de nouvelles informations. Par contre la répétition de ce processus ne crée plus rien de nouveau.

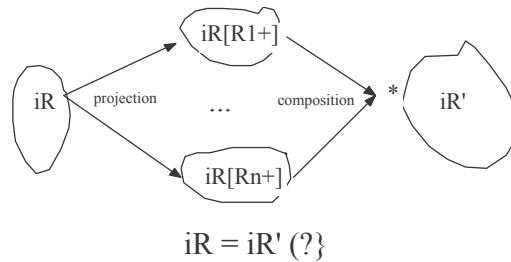


Figure 15-2 : équivalence de iR et iR'

Décomposition totale

Définition

Si un schéma $(R;F)$ est décomposé dans les relations $d_R = \{R_1, R_2, \dots, R_n\}$. On dit que la décomposition est *totale* si pour chaque instance iR validant F , on a

$$iR = \prod_{j=1..n} iR[R_j^+]$$

Si la décomposition est totale, on peut alors ne plus faire de différence entre la relation et sa décomposition. En utilisant une base de données où les informations sont stockées dans des tables correspondant à une décomposition totale d_R , on ne perd aucune information par rapport à une seule table correspondant à R et contrainte par F .

Comment tester une décomposition sans passer par les instances de la relation ?

Tester si une décomposition est totale

Soit les paramètres

$$R^+ = \{C_1, C_2, \dots, C_n\}$$

F un ensemble de df

$$d_R = \{R_1, R_2, \dots, R_k\}$$

Total(R^+, F, d_R) -- Algorithme de test [ULL82]

-- initialiser un tableau t de k lignes et n colonnes

si $C_j \in R_i^+$ alors $t_{ij} \leftarrow C_j$

 sinon $t_{ij} \leftarrow b_{ij}$

changement \leftarrow vrai

tant que changement faire

 changement \leftarrow faux

 pour chaque $f \in F$ faire

 chercher une ligne i et une ligne j

 tel que $\text{ligne}_i[\text{gauche}(f)] = \text{ligne}_j[\text{gauche}(f)]$

 et $\text{ligne}_i[\text{droite}(f)] \neq \text{ligne}_j[\text{droite}(f)]$

 si trouver i et j alors

```

pour chaque  $C_1 \in \text{droite}(f)$  faire
  -- l un indice de colonne
  si  $t_{i1}=C_1$  alors  $t_{j1} \leftarrow C_1$ 
    sinon  $t_{i1} \leftarrow t_{j1}$ 
  changement  $\leftarrow$  vrai
  refaire
refaire
refaire
si une ligne est remplie de  $C_i$  alors total  $\leftarrow$  vrai
  sinon total  $\leftarrow$  faux

fin total

```

Exemple: Hôtel

Nous avons pris les abréviations suivantes:

NCL NumClient

N Nom

PPrenom

A Adresse

NCH NumChambre

PX Prix

C Confort

EÉquipement

LNbrLit

DA DateArr

DD DateDep

R=Hôtel(NCL,N,P,A,NCH,PX,C,E,L,DA,DD)

dR={ R₁=Clients(NCL,N,P,A)

R₂=Chambres(NCH,PX,C,E,L)

R₃=Réservation(NCL,NCH,DA,DD)}

F={ f₁=NCL → N,P,A

f₂=NCH → PX,C,E,L

F₃=NCL NCH DA → DD}

En suivant l'algorithme on obtient le tableau initial suivant:

	1	2	3	4	5	6	7	8	9	α	β
	NC	N	P	A	NC	PX	C	E	L	DA	DD
	L				H						
R ₁	NC L	N	P	A	b ₁₅	b ₁₆	b ₁₇	b ₁₈	b ₁₉	b _{1α}	b _{1β}
R ₂	b ₂₁	b ₂₂	b ₂₃	b ₂₄	NC H	PX	C	E	L	b _{2α}	b _{2β}
R ₃	NC L	b ₃₂	b ₃₃	b ₃₄	NC H	b ₃₆	b ₃₇	b ₃₈	b ₃₉	DA	DD

On cherche à appliquer une df dont la partie gauche existe dans deux lignes et dont la partie droite n'existe pas dans ces deux lignes

avec f_1 on a gauche(f_1)=NCL et droite(f_1)={N,P,A}

les lignes 1 et 3 sont égales pour NCL

la ligne 3 est modifiée selon l'algorithme de la façon suivante pour N,P,A:

	1	2	3	4	5	6	7	8	9	α	β
	NC L	N	P	A	NC H	PX	C	E	L	ΔA	DD
R ₁	NC L	N	P	A	b ₁₅	b ₁₆	b ₁₇	b ₁₈	b ₁₉	b _{1α}	b _{1β}
R ₂	b ₂₁	b ₂₂	b ₂₃	b ₂₄	NC H	PX	C	E	L	b _{2α}	b _{2β}
R ₃	NC L	N	P	A	NC H	b ₃₆	b ₃₇	b ₃₈	b ₃₉	DA	DD

avec f_2 on a gauche(f_2)=NCH et droite(f_2)={PX,C,E,L}

les lignes 2 et 3 sont égales pour NCH

la ligne 3 est modifiée selon l'algorithme de la façon suivante pour PX,C,E,L:

	1	2	3	4	5	6	7	8	9	α	β
	NC L	N	P	A	NC H	PX	C	E	L	DA	DD
R ₁	NC L	N	P	A	b ₁₅	b ₁₆	b ₁₇	b ₁₈	b ₁₉	b _{1α}	b _{1β}
R ₂	b ₂₁	b ₂₂	b ₂₃	b ₂₄	NC H	PX	C	E	L	b _{2α}	b _{2β}
R ₃	NC L	N	P	A	NC H	PX	C	E	L	DA	DD

La ligne 3 est complète donc d_R est totale

Cet algorithme nous donne en plus l'ordre de jointure des relations. Nous avons utilisé les lignes 1 et 3 et ensuite la ligne 2. Nous avons donc $(R_1 * R_3) * R_2$.

En utilisant une base de données où les informations sont stockées dans des tables correspondant à une décomposition totale d_R , on ne perd aucune information par rapport à une seule table correspondant à R et contrainte par F.

En examinant l'algorithme de plus près, on constate que deux relations sont composables si elles possèdent en commun la partie gauche d'une df et que la projection de cette df est contenue dans l'une des deux. Cette propriété est connue sous le nom du théorème de décomposition.

Théorème: décomposition binaire

Soit $d_R = \{R_1, R_2\}$ une décomposition de $(R; F)$.

R_1, R_2 est une décomposition totale si et seulement si

$$(R_1^+ \cap R_2^+) \rightarrow (R_1^+ - R_2^+) \in F^{++}$$

ou $(R_1^+ \cap R_2^+) \rightarrow (R_2^+ - R_1^+) \in F^{++}$

preuve:

"astuce" de la preuve: utiliser l'algorithme pour tester si une décomposition est totale avec la table suivante:

	$R_1^+ \cap R_2^+$	$R_1^+ - R_2^+$	$R_2^+ - R_1^+$
R_1	C ... C	C ... C	b ... b
R_2	C ... C	b ... b	C ... C

On voit que la seule façon de compléter la ligne 1 ou 2 correspond bien à l'existence d'une des deux df énoncées dans le théorème.

Ce théorème nous permet de concevoir un algorithme de décomposition binaire qui nous permet de chercher les décompositions totales d'une relation. Pour cela nous utilisons les df du schéma pour couper les relations en deux.

Décomposition_de (R, F)

```

DR ← R
nouveau ← vrai -- si on a trouvé une décomposition
tant que nouveau faire
  nouveau ← faux
  pour chaque r ∈ DR faire
    pour chaque f ∈ F faire
      -- df strictement contenue dans r
      si (r+ ⊃ gauche(f) ∪ droite(f))
        alors -- on substitue à r les deux relations
          -- formées à partir de f
            DR ← DR - r
            DR ← DR ∪ ( gauche(f) ∪ droite(f) )
            DR ← DR ∪ ( r+ - droite(f) )
            nouveau ← vrai
    finsi
  refaire
refaire
refaire
Décomposition_de(R, F) ← DR
fin Décomposition_de

```

En utilisant l'algorithme sur HOTEL, on retrouve la décomposition que l'on a utilisée.

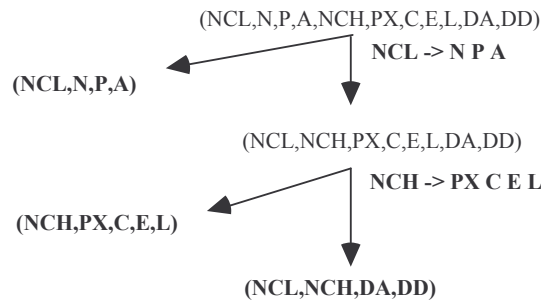


Figure 15-3 : décomposition de l'exemple HOTEL

Pour appliquer cet algorithme, on part avec une base irréductible de F. Dans le cas où il existe des cycles, il existera plusieurs décompositions possibles car il existe plusieurs bases possibles. Le choix de l'une ou l'autre dépend du champ d'application et du contexte d'utilisation.

Préservation des df

La question est de savoir si les dépendances fonctionnelles F qui étaient impliquées par les instances de iR sont toujours impliquées par les instances iRj (et leur composition) de la décomposition dR.

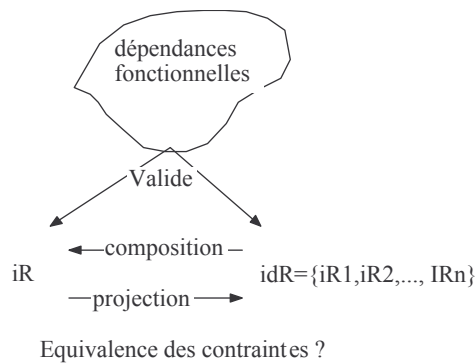


Figure 15-4 : Les df sont-elles encore la conséquence logique pour la décomposition ?

Définitions:

La projection de F sur un ensemble Z de constituants

$$F[Z]=\{X \rightarrow Y \in F^{++} \mid XY \subseteq Z\}$$

On dira que la *décomposition dR préserve F* si l'union des F[Ri] est une couverture de F soit:

$$(\bigcup F[R_i])^{++} = F^{++}$$

On veut pouvoir vérifier que F est préservée dans les iRj sans devoir recourir à leur composition. Les df doivent donc être localement validées par les instances des relations de la décomposition

Exemple:

Ville = V, Rue = R, Numéro postal = N
 R=(RVN)
 F={VR → N, N → V}

Cette décomposition est totale

car $(\{VN\} \cap \{RN\}) \rightarrow (\{VN\} - \{RN\})$

$N \rightarrow V$ est une df de F

Soit les instances suivantes:

V	N
Genève	1207
Genève	1205

R	N
12 rue du Lac	1207
12 rue du Lac	1205

Calculons les df projetées:

$F[VN] = \{N \rightarrow V\}^+$

$F[RN] = \{\}^+$

$i(VN)$ et $i(RN)$ satisfont les df des F projetées. Mais $i(VN)*i(RN)$ ne satisfait plus F malgré la décomposition totale car $RV \rightarrow N$ n'est pas vérifiée

$i(VN)*i(RN)$

V	R	N
Genève	12 rue du Lac	1207
Genève	12 rue du Lac	1205

La propriété de décomposition totale et de préservation des df sont des propriétés indépendantes.

Tester la préservation des df

Soit $(R;F)$ et $d_R = \{R_1, R_2, \dots, R_k\}$, en théorie, il faut:

- 1) calculer F^{++}
- 2) projeter F^{++} sur chaque R_i
- 3) union des $F[R_i] = G$
- 4) calculer la fermeture de cette union G^{++}
- 5) tester l'égalité de F^{++} et G^{++}

Rappelons que calculer F^{++} est de complexité exponentielle, par exemple:

pour $F = \{X_0 \rightarrow X_1, X_0 \rightarrow X_2, \dots, X_0 \rightarrow X_n\}$

on a $X_0 \rightarrow 2^{\{X_0, X_1, X_2, \dots, X_n\}} \in F^{++}$

L'idée de l'algorithme qui suit réside dans le fait que pour $X \rightarrow Y \in F$ si G était connu alors il suffirait de tester si $Y \subseteq X^{++}$ pour G .

Il reste à trouver comment calculer X^{++} sans calculer G . Ceci est possible en restreignant la saturation de X à la projection de chaque R_i lors du calcul (on élimine tout ce qui dépasse!)

Algorithme: test de la préservation des df

Paramètres de Préservation (d_R, F) :

$R^+ = \{C_1, C_2, \dots, C_n\}$
 F un ensemble de df
 $d_R = \{R_1, R_2, \dots, R_k\}$

Préservation (d_R, F)

```

pour chaque  $f \in F$  faire
   $Z \leftarrow$  gauche( $f$ )
  tant que changement de  $Z$  faire
    pour  $i=1$  à  $k$  faire
       $Z \leftarrow Z \cup (\text{saturation}(Z \cap R_i^+) \cap R_i^+)$ 
    refaire
  refaire
  si droite( $f$ ) pas contenu dans  $Z$ 
    alors Préservation  $(d_R, F) \leftarrow$  non
  fin si
refaire
Préservation  $(d_R, F) \leftarrow$  oui
fin préservation

```

exemple: test préservation

$R = (RVN)$, $d_R = \{R_1 = (VN), R_2 = (RN)\}$

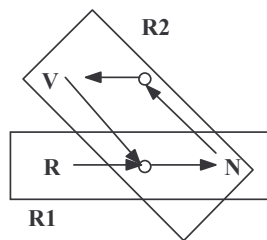
$F = \{VR \rightarrow N, N \rightarrow V\}$

les saturations utilisées dans l'algorithme sont:

$VR^{++} = \{VRN\}$

$N^{++} = \{NV\}$

$R^{++} = \{R\}$



initialisation

$Z \leftarrow$ gauche($VR \rightarrow N$) = VR

itération avec R_1

$((VR \cap RN)^{++} \cap RN) \cup VR$
 $= (R^{++} \cap RN) \cup VR$
 $= R \cup VR$
 $= VR$

itération avec R_2

$$\begin{aligned}
& ((VR \cap VN)^{++} \cap VN) \cup VR \\
&= (V^{++} \cap VN) \cup VR \\
&= V \cup VR \\
&= VR
\end{aligned}$$

Z n'a pas été modifié

On a pas droite(f)=N \subseteq VR, donc les df ne sont pas préservées

Inclusion des df dans les Ri

Dans l'exemple Hôtel, la saturation des df (partie gauche) peut être entièrement calculée à l'intérieur d'un élément de la décomposition. La partie gauche sera donc contenue dans la saturation.

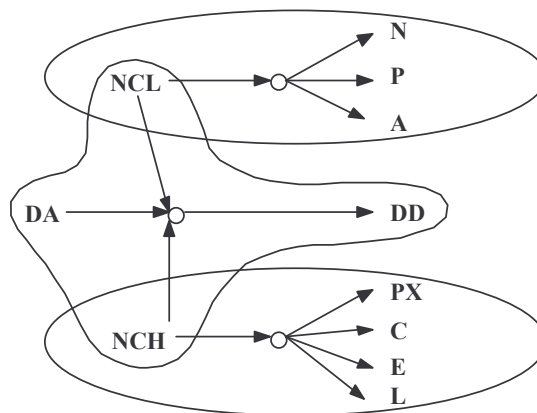


Figure 15-5 : Les df sont toutes contenues dans les relations de la décomposition

On peut donc énoncer la proposition

Proposition:

Soit une décomposition d_R et les df F .

Si $\forall f \in F, \exists R_i \in d_R$ telle que gauche(f) \cup droite(f) \subseteq R_i

alors d_R préserve F

Elimination des anomalies de mise à jour

Une anomalie de mise à jour est mise en évidence dans un schéma lorsque l'exécution d'une primitive de modification sur une relation pour une entité n'a pas l'effet désiré. Une première modélisation de Hôtel (Chapitre 3) était:

```
Hôtel(NumChambre, NumClient, Nom, Prenom, Adresse,
      Prix, NbrLit, NbrPers, DateArr, DateDep, Confort,
      Equipement)
```

Cette modélisation accumule toutes les anomalies de mise à jour:

- lors de la création d'une entité, il est possible d'introduire une entité qui respecte la dépendance induite par la clé de la relation, mais qui n'en valide aucune autre.

- lors de la suppression d'une entité, il est possible que pour supprimer un client, il soit nécessaire de détruire d'autres entités faisant référence à celui-ci. De plus, la suppression d'un client peut aussi entraîner la suppression de la définition d'une chambre.
- lors de la mise à jour d'une entité, il est possible que pour mettre à jour une information, il soit nécessaire de mettre à jour d'autres entités faisant référence à celle-ci .

Dans la deuxième modélisation proposée: Aucun de ces effets n'apparaît:

Chambres(NumChambre, Prix, NbrLit, NbrPers, Confort, Equipement)

Clients(NumClient, Nom, Prenom, Adresse)

- lors de la création d'une entité, si elle respecte les df induit par la clé de la relation alors elle valide toutes les autres df du schéma.
- lors de la suppression d'une entité, l'objet est supprimé sans effet de bord.
- lors de la mise à jour, une seule entité est affectée.

Les anomalies de mise à jour sont synonymes de redondance de l'information. Dans les "bonnes" décompositions, une entité ne doit pas contenir plusieurs objets ayant des existences indépendantes. L'absence de dépendance transitive et de dépendance partielle traduit cette bonne propriété.

Les Formes Normales remédient (en partie) à ces anomalies

1er forme normale 1NF

L'objectif de la 1NF est de simplifier la structure d'une relation en éliminant les structures internes

Définition : 1NF

Une relation est en 1NF si tous les constituants sont atomiques.

La 1NF évite donc la confusion entre le constituant et la relation. Un constituant atomique ne peut pas avoir une structure interne (vecteur, enregistrement, champ répétitif, pointeur)

Dans un langage de programmation structuré, nous avons pour déclarer un fichier d'employés le code suivant:

```

date = record of
    jour : 1..31;
    mois : 1..12;
    année: 1900..2100;
end;

enfant = record of
    prénom: char(20);
    datenaiss : date;
end;

```

```

pntpersonne    = ^personne;
personne      = record of
    nom: char(20);
    sexe : (F,M);
    pere: pntpersonne ;
    mere: pntpersonne ;
end;

employé      = record of
    nom,prenom: char(20);
    nbenfant: 1..12;
    descriptenfant: array[1..nbenfant]
        of enfant;
    salaire: array [1..12] of real;
end;
var fp:file of employé;

```

Dans la forme 1FN, pour un SGBD, toutes les structures doivent être décomposées, nous avons les trois relations suivantes qui sont équivalentes au fichier fp déclaré ci-dessus.

```

Create table personne(idpers number,
    nom char(20),
    prénom char(20) ,
    nbenfant number )

Create table salempl(idpers number,
    période number(2),
    salaire number)

Create table enfant(idpers number,
    prénom char(20),
    jour number(2),
    mois number(2),
    année number(4))

```

Un autre exemple de déclaration en pseudo-pascal donne:

```

statistiqueTrimeste    = array[1900..1999,1..4] of
real;

```

Ici, nous avons la relation suivante qui à chaque trimestre associe une entité.

```

stattrim(année,numtrimeste,valeur)

```

Dans ce cas, l'année est restituée avec une triple jointure qui peut être relativement coûteuse . Si l'information trimestrielle n'est pas exploitée de manière indépendante, on peut choisir la relation suivante:

```

stattribis(année,trim1,trim2,trim3,trim4)

```

Elle est "non normalisé" , mais elle peut être plus efficace pour certains traitements.

2eme forme normale 2NF

La 2NF élimine les objets décrits d'une manière dépendante par rapport à d'autres. Elle élimine ainsi les anomalies suivantes:

- Lors de la création: on ne peut pas créer l'objet sans l'attacher à un autre
- Lors de la suppression: elle entraîne celle de celui qui dépend de lui
- Lors de la maj : la modification doit être reproduite s'il existe plusieurs occurrences de l'objet

Définition: *dépendance partielle*

Soit $R, X \rightarrow A$ et K une clé de R , on dit que $X \rightarrow A$ est une dépendance partielle de R si X est contenu dans K et A n'est pas contenu dans X

Dans la Figure 15-6, on peut voir que le groupe de constituants X et A définit un objet indépendant. En décomposant R en R_1 et R_2 , on élimine la dépendance transitive où:

$$R_1 = R^+ - A$$

$$R_2 = X \cup A$$

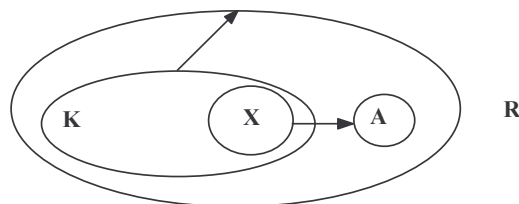


Figure 15-6 : schéma de la dépendance partielle

exemple de relation contenant une dépendance partielle:

Réservation(Num_chambre, num_client, date_arrivée,
date_départ, prix)

$NCL, NCH, DA \rightarrow DD$

$NCH \rightarrow PX$

NCH est contenu dans la clé de Réservation qui est
 NCL, NCH, DA

Définition : 2NF

Une relation en 1NF est en 2NF s'il n'existe pas de dépendance partielle

Exemple: soit les deux relations suivantes et leurs df

Commande(NoCom, NoClient, Nom, Adresse, PrixTotal)

Ligne(NoCom, NoArt, Prix, Qte, Art_total)

$NoCom \rightarrow PrixTotal, NoClient$

$NoClient \rightarrow Nom, Adresse$

$NoArt \rightarrow Prix$

$NoCom, NoArt \rightarrow Qte$

Nous avons $gauche(NoArt \rightarrow Prix) \subseteq clé(Ligne)$ donc Ligne n'est pas en 2NF

Pour transformer le schéma en 2NF il convient de le réécrire comme suit:

Commande (NoCom, NoClient, Nom, Adresse, PrixTotal)

LigneCommande (NoCom, NoArt, Qte, Art_total)

Article (NoArt, Prix)

3eme forme normale 3NF

La 3NF élimine aussi les objets décrits d'une manière dépendante par rapport à d'autres. Dans ce cas, la dépendance est transitive.

Définition: dépendance transitive

Soit R, $X \rightarrow A$ et K une clé de R, on dit que $X \rightarrow A$ est une dépendance transitive de R si X n'est pas contenu dans K et A n'est pas contenu dans X

Dans la Figure 15-7, on peut voir que le groupe de constituants X et A définit un objet indépendant. En décomposant R en R1 et R2, on élimine la dépendance transitive où:

$R1 = R^+ - A$

$R2 = X \cup A$

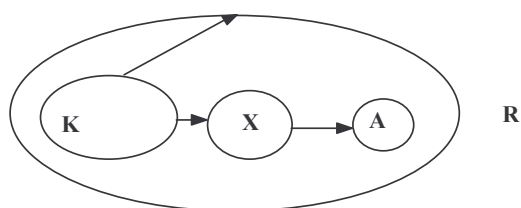


Figure 15-7 : schéma de la dépendance transitive

Exemple de relation contenant une dépendance transitive

géographie (Ville, Pays, Superficie)

Ville \rightarrow Pays

Pays \rightarrow Superficie est une df transitive

Définition : 3NF

Une relation en 2NF est en 3NF s'il n'existe pas de dépendance transitive

Exemple: soit les trois relations suivantes et leurs df

Commande (NoCom, NoClient, Nom, Adresse, PrixTotal)

LigneCommande (NoCom, NoArt, Qte, Art_total)

Article (NoArt, Prix)

NoCom \rightarrow PrixTotal, NoClient

NoClient \rightarrow Nom, Adresse (1)

NoArt \rightarrow Prix

NoCom, NoArt \rightarrow Qte

Nous avons gauche (NoClient \rightarrow Nom, Adresse) qui n'est pas contenu dans la clé (Commande) donc Commande n'est pas en 3FN

Pour transformer le schéma en 3NF il convient de le réécrire comme suit:

```

Commande (NoCom, NoClient, PrixTotal)
Client (NoClient, Nom, Adresse)
LigneCommande (NoCom, NoArt, Qte, Art_total)
Article (NoArt, Prix)

```

Forme normale Boyce-Codd (BCFN)

Dans cette forme normale toutes les parties gauches des df d'une relation contiennent une clé

Définition BCFN

On dit que le schéma (R;F) est en BCFN si pour $X \rightarrow A$ une dépendance à valider sur R telle que A n'est pas contenu dans X alors on a K une clé de R contenue dans X.

Exemple de relation qui n'est pas en BCNF:

```

R = (RVN)
F = {VR → N, N → V}

```

R n'est pas en BCFN, car $N \rightarrow V$ est à valider et N ne contient pas de clé (clé de R est VR)

La BCFN est une forme (trop) contraignante car on ne peut pas toujours trouver pour R et F une décomposition dont les schémas ont simultanément les propriétés de BCFN, de décomposition totale et de préservation des df.

Exemple: soit les relations suivantes et leurs df

```

Commande (NoCom, NoClient, PrixTotal)
Client (NoClient, Nom, Adresse)
LigneCommande (NoCom, NoArt, Qte, Art_total)
Article (NoArt, Prix)
NoCom → PrixTotal, NoClient
NoClient → Nom, Adresse
NoArt → Prix
NoCom, NoArt → Qte

```

La décomposition est en BCNF

Cependant en ajoutant

```
prix , Qte → Art_total
```

Dans les df projetées sur LigneCommande, on a la df:

```
NoArt, Qte → Art_total
```

dont la partie gauche NoArt, Qte ne contient pas de clé de LigneCommande, donc LigneCommande n'est pas en BCNF

A l'extrême LigneCommande doit être décomposée en

```
LigneCommande (NoCom, NoArt, Qte)
```

```
Table_multiplication(Prix, Qte, Art_total)
```

Nous verrons dans le chapitre suivant que les vues permettent d'éviter une telle extrémité; la df prix , Qte \rightarrow Art_total exprime en fait une dépendance de calcul.

La 3FN une relaxation de la BCFN

Une définition équivalente de la 3FN permet de voir que la 3FN est une relaxation de la BCFN.

Définition: constituant premier

Nous dirons qu'un constituant C est premier s'il appartient à une clé sinon il est non-premier.

Définition 2 : 3FN

On dit que le schéma (R;F) est en 3FN si pour $X \rightarrow A$ une dépendance à valider sur R telle que A n'est pas contenu dans X alors on vérifie une des deux conditions suivantes:

- 1) il existe K une clé de R contenu dans X
- 2) A est premier

La condition 1 est identique à celle de la BCFN. La 3FN est donc bien une relaxation des contraintes 3FN. Donc toutes les relations BCFN sont en 3FN

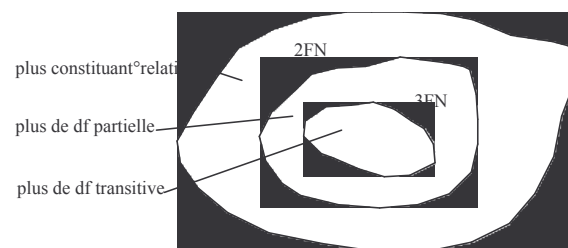


Figure 15-8 : Imbrications des formes normales

Les moyens de ne pas être en 3FN sont donc:

- La 1FN: spécifie qu'un constituant ne soit pas un schéma R
- la 2FN: X est un sous-ensemble strict d'une clé
- on dit que $X \rightarrow A$ est une dépendance partielle
- la 3FN: X est sous ensemble d'aucune clé
- on dit que $X \rightarrow A$ est une dépendance transitive

Sans entrer dans les détails, nous dirons qu'il existe toujours une décomposition (ou plusieurs si le graphe des df a des cycles) qui possède simultanément les trois propriétés que nous avons examinées (totale, préserve les df et 3FN). Par contre, le triplet (total, préserve les df et BCFN) peut être vide.

Notre objectif n'étant pas de faire de la conception, nous ne donnons pas d'exercice sur la décomposition. Toutefois le lecteur peut constater que pratiquement toutes les modélisations des études de cas sont en 3FN et possèdent les bonnes propriétés de la décomposition totale et de la

préservation des dépendances fonctionnelles (quand cela n'est pas le cas, une question y fait référence).

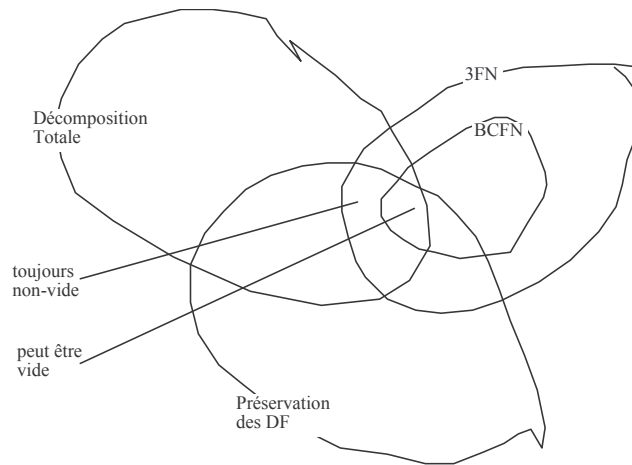


Figure 15-9 : Interdépendances des propriétés sur la décomposition

16. Vues

"Le rêve semblait toujours si vivant, si exact, moins distorsion du réel que simulacre, illusion si riche en détails de la vie éveillée que Nashe ne soupçonnait jamais qu'il était en train de rêver." Paul Auster - La musique du hasard

Jusqu'à présent, une relation était toujours associée à une instance qui stockait physiquement les entités. Une vue permet de déclarer un schéma de relations qui est défini à partir d'autres relations (ou éventuellement d'autres vues). L'instance d'une vue est évaluée au moment où elle est utilisée dans une requête; une vue ne possède donc pas d'instance physique.

Ceci permet de créer des redondances logiques dans le schéma sans induire pour autant des anomalies de mise à jour. Par contre, une vue devant être réévaluée à chaque fois que l'on s'y réfère, cela peut entraîner une diminution de performance dans certains cas.

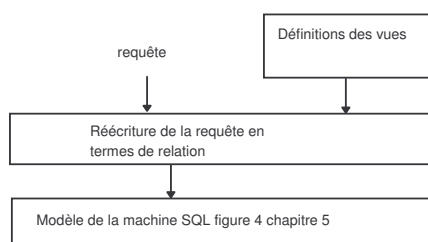


Figure 16-1 : Réécriture de la requête SQL

Le modèle d'exécution d'une requête SQL doit être complété, par un module de réécriture de la requête qui substitue lexicalement les termes de la requête par ceux de la vue (parfois, c'est l'inverse qui est pratiqué). Au minimum, nous avons les substitutions suivantes:

- a) l'identificateur de la vue dans la clause `FROM` de la requête est substitué par les identificateurs spécifiés dans la clause `FROM` de la vue.
- b) les colonnes de la vue sont substituées par les expressions spécifiées dans la clause `SELECT` de la vue.
- c) la clause `WHERE` de la vue est ajoutée à la clause `WHERE` de la requête.
- d) Dans les cas les plus simples, la clause `GROUP BY` de la vue est ajoutée à la requête (ce type de cas est problématique comme on le verra plus loin).

Examinons un cas, soit la vue et la requête suivante:

```
CREATE VIEW Chambre_avec_tv_2
  (Num_chambre, prix_par_pers)
AS SELECT num_chambre, prix/nbr_pers
  FROM Chambres
  WHERE equipement='TV';

SELECT num_chambre, prix_par_pers
  FROM Chambre_avec_tv_2
  WHERE num_chambre=44;
```

L'application de la règle a) donne:

```
SELECT num_chambre, prix_par_pers
  FROM Chambres
  WHERE num_chambre=44;
```

L'application de la règle b) donne:

```
SELECT num_chambre, prix/nbr_pers
  FROM Chambres
  WHERE num_chambre=44;
```

L'application de la règle c) donne:

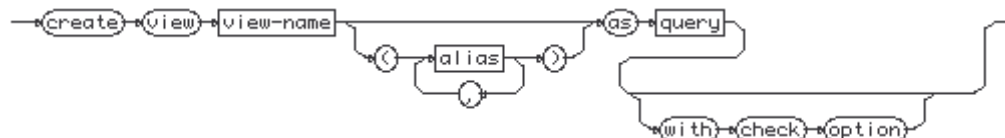
```
SELECT num_chambre, prix/nbr_pers
  FROM Chambres
  WHERE num_chambre=44
     AND equipement='TV';
```

C'est la requête que nous aurions écrit si la vue n'existait pas.

Vues en SQL

La clause `CREATE-VIEW` permet d'associer un schéma de vue à une requête.

CREATE-VIEW



Par défaut, le schéma de la vue est celui de la requête. Les noms et les types des colonnes sont ceux qui apparaissent dans la clause de projection de la requête.

Exemple: Une vue définie pour "les chambres ayant une TV" ?

```
CREATE VIEW Chambre_avec_tv_1
AS SELECT *
  FROM Chambres
  WHERE equipement='TV';
```

En interrogeant la vue on obtient:

```
SELECT * FROM Chambre_avec_tv_1;
```

Il est aussi possible de donner un nom à chaque colonne de la vue en les mentionnant après le nom de la vue.

```
CREATE VIEW Chambre_avec_tv_2
  (Num_chambre, prix_par_pers)
AS SELECT num_chambre, prix/nbr_pers
  FROM Chambres
  WHERE equipement='TV';
```

En interrogeant la vue on obtient:

```
SELECT * FROM Chambre_avec_tv_2;
```

Dans l'exemple suivant, on crée une vue qui correspond à l'agrégation des prix moyens.

```
CREATE VIEW Prix_moyen_par_confort
  (confort, prix_moyen)
AS SELECT confort, avg(prix)
  FROM Chambres
  GROUP BY confort;

SELECT * FROM Prix_moyen_par_confort;
```

Normalement l'implémentation des vues consiste en une réécriture de la requête utilisant la vue, donc les deux requêtes suivantes devraient être illégales, comme le souligne C.J. Date [DAT87]. Cependant le SGBD Oracle (que l'on utilise pour tester les requêtes de ce livre) les accepte. Ceci s'explique par le fait que pour ces cas, il crée temporairement une table correspondant à la vue.

Cette particularité est à première vue très pratique, mais dans certains cas elle peut pénaliser fortement la performance de l'exécution. Surtout dans le cas de jointure sur une vue agrégée car les optimisations qui peuvent normalement s'effectuer ne sont plus possibles et le système doit évaluer et stocker temporairement la vue à chaque appel.

```
SELECT * FROM Prix_moyen_par_confort
  WHERE prix_moyen>100;13

SELECT avg(prix_moyen) FROM Prix_moyen_par_confort;14
```

Les modifications à travers une vue sont restreintes aux vues qui n'utilisent pas les opérations suivantes:

- La jointure
- Les agrégations (GROUP BY)
- Les connexions (connect by)
- La clause distinct

Dans le cas de colonnes résultats d'expression, ces dernières ne peuvent pas être mises à jour. Tous ces cas sont indéterminés, car il correspondent à la modification , à partir d'une valeur, de plusieurs rangées ou de plusieurs colonnes d'une table. Si une colonne est absente dans la table de référence alors elle prend la valeur null.

¹³ La réécriture devrait donner `where avg(prix)>100`, ce qui est interdit car on doit utiliser la clause `having`

¹⁴ Ici on devrait avoir `avg(avg(prix))`, ce qui n'a pas de sens.

La clause `with check option` permet de spécifier que les modifications sont faites à l'intérieur de l'instance de la vue. C'est à dire qu'une sélection à travers la vue doit pouvoir sélectionner le résultat de la modification (On peut donner un nom à cette contrainte).

Exemple: la mise à jour de la colonne équipement fait sortir la rangée de l'instance de la vue.

```
insert into Chambre_avec_tv_1
  (num_chambre,prix,nbr_lits,
   nbr_pers,confort,equipement)
values (99,100,2,2,'BAIN','TV');
update Chambre_avec_tv_1
  set equipement='VD'
  WHERE num_chambre=99;
delete FROM Chambre_avec_tv_1;
```

Créons une vue similaire avec la clause de vérification d'appartenance à la vue et soumettons les mêmes modifications.

```
CREATE VIEW Chambre_avec_tv_1_valide
AS SELECT *
  FROM Chambres
  WHERE equipement='TV'
with check option constraint valide_la_vue;
insert into Chambre_avec_tv_1_valide
  (num_chambre,prix,nbr_lits,
   nbr_pers,confort,equipement)
values (99,100,2,2,'BAIN','TV');
update Chambre_avec_tv_1_valide
  set equipement='VD'
  WHERE num_chambre=99;
delete FROM Chambre_avec_tv_1_valide;
```

L'exécution de la mise à jour a été rejetée car elle viole notre contrainte d'intégrité.

Multiplier les représentations logiques

Nous donnons un certain nombre de cas où les vues sont utiles. L'idée central étant de créer une indépendance logique entre le schéma de la base de données et les applications qui l'utilisent. Celle-ci n'est pleinement réalisée que dans le cas de l'interrogation et dans un nombre de cas limités de modification de la base.

Créer des vues contextuelles

La création des vues contextuelles permet de construire à partir d'un ensemble de relations, une nouvelle relation. L'utilisateur en utilisant le contexte devient alors indépendant des modifications qui peuvent être apportées au schéma sous-jacent.

Dans l'exemple suivant, on crée la vue sur la relation universelle. (Elle n'est pas équivalente à la relation universelle car elle ne prend en compte que les entités qui sont jointes. Pour une discussion approfondie sur ce thème consulter le travail de G. Falquet [FAL89])

```
CREATE VIEW hotel
  (NUM_CLIENT, NOM, PRENOM, ADRESSE,
   NUM_CHAMBRE, PRIX, NBR_LITS,
   NBR_PERS, CONFORT, EQUIPEMENT,
   DATE_ARR, DATE_DEP)
AS SELECT CLIENTS.NUM_CLIENT, NOM, PRENOM, ADRESSE,
        CHAMBRES.NUM_CHAMBRE, PRIX, NBR_LITS,
        NBR_PERS, CONFORT, EQUIPEMENT,
        DATE_ARR, DATE_DEP
   FROM CHAMBRES, CLIENTS, RESERVATIONS
  WHERE Clients.num_client=Reservations.num_client
     AND Chambres.num_chambre=Reservations.num_chambre
```

La vue hotel permet à un utilisateur de faire abstraction des schémas CHAMBRES, CLIENTS, RESERVATIONS. Ces derniers peuvent être modifiés. Il suffit de maintenir une vue équivalente pour satisfaire l'utilisateur de la vue.

Interfacer un schéma

Supposons que vous ayez acheté une application BD, celle-ci possède déjà un schéma de relations. La société qui l'a développé est susceptible de le modifier. Cependant vous aimeriez développer une extension. Pour être indépendant, il suffit de travailler sur des vues du schéma de l'application. En cas de modification, vous adaptez vos vues afin de les rendre sémantiquement équivalentes. Ceci permet aussi de créer un schéma traduit dans une autre langue à peu de frais.

exemple: Une chaîne anglaise d'hôtels étend notre application HOTEL. Elle aura intérêt à travailler sur la vue Clients suivante:

```
CREATE VIEW Customer
  (ident_cust, name, surname, address)
AS SELECT NUM_CLIENT, NOM, PRENOM, ADRESSE
   FROM Clients
```

Créer des dépendances calculées

Souvent la valeur d'un constituant est entièrement déterminée par de l'information qui existe déjà dans la base de données. Stocker physiquement ces informations demande qu'à chaque modification elles soient remises à jour, ce qui rend plus complexe les programmes de modification.

Examinons le cas suivant:

Un client a un nom et une adresse. Une commande est associée à un client et le prix total de la commande est la somme des lignes de cette commande. Une ligne de commande correspond à un article commandé selon une

certaine quantité. Le prix total de la ligne correspond au produit du prix de l'article par sa quantité. A chaque article est associé un prix.

Ce champ est soumis aux dépendances fonctionnelles suivantes:

NoCom → PrixTotal, NoClient

NoClient → Nom, Adresse

NoArt → Prix

NoCom, NoArt → Qte

La décomposition suivante est tout à fait admissible:

Client (NoClient, Nom, Adresse)

Commande(NoCom, NoClient, PrixTotal)

LigneCommande (NoCom, NoArt, Qte, Art_total)

Article (NoArt, Prix)

Le fait de stocker PrixTotal et Art_total implique que les programmes de l'application modifiant les lignes de commandes et les articles doivent les remettre à jour.

Formellement on devrait ajouter la dépendance fonctionnelle suivante (qui encode la multiplication)

- prix , Qte → Art_total

On a alors les relations suivantes:

LigneCommande (NoCom, NoArt, Qte)

CalculPrix (Prix, Qte, Art_total)

Les vues permettent d'éviter de telles extrémités

Supposons le schéma suivant :

Client (NoClient, Nom, Adresse)

Commande(NoCom, NoClient)

LigneCommande (NoCom, NoArt, Qte)

Article (NoArt, Prix)

```
CREATE VIEW
```

```
  LigneCommAvecPrix(NoCom, NoArt, Qte, Art_total)
AS SELECT Nocom, a.NoArt, Qte, Prix*Qte
   FROM lignecommande a, article b
   WHERE a.NoArt=b.NoArt
```

```
CREATE VIEW
```

```
  CommAvecPrix (NoCom, NoClient, PrixTotal)
AS SELECT b.Nocom, NoClient, sum(Art_total)
   FROM lignecommande a, Commande b
   WHERE a.NoCom=b.NoCom
   GROUP BY b.Nocom, NoClient
```

On a éliminé les constituants calculés des relations stockées. Par contre, on a spécifié des vues qui permettent à tout moment de les recalculer.

Nous retrouvons la dualité de la redondance contre l'évaluation, l'une avec son coût en complexité accrue de programmation (la place physique n'étant

plus généralement l'argument principal) contre un coût en ressource de calcul (directement lié à la performance)

Déduire de l'information

A partir d'un ensemble de faits et d'un ensemble de règles de déduction, il est possible de produire de nouvelles informations. La programmation logique, exprimée formellement par les clauses de Horn (voir l'ouvrage de R. Kowalski [KOW79]), permet de définir de telles règles de déduction. Le langage de programmation PROLOG [CLO81] est une implémentation de cette approche déductive. Les bases de données relationnelles peuvent être considérées comme un cas particulier de la programmation logique. Les faits sont représentés par les entités et les requêtes sont des clauses logiques dont les réponses sont des interprétations possibles. Dans ce contexte, les vues peuvent être perçues comme des règles de déduction (la limite étant que l'on ne peut pas exprimer des règles récursivement)

Examinons l'exemple classique de l'arbre généalogique. Soit les deux relations suivantes. Les personnes sont identifiées par un nom et associées à un sexe. La relation GENI détermine que le parent est un des géniteurs de l'enfant.

```
CREATE TABLE pers (nom char(20),
                  sexe char(1));
CREATE TABLE geni (parent char(20),
                  enfant char(20));
```

Nous avons entré les données suivantes:

A partir de ces données, nous aimerions poser les questions:

- - qui est la soeur de ... ?
- - qui sont les grand-parents ... ?
- - qui est le cousin de ... ?
- - qui est l'ancêtre de ... ?

Les vues suivantes répondent à ces questions:

```
CREATE VIEW femme
AS SELECT nom
   FROM pers
   WHERE sexe='F';
CREATE VIEW homme
AS SELECT nom
   FROM pers
   WHERE sexe='H';
CREATE VIEW pere_de
AS SELECT parent pere15, enfant
   FROM geni, homme
   WHERE geni.parent=homme.nom;
```

¹⁵ L'utilisation d'un alias permet de renommer la colonne de la vue

```

CREATE VIEW mere_de
  AS SELECT parent mere, enfant
     FROM geni, femme
     WHERE geni.parent=femme.nom;
CREATE VIEW soeur_de16
  AS SELECT a.enfant soeur, b.enfant nom
     FROM geni a, geni b, femme
     WHERE a.parent=b.parent
        AND a.enfant=femme.nom
        AND a.enfant<>b.enfant17;
CREATE VIEW frere_de
  AS SELECT a.enfant frere, b.enfant nom
     FROM geni a, geni b, homme
     WHERE a.parent=b.parent
        AND a.enfant=homme.nom
        AND a.enfant<>b.enfant;
CREATE VIEW grandpere_de
  AS SELECT a.pere grandpere, b.enfant petitenfant
     FROM pere_de a, geni18 b
     WHERE a.enfant=b.parent;
CREATE VIEW grandmere_de
  AS SELECT a.mere grandmere, b.enfant petitenfant
     FROM mere_de a, geni b
     WHERE a.enfant=b.parent;
CREATE VIEW grandparent_de(grandparent, petitenfant)
  AS SELECT grandpere, petitenfant
     FROM grandpere_de
  union
  SELECT grandmere, petitenfant
     FROM grandmere_de;

```

Nous pouvons faire un test sur nos données.

```

SELECT distinct * FROM soeur_de
  WHERE nom='jacques';
SELECT distinct * FROM grandparent_de
  WHERE petitenfant='amélie';

```

La question des ancêtres est plus délicate car elle fait intervenir la notion de récursivité. C'est à dire:

X ancêtre de Z si

- 1) X est le géniteur de Z
- ou bien 2) si X est ancêtre de Y et Y est le géniteur de Z

On constate que l'on peut appliquer à nouveau la définition d'ancêtre dans la condition 2). En toute généralité, il n'est pas possible de créer des vues

¹⁶ Aussi les demi-soeurs

¹⁷ Cette condition interdit que l'on soit la soeur de soi-même.

¹⁸ En remplaçant GENI par PERE_DE, on a le grand père parternel

récurives. Au plus, on peut déplier la définition et créer une vue de géniteur(géniteur(géniteur(...))) en joignant plusieurs fois la relation GENI. Cependant, l'extension CONNECT BY au standard SQL permet de rendre récurives certaines requêtes.

La sous-requête de la vue ancetre_de calcule les descendants de A. La définition d'ancêtre devient alors: A est ancêtre de B si B est dans les descendants de A.

```
CREATE VIEW ancetre_de
AS SELECT a.nom ancetre,b.nom descendant
   FROM pers a, pers b
   WHERE b.nom in (SELECT enfant FROM geni
                   connect by prior enfant=parent
                   start with parent=a.nom);

SELECT * FROM ancetre_de
   WHERE descendant='amélie';
```

Pour terminer cet exemple on peut se poser la question de savoir si deux personnes ont des ancêtres communs.

```
CREATE VIEW meme_famille(dans_la, de)
AS SELECT a.descendant,b.descendant
   FROM ancetre_de a, ancetre_de b
   WHERE a.ancetre=b.ancetre;

SELECT * FROM meme_famille
   WHERE dans_la='amélie' AND de='eliot';

SELECT * FROM meme_famille
   WHERE dans_la='amélie' AND de='nolwenn';
```

Le nombre de réponses correspond aux six ancêtres communs.

Nous voyons que l'introduction des vues dépasse le cadre de la représentation des données. Elles permettent de construire, au dessus des faits, des règles structurées sur les liens sématiques existant entre ces faits et ainsi de déduire de nouvelles informations.

Exercices

Les deux exercices suivants sont destinés à montrer la puissance des vues.

Le jeu des trois dés

Soit le dé modélisé de la façon suivante:

```
CREATE TABLE D (v number(1));
insert into D values(1);
insert into D values(2);
insert into D values(3);
insert into D values(4);
insert into D values(5);
insert into D values(6);
```

A partir de cette unique table créer des vues et des requêtes afin de:

- - créer trois dés indépendants
- - chercher la somme pour un jeté de 3 dés
- - le nombre de jetés de 3 dés possibles
- - les différentes façons d'obtenir 7 avec trois dés
- - le nombre de possibilités de jetés de trois dés associés à chaque somme
- - dresser un tableau en terme de statistique pour chaque somme

Produit scalaire et le produit matriciel

Il est intéressant de remarquer qu'une relation peut facilement représenter un vecteur. La relation Vect donne à chaque élément le nom du vecteur auquel il appartient, sa position et sa valeur

```
CREATE TABLE vect (nomvect char(10),
                  i number(3),
                  value number);
```

Ainsi les vecteurs $V=(1,2,4)$ et $W=(1,0.5,0.25)$ sont représentés par l'instance VECT suivante:

Le produit scalaire de deux vecteurs est égal à la somme des produits de chaque élément des deux vecteurs occupant la même position.

On a donc $V*W=(1*1+2*.5+4*.25)=3$

et $V*V=(1*1+2*2+4*4)=21$

Chercher une vue qui calcule tous les produits scalaires de Vect (normalement les deux vecteurs doivent avoir la même longueur)

Une matrice correspond à un tableau de valeurs, on peut étendre notre définition de VECT à la relation MAT qui définit le nom de la matrice, la ligne, la colonne et la valeur pour chaque élément.

```
CREATE TABLE mat (nommat char(10),
                  l number(3), c number(4),
                  value number);
```

Les deux matrices A et B sont donc représentées par l'instance suivante de MAT.

	1	2	B =	1	2	3
A =	2	4		1	2	3
	3	6				

Le produit de deux matrices donne une nouvelle matrice dont chaque élément correspond au produit scalaire d'une ligne et d'une colonne des matrices multipliées. Ainsi $A*B$ donnera pour l'élément 1,1 du résultat le produit scalaire de la ligne 1 de A et de la colonne 1 de B soit 3 l'élément 1,2 du résultat est le produit scalaire de la ligne 1 de A et de la colonne 2 de B soit 6, etc.

	3	6	9
A * B =	6	12	18
	9	18	27

Ecrire une vue qui effectue le produit matriciel de toutes les matrices de MAT (normalement le nombre de colonnes de la première matrice doit être égale au nombre de lignes de la deuxième matrice)

Sur TT3

Etablir une vue qui permette de répondre aux questions suivantes, par des requêtes sans de jointure .

- - liste des véhicules et leur catégorie de permis
- - liste des véhicules de moins de 20 places
- - liste des véhicules ayant entre 5 et 8 places

Etendre la vue précédente pour tenir compte de la relation carburant.

Il doit maintenant être facile de donner la consommation par modèle, par catégorie, par nbr de place.

Etablir une vue qui regroupe ces statistiques.

Donner les groupes qui consomment moins de 10 litres aux 100.

Réponses

Le jeu des trois dés

- créer trois dés indépendants

```
CREATE VIEW D1
AS SELECT * FROM D;
CREATE VIEW D2
AS SELECT * FROM D;
CREATE VIEW D3
AS SELECT * FROM D;
```

- chercher la somme pour un jeté de 3 dés

```
CREATE VIEW sum3D (v1,v2,v3,sum3)
AS SELECT D1.v,D2.v,D3.v,D1.v+D2.v+D3.v
FROM D1,D2,D3;
```

- le nombre de jetés de 3 dés possibles

```
SELECT count (*)
FROM sum3D;
```

- les différentes façons d'obtenir 7 avec trois dés

```
SELECT * FROM
sum3D
WHERE sum3=7;
```

- le nombre de possibilités associées à chaque somme

```
SELECT sum3,count (sum3)
FROM sum3D
GROUP BY sum3;
```

- dresser un tableau en terme de statistique pour chaque somme

```
SELECT sum3 somme,
       (count(sum3)/216)*100 probabilite,'% '
FROM sum3D
GROUP BY sum3;
```

Les vecteurs et les matrices

La condition de jointure porte sur la position dans le vecteur.

```
CREATE VIEW prod_scalaire(a,b,prod)
AS SELECT a.nomvect,b.nomvect,sum(a.value*b.value)
FROM vect a,vect b
WHERE a.i=b.i
GROUP BY a.nomvect,b.nomvect;
```

```
SELECT * FROM prod_scalaire;
```

La vue `prod_scalaire` calcule l'élément (l,c) du résultat, la condition de jointure porte sur la position dans la ligne l et la colonne c.

```
CREATE VIEW prod_scalaire_LC(a,b,l,c,prod)
AS SELECT a.nommat,b.nommat,a.l,b.c,sum(a.value*b.value)
FROM mat a,mat b
WHERE a.c=b.l
GROUP BY a.nommat,b.nommat,a.l,b.c;
```

Le produit matriciel est l'ensemble possible des valeurs des produits scalaires. Nous avons ajouté la condition qui empêche de donner le résultat pour des matrices non-compatibles

```
CREATE VIEW prod_mat(a,b,l,c,value)
AS SELECT a,b,l,c,prod
FROM prod_scalaire_lc
WHERE exists (SELECT 'vrai'
              FROM mat a, mat b
              WHERE a.nommat=a AND b.nommat=b
              GROUP BY a.nommat, b.nommat
              having max(a.c)=max(b.l));
```

Seules $A * B$ et $B * A$ sont possibles

```
SELECT * FROM prod_mat;
```

sur TT3

```
CREATE VIEW vhc_type
AS SELECT nochassis,noplaque,miseenservice,v.modele,nostation,
nbplaces,categorie,typecarburant,automatique,poids
FROM vehicule v, type t
WHERE v.modele=t.modele;

SELECT nochassis,categorie
FROM vhc_type;

SELECT nochassis,nbplaces
FROM vhc_type
WHERE nbplaces<20;

SELECT nochassis,nbplaces
FROM vhc_type
WHERE nbplaces between 5 AND 8;
```

```
CREATE VIEW vhc_type_carb
AS SELECT nochassis,v.noplaque,miseenservice,
      v.modele,nostation,
      nbplaces,categorie,v.typecarburant
      carburantrequis,automatique,poids,
      nojour,kilometrage,litres,
      c.typecarburant carburantplein
      FROM vhc_type v, carburant c
      WHERE v.noplaque=c.noplaque;
SELECT * FROM vhc_type_carb
      WHERE carburantrequis<>carburantplein;
CREATE VIEW analyse_consomation
      (critere,valeur,consom100)
AS SELECT 'modele',
      modele,(sum(litres)/sum(kilometrage))*100
      FROM vhc_type_carb
      GROUP BY modele
union
      SELECT 'nbplaces',
      to_char(nbplaces),(sum(litres)/sum(kilometrage))*100
      FROM vhc_type_carb
      GROUP BY to_char(nbplaces)
union
      SELECT 'carburant',
      carburantplein,(sum(litres)/sum(kilometrage))*100
      FROM vhc_type_carb
      GROUP BY carburantplein;
SELECT *
      FROM analyse_consomation;
SELECT *
      FROM analyse_consomation
      WHERE consom100<10;
```


17. Optimisation et performance

"Ainsi certaines fourmis naissent avec d'énormes mandibules cisailles pour être soldat, d'autres possèdent des mandibules broyantes pour produire de la farine de céréales, d'autres sont équipées de glandes salivaires surdéveloppées pour mouiller et désinfecter les jeunes larves." Bernard Weber - Les fourmis.

La première partie de ce chapitre est consacrée à l'optimisation des requêtes SQL. Jusqu'à présent, nous avons fait abstraction de l'implémentation physique des relations et nous avons décrit le fonctionnement de la machine SQL en termes purement logiques. Les instances des relations dans un SGBD sont stockées physiquement; au schéma logique des relations est associé un schéma physique. Ce dernier spécifie pour chaque relation le modèle physique et les paramètres de stockage utilisés.

La richesse des modèles utilisés dépend du constructeur du SGBD. Le choix du schéma physique dépend alors de l'utilisation de la base de données. Pour que les requêtes SQL restent logiquement indépendantes du schéma physique, le SGBD est muni d'un *optimiseur* de requêtes qui traduit la requête SQL en une séquence d'actions la plus appropriée au schéma physique existant. Une même requête a donc une traduction directement dépendante du schéma physique. L'optimiseur utilise les informations du dictionnaire et les propriétés théoriques pour effectuer sa traduction.

Dans la deuxième partie ce chapitre, nous traitons des pièges de la normalisation et de l'utilisation généralisée des vues. Nous montrons que l'approche généralement préconisée consistant à:

- analyser les données et concevoir le schéma
- établir des vues pour faciliter la programmation des traitements
- choisir un schéma physique de stockage (placer judicieusement des index)

peut mener à des systèmes ayant de mauvaises performances. Ceci montre que des solutions localement idéales peuvent mener à une solution globalement insatisfaisante.

Afin d'éviter ceci, il faut que les contraintes de performance, d'accès aux données et la spécification des traitements usuels et fréquents soient considérés en parallèle avec la conception des données. Nous préconisons donc une approche qui ne privilégie pas les données aux dépens des traitements.

Optimisation des requêtes SQL

L'optimiseur utilise d'une part les propriétés de l'algèbre relationnelle et d'autre part les structures de données accompagnant les tables mémorisées.

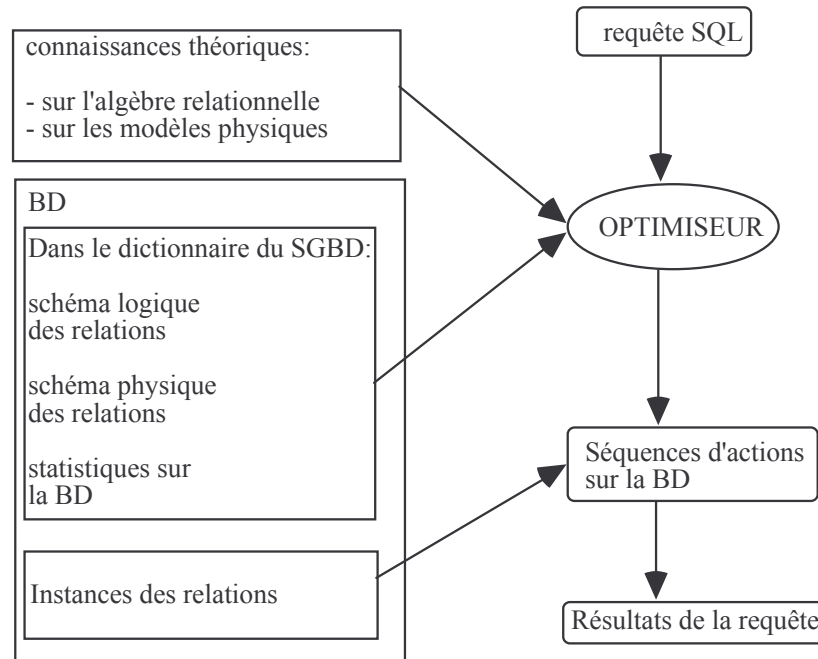


Figure 17-1 : Optimisation des requêtes SQL

Les propriétés de l'algèbre relationnelle nous donnent des règles de réécriture ou des équivalences entre expressions. Ces équivalences sont utilisées pour éliminer les n-uplets qui ne satisferont pas le prédicat de la requête. Par exemple, si un prédicat de sélection ne fait intervenir qu'une relation, alors on a l'équivalence suivante:

$$(*FR_1 \wedge FR_2 \wedge \dots \wedge FR_n) (R_1 * R_2 * \dots * R_n) + (*FR_1 R_1) * (*FR_2 R_2) * \dots * (*FR_n R_n)$$

En utilisant cette équivalence, on voit qu'il est possible de déplacer le produit cartésien après les sélections sur les relations. En procédant ainsi, on élimine un très grand nombre de n-uplets qui n'auraient pas satisfait le prédicat. Imaginons que nous ayons 1000 clients, 1000 réservations et 100 chambres dans notre hôtel. La requête suivante, produirait 100'000'000 de n-uplets à tester, alors qu'en effectuant d'abord la sélection du numéro de client sur CLIENT, il nous reste que 100'000 n-uplets à joindre

```

SELECT Nom, prix, datedeb, datefin
  FROM CHAMBRES, CLIENTS, RESERVATIONS
 WHERE Clients.num_client=Reservations.num_client
 AND Chambres.num_chambre=Reservations.num_chambre
 AND Clients.num_client=1001
  
```

SQL permet de mémoriser des relations et de les interroger sans spécifier les structures de données ou les chemins d'accès à ces données. Cependant si de tels chemins d'accès existent, le SGBD pourra optimiser la requête afin d'éliminer à priori des n-uplets qui ne satisfont pas le prédicat. Ces chemins d'accès sont surtout utiles pour l'opération d'équi-jointure. Ils ne jouent pas

de rôle au niveau sémantique de la relation. On peut à tout moment les ajouter, les supprimer ou modifier la structure de stockage des tables. Les structures utilisées sont celles que l'on étudie dans un cours de structures de données: les arbres, les B-arbres, les fonctions de "hash-coding", les partitionnements, les index, etc.

Les index sont une information supplémentaire qui permet directement d'accéder aux rangées ayant une certaine valeur. En prenant comme métaphore les livres, s'il n'existe pas d'index, pour rechercher une information, on est obligé de parcourir tout le livre, l'index permet de retrouver directement la page intéressante. Il en va de même pour une relation, sans index, le SGBD doit la parcourir complètement, avec un index, il pointe directement sur la partie intéressante. Comme pour un livre, l'index n'ajoute aucune information intéressante à part une facilité d'accès. En construisant un index sur les numéros de client pour la relation CLIENTS, les numéros de client pour la relation RESERVATIONS et un index sur les numéros de chambre pour la relation CHAMBRES, la requête précédente qui exigeait 100'000'000 de tests sur des n-uplets se réduit aux parcours de pointeurs dans un B-arbre (la structure de l'index) et à quelques entrées/sorties sur les disques. Dans la Figure 17-2, on peut suivre le cheminement de l'optimisation suivante produite par le SGBD:

Dans la relation CLIENTS

```
chercher Clients.num_client=1001
```

```
pour chaque n-uplet trouvé
```

```
  Dans la table RESERVATIONS
```

```
  chercher Clients.num_client=Reservations.num_client
```

```
  pour chaque n-uplet trouvé
```

```
    Dans la table CHAMBRES
```

```
    chercher Chambres.num_chambre=Reservations.num_chambre
```

```
    pour chaque n-uplet trouvé
```

```
      effectuer la projection.
```

Cette requête optimisée aurait pu être écrite par un programmeur. La différence réside dans le fait que le SGBD fait évoluer dynamiquement le résultat de ses optimisations en fonction des chemins d'accès existant au moment de l'exécution de la requête. Alors que le programme écrit reste statique et ne peut aucunement évoluer en fonction des besoins émergent du champ d'application ou d'une meilleure connaissance de ce dernier.

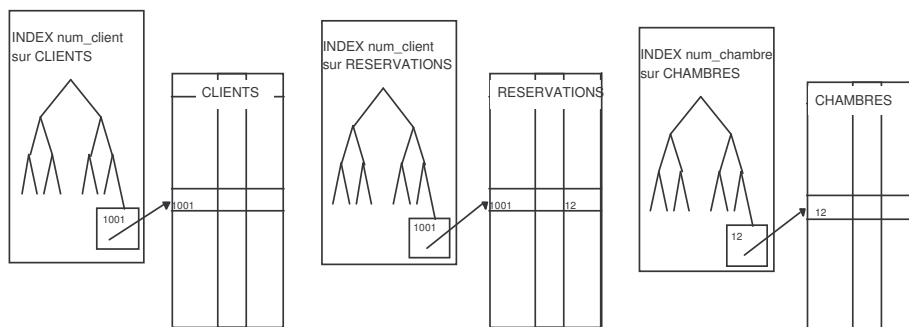


Figure 17-2 : Utilisation des index lors de la sélection.

Le processus d'optimisation dépend fortement de l'implantation du SGBD. L'optimisation se découpe généralement dans les étapes suivantes:

- Par la réécriture de la requête SQL. Cette phase traduit une requête SQL en une autre requête SQL équivalente du point de vue sémantique, mais qui exploitera au maximum les connaissances des modèles physiques de stockage. La requête obtenue est plus performante car elle sera plus sélective du point de son exécution. Cependant cette phase est entièrement syntaxique et peut s'exécuter sans utiliser les informations sur les schémas physiques.
- Par la transformation de la requête SQL en une séquence d'actions sur les données physiques. Ce "comment on va réelement le faire" doit déterminer et choisir les chemins d'accès aux données, l'ordre des jointures et les méthodes pour effectuer les jointures. Cette phase s'exécute sur la base des schémas physiques et éventuellement sur la base d'informations statistiques sur les instances.

En général les SGBD permettent de connaître la planification d'actions que l'optimiseur a effectuées.

Réécriture de la requête SQL

L'idée sous-jacente à chaque transformation est d'obtenir des expressions qui accéderont plus directement à l'information. C'est à dire qui pourront être utilisées directement lors de la planification des actions. Par exemple dans:

```
SELECT * FROM Chambres WHERE not (prix>100)
```

On souhaite obtenir toutes les rangées qui ne sont pas sélectionnées. Le SGBD ne sait que faire des sélections il est donc préférable de lui faire exécuter la requête équivalente suivante:

```
SELECT * FROM Chambres WHERE prix<=100
```

Sans être exhaustif, nous donnons une liste des principales transformations syntaxiques que l'optimiseur peut effectuer.

élimination des expressions constantes (l'expression est évaluée qu'une seule fois)

```
    prix <840/7  -- (prix à la semaine)
->  prix <120
```

transformation des recherches dans les chaînes de caractères en égalité (utilisable par les index)

```
    nom like 'Dumas'
->  nom = 'Dumas'
```

transformation des expressions d'appartenance en une expression disjonctive (utilisable par les index)

```
    confort in ('BAIN', 'DOUCHE')
->  confort='BAIN' OR confort='DOUCHE'
```

transformation des expressions ANY explicitées en une expression disjonctive (utilisable par les index)

```
prix < any (100,840/7)
-> prix < 100 OR prix < 840/7
```

transformation des expressions ALL explicitées en une expression conjonctive (utilisable par les index)

```
prix < all (100,840/7)
-> prix < 100 AND prix < 840/7
```

transformation des expressions ANY implicites en une expression EXISTS augmentée de la condition (plus sélective)

```
a < any (SELECT prix FROM chambres
        WHERE confort ='BAIN'))
-> Exists (SELECT 'vrai' FROM chambres
        WHERE confort ='BAIN'
        AND a<prix)
```

transformation des expressions ALL implicites en une expression EXISTS augmentée de la condition inversée (plus sélective)

```
a < all (SELECT prix FROM chambres
        WHERE confort ='BAIN'))
-> not Exists (SELECT 'vrai' FROM chambres
        WHERE confort ='BAIN'
        AND a>=prix)
```

transformation des expressions BETWEEN en une expression conjonctive

```
prix between 100 AND 120
-> prix >= 100 AND prix <= 120
```

simplification des expressions NOT en utilisant l'opérateur inverse

```
not (prix < 100 OR confort ='WC')
-> prix >=100 AND confort <>'WC'
```

transformation des clauses disjonctives en sous-requêtes liées par un UNION ALL (utilisation possible des index dans les sous-requêtes)

```
SELECT * FROM chambres
WHERE confort='BAIN' OR confort='DOUCHE'
-> SELECT * FROM chambres WHERE confort='BAIN'
UNION ALL
SELECT * FROM chambres WHERE confort='DOUCHE'
```

transformation des sous-requêtes en une requête équivalente utilisant une jointure (utilisation possible des index dans les sous-requêtes)

```
SELECT * FROM reervation
WHERE num_chambre in (SELECT chambres.num_chambre
                     FROM chambres
                     WHERE confort='BAIN')
```

```
-> SELECT reversion.* FROM reversion, chambres
      WHERE reversion.num_chambre=chambres.num_chambre
            AND confort='BAIN')
```

Ces transformations ne sont pas toujours possibles, la sous requête doit alors être évaluée pour chaque rangée de la requête si elle en dépend.

```
SELECT * FROM chambres c1
      WHERE c1.prix > (SELECT avg(c2.prix)
                      FROM chambres c2
                      WHERE c1.confort=c2.confort)
```

-> pas de requête équivalente avec une jointure

L'optimiseur intègre aussi les vues dans les requêtes (déjà discuté dans le chapitre sur les vues). Les transformations que nous avons examinées sont toutes syntaxiques, c'est à dire que l'on peut les effectuer sans connaissance du schéma physique de la base de données. L'étape suivante va consister à sélectionner les chemins d'accès.

Chemins d'accès

Les données sont stockées dans des structures physiques qui ont chacune des méthodes d'accès particulières. Pour notre exposé, nous ne considérons que les structures de tables séquentielles et d'index. Cependant les SGBD mettent d'autres structures à disposition du concepteur telles que les tables dont les données sont regroupées physiquement si elles possèdent des valeurs communes pour certains constituants (*cluster*); les tables dont la place des données est fixée par un algorithme de *hash coding*; ou bien des structures de tables séquentiellement indexées. Pour ces structures, on applique les mêmes critères de choix que pour les deux cas que nous examinons.

Table séquentielle

Dans cette structure, les données sont rangées séquentiellement dans l'espace qui est alloué à la table. Chaque rangée est identifiée par un identificateur de rangée (*rowid*¹⁹) entièrement dépendant de la place physique de la rangée.

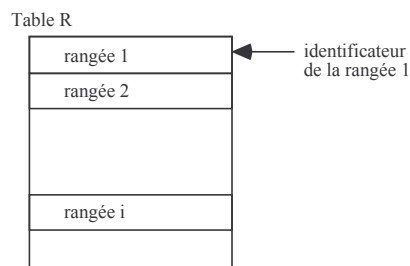


Figure 17-3 : Structure de la table séquentielle

¹⁹ L'identificateur de rangée dans ORACLE indique le numéro du fichier, le numéro du bloc dans ce fichier, la place dans ce bloc de la rangée

Les méthodes de recherche d'une rangée pour cette structure sont:

- la lecture séquentielle de toute la table, rangée après rangée (*full scan table*)
- l'accès à une rangée par son identificateur (*by rowid*)

Exemple sur Hotel (les plans d'exécutions sont obtenus avec l'ordre `Explain plan` du SGBD Oracle. Les plans décrivent la méthode et l'accès utilisé. Les conditions de sélection sont utilisées lors de l'accès soit comme clé d'accès ou bien comme filtre):

```
SELECT * FROM chambres WHERE num_chambre=12;
```

Cherchons le rowid d'une rangée

```
SELECT rowid FROM chambres WHERE num_chambre=12;
```

```
SELECT * FROM chambres
WHERE rowid='0000004E.0008.0002';
```

L'exécution est bien conforme à ce qui était prévu

L'identificateur de rangée est accessible, mais son utilisation est associée à d'autres structures de données telles que les index.

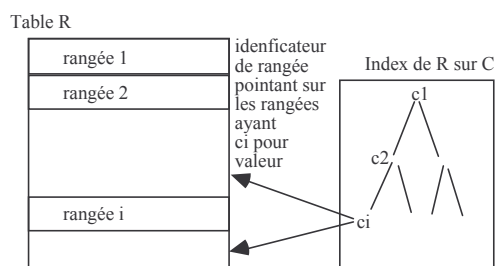
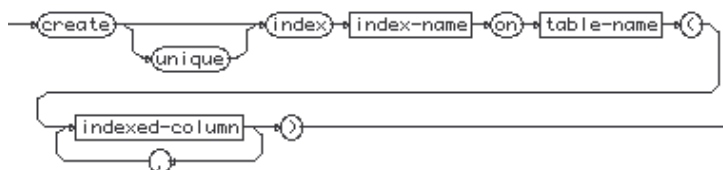


Figure 17-4 : Structure d'index associé à une table

Les index

Cette structure est associée à une table, elle est construite sur un certain nombre de constituants de la table. A chaque groupe de valeurs est associé un ou plusieurs identificateurs de rangée. Les valeurs sont elles-mêmes organisées dans une structure qui facilite la recherche, souvent sous forme d'arbre binaire (pour une information de base, examiner l'ouvrage de N. Wirth [WIR76]).

La commande de création de ces index est une clause SQL qui est ajoutée au standard. Dans le SGBD Oracle, nous avons la clause suivante:



On associe donc un identificateur d'index à une table et une liste de colonnes. La clause unique spécifie que ce groupe est une clé de la table donc qu'il n'y a qu'un seul identificateur de rangée pour une valeur des colonnes indexées.

Exemple:

```
CREATE unique index chambres_by_number
  on chambres(num_chambre);
```

Les méthodes principales de recherche d'un identificateur de rangée pour cette structure sont:

- l'accès à un identificateur de rangée avec un index unique(*Unique scan*)
- l'accès à une liste d'identificateurs de rangée avec un index (*range scan*)

Pour que ces accès soient possibles, il faut connaître les valeurs des constituants définissant l'index. Le plan d'exécution de la requête précédente devient alors:

- Chercher le rowid dans l'index pour num_chambre=12
- Chercher la rangée portant ce rowid dans la table Chambres²⁰

```
SELECT * FROM chambres WHERE num_chambre=12;
```

En créant un index sur prix, on a des recherches qui nous ramènent tous les identificateurs ayant un même prix.

```
CREATE index chambres_by_prix on chambres(prix);
SELECT * FROM chambres WHERE prix=100;
```

Choix des chemins d'accès

L'optimiseur va donc examiner toutes les possibilités d'accéder aux rangées des tables. Pour cela il utilisera donc les conditions de sélection de la requête.

Le choix d'un chemin est déterminé simplement par une pondération de la préférence qui reflète directement la performance de l'accès (liste par ordre de préférence):

- accès à une rangée par un identificateur de rangée
- accès par un index unique
- accès par un index
- accès séquentiel à la table

Le critère d'accès peut être raffiné si l'on possède des informations statistiques sur l'instance de la base de données (les propriétés discriminantes des index, la taille des tables par exemple, etc). Des règles du modèle physique peuvent alors être utilisées automatiquement. Une telle règle sera par exemple, "si l'on accède à plus de 25% des rangées d'une table en utilisant un index alors il faut préférer l'accès séquentiel" ou encore "si la taille d'une table est plus petite que 4 Koctets alors préférer l'accès

²⁰Les plans d'exécution se lisent à l'envers, ils correspondent à un arbre, les commandes les plus imbriquées sont exécutées en premier.

séquentiel". Les statistiques permettent de vérifier que les accès sophistiqués sont justifiés.

Le travail effectué par l'optimiseur lors de la réécriture est exploité dans la recherche des meilleurs chemins d'accès. Ajoutons un index sur confort.

```
CREATE index chambres_by_confort on chambres(confort);
```

Le plan d'exécution montre bien que la requête à été réécrite sous forme d'une concaténation de deux sélections, l'une portant sur le prix et l'autre sur le confort

```
SELECT * FROM chambres
  WHERE prix<100 OR confort='BAIN';
```

Choix de la méthode de jointure

La dernière étape de l'optimisation consiste à choisir une méthode et un ordre de jointure entre les relations. Les méthodes de jointure sont dépendantes des chemins d'accès (et donc des structures physiques de stockage). Avec les structures retenues, nous avons deux méthodes:

- la jointure par tri et fusion
- la jointure par boucle imbriquée

Jointure par tri et fusion

Soit deux tables R et S à joindre par le groupe de constituants C, alors l'algorithme est le suivant

```
LE TRI
  trier R avec C comme clé de tri
  placer le résultat dans une table Rt=(C, rowid)
  trier S avec C comme clé de tri
  placer le résultat dans une table St=(C, rowid)
LA FUSION21
  lire la première rangée de Rt
  lire la première rangée de St
  pour chaque rangée de Rt faire
    tant que st.C <=rt.C faire
      lire prochaine rangée de St
      si st.C =rt.C alors
        joindre st.rowid & rt.rowid
        mettre dans la relation RES
  refaire
refaire
```

Cette méthode de jointure nécessite une lecture séquentielle de chaque table, les tris et la fusion. Le tri est l'opération la plus sensible car sa

²¹La fusion proposée est correcte si C est une clé de R, si cela n'est pas le cas, il faut compléter l'algorithme par une possibilité de retourner en arrière dans la table St pour relire des valeurs lorsqu'on rencontre deux valeurs identiques se succédant dans Rt.

complexité est de $O(n \cdot \log(n))$, les autres sont d'ordre $O(n)$ où n est la cardinalité de la plus grande table.

Elle ne nécessite pas de structure physique particulière.

Jointure par boucle imbriquée

Soit deux tables R et S à joindre par le groupe de constituants C, alors l'algorithme est le suivant

```
BOUCLE IMBRIQUEE
  pour chaque rangée de R faire
    pour chaque rangée de S faire
      si s.C =r.C alors
        joindre s & r
        mettre dans la relation RES
    refaire
  refaire
```

Sans structure physique particulière, cette méthode de jointure nécessite une lecture séquentielle de la table R et n lectures séquentielles de la table S où n est la cardinalité de R. Du point de vue performance, elle est catastrophique si n est grand ou si S est grande. La jointure par tri-fusion est préférable. Cependant si on peut accéder aux rangées de S à travers un index sur la charnière C de la jointure, cet algorithme est très performant.

Ordre et méthode

Dans le cas de jointure de plus de 2 relations, les relations sont toujours jointes deux par deux, mais alors une des relations est le résultat d'une jointure. L'optimiseur examine et attribue une note à toutes les possibilités de jointures. Il examine aussi les couples équivalents $R*S$ et $S*R$ qui peuvent différer dans leur taille et les index à exploiter. Ces algorithmes d'évaluation sont assez complexes et dépendent étroitement des structures physiques mises en oeuvre dans le SGBD.

Pour terminer examinons deux exemples:

Cette requête dans une structure sans index utilise la méthode de tri-fusion

```
SELECT r.num_chambre,prix
  FROM chambres c,reservations r
 WHERE c.num_chambre=r.num_chambre;
```

Après la création d'un index sur les numéro de chambre, l'optimiseur utilise, pour la même requête, la fusion par boucle imbriquée où la boucle intérieure est un accès à travers un index portant sur la charnière de la jointure.

```
CREATE unique index chambres_by_number
  on chambres(num_chambre);
```

La requête suivante porte sur trois relations, sans structure d'index. Un tri-fusion est effectué entre les clients et les réservations (on utilise la

condition de sélection sur le nom pour diminuer le nombre de rangées du résultat). Le résultat est repris dans un deuxième tri-fusion avec la relation chambres.

```
SELECT nom, r.num_chambre, prix
      FROM chambres c, reservations r, clients cl
      WHERE c.num_chambre=r.num_chambre
            AND cl.num_client=r.num_client
            AND cl.nom='DUMAS';
```

```
CREATE unique index chambres_by_number
      on chambres (num_chambre);
```

```
CREATE index reservations_by_num_client
      on reservations (num_client);
```

```
CREATE index clients_by_nom
      on clients (nom);
```

Après avoir créé trois index (similaire à la Figure 17-2), nous obtenons un plan d'exécution qui utilise tous les index. Il s'agit de boucles imbriquées qui portent seulement sur quelques rangées

Pour créer une application performante, le concepteur doit connaître:

- les mécanismes utilisés par le SGBD
- les schémas physiques de son application
- les traitements de l'application

Dans la deuxième partie de ce chapitre, nous examinons comment ces différents points sont interdépendants.

Une perception holistique de la conception

Nous illustrons notre propos par un exemple simplifié, tiré de la comptabilité. Nous avons gardé, le coeur d'une base de données comptable, à savoir enregistrer des écritures ventilant, sur des comptes, des pièces comptables regroupées dans des journaux comptabilisés pour une période comptable. Nous avons conservé l'aspect multi-société.

Nous avons retenu les constituants suivants:

Date_jour	dom	date	
Date_Valeur	dom	date	
Debit_Credit	dom	mot ('D', 'C')	
Lib_Ecriture	dom	texte	
Lib_journal	dom	texte	
Lib_Piece	dom	texte	
Montant	dom	numérique réel	
Num_Compte	dom	numérique entier	[1..999999]
Num_Ecriture	dom	numérique entier	[1..999999]
Num_journal	dom	numérique entier	[1..999999]
Num_Piece	dom	numérique entier	[1..999999]
Periode_comptable	dom	mot ('9301', '9302', ...)	
Societe	dom	mot ('SA', ...)	

Le prédicat portant sur ces constituants est:

```
Tout (Date_jour, Date_Valeur, Debit_Credit,
      Lib_Ecriture, Lib_journal, Lib_Piece, Montant,
      Num_Compte, Num_Ecriture, Num_journal, Num_Piece,
      Periode_comptable, Societe)
predicat: "||Tout (dj, dv, dc, le, lj, lp, mt, nc, ne, nj, np, pc, s
o)|| L'écriture libellée le, au (débit/crédit) dc, du
montant mt, portant sur le compte nc appartient à la
pièce np, libellée lp, à la date de valeur dv, et est
comptabilisée dans le journal nj, libellé lj, à la
date dj, pour la période pc"
```

Le schéma valide les dépendances fonctionnelles suivantes:

```
Societe, Date_jour → Periode_comptable
Societe, Num_journal → Lib_journal, Date_jour
Societe, Num_Piece → Num_journal, Date_Valeur, Lib_Piece
Societe, Num_Piece, Num_Compte → Montant, Debit_Credit, Lib
_Ecriture
```

Ces dépendances fonctionnelles forment un base irredondante (pas de cycle), nous avons donc directement une décomposition en 3FN (les clés sont soulignées).

```
Periode (Societe, Date_jour, Periode_comptable)
Journal (Societe, Num_journal, Lib_journal, Date_jour)
Piece (Societe, Num_Piece, Num_journal, Date_Valeur,
       Lib_Piece)
Ecriture (Societe, Num_Piece, Num_Compte, Montant,
          Debit_Credit, Lib_Ecriture)
```

Nous obtenons donc le schéma de création suivant pour le SGBD:

```
CREATE TABLE PERIODE (
    Societe CHAR(3) not null,
    Date_Jour DATE not null,
    Periode_Comptable CHAR(4) not null)
CREATE TABLE JOURNAL (
    Societe CHAR(3) not null,
    Num_Journal NUMBER (6) not null,
    Date_Jour DATE not null,
    Lib_Journal CHAR(30))
CREATE TABLE PIECE (
    Societe CHAR(3) not null,
    Num_Piece NUMBER (6) not null,
    Num_Journal NUMBER (6) not null,
    Date_Valeur Date not null,
    Lib_Piece CHAR(30))
```

```
CREATE TABLE ECRITURE (
    Societe CHAR(3) not null,
    Num_Piece NUMBER (6) not null,
    Num_Compte NUMBER (6) not null,
    Montant Number(13,2) not null,
    Debit_Credit char(1) not null,
    Lib_Ecriture CHAR(30))
```

Pour faciliter notre travail , on peut définir la vue *Tout*. D'un point de vue méthodologique, travailler sur des vues devrait minimiser les modifications dans les traitements si l'on doit changer le schéma de la base de données.

```
CREATE VIEW tout AS
SELECT
    D.societe,D.date_jour,D.periode_comptable,
    J.Num_journal,J.lib_journal,
    P.num_piece,P.date_valeur,P.lib_piece,
    E.num_compte,E.Montant,E.debit_credit,E.lib_ecriture
FROM periode D,journal J, piece P, Ecriture E
WHERE D.societe=J.societe
    AND D.date_jour=J.date_jour
    AND J.societe=P.societe
    AND J.num_journal=P.num_journal
    AND P.societe=E.societe
    AND P.num_piece=E.num_piece;
```

Nous avons retenu les traitements suivants:

- 1) calculer le mouvement d'un compte pour une période
- 2) rechercher les pièces et les écritures d'un journal
- 3) rechercher les écritures d'une pièce
- 4) vérifier qu'un groupe d'écritures est balancé (somme des débits est égale à la somme des crédits pour chaque pièce)

Pour effectuer les jointures et effectuer les différents accès, nous définissons les index suivants:

```
CREATE unique index periode_i1
    on periode(societe,date_jour);
CREATE unique index journal_i1
    on journal(societe,num_journal);
CREATE unique index piece_i1
    on piece(societe,num_piece); -- pour accéder par les
pièces
CREATE index piece_i2;
    on piece(societe,num_journal);
CREATE index ecriture_i1
    on ecriture(societe,num_piece);
```

```
CREATE index ecriture_i2
  on ecriture(societe,num_compte); -- pour accéder par les
  comptes
```

Pour ces quatre questions, nous avons écrit une requête SQL utilisant la vue Tout. Les résultats sont présentés dans un tableau comportant la requête, son coût d'exécution et son plan d'exécution.

Q	SQL	coût	Accès utilisé
1	<pre>SELECT decode(debit_credit,'D',- montant,montant) FROM tout WHERE societe='SA' AND num_compte=11 AND periode_comptable='9304';</pre>	17	<pre>NESTED LOOPS NESTED LOOPS NESTED LOOPS TABLE ACCESS BY ROWID PERIODE INDEX RANGE SCAN PERIODE_I1 TABLE ACCESS BY ROWID JOURNAL INDEX RANGE SCAN JOURNAL_I1 TABLE ACCESS BY ROWID PIECE INDEX RANGE SCAN PIECE_I2 TABLE ACCESS BY ROWID ECRITURE INDEX RANGE SCAN ECRITURE_I2</pre>
2	<pre>SELECT * FROM tout WHERE societe='SA' AND num_journal=11;</pre>	3	<pre>NESTED LOOPS NESTED LOOPS NESTED LOOPS TABLE ACCESS BY ROWID PERIODE INDEX RANGE SCAN PERIODE_I1 TABLE ACCESS BY ROWID JOURNAL INDEX UNIQUE SCAN JOURNAL_I1 TABLE ACCESS BY ROWID PIECE INDEX RANGE SCAN PIECE_I2 TABLE ACCESS BY ROWID ECRITURE INDEX RANGE SCAN ECRITURE_I1</pre>
3	<pre>SELECT * FROM tout WHERE societe='SA' AND num_piece=1;</pre>	15	<pre>NESTED LOOPS NESTED LOOPS NESTED LOOPS TABLE ACCESS BY ROWID PERIODE INDEX RANGE SCAN PERIODE_I1 TABLE ACCESS BY ROWID JOURNAL INDEX RANGE SCAN JOURNAL_I1 TABLE ACCESS BY ROWID PIECE INDEX UNIQUE SCAN PIECE_I1 TABLE ACCESS BY ROWID ECRITURE INDEX RANGE SCAN ECRITURE_I1</pre>
4	<pre>SELECT num_piece,sum(decode(debit_credit,'D',- montant,montant)) FROM tout WHERE societe='SA' AND num_piece between 1 AND 100 GROUP BY num_piece having sum(decode(debit_credit,'D',- montant,montant))<>0;</pre>	36	<pre>NESTED LOOPS NESTED LOOPS NESTED LOOPS TABLE ACCESS BY ROWID PERIODE INDEX RANGE SCAN PERIODE_I1 TABLE ACCESS BY ROWID JOURNAL INDEX RANGE SCAN JOURNAL_I1 TABLE ACCESS BY ROWID PIECE INDEX UNIQUE SCAN PIECE_I1 TABLE ACCESS BY ROWID ECRITURE INDEX RANGE SCAN ECRITURE_I1</pre>

Pour ces tests, nous avons utilisé une petite base de données d'environ 5000 écritures, portant sur 4 périodes.

A première vue, pas de "FULL SCAN", tous les accès se font par des index. Effectivement dans chaque requête, on spécifie la société donc on peut toujours accéder avec un index (juste sur la première clé). Malgré la

diversité des requêtes, on accède toujours dans le même ordre aux tables PERIODE -> JOURNAL -> PIECE -> ECRITURE.

Cette ordre est fixé par la vue Tout. En effet, les valeurs d'accès sont substituées lexicalement dans la vue. La contrainte sur la société portera toujours sur le constituant société de période, alors que si l'on désire accéder par le numéro de journal, il faut contraindre celui de la table Journal pour profiter des index.

En effectuant des test sur dix fois plus d'entités, les coûts peuvent facilement centupler. Le relativement bon score de la question deux s'explique par le fait que le SGBD a utilisé l'index sur le numéro de pièce et que le nombre de rangées à ce stade de la résolution était faible (500). La vue Tout telle qu'elle est définie ne correspond en fait à aucun type d'accès qu'il faut privilégier du point de vue des traitements.

Spécialiser les vues pour un accès principal

En examinant la vue Tout et nos traitements, on peut constater que l'on doit accéder, par les journaux, par les pièces et par les écritures. Il faut donc créer trois vues, préservant un accès privilégié pour ces types de sélection (en gras, on trouve les différences avec la vue Tout.

```
CREATE VIEW tout_E AS -- par les écritures
SELECT
    E.societe, J.date_jour, D.periode_comptable,
    J.Num_journal, J.lib_journal,
    E.num_piece, P.date_valeur, P.lib_piece,
    E.num_compte, E.Montant, E.debit_credit, E.lib_ecriture
FROM
    periode D, journal J, piece P, Ecriture E
WHERE E.societe=J.societe
    AND D.date_jour=J.date_jour
    AND E.societe=P.societe
    AND J.num_journal=P.num_journal
    AND P.societe=E.societe
    AND P.num_piece=E.num_piece;

CREATE VIEW tout_J AS -- par les journaux
SELECT
    J.societe, J.date_jour, D.periode_comptable,
    J.Num_journal, J.lib_journal,
    P.num_piece, P.date_valeur, P.lib_piece,
    E.num_compte, E.Montant, E.debit_credit, E.lib_ecriture
FROM
    periode D, piece P, Ecriture E, journal J
WHERE J.societe=D.societe
    AND J.date_jour=D.date_jour
    AND J.societe=P.societe
    AND J.num_journal=P.num_journal
```

```

AND J.societe=E.societe
AND P.num_piece=E.num_piece;

```

```

CREATE VIEW tout_P AS -- par les pièces
SELECT
  P.societe, J.date_jour, D.periode_comptable,
  P.Num_journal, J.lib_journal,
  P.num_piece, P.date_valeur, P.lib_piece,
  E.num_compte, E.Montant,
  E.debit_credit, E.lib_ecriture
FROM
  periode D, piece P, Ecriture E, journal J
WHERE P.societe=D.societe
  AND J.date_jour=D.date_jour
  AND P.societe=P.societe
  AND J.num_journal=P.num_journal
  AND J.societe=E.societe
  AND P.num_piece=E.num_piece;

```

En utilisant ces vues, on obtient le tableau qui suit

Q	SQL	coût	Accès utilisé
1	<pre> SELECT decode(debit_credit,'D',- montant,montant) FROM tout_E WHERE societe='SA' AND num_compte=11 AND periode_comptable='9304'; </pre>	15	NESTED LOOPS NESTED LOOPS NESTED LOOPS TABLE ACCESS FULL PERIODE TABLE ACCESS BY ROWID ECRITURE INDEX RANGE SCAN ECRITURE_I2 TABLE ACCESS BY ROWID PIECE INDEX UNIQUE SCAN PIECE_I1 TABLE ACCESS BY ROWID JOURNAL INDEX UNIQUE SCAN JOURNAL_I1
2	<pre> SELECT * FROM tout_J WHERE societe='SA' AND num_journal=11; </pre>	2	NESTED LOOPS NESTED LOOPS NESTED LOOPS TABLE ACCESS BY ROWID JOURNAL INDEX UNIQUE SCAN JOURNAL_I1 TABLE ACCESS BY ROWID PERIODE INDEX UNIQUE SCAN PERIODE_I1 TABLE ACCESS BY ROWID PIECE INDEX RANGE SCAN PIECE_I2 TABLE ACCESS BY ROWID ECRITURE INDEX RANGE SCAN ECRITURE_I1
3	<pre> SELECT * FROM tout_P WHERE societe='SA' AND num_piece=1; </pre>	2	NESTED LOOPS NESTED LOOPS NESTED LOOPS TABLE ACCESS BY ROWID PIECE INDEX UNIQUE SCAN PIECE_I1 FILTER TABLE ACCESS FULL JOURNAL TABLE ACCESS BY ROWID PERIODE INDEX UNIQUE SCAN PERIODE_I1 TABLE ACCESS BY ROWID ECRITURE INDEX RANGE SCAN ECRITURE_I1

4	<pre>SELECT num_piece,sum(decode(debit_credit,'D',- montant,montant)) FROM tout WHERE societe='SA' AND num_piece between 1 AND 100 GROUP BY num_piece having sum(decode(debit_credit,'D',- montant,montant))<>0;</pre>	23	<pre>FILTER SORT GROUP BY NESTED LOOPS NESTED LOOPS NESTED LOOPS TABLE ACCESS FULL JOURNAL TABLE ACCESS BY ROWID PIECE INDEX RANGE SCAN PIECE_I2 INDEX UNIQUE SCAN PERIODE_I1 TABLE ACCESS BY ROWID ECRITURE INDEX RANGE SCAN ECRITURE_I1</pre>
---	--	----	---

On constate que les questions 2 et 3 ont bénéficié de ce remaniement. Leur exécution accède maintenant directement à la table la plus discriminante. L'exécution de 4 reste coûteuse, car elle est mal formulée, la réponse à cette question peut être calculée uniquement dans le contexte de la table des écritures. La vue nous oblige à faire des jointures pour obtenir des informations qui sont inutiles pour cette question.

Q	SQL	coût	Accès utilisé
4	<pre>SELECT num_piece,sum(decode(debit_credit,'D',- montant,montant)) FROM Ecriture WHERE societe='SA' AND num_piece between 1 AND 100 GROUP BY num_piece having sum(decode(debit_credit,'D',- montant,montant))<>0;</pre>	1	<pre>FILTER SORT GROUP BY TABLE ACCESS BY ROWID ECRITURE INDEX RANGE SCAN ECRITURE_I1</pre>

Dénormalisation

La question 1 ne peut pas être optimisée par une vue, ou par une réduction de contexte. En effet, dans ce schéma, elle est intrinsèquement coûteuse, car elle nécessite trois jointures, pour accéder simultanément à la période et au numéro de compte. La seule solution consiste à dénormaliser, c'est à dire à créer une redondance dans les données. Dans notre cas il faut ajouter l'information période sur l'écriture. Les programmes de saisie des données doivent tenir compte de cette redondance. Le coût de cette dénormalisation se traduit aussi dans une augmentation de la place utilisée pour stocker cette information supplémentaire pour chaque écriture. (dilemme entre calculer et mémoriser). Nous ajoutons donc le constituant période à la table écriture. Il prend pour valeur celle associée à la pièce et au journal.

```
CREATE TABLE ECRITURE (
  Societe CHAR(3) not null,
  Num_Piece NUMBER (6) not null,
  Num_Compte NUMBER (6) not null,
  Période CHAR(4) not null,
  Montant Number(13,2) not null,
  Debit_Credit char(1) not null,
  Lib_Ecriture CHAR(30))
```

Nous remplaçons l'index *écriture_i2* par l'index suivant portant en plus sur la période:

```
CREATE index ecriture_i2bis on
ecriture(societe,num_compte,periode);
```

La question 1 peut maintenant se calculer localement dans le contexte de la table écriture.

Q	SQL	coût	Accès utilisé
1	SELECT decode(debit_credit,'D',- montant,montant) FROM ecriture WHERE societe='SA' AND num_compte=11 AND periode='9304';	1	TABLE ACCESS BY ROWID ECRITURE INDEX RANGE SCAN ECRITURE_I2BIS

Une requête SQL donne toujours une réponse, mais celle-ci peut arriver après une courte éternité ... Il faut donc être très vigilant dans l'écriture des requêtes. La durée maximale d'une requête devrait être inverse à sa fréquence.

Fréquence	durée
mois	heures
jour	quart d'heure
heure	minute
minute	seconde

Avant de penser à changer de machine, il faut examiner minutieusement les requêtes qui sont un goulot d'étranglement. En changeant de machine, on peut gagner un facteur 10 (selon ses moyens). En réécrivant une requête, on peut gagner parfois un facteur 1000.

Nous avons utilisé l'approche préconisée consistant à :

- analyser les données
- concevoir le schéma normalisé
- établir des vues pour faciliter la programmation des traitements
- choisir un schéma physique de stockage
- programmer les traitements

Chaque étape était optimale

- Le schéma était en troisième forme normale
- Les traitements opéraient à travers une vue
- Tous les index nécessaires étaient présents

Pourtant le résultat de l'exécution des traitements avaient de mauvaises performances. La normalisation peut éloigner des informations qu'il serait souhaitable de garder couplées, car elles prennent sens lorsqu'elles sont ensemble. Les vues peuvent rendre inopérantes les optimisations. Les index peuvent être utilisés partiellement. Ceci montre que des solutions

localement idéales peuvent mener à une solution globalement insatisfaisante.

Afin d'éviter ceci, il faut que les contraintes de performance, d'accès aux données et de spécification des traitements usuels et fréquents soient considérés en parallèle avec la conception des données. Nous préconisons donc une approche:

- ne privilégiant pas les données aux dépends des traitements.
- validant les choix du concepteur (par prototypage par exemple).

L'indépendance entre les niveaux conceptuels, logiques et physiques; entre les modèles de données, de traitements et des règles d'intégrité donne un cadre théorique pour le chercheur et un cadre cognitif pour le concepteur. Cependant lors de la réalisation d'un système d'information, les frontières de ces cadres doivent être assouplies, la conception s'y effectue en parallèle, les itérations étant nombreuses. Les niveaux et les modèles deviennent interdépendants. Cette interdépendance est liée aux objectifs et aux choix des concepteurs.

18. INVEST

Énoncé

Une société de gestion de portefeuille a décidé de faire évoluer ses applications de gestion. Elle a choisi de les implanter à l'aide d'un système de gestion de bases de données relationnel et de remplacer ainsi ses anciennes applications écrites en BASIC avec des fichiers séquentiels indexés. Invest gère des portefeuilles de valeur pour différents clients. Elle achète et elle vend des titres. Elle perçoit les revenus. Elle gère les portefeuilles en francs suisses, mais peut acheter des valeurs dans une autre monnaie. Tous les jours, elle interroge un serveur à la bourse qui lui indique le cours du jour pour un titre et les cours de change.

Un consultant a étudié les informations de Invest et propose l'analyse qui suit. Il a volontairement écarté les problèmes touchant à la comptabilité et à la fiscalité dans la gestion des portefeuilles.

Invest gère des valeurs qui sont identifiées par un numéro **NumValeur**. Chaque valeur possède un **Genre** qui spécifie le type de valeur (une action ou une obligation par exemple); une **Branche** indiquant le domaine d'activité économique (chimie, assurance, banque, métallurgie ...); le **Pays** émetteur de la valeur; la monnaie dans laquelle cette valeur est cotée **MonnaieCotation**; un libellé pour la valeur **LibelleValeur**. On lui associe aussi une appréciation interne **Catégorie** permettant de spécifier le type de placement par rapport aux clients (Sécurité, RevenuFixe, Spéculatif, ...). Pour les obligations, on trouve un supplément d'information, indiquant le taux d'intérêt **Taux**, ainsi que la première et la dernière date d'échéance des coupons des obligations **PremEcheance**, **DernEcheance**. La première date d'échéance correspond à une année après la date d'émission.

Pour chaque valeur, on mémorise un historique journalier **DateCotation** des **Cours** de la valeur dans sa monnaie de cotation. Les portefeuilles sont gérés en francs suisses; pour effectuer les conversions de monnaies, on possède un historique journalier **DateChange** des cours de change **CoursChange** des monnaies étrangère **MonnaieEtrangere**.

Un dossier est constitué pour chaque client identifié par un numéro **NumClient**. Pour chaque client, on enregistre son **Nom**, **Prénom**, **Adresse** et numéro de **Téléphone**. Le client peut émettre des contraintes dans la composition de son portefeuille, par exemple qu'il ne veut pas plus de 100'000 FRS de valeurs spéculatives. Une contrainte associe donc un montant maximum **MontantMax** pour une catégorie et un client.

Les transactions sont identifiées par un numéro de transaction unique **NumTrans**. On spécifie le client, la valeur, le type de transaction **TypeTrans**

(Achat, Vente, Split,), la quantité **Qte**, le montant de la transaction **MontantTrans** en francs suisses et la date de la transaction **DateTrans**.

Lorsqu'un client touche un revenu, on identifie celui-ci par un numéro unique **NumRevenu**. On spécifie le client, la valeur, le type de revenu **TypeRevenu** (Dividende, CouponOblig, ...), le montant du revenu **MontantRevenu** en francs suisses et la date du revenu **DateRevenu**.

Le consultant pense qu'avec ces données, il peut faire des estimatifs, des inventaires, vérifier des règles de gestion (ne pas vendre des titres que l'on ne possède pas), etc ...

Domaine des constituants

Adresse	texte Branchemot (chimie, assurance, banque, métallurgie)
Categorie	mot (Sécurité, RevenuFixe, Spéculatif, ...)
CoursChange	nombre
Cours	nombre
DateChange	date
DateCotation	date
DateRevenu	date
DateTrans	date
DernEcheance	date
Genre	mot (action, obligation,)
LibelleValeur	texte
MonnaieCotation	mot (FRS, FF, DM, \$US, ...)
MonnaieEtranger	mot (FRS, FF, DM, \$US, ...)
MontantMax	nombre
MontantRevenu	nombre
MontantTrans	nombre
Nom	mot
NumClient	entier [1..99999]
NumRevenu	entier
NumTrans	entier
NumValeur	entier [1..9999999999]
Pays	mot
PremEcheance	date
Prenom	mot
Qte	entier
Taux	nombre [0..100]
Telephone	texte
TypeRevenu	mot (Dividende, CouponOblig, ...)

TypeTrans mot (Achat, Vente, Split,)

Sémantique

Exemple de faits à mémoriser dans notre modèle

"Le client numéro 342 se nommant Jean Dupont a acheté hier 100 actions américaines de BigHamburger (Numvaleur=140140) de la branche Alimentaire au cours de 143\$. Le taux de change était de 1.45 FRS/\$. Aujourd'hui, il a touché un dividende de 1030 FRS sur ces mêmes actions. Ce client a spécifié qu'il voulait un maximum de 100'000 dans les titres spéculatifs. L'obligation de la "banque du Casino" (NumValeur = 130130) est émise pour 5 ans à partir du 1 avril 2002 à 18 %"

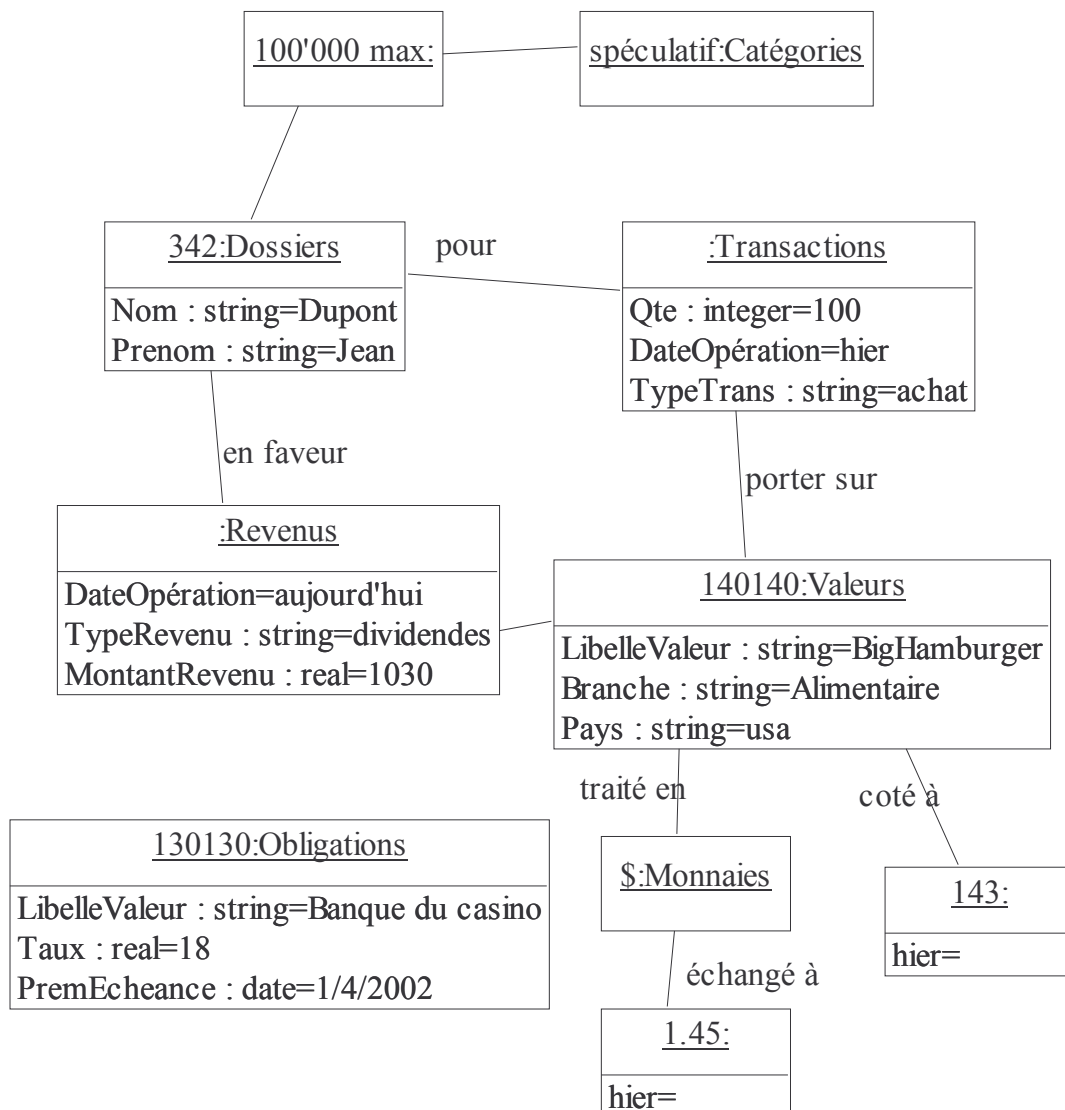
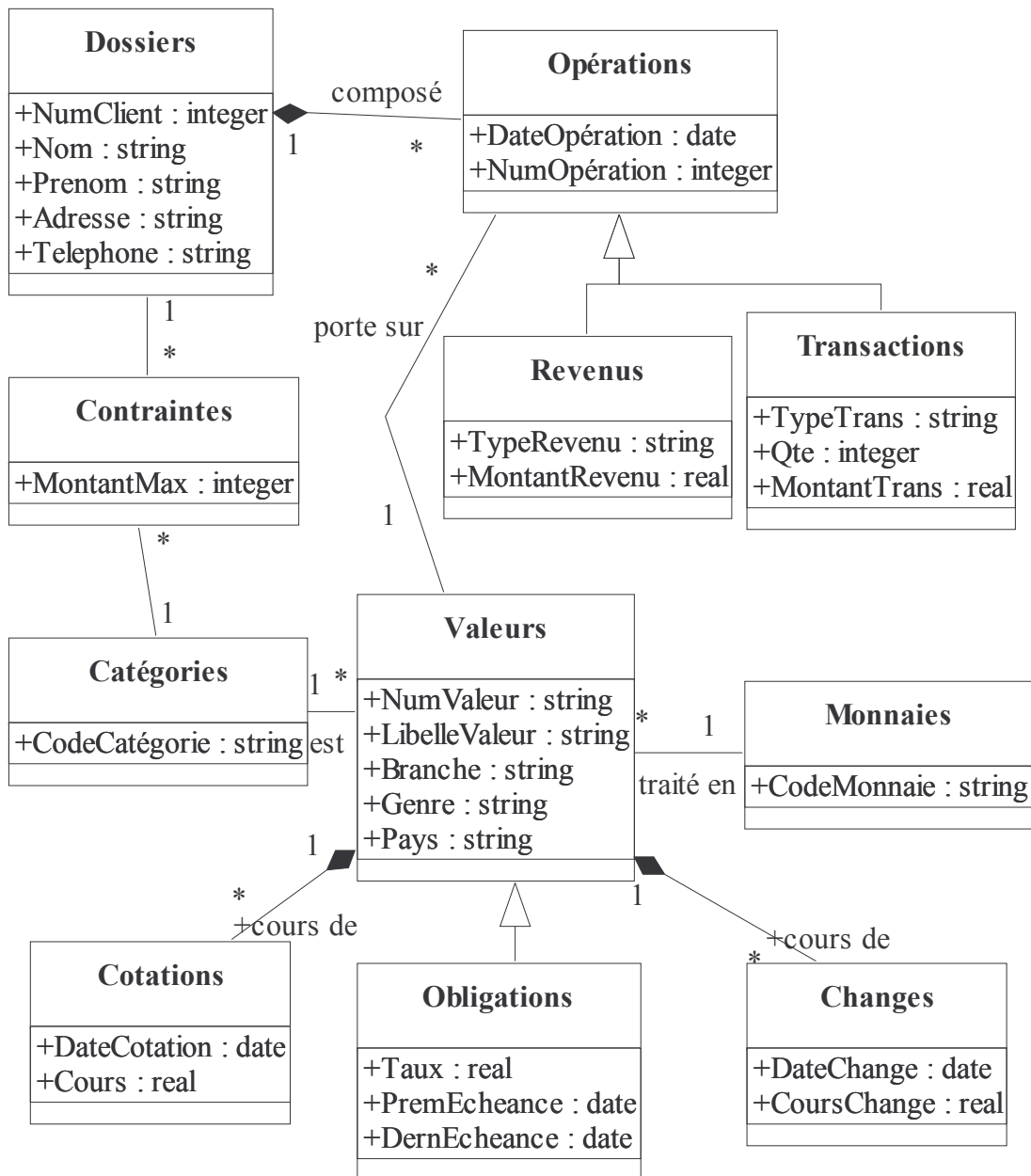


Diagramme des classes



Relations

Valeur	(<u>NumValeur</u> , LibelleValeur, Branche, Genre, Pays, MonnaieCotation, Catégorie)
Obligation	(<u>NumValeur</u> , Taux, PremEcheance, DernEcheance)
Change	(<u>DateChange</u> , MonnaieEtrangere, CoursChange)
Cotation	(<u>NumValeur</u> , <u>DateCotation</u> , Cours)
Dossier	(<u>NumClient</u> , Nom, Prenom, Adresse, Telephone)

Revenu	NumRevenu	NumClient	NumValeur	Type Revenu	Date Revenu	Montant Revenu

Contrainte	NumClient	Categorie	MontantMax

Sur la modélisation

Répondre aux questions suivantes en basant vos arguments uniquement sur la modélisation

- 1) Peut-on acheter une même quantité d'une valeur, le même jour, à un même prix, pour un même client?
- 2) Si on réunit les relations Valeur et Obligation dans une unique relation: Valeurbis(NumValeur, LibelleValeur, Branche, Genre, Pays, MonnaieCotation, Categorie, Taux, PremEcheance, DernEcheance) donner les avantages et désavantages qu'on obtient, chercher la clé de cette nouvelle relation et discuter des anomalies de mise à jour ainsi que du respect de la décomposition
- 3) Si on réunit les relations Valeur et Cotation dans une unique relation: Valeurter (NumValeur, LibelleValeur, Branche, Genre, Pays, MonnaieCotation, Categorie, DateCotation, Cours) donner les avantages et désavantages qu'on obtient, chercher la clé de cette nouvelle relation et discuter des anomalies de mise à jour ainsi que du respect de la décomposition
- 4) Explicitez **toutes** les dépendances existentielles à vérifier sur la modélisation (sous la forme $R[C] \subseteq S[C]$)
- 5) Donnez l'ordre de création qui permet de créer la relation Transaction (NumTrans, NumClient, NumValeur, TypeTrans, DateTrans, Qte, MontantTrans) avec les clauses sur toutes les contraintes d'intégrité
- 6) Imaginez une ri qui existerait dans le champ d'application telle que:
 - a) une primitive de la relation **Transaction** ferait partie de la portée de cette ri et
 - b) cette ri ne pourrait pas être prise en compte lors du CREATE de la relation **Transaction** (donc elle ne peut figurer dans la réponse à 5)

Requêtes

Répondre en SQL aux questions suivantes:

- 1) Liste des valeurs émises en "Espagne", pour lesquelles aucune transaction n'a été émise, par ordre alphabétique

- 2) Liste des obligations françaises et suisses ayant une échéance dans le XXI siècle. (rem : pour utiliser une date après 1999, utiliser la fonction to_date. ex: to_date('05-MAR-2020', 'DD-MMM-YYYY'))
- 3) Etablir un inventaire (QTE actuelle pour chaque valeur d'un portefeuille) pour le client 342
- 4) Liste des cotations du jour (22-6-92) dans la monnaie de base
- 5) Liste par pays des revenus perçus entre le 1-1-91 et le 31-12-91 pour le client "Dupont" (en le supposant unique)
- 6) Liste des valeurs ayant gagné 20% dans les 6 derniers mois (dans leur monnaie de cotation). (rem : utiliser la fonction months_between(date1,date2))

Règles d'intégrité

"...Le client peut émettre des contraintes dans la composition de son portefeuille, par exemple qu'il ne veut pas plus de 100'000 FRS de valeurs spéculatives. Une contrainte associe donc un montant maximum pour une catégorie et un client..." Ce montant est à comparer avec la somme des montants engagés pour les valeurs de cette catégorie. Le montant engagé pour une valeur est la somme des montants des transactions concernant cette valeur.

Donnez la portée de cette ri en remplissant le tableau de contexte suivant (mettre une croix si la case appartient à la portée)

Primitive /relation	insérer	Sup- primer	maj	maj	maj	maj	maj	maj	maj
Transaction			Num Tran s	Num Clie nt	Num Vale ur	Type Tran s	Date Trans	Qte	Mont antT rans
Valeur			Num Vale ur	Libel leVal eur	Bran che	Genr e	Pays	Monn aieC otati on	Cate gorie
Dossier			Num Clie nt	Nom	Pren om	Adre sse	Tele phon e		
contrainte			Num Clie nt	Cate gori e	Mont antM ax				

Donner une requête SQL qui permette de détecter les contraintes qui ne sont pas respectées dans un dossier

19. SECUR

Enoncé

La fiduciaire SECUR est spécialisée dans la gestion des fonds de prévoyance. Ce marché est en pleine expansion depuis que l'état avait rendu obligatoire l'adhésion de tous les salariés d'une entreprise à un fond de prévoyance couvrant l'invalidité et la retraite. Des centaines de fond de prévoyance s'étaient créés à travers tout le pays. Les entreprises avaient confiés la gestion des fonds à des institutions financières et s'étaient déchargées de la partie comptable et administrative sur les fiduciaires.

SECUR dans une phase de nouveau développement informatique a entrepris l'étude d'une base de données permettant de gérer les informations nécessaires à ces fonds de prévoyance.

Le champ d'application couvert par les impératifs de la gestion de cette compagnie de transport concerne donc la parc de véhicules, son entretien, l'administration de chauffeurs et de leur emploi du temps, la gestion des appels des clients à la centrale téléphonique.

Le texte qui suit est une description du champ d'application tel qu'il apparaît à la suite d'une étude avec les comptables (les mots en style gras sont les constituants qui sont retenus dans la modélisation)

Extrait du rapport

Un fond de prévoyance est identifié par un identificateur unique **IdentFond**. Le fond est connu sous une raison sociale **RaisonFond** ayant une adresse **AdrFond** pour la correspondance. Une personne cotisant à un fond est un assuré, identifié par un numéro **IdentAssuré**. Une personne peut être assurée par plusieurs fonds. Un assuré est décrit par un numéro de Sécurité sociale **NumAVS**, il possède un nom **Nom**, un prénom **Prénom**, une date de naissance **NaissAssuré**, un sexe **Sexe**, un état civil **EtatCivil**.

Pour effectuer des calculs actuariels, il était aussi nécessaire de connaître des informations sur les membres de la famille de l'assuré. Pour les enfants d'un assuré, on enregistre leur prénom **PrénomEnfant** et leur date de naissance **NaissEnfant**. Dans le cas où un assuré est marié, on doit aussi connaître le prénom du conjoint **PrénomConjoint** et sa date de naissance **NaissConjoint**.

Un assuré fait partie d'une certaine catégorie **CatégorieAssuré** par rapport aux fonds, il est soit actif (verse ses cotisations annuelles), invalide (est au bénéfice d'une rente d'invalidité), retraité (est au bénéfice d'une rente de retraite), décédé (son conjoint est au bénéfice d'une rente de survivant et/ou ses enfants sont au bénéfice d'une rente d'orphelin) ou quitté (à quitter ce fond, ses cotisations ont été versée sur un autre fond) Un taux

Taux d'activité ou d'invalidité doit être indiqué pour ces catégories. L'appartenance à ces catégories est historisée, c'est à dire que l'on indique le début **DébutPériode** et la fin de la période **FinPériode** où l'assuré appartenait à une catégorie. En ne spécifiant pas la fin de la période (valeur nulle), on indique qu'il s'agit de la catégorie actuelle. Ces catégories sont exclusives entre elles, sauf dans le cas d'une invalidité partielle où l'on admet que l'assuré peut avoir une activité partielle (la somme des taux ne devant pas excéder 100%)

Chaque année, les assurés actifs doivent verser une cotisation **MntCotisation**. Cette cotisation est déterminée en appliquant un certain taux personnel **TauxPersonnel** au salaire annuelle **SalaireAnnuel** de l'assuré. Ce taux de participation personnel **TauxParticipation** est établi en fonction du sexe de l'assuré et de son age. De plus chaque année, l'état fixe une fourchette minimum **SalaireMin** et maximum **SalaireMax** pour les salaires soumis à la participation au fond de prévoyance. D'autres types de cotisation **TypeCotisation** sont possibles, l'apport initial quand un assuré rentre dans le fond, la clôture quand un assuré quitte le fond, les intérêts annuels. Le montant d'une cotisation est positif sauf dans le cas de la clôture.

Les prestations **MntPrestation** versées mensuellement à un assuré. On note le début **DébutPrestation** et la fin **FinPrestation** de la période pendant laquelle on verse une certaine rente. On indique aussi le type de prestation versée **TypePrestation**.

Domaine des constituants

AdrFond	texte
Age	entier [18..120]
Année	entier [1980..2100]
CatégorieAssuré	mot (actif, invalide, retraité, décédé, quitté)
DateCotisation	date
DébutPériode	date
DébutPrestation	date
EtatCivil	mot (célibataire, marié, divorcé, veuf_ve)
FinPériode	date
FinPrestation	date
IdentAssuré	entier [10000..99999]
IdentFond	entier[100..999]
MntCotisation	réel
MntPrestation	réel
NaissAssuré	date
NaissConjoint	date
NaissEnfant	date
Nom	texte

NumAVS	entier [10000000000..9999999999]
Prénom	texte
PrénomConjoint	texte
PrénomEnfant	texte
RaisonFond	texte
SalaireAnnuel	réel
SalaireMax	réel
SalaireMin	réel
Sexe	mot (H,F)
TauxParticipation	réel [0..1]
TauxPersonnel	réel
Taux	réel [0..100]
TypeCotisation	mot(ApportInitial, CotisAnnuelle, IntérêtAnnuel, Cloture)
TypePrestation	mot (rente,invalidité,survivant,orphelin)

Schéma des relations

Barême	(<u>Sexe</u> , <u>Age</u> , TauxParticipation)
Fourchette	(<u>Année</u> , SalaireMin, SalaireMax)
Assuré	(<u>IdentAssuré</u> , NumAVS, Nom, Prénom, Sexe, NaissAssuré, EtatCivil)
DansFond	(<u>IdentAssuré</u> , <u>IdentFond</u>)
Conjoint	(<u>IdentAssuré</u> , PrénomConjoint, NaissConjoint)
Fond	(<u>IdentFond</u> , RaisonFond, AdrFond)
Enfant	(<u>IdentAssuré</u> , <u>PrénomEnfant</u> , NaissEnfant)
Catégorie	(<u>IdentAssuré</u> , <u>DébutPériode</u> , <u>CatégorieAssuré</u> , <u>FinPériode</u> , Taux)
Cotisation	(<u>IdentAssuré</u> , <u>IdentFond</u> , <u>TypeCotisation</u> , <u>DateCotisation</u> , SalaireAnnuel, TauxPersonnel, MntCotisation)
Prestation	(<u>IdentAssuré</u> , <u>IdentFond</u> , <u>TypePrestation</u> , <u>DébutPrestation</u> , <u>FinPrestation</u> , MntPrestation)

Dépendances fonctionnelles

(les instances des relations valident ces d.f. !)

- 1) Sexe, Age -> TauxParticipation
- 2) Année -> SalaireMin, SalaireMax
- 3) IdentAssuré -> NumAVS, Nom, Prénom, Sexe, NaissAssuré, EtatCivil, PrénomConjoint, NaissConjoint
- 4) IdentFond -> RaisonFond, AdrFond
- 5) IdentAssuré, PrénomEnfant -> NaissEnfant

- 6) IdentAssuré, DébutPériode, CatégorieAssuré -> FinPériode, Taux
- 7) IdentFond, IdentAssuré, TypeCotisation, DateCotisation -> SalaireAnnuel, TauxPersonnel, MntCotisation
- 8) IdentFond, IdentAssuré, TypePrestation, DébutPrestation -> FinPrestation, MntPrestation
- 9) DateCotisation, NaissAssuré -> Age
- 10) NumAVS -> IdentAssuré

Diagramme des cas d'utilisation

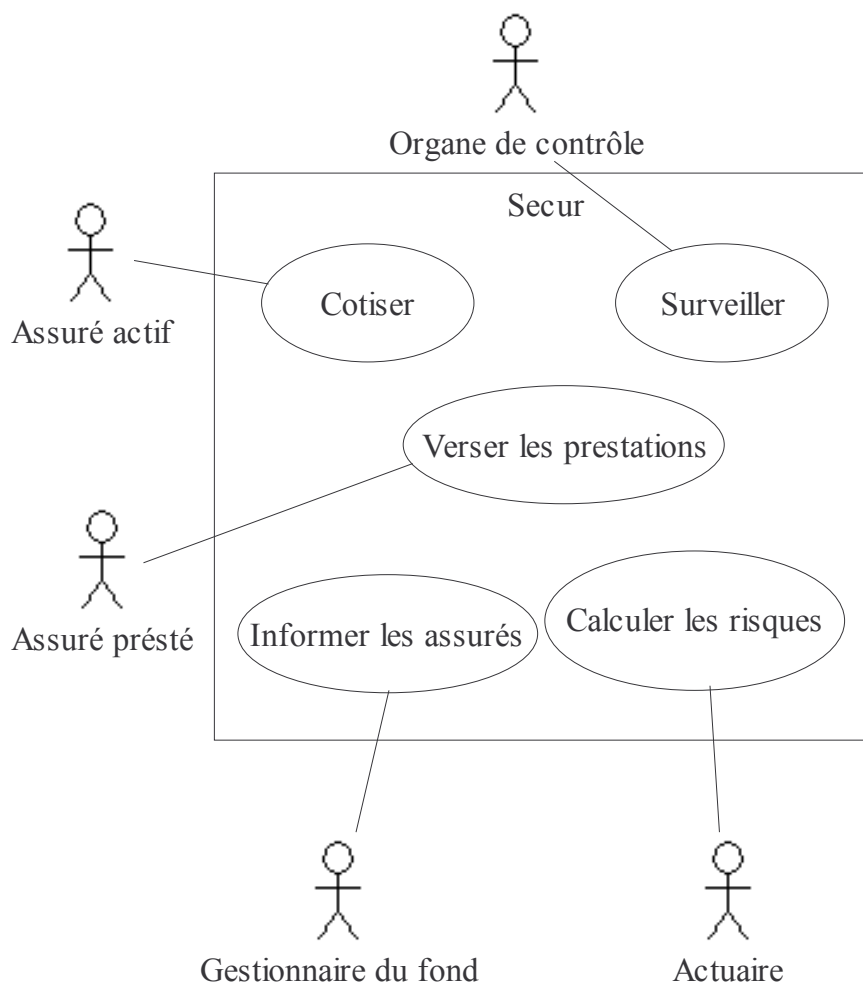
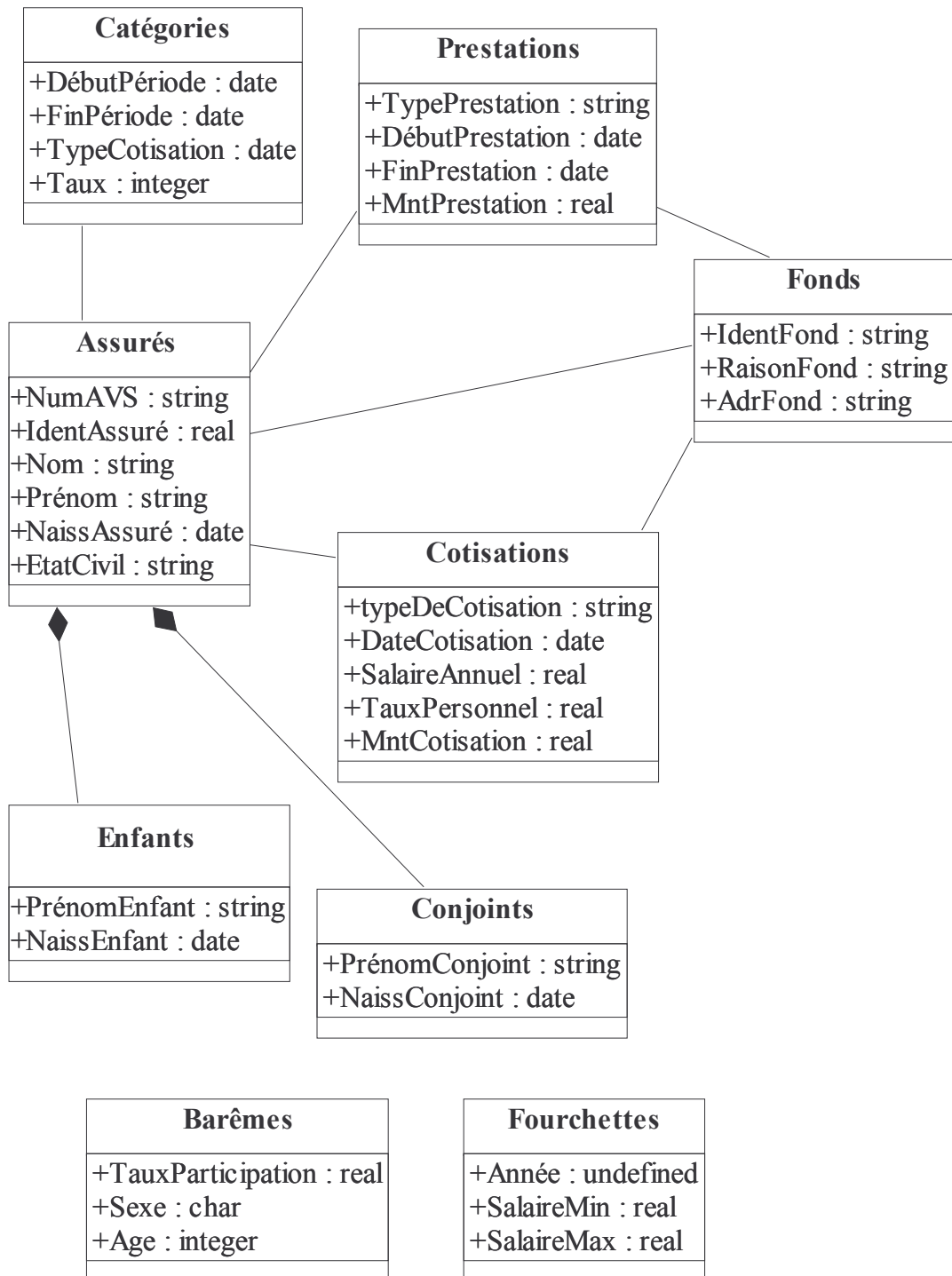


Diagramme des classes



Sémantique

Remplir les tables correspondant aux instances des relations afin d'exprimer le texte qui suit:

--	--	--	--	--	--	--

Prestation	Ident Assuré	Ident Fond	Type Prestation	Début Prestation	Fin Prestation	Mnt Prestation

Sur la modélisation

Répondre aux questions suivantes en basant vos arguments uniquement sur la modélisation

- 1) Un assuré peut-il avoir un nombre différent d'enfant pour deux fonds dont il est adhérent?
- 2) Un assuré peut-il avoir pour une date de cotisation donnée plusieurs taux de participation?
- 3) Un assuré peut-il être simultanément dans les catégories invalide, retraité et quitté ?
- 4) En utilisant FD1 (l'outil à rechercher des clés du laboratoire de BD), avec les dépendances fonctionnelles de l'énoncé, on trouve les clé suivantes.

**Année IdentAssuré IdentFond PrénomEnfant DébutPériode
CatégorieAssuré TypeCotisation DateCotisation TypePrestation
DébutPrestation**

**Année NumAvs IdentFond PrénomEnfant DébutPériode
CatégorieAssuré TypeCotisation DateCotisation TypePrestation
DébutPrestation**

En supposant que l'on substitue partout le constituant **NumAVS** au constituant **IdentAssuré**, en justifiant votre réponse avec la théorie, définir ce que trouvera FD1 après une telle substitution.

- 5) Décrivez ce qui se passe, si l'on supprime la relation **DansFond (IdentAssuré, IdentFond)** et que l'elle soit remplacée par la vue suivante:

```
CREATE VIEW DansFond (IdentAssuré, IdentFond)
AS SELECT distinct IdentAssuré, IdentFond
FROM cotisation;
```

- 6) Explicitez les dépendances fonctionnelles existentielles à vérifier sur la modélisation (sous la forme $R[C] \subseteq S[C]$)?

Requêtes

Répondre en SQL aux questions suivantes:

- 1) Donnez une liste de tous les fonds par ordre alphabétique
- 2) Donnez la liste des assurés du fond 'ForEver' par ordre alphabétique

- 3) Donnez le montant total des cotisations de M.Smith (pour tous les fonds)
- 4) Donnez le montant total des prestations par type de prestation pour le fond No 421
- 5) Donnez la répartition des états civil en % pour l'ensemble des fonds
- 6) Donnez la liste des personnes qui étaient retraitées le 1-jan-90 dans le fond 'ForEver'
- 7) Trouver toutes les cotisations qui ne font pas référence à un assuré

Règles d'intégrité

"...Cette cotisation est déterminée en appliquant un certain taux personnel **TauxPersonnel** au salaire annuelle **SalaireAnnuel** de l'assuré. Ce taux de participation personnel **TauxParticipation** est établi en fonction du sexe de l'assuré et de son age..."

Donnez la portée de cette ri en remplissant le tableau de contexte suivant (mettre une croix si la case appartient à la portée)

Primitive /relation	insérer	Sup- primer	maj	maj	maj	maj	maj	maj	maj
Assuré			Ide ntA ssur é	Nu mA VS	Nom	Prén om	Sexe	Nais sAss uré	Eta tciv il
Cotisation			Ide ntA ssur é	Typ eCo tisa tion	Date Coti satio n	Salai reAn nuel	Taux Pers onne l	Mnt Coti sati on	Ide ntF ond
Barême			Sex e	Age	Taux Parti cipa tion				

Donner une requête SQL qui permette de détecter les cotisations qui n'ont pas le bon taux par rapport à l'age de la personne au moment de la cotisation .

(l'expression suivante calcule l'age de la personne !)

`to_char(DateCotisation),'YYYY')-to_char(DateCotisation),'YYYY')`

20. ULTIMA

Énoncé

Une société fabriquant des composants électroniques a décidé de faire évoluer ses applications de gestion. Elle a choisi de les implanter à l'aide d'un système de gestion de base de données relationnelle et ainsi de remplacer ses applications écrites en COBOL avec des fichiers séquentiels indexés. Cependant l'intégration et le partage des informations par plusieurs applications ont fait émerger de manière aiguë le problème de la sécurité et de la confidentialité. En effet, comment donner accès à la table des personnes aux applications de contrôle de production sans leur donner accès aux salaires de ces mêmes personnes.

Un groupe sécurité "ULTIMA" a donc été formé pour étudier ce problème. L'idée étant que la sécurité doit être intégrée dès le début dans la conception des autres applications.

Le groupe ULTIMA a étudié la structure organisationnelle de l'entreprise, à savoir qu'une personne est identifiée par un numéro d'employé **NumEmp** et décrite par un **Nom**, un **Prénom** et qu'elle travaille pour un seul département **Dept**. Un département peut dépendre hiérarchiquement d'un autre département **DeptSup**. **LibelleDept** est une description étendue du département.

Un résultat de l'étude de ULTIMA est que la structure organisationnelle de l'entreprise n'est pas adéquate pour gérer la confidentialité des données. En effet, des personnes travaillant au sein d'un même département ont accès à des informations différentes, l'apprenti du bureau du personnel et son directeur ont des droits différents. Cependant cette structure a été conservée car elle peut mettre en évidence les informations partagées par les différents départements.

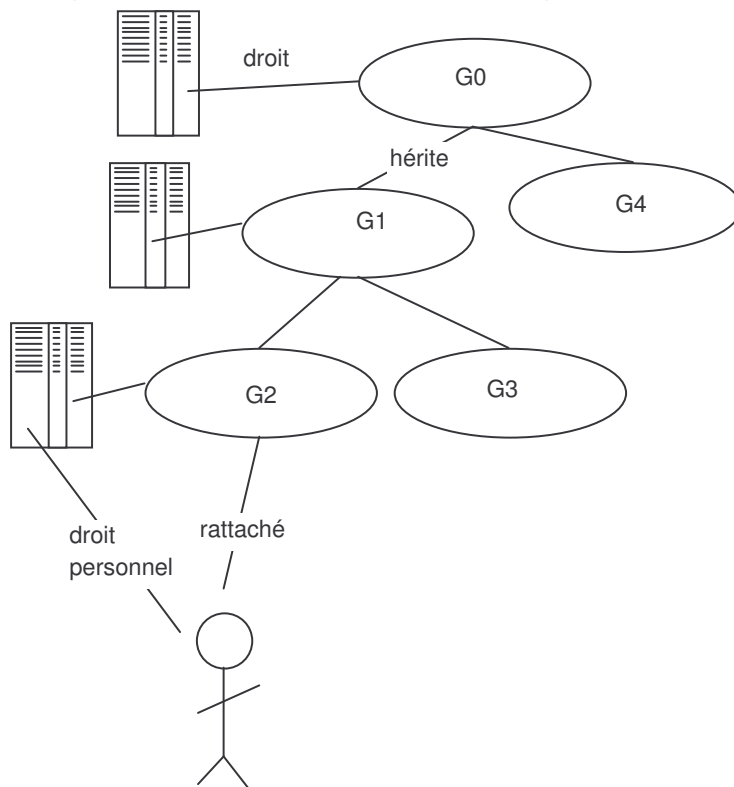
Une autre structure était donc nécessaire. Elle est décrite par un groupe d'informations identifié par **Groupe**, (avec un libellé **LibelleGroupe**). De plus un groupe peut hériter les informations d'un autre groupe **HeriteDe**. Un employé peut ainsi être rattaché aux différents groupes d'information auxquels il a droit par défaut.

Ce système étant dans un premier temps limité aux informations du SGBD (lettres, feuilles de calcul ne sont pas incluses), les informations sont donc les tables identifiées par **NomTable** et une description **Semantique** ainsi que le nombre d'entités approximatifs **NbrEntites**. Les constituants des tables sont identifiés par **NomConstituant**, un **Type** et un degré de **Securite** (0 public ... 9 = très confidentiel) est aussi donné

La description des droits par défaut est donnée par un **Droit**, que l'on attribue à un groupe sur le constituant d'une table sur une certaine période (de **Début** à **Fin**)

La description des droits personnels est donnée par un **Droit**, que l'on attribue à un groupe sur le constituant d'une table sur une certaine période (de **Début** à **Fin**). Ces droits sont ceux qu'une personne a en plus des droits donnés par son rattachement aux groupes d'information.

La figure suivante montre les liens d'héritage, la personne rattachée au groupe G2 possède par héritage les droits du groupe G0.



Domaine des constituants

Debut	date
Dept	mot (Compta Salaire, Fabric, Pers, Inform)
DeptSup	mot (Compta Salaire, Fabric, Pers, Inform)
Droit	mot (SELECT, CREATE, delete, update)
Fin	date
Groupe	mot (RessHum, Adresse, Prix, ComtaGen, ...)
HeriteDe	mot (RessHum, Adresse, Prix, ComtaGen, ...)
LibelleDept	texte
LibelleGroupe	texte
NbrEntites	entier
NomConstituant	mot (Tous_Cst, Nom, DateValeur, Piece, ...)
NomTable	mot (Employe, Ecriture, Nomemclature, ...)

Nom	texte
NumEmp	entier [1..99999]
Prenom	texte
Securite	entier [0..9]
Semantique	texte
Type	mot (integer, char, date, real)

Relations

(une clé de la relation est indiquée par un soulignement)

Departement	(<u>Dept</u> , DeptSup, LibelleDept)
Personnes	(<u>NumEmp</u> , Dept, Nom, Prenom)
Groupes	(<u>Groupe</u> , HériteDe, LibelleGroupe)
Rattacher	(<u>NumEmp</u> , <u>Groupe</u>)
Tables	(<u>NomTable</u> , Semantique, NbrEntites)
Constituants	(<u>NomTable</u> , <u>NomConstituant</u> , Type, Securite)
DroitGroupe	(<u>Groupe</u> , <u>NomTable</u> , <u>NomConstituant</u> , <u>Droit</u> , <u>Début</u> , <u>Fin</u>)
DroitPers	(<u>NumEmp</u> , <u>NomTable</u> , <u>NomConstituant</u> , <u>Droit</u> , <u>Début</u> , <u>Fin</u>)

On admet aussi pour la suite de l'énoncé qu'il existe une vue

TousLesGroupes(GrpRacine, GrpAncetre)

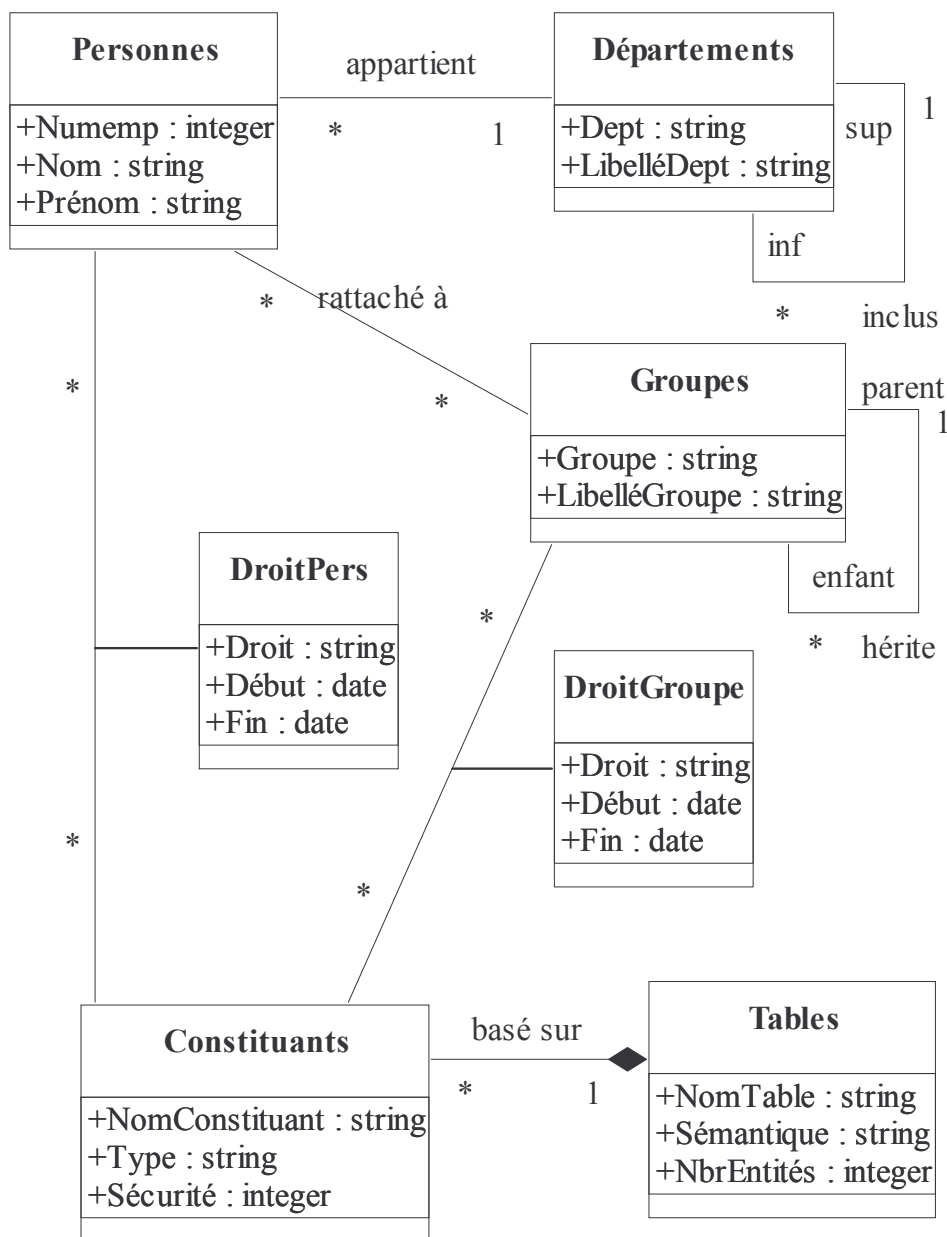
qui pour un groupe racine donne tous les ancêtres (c'est à dire tous les groupes dont il hérite) Ceci permet d'avoir explicitement la liste des groupes auxquels on a accès à travers le mécanisme d'héritage.

Dépendances fonctionnelles

(les instances des relations valident ces d.f. !)

- 1) Dept -> DeptSup, LibelleDept
- 2) NumEmp -> Dept, Nom, Prenom
- 3) Groupe -> HériteDe, LibelleGroupe
- 4) NomTable -> Semantique, NbrEntites
- 5) NomTable, NomConstituant -> Type, Sécurité
- 6) Groupe, NomTable, NomConstituant, Droit, Début -> Fin
- 7) NumEmp, NomTable, NomConstituant, Droit, Début -> Fin

Diagramme des classes



Sémantique

Remplir les tables correspondant aux instances des relations afin d'exprimer le texte qui suit:

"Jean Sérien (employé numéro 54) travaille au département "Fourn" (les fournisseurs) qui dépend de la "Compta". Il a accès au groupe "Four_CA" qui contient les informations sur les chiffres d'affaires et hérite du groupe "Four_Social", les informations sur les raisons sociales des fournisseurs. La table "Histo_Four_CA", l'historique des chiffres d'affaire contient environ 10'000 rangées. Elle est composée entre autre d'une colonne "Annee" et d'une

colonne "CA". CA est de type "real" avec un degré de sécurité 6. Année est de type integer avec un degré de sécurité 0. Le groupe "Four_CA" a le droit SELECT sur CA et Jean Sérien a le droit update sur CA depuis le 1-1-92"

Tables Type

Departement	Dept	Deptsup	LibelleDept

Groupes	Groupe	HeriteDe	LibelleGroupe

Personnes	Numemp	Dept	Nom	Prenom

Rattacher	Numemp	Groupe

Tables	NomTable	Semantique	NbrEntites

Constituants	NomTable	NomConstituant	Type	Securite

DroitGroupe	Groupe	NomTable	NomConstituant	Droit	Debut	Fin

DroitPers	Numemp	NomTable	NomConstituant	Droit	Debut	Fin

Sur la modélisation

Répondre aux questions suivantes en basant vos arguments uniquement sur la modélisation

- 1) Une personne peut-elle être associée à une table, un constituant et un droit identique pour des périodes différentes (début fin)?

- 2) En supprimant le constituant HeriteDe, peut-on encore exprimer des informations identiques dans la BD (si oui comment, sinon pourquoi)?
- 3) Explicitez les dépendances fonctionnelles existentielles à vérifier sur la modélisation (sous la forme $R[C] \sqsubseteq S[C]$)
- 4) Donnez l'ordre de création qui permet de créer la relation DroitGroupe (Groupe, NomTable, NomConstituant, Droit, Début, Fin) avec les clauses sur les contraintes d'intégrité
- 5) Donnez un exemple de ri qui pourrait exister dans ce champ d'application et qui ne peut être pris en compte lors du CREATE pour la relation DroitGroupe (donc qui ne peut figurer dans la réponse à 4)
- 6) En ajoutant la df NumEmp -> Groupe, que cela signifie-t'il et comment peut-t'on en tenir compte dans la décomposition?

Requêtes

Répondre en SQL aux questions suivantes:

- 1a) Trouver la liste des employés ayant explicitement accès au groupe d'information "Four_CA"
- 1b) Trouver la liste des employés ayant implicitement accès au groupe d'information "Four_CA"
- 2) Trouver les employés qui ont accès (SELECT,maj,...) au constituant "Salaire", à travers les droits de groupe.
- 3) Donner le degré moyen, max de sécurité du groupe "Four_CA" (sans tenir compte des constituants hérités) en fonction du degré de sécurité des constituants auxquels il a accès
- 4) Donnez les nouveaux droits du groupe "Four_CA" entre le 1-1-91 et le 1-1-92
- 5) Le nombre d'employés par département
- 6) Les groupes dont aucun autre est héritier.

Règles d'intégrité

En reprenant l'énoncé, le groupe ULTIMA à ajouter que l'héritage doit se faire de manière à préserver un ordre par rapport au degré de sécurité des groupes (défini comme le maximum du degré des constituants auxquels il a accès). Donc "Un groupe ne peut hériter que d'un groupe ayant un degré inférieur ou égal"

Donnez la portée de cette ri en remplissant le tableau de contexte suivant (mettre une croix si la case appartient à la portée)

Primitive /relation	insérer	Sup- primer	maj	maj	maj	maj	maj	maj
DroitGroupe			Groupe	NomTable	NomConstituant	Droit	Debut	Fin

Constituants			NomTable	NomConstituant	Type	Securite
Groupe			Groupe	HeriteDe	LibelleGroupe	

Donner une requête SQL qui permette de détecter les groupes qui ne satisfont pas à la règle.

21. BIGSMAC

La société BIGsmac est, depuis cinq ans, spécialisée dans la vente d'ordinateurs individuels. Elle a fait face à plusieurs tourmentes dans le passé: changement de structures du marché, chute des prix, diversification des produits et des services. Actuellement, elle veut prendre une part active sur un marché décentralisé, étendu aux clients éloignés des grandes concentrations urbaines. Pour cela, elle désire mettre au point une base de données accessible par minitel. Le client peut y trouver les services suivants:

- être conseillé sur la configuration adaptée à ses besoins
- acheter des articles au détail
- accéder aux données techniques des produits
- connaître la compatibilité entre les produits
- connaître les délais de livraison pour les articles hors stock

Après une première réunion, on a obtenu les informations qui suivent.

Un client est identifié par son **NoClient**, on lui connaît un **nom**, un **prénom**, une **adresse** et un numéro de Carte de Crédit **NoCarteCrédit** par lequel on lui débite son compte pour ses commandes. Une commande est identifiée par son **NoCmd**; on lui associe une date **dateCmd**, un **délai** qui correspond au délai maximum de livraison des articles de la commande et le prix total de la commande **prix_Cmd**. La commande se décompose en lignes, chacune correspondant à un article (quantité de 1). Une information indique si l'article est **réservé** dans le stock dans le cas où toute la commande n'est pas encore livrable. Un **prix** est indiqué pour chaque article. Un article est identifié par sa catégorie et un **NoArt**.

Pour définir certaines configurations type, BIGsmac a défini des usages (domestique, bureau, serveur, CAO, PAO, ...) pour orienter le client. **Use** est le code de l'usage et **lib_usage** son libellé. Aux usages sont associés des logiciels et des configurations. Une configuration est identifiée par un **NoConfig**; elle est décrite par le libellé **lib_config**, un **rabais** est accordé sur tous les articles de la configuration et le prix total correspond à **prix_config**. Dans une configuration peut entrer les catégories d'articles suivantes:

CatInOut: les imprimantes et scanners qui sont décrits par les paramètres techniques.

- **taille_doc** la taille du document (A4, A3)
- **précision_dpi** la précision des impressions (300, 600, ...)
- **color** la possibilité d'imprimer en couleur
- **nbr_color** le nombre de couleurs
- **vitesse** la vitesse d'impression en pages par minute

CatDisque: les disques et les moyens de stockages auxiliaires qui sont décrits par les paramètres techniques.

- **Vitesse_de transfert** en MBytes par seconde
- **accès_moyen** le temps moyen d'un accès à une information
- **capacité** la capacité de stockage
- **écriture** la possibilité d'écrire (CDROM)

CatEcran: les écrans qui sont décrits par les paramètres techniques.

- **Taille** en pouces
- **pt_haut** le nombre de points en hauteur
- **pt_larg** le nombre de points en largeur
- **color** la possibilité d'afficher en couleur
- **nbr_color** le nombre de couleurs

CatCPU: les unités centrales avec processeur qui sont décrites par les paramètres techniques.

- **processeur** son type
- **horloge** la vitesse de son horloge
- **FPU** l'existence d'une unité arithmétique
- **ethernet** l'existence d'une interface de communication
- **nb_slot** le nombre de cartes d'interface possible

CatAcc: les accessoires qui sont décrits par les paramètres techniques.

- **lib_accessoire** une description

CatLog: les logiciels qui sont décrits par les paramètres techniques.

- **logiciel** une description
- **version** le numéro de version
- **domaine** le domaine d'utilisation
- **mémoire_exigée** la mémoire recommandée par le fabricant

Une configuration associe donc un certain nombre de ces articles. Au minimum, il faut un CPU, un disque et un écran. Le client peut aussi effectuer sa propre configuration; pour cela il est aidé dans la détection des incompatibilités entre un CPU et des articles.

Pour chaque article, on constitue un historique des prix et des quantités achetées à ces conditions. Dans un intervalle de temps (**depuis jusqu'à**), on a le prix d'achat (**prix_achat**), le prix client (**prix_client**), la quantité encore en stock (**qte_stock**) et le **décali** de réapprovisionnement. Ces données permettent de fixer le prix des configurations et des commandes. On peut aussi calculer les marges et les prix moyens d'achat et de ventes sur des articles restant en stock.

Le consultant pense qu'avec ces données, il peut satisfaire les demandes principales exprimées par la société.

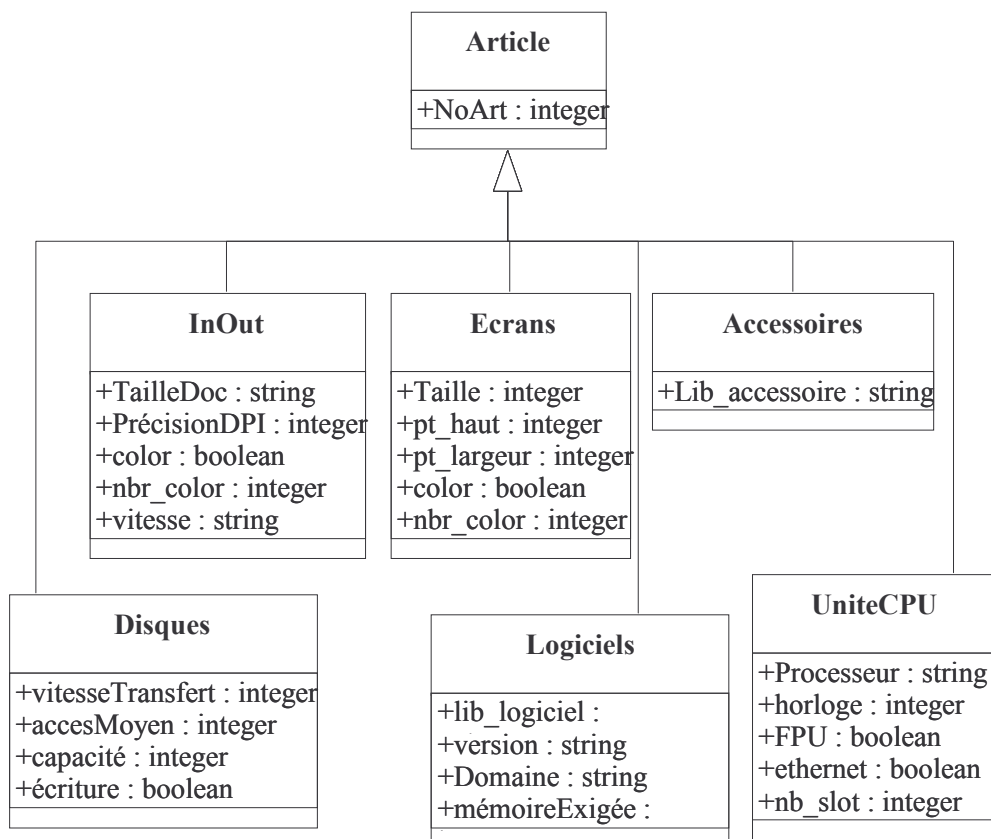
Domaine des constituants

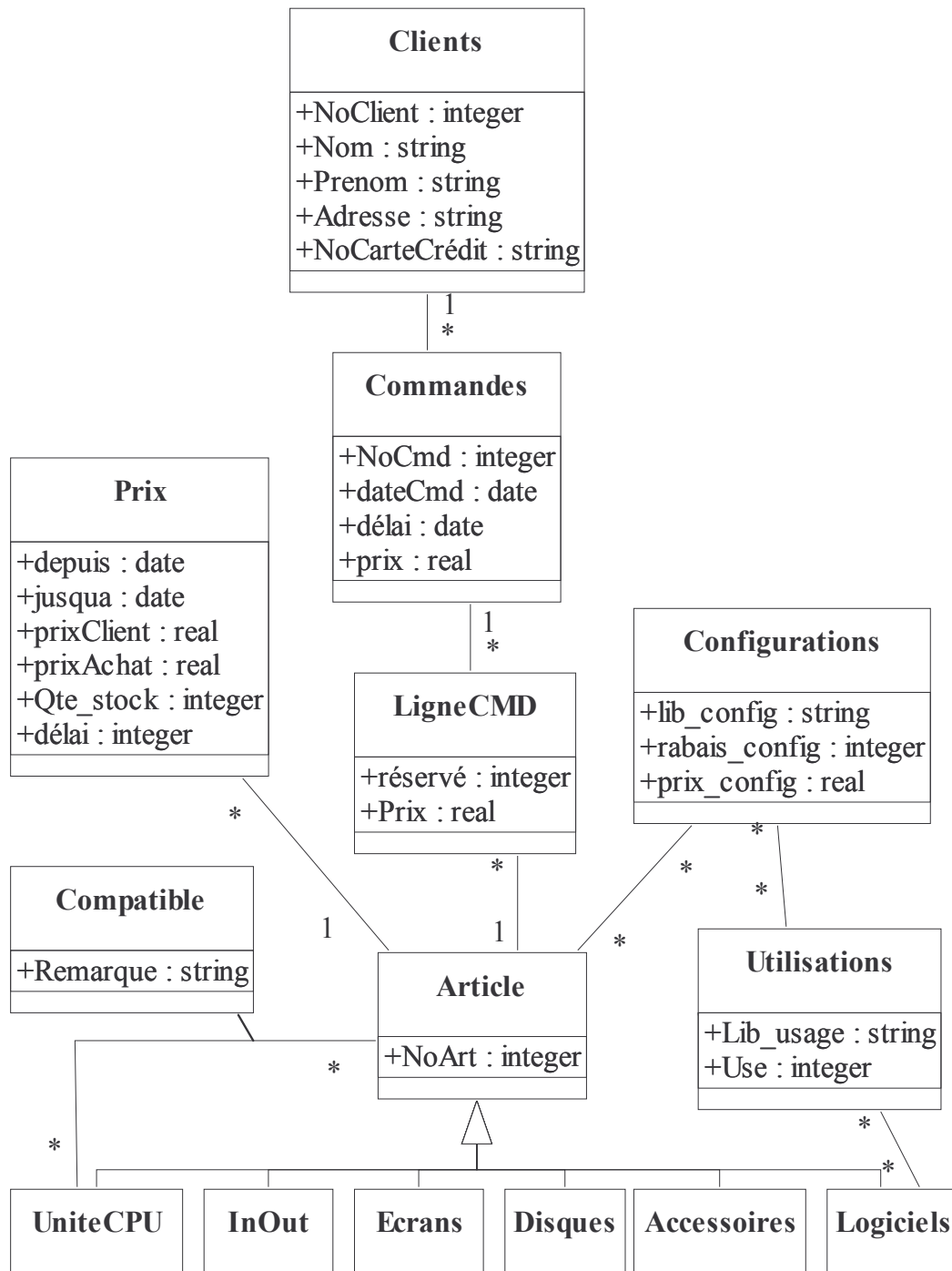
accès_moyen	nombre (en milli-secondes)
adresse	texte
capacité	entier [1..99999] (en MBytes)
Cat	mot (ACC, CPU, DSK, ECR, IO, LOG)
CatAcc	mot (ACC)
CatCPU	mot (CPU)
CatDisque	mot (DSK)
CatEcran	mot (ECR)
CatInOut	mot (IO)
CatLog	mot (LOG)
color	mot (non, oui)
dateCmd	date
délai	entier [1..52] (en semaines)
depuis	date
domaine	mot (jeux, trait_texte, tableur, sgbd, dessin,...)
écriture	mot (non, oui)
ethernet	mot (non, oui)
FPU	mot (non, oui)
horloge	nombre [10..1000] (en MHz)
jusqu'à	date
lib_accessoire	texte
lib_config	texte
lib_logiciel	texte
lib_usage	texte
mémoire_exigée	entier [1..16000] (en KBytes)
nbr_color	entier [1..24]
nb_slot	entier [1..8]
NoArt	entier [1..99999]
NoArt_cpu	(idem que NoArt)
NoCarteCrédit	texte
NoClient	entier [1..99999]
NoCmd	entier [1..99999]
NoConfig	entier [1..99999]
nom	texte
prénom	texte
précision_dpi	entier [1..2400] (en points par pouce)
prix	nombre

prix_achat	nombre
prix_client	nombre
prix_Cmd	nombre
prix_config	nombre
processeur	mot(68040, 68060, 80386, 80486, ...)
pt_haut	entier [100..2400]
pt_larg	entier [100..2400]
qte_stock	entier [0..1000]
rabais_config	nombre
remarque	texte
réservé	entier [0..1]
Taille	entier [9..21] (en pouces)
taille_doc	mot(A4, A3,)
use	mot (domestique, bureau, serveur, CAO, PAO, ...)
version	texte
vitesse	entier [1..100] (en pages par minute)
vitesse_transfert	nombre (en MBytes par seconde)

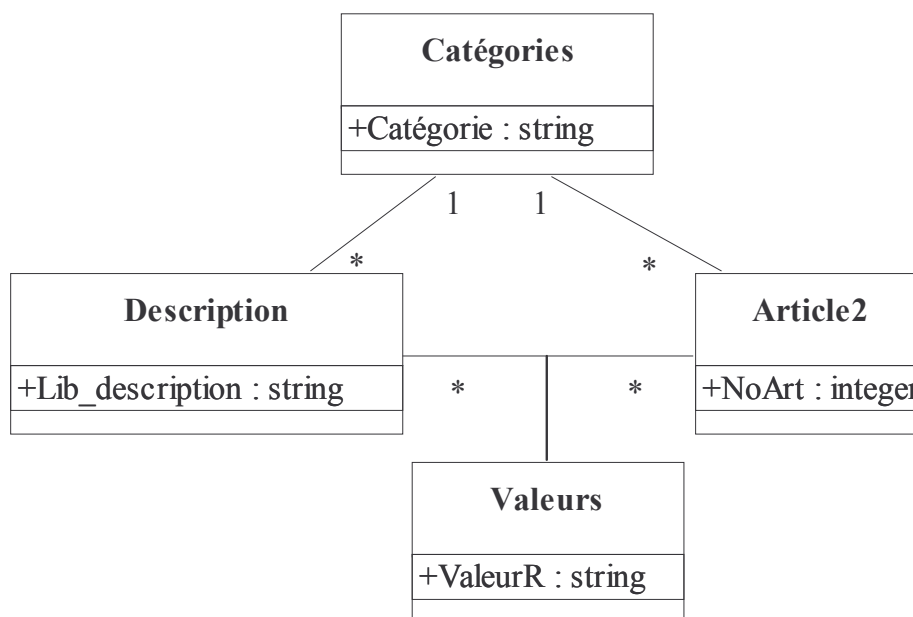
Diagramme des classes

L'évidence de ce cas est la relation d'héritage suivante :





Justement les évidences sont trompeuses et dans le cas présent, la diversité et l'évolution possible du matériel à vendre serait mieux géré par la modélisation suivante :



Relations

(une clé de la relation est indiquée par un soulignement)

Clients	(<u>NoClient</u> , nom, prénom, adresse, NoCarteCrédit)
Commande	(<u>NoCmd</u> , NoClient, dateCmd, délai, prix_Cmd)
Ligne_Cmd	(<u>NoCmd</u> , <u>Cat</u> , <u>NoArt</u> , réservé, prix)
Utilisation	(<u>use</u> , lib_usage)
Fait_pour	(<u>use</u> , <u>CatLog</u> , <u>NoArt</u>)
Conseillé	(<u>use</u> , NoConfig)
Configuration	(<u>NoConfig</u> , lib_config, rabais_config, prix_config)
InConfig	(<u>NoConfig</u> , <u>Cat</u> , <u>NoArt</u>)
InOut	(<u>CatInOut</u> , <u>NoArt</u> , taille_doc, précision_dpi, color, nbr_color, vitesse)
Disques	(<u>CatDisque</u> , <u>NoArt</u> , vitesse_transfert, accès_moyen, capacité, écriture)
Ecrans	(<u>CatEcran</u> , <u>NoArt</u> , Taille, pt_haut, pt_larg, color, nbr_color)
UniteCPU	(<u>CatCPU</u> , <u>NoArt</u> , processeur, horloge, FPU, ethernet, nb_slot)
Accessoires	(<u>CatAcc</u> , <u>NoArt</u> , lib_accessoire)
Logiciels	(<u>CatLog</u> , <u>NoArt</u> , lib_logiciel, version, domaine, mémoire_exigée)
Compatible	(<u>CatCPU</u> , <u>NoArt_cpu</u> , <u>Cat</u> , <u>NoArt</u> , remarque)
prix	(<u>Cat</u> , <u>NoArt</u> , <u>depuis</u> , jusqu'à, prix_client, prix_achat, qte_stock, délai)

Dépendances fonctionnelles

(les instances des relations valident ces d.f. !)

- 1) NoClient -> nom, prénom, adresse, NoCarteCrédit
- 2) NoCmd -> NoClient, dateCmd, délai, prix_Cmd
- 3) NoCmd, Cat, NoArt -> réservé, prix
- 4) use -> lib_usage
- 5) NoConfig -> lib_config, rabais_config, prix_config
- 6) CatInOut, NoArt -> taille_doc, précision_dpi, color, nbr_color, vitesse
- 7) CatDisque, NoArt -> vitesse_transfert, accès_moyen, capacité, écriture
- 8) CatEcran, NoArt -> Taille, pt_haut, pt_larg, color, nbr_color
- 9) CatCPU, NoArt -> processeur, horloge, FPU, ethernet, nb_slot
- 10) CatAcc, NoArt -> lib_accessoire
- 11) CatLog, NoArt -> lib_logiciel, version, domaine, mémoire_exigée
- 12) CatCPU, NoArt_cpu, Cat, NoArt -> remarque
- 13) Cat, NoArt , depuis -> jusqu'à, prix_client, prix_achat, qte_stock
- 14) Cat, NoArt -> délai

Sémantique

Remplir les tables correspondant aux instances des relations afin d'exprimer le texte qui suit:

" Hier le client no 577 Paul Nioumane (carte crédit 8798798) a commandé (no 343) une configuration "STARTER" (config. no 5) avec 10% de rabais. Celle-ci se compose de:

- un processeur 680486 à 100 MHZ avec FPU et 2 slot (art. no 100)
- un écran 16", 256 couleurs (art. no 200)
- un disque de 320 Mbyte (art. no 300)

Pour la période de 1 juin 93 au 1 septembre 93, le délai d'approvisionnement de ces articles est de 5 semaines et les prix de vente de ces articles sont les suivants:

- no 100, 1200 FRF.
- no 200, 600 FRF.
- no 300, 1000 FRF. "

Tables type

Clients	<u>NoClient</u>	nom	prénom	adresse	NoCarteCrédit

Commande	<u>NoCmd</u>	NoClient	dateCmd	délai	prix_Cmd
----------	--------------	----------	---------	-------	----------

Ligne_Cmd	<u>NoCmd</u>	<u>Cat</u>	<u>NoArt</u>	réservé	prix

Configuration	<u>NoConfig</u>	<u>Lib_config</u>	<u>rabais_config</u>	<u>prix_config</u>

InConfig	<u>NoConfig</u>	<u>Cat</u>	<u>NoArt</u>

UniteCPU	<u>CatCPU</u>	<u>NoArt</u>	processeur	horloge	FPU	ethernet	nb_slot

Ecrans	<u>CatEcran</u>	<u>NoArt</u>	Taille	pt_haut	pt_larg	color	nbr_color

Disques	<u>CatDisque</u>	<u>NoArt</u>	vitesse_transfert	accès_moyen	capacité	écriture

InOut	<u>CatInOut</u>	<u>NoArt</u>	taille_doc	precision_dpi	color	nbr_color	vitesse

Compatible	<u>CatCPU</u>	<u>NoArt_cpu</u>	<u>Cat</u>	<u>NoArt</u>	remarque

prix	<u>Cat</u>	<u>NoArt</u>	<u>depuis</u>	<u>jusqu'à</u>	<u>prix_client</u>	<u>prix_achat</u>	<u>qte_stock</u>	<u>délai</u>

Sur la modélisation

Répondre aux questions suivantes en basant vos arguments uniquement sur la modélisation

- 1) Peut-on avoir plusieurs configurations conseillées pour un même usage?
- 2) Soit la df et la relation suivantes:
Cat, NoArt -> délai

prix(Cat, NoArt, depuis, jusqu'à, prix_client, prix_achat, qte_stock, délai)

Discuter des anomalies de mise à jour. Peut-on proposer une amélioration de la décomposition ?

- 3) Si on réunit les relations InOut, Disques, Ecrans, UniteCPU et Accessoires dans une unique relation:

Matériel (Cat, NoArt, taille_doc, précision_dpi, color, nbr_color, vitesse, vitesse_transfert, accès_moyen, capacité, écriture, Taille, pt_haut, pt_larg, processeur, horloge, FPU, ethernet, nb_slot, lib_accessoire)

donner les avantages et désavantages qu'on obtient, chercher la clé de cette nouvelle relation et discuter des anomalies de mise à jour

- 4) Explicitiez toutes les dépendances d'inclusion à vérifier sur la modélisation (sous la forme $R[C] \sqsupseteq S[D]$)
- 5) Donnez l'ordre de création SQL qui permet de créer la relation Ligne_Cmd(NoCmd, Cat, NoArt, réservé, prix) avec les clauses sur toutes les contraintes d'intégrité
- 6) Imaginez une ri qui existerait dans le champ d'application telle que:
- a) une primitive de la relation Commande ferait partie de la portée de cette ri et
 - b) cette ri ne pourrait pas être prise en compte lors du CREATE de la relation Commande

Requêtes

Répondre en SQL aux questions suivantes:

- 1) Liste des articles des configurations dédiées à la CAO par ordre croissant du prix de la configuration, de la catégorie et du numéro de l'article.
- 2) Donner les prix actuels des écrans en couleur
- 3) Donner le libellé des usages pour lesquels le nombre de logiciels dépasse 20.
- 4) Etablir la liste des commandes dont le délai de livraison a été dépassé ($date1 - date2 = \text{nbr de jours}$) ou est inconnu
- 5) Donner la période où l'imprimante no 100 a atteint son prix le plus bas
- 6) Chercher les noms et prénoms des clients qui ont acheté des articles, lors d'une commande, sans acheter de CPU

Règles d'intégrité

"...Les configurations proposées doivent être compatibles..." .

Donner la portée de cette ri en remplissant le tableau de contexte suivant (mettre une croix si la case appartient à la portée)

primitive/ relation	insérer	supprimer	maj	maj	maj	maj	maj	maj	maj

Donner une requête SQL qui permette de détecter les configurations qui contiennent des incompatibilités (CPU-imprimantes, CPU-disques, etc...)

(indication: commencer par définir une vue qui associe à une configuration, son CPU et ses autres articles)

22. DUF

La société DUF (Distributeur Universel Français) est, depuis cinquante ans, spécialisée dans la distribution des livres et des revues. Elle joue le rôle d'intermédiaire entre les éditeurs qui sont membres de son groupe et les libraires. Elle a fait face à plusieurs tourmentes dans le passé: changement de structures du marché, chute des prix, diversification des services. Actuellement, elle veut transformer son outil de gestion en un outil d'aide à la décision pour faciliter le dimensionnement des "offices" (les libraires reçoivent chaque mois un ensemble de livres d'office, qu'ils payent mais qu'ils peuvent retourner au distributeur) qui sont envoyés aux libraires. Pour cela, elle désire mettre au point une base de données. Les éditeurs et les libraires pourront y trouver les services suivants:

- ventes, retours pour chaque titre, auteur
- ventes, retours par région
- ventes, retours par genre littéraire
- être un outil de gestion des stocks
- être un outil de gestion financier

Après une première réunion, on a obtenu les informations qui suivent.

Le consultant pense qu'avec ces données, il peut satisfaire les demandes principales exprimées par la société.

Les éditeurs

Les éditeurs sont identifiés par un numéro d'éditeur **NoEditeur**, on connaît pour chacun une raison Sociale **RaisonSocialeEd**, une adresse **AdresseEd** et un **solde** financier représentant le total à payer à cet éditeur. Un éditeur produit des livres qui sont identifiés par un numéro de référence **NoRef**, on connaît pour chacun son **titre**, son auteur, sa date de parution **datepublication**, son prix de vente en librairie **prixvente** et son prix de distribution **prixdistribution**. Un livre peut être associé à plusieurs genres (numéro de genre **Noggenre**). Les auteurs sont identifiés par un numéro d'auteur **NoAuteur**, on connaît pour chacun son **nom** et son **prénom**.

L'éditeur livre périodiquement des nouveaux ouvrages (ou des rééditions) pour réapprovisionner le stock de DUF. Chaque livraison est associée une date et une quantité.

Chaque éditeur possède un journal des écritures comptables qui lui sont imputables. Une écriture est associée à un éditeur, un libellé **Lib**, une date d'opération **date_op** et un montant **mnt**.

Les librairies

Les librairies sont identifiées par un numéro de librairie **Nolibrairie**, on connaît pour chacune une raison Sociale **RaisonSocialeLib**, une adresse

AdresseLib et un **solde** financier représentant le total à recevoir de cette librairie.

Une librairie effectue régulièrement des transactions avec DUF. Chaque transaction est décomposée en lignes concernant un livre pour une certaine quantité **qte** (on en déduit un **prix**) et un type de transaction **type_trans**. La transaction est effectuée à une date **dateTrans** et **TotalTrans** récapitule le total des lignes de la transaction.

Chaque librairie possède un journal des écritures comptables qui lui sont imputables. Une écriture est associée à une librairie, un libellé **Lib**, une date d'opération **date_op** et un montant **mnt**.

Pour simplifier l'étude, le consultant a fait quelques impasses sur les paiements en monnaies étrangères, les représentants chargés du contact avec les libraires, ...

Domaine des constituants

AdresseEd	texte
AdresseLib	texte
datelivraison	date
dateTrans	date
date_op	date
datepublication	date
lib	mot ('paiement','facture', ...)
libgenre	mot ('sociologie','roman','histoire','art', ...)
mnt	nombre
NoAuteur	entier [1..99999]
NoEditeur	entier [1..99999]
NoGenre	entier [1..99999]
NoLibrairie	entier [1..99999]
NoLigne	entier [1..999]
nom	mot ('DUPONT', ...)
NoRef	entier [1..99999]
NoTrans	entier [1..99999]
prénom	Texte
prix	nombre
prixdistribution	nombre
prixvente	nombre
Qte	entier [-999..999]
qte_livree	entier [-99999..99999]
RaisonSocialeEd	mot ('La renaissance', 'Kalimard...')
RaisonSocialeLib	mot ('Aux vieux livres', ...)

Region	mot ('Paris','Lyon','Canada',...)
Solde	nombre
Titre	texte
TotalTrans	nombre
type_trans	mot ('dépot','office','retour','commande',...)

Relations

(une clé de la relation est indiquée par un soulignement)

Auteur	(<u>NoAuteur</u> , nom, prénom)
Editeur	(<u>NoEditeur</u> , RaisonSocialeEd, AdresseEd, Solde)
Ecriture_Ed	(<u>NoEditeur</u> , date_op, Lib, mnt)
Ouvrage	(<u>NoRef</u> , Titre, NoEditeur, NoAuteur, datepublication, prixvente, prixdistribution)
GenreOuvrage	(<u>NoRef</u> , <u>Noggenre</u>)
Genre	(<u>Noggenre</u> , genre)
Stock	(<u>NoRef</u> , <u>datelivraison</u> , qte_livree)
Librairie	(<u>NoLibrairie</u> , RaisonSocialeLib, AdresseLib, Region, Solde)
Transaction	(<u>NoTrans</u> , NoLibrairie, dateTrans, TotalTrans)
Ligne_Trans	(<u>NoTrans</u> , <u>NoLigne</u> , NoRef, type_trans, Qte, prix)
Ecriture_Lib	(<u>NoLibrairie</u> , date_op, Lib, mnt)

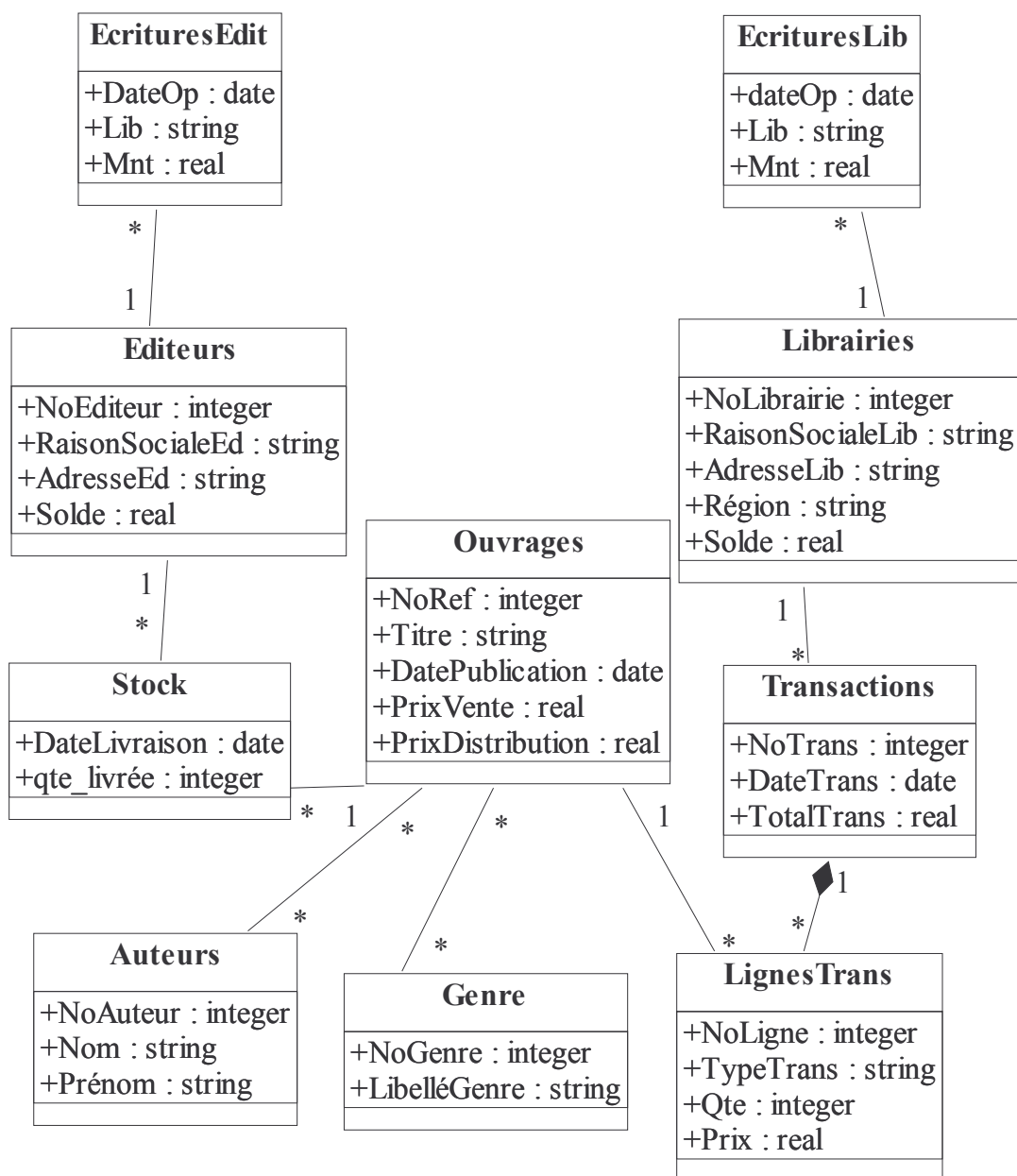
Dépendances fonctionnelles

(les instances des relations valident ces d.f. !)

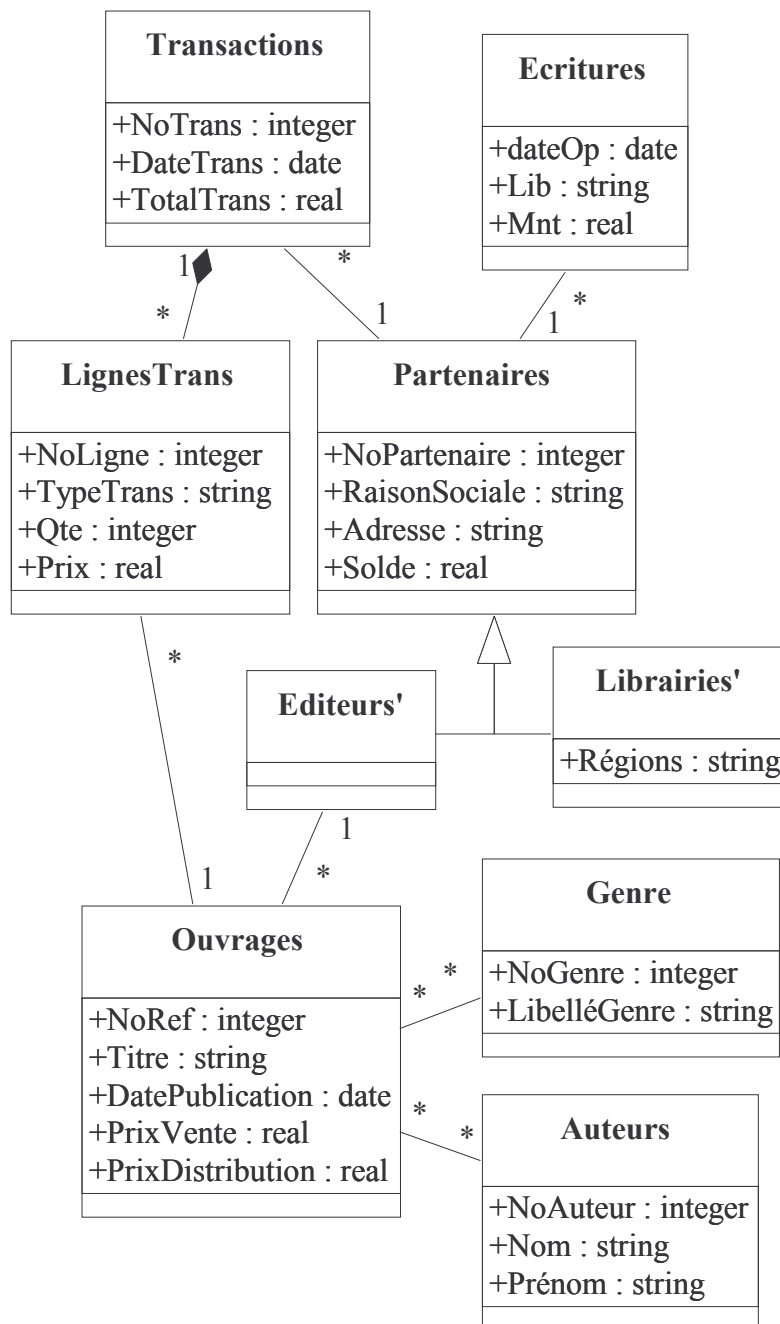
- 1) NoAuteur -> nom, prénom)
- 2) NoEditeur -> RaisonSocialeEd, AdresseEd, Solde)
- 3) NoEditeur, date_op -> mnt
- 4) NoRef -> Titre, NoEditeur, NoAuteur, datepublication, prixvente, prixdistribution
- 5) Noggenre -> genre)
- 6) NoRef, datelivraison -> qte_livree
- 7) NoLibrairie -> RaisonSocialeLib, AdresseLib, Region, Solde
- 8) NoTrans -> NoLibrairie, dateTrans, TotalTrans
- 9) NoTrans, NoLigne -> NoRef, type_trans, Qte, prix
- 10) NoLibrairie, date_op -> mnt

Diagramme des classes

Cette première modélisation présente les éditeurs et les libraires comme deux entités différentes



La modélisation suivante ne fait plus de distinction entre les éditeurs et les libraires.



Sémantique

Remplir les tables correspondant aux instances des relations afin d'exprimer le texte qui suit:

"L'éditeur NET (no 10) a livré 2000 livres de son dernier ouvrage "Java et Internet" (ref 123) de James Sun (auteur 25). L'ouvrage est classé dans les genres technologie (no 4) et informatique (no 7). Son prix de distribution est de 60 francs. L'éditeur a été créditer de 20,000 francs pour avance sur recette."

"La librairie GUTENBERG de Lyon (n0 333) à reçu son office du mois (1.7.96) pour une valeur de 12400 francs (transaction 555), on y trouvait 10 exemplaires de "Java et Internet". Cette transaction a fait l'objet d'une facture."

•

Tables type

Auteur	<u>NoAuteur</u>	nom	prénom

Editeur	<u>NoEditeur</u>	RaisonSocialeEd	AdresseEd	Solde

Ecriture_Ed	<u>NoEditeur</u>	date_op	Lib	mnt

Ouvrage	<u>NoRef</u>	Titre	NoEditeur	NoAuteur	datepublication	prixvente	prixdistribution

GenreOuvrage	<u>NoRef</u>	<u>Nogenre</u>

Genre	<u>Nogenre</u>	genre

Stock	<u>NoRef</u>	<u>datelivraison</u>	qte_livree

Librairie	<u>NoLibrairie</u>	RaisonSocialeLib	AdresseLib	Region	Solde

Transaction	<u>NoTrans</u>	NoLibrairie	dateTrans	TotalTrans

Ligne_Trans	<u>NoTrans</u>	<u>NoLigne</u>	NoRef	type_trans	Qte	prix

Ecriture_Lib	NoLibrairie	date_op	Lib	mnt

Sur la modélisation

Répondre aux questions suivantes en basant vos arguments uniquement sur la modélisation (domaine des constituants, relations et dépendances fonctionnelles).

- Un livre peut-il être écrit par plusieurs auteurs?
 - si oui - comment socker "introduction à C" écrit par A.dunix et B.Chelle?
 - si non - comment modifier le schéma des relations et des df?
- Ecriture_Lib et Ecriture_Ed sont très semblables peut-on faire une fusion et sous quelles conditions ?
- Expliciter **toutes** les dépendances d'inclusion à vérifier sur la modélisation (sous la forme R[C] \sqcap S[D])
- Donner la requête SQL qui permet de créer la relation :
 - **Ouvrage** (NoRef, Titre, NoEditeur, NoAuteur, datepublication, prixvente, prixdistribution)
 - **avec les clauses sur toutes les contraintes d'intégrité**
- Imaginer une règle d'intégrité dont le contexte serait {Auteur, Ouvrage, Editeur} mais que l'on ne peut exprimer lors de la création du schéma (pas un check, primary key, ou une foreign key ...).

Requêtes

Répondre en SQL aux questions suivantes:

(tracez les mots-clés non-utilisés si nécessaire)

- 1) Donner la liste des références par ordre croissant de la date de publication.
- 2) Donner le nom et le prénom des auteurs ayant publié chez "NET" en 1996.
- 3) Donner le nombre total d'exemplaires vendus de "Java et Internet" en juin 1996.
- 4) Donner le chiffre d'affaire des ventes par région en 1995.
- 5) Donner la liste des références ayant un retour de plus de 50% (2 livrées plus (qte positive) de 1 retournée (qte négative))

Règles d'intégrité

Pour éviter des abus sur les retours, DUF veut attribuer un % maximum de retours par genre (Genre (Noggenre, genre, max_retour))

Donner la portée de cette ri en remplissant le tableau de contexte suivant (mettre une croix si la case appartient à la portée)

primitive/ relation	insérer	supprimer	maj	maj	maj	maj	maj	maj

Ecrire une vue permettant de retrouver les anomalies sur % maximum de retour.

23. FAME

Énoncé

"Fame²²" est une école célèbre qui forme de jeunes danseurs, musiciens et comédiens. Réputée pour son sérieux et son niveau élevé, elle attire les jeunes du monde entier. Grâce au dynamisme de la directrice de l'école, "Fame" n'a cessé de s'agrandir et de se développer ces dernières années: augmentation du nombre des salles, diversification des cours, renouvellement du matériel, augmentation du nombre d'étudiants et d'enseignants. Victime de ce succès, la directrice de l'école a de la peine à gérer toutes ces informations. Des amis lui conseillent d'utiliser un système de gestion de bases de données relationnel pour stocker et manipuler les données de l'école. Elle fait donc appel à des informaticiens de gestion et leur décrit le champ d'application suivant au cours d'une réunion:

L'école offre trois filières: danse, musique et théâtre. Elle décerne des diplômes dans chacune de ces filières (p.ex: "danse classique", "art dramatique", etc...). Un diplôme est identifié par un sigle (**SigleDip**) et il a un libellé (**LibDip**). Il est rattaché à une filière (**Filière**). Son obtention nécessite un nombre minimum de crédits (**NbCrédits**).

Le plan d'études d'un diplôme est composé d'un ensemble de cours. Un cours est identifié par un numéro (**NoCours**). Il possède un titre (**Titre**) et il est rattaché à une filière (p.ex: le cours de "ballet" est rattaché à la filière "danse"). Lorsqu'un cours est dans le plan d'études d'un diplôme, il est obligatoire ou optionnel (**Type**) et il est pondéré par un nombre de crédits (**Crédits**). Un cours d'une filière peut figurer dans le plan d'études d'un diplôme rattaché à une autre filière (p.ex: le cours "ballet" est un cours à option pour le diplôme "art dramatique" de la filière "théâtre").

L'école comporte vingt salles. Chaque salle est identifiée par un numéro (**NoSalle**), décrite par un libellé (**LibelléSalle**) et contient du matériel (**Equipement**). Un cours est donné par un enseignant dans une salle de l'école; l'enseignant et la salle peuvent changer d'une année à une autre. Chaque enseignant est identifié par un numéro (**NoEnseignant**). On connaît son nom (**Nom**), son prénom (**Prénom**), sa nationalité (**Pays**) et son adresse (**AdresseE**).

Un étudiant est identifié par un numéro d'immatriculation (**NoImmat**). Il a un nom (**Nom**), un prénom (**Prénom**), une nationalité (**Pays**) et une adresse (**Adresse**). Il s'inscrit en une année (**AnnéeIns**) à un diplôme et l'obtient plus tard (**AnnéeObt**) avec une mention (**Mention**). Un étudiant ne peut pas

²² L'énoncé de base de ce travail est due à Lina Aljadir (et est un film connu des années 80)

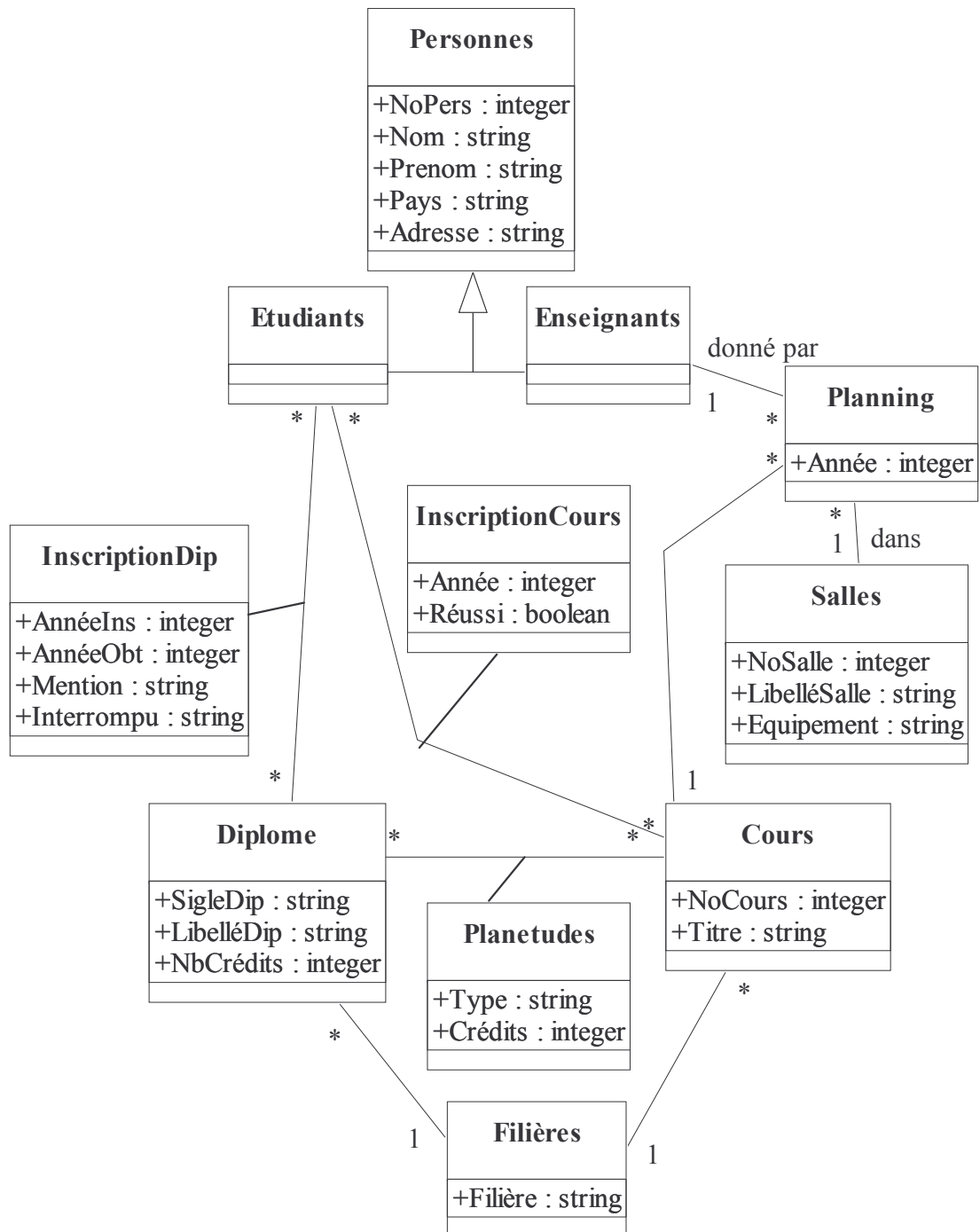
effectuer deux diplômes en même temps. Par contre, il peut commencer un deuxième diplôme après avoir terminé le premier. Il se peut qu'un étudiant n'obtienne pas son diplôme (**Interrompu**) en raison d'un départ volontaire ou d'une exclusion.

Pour obtenir son diplôme, un étudiant doit suivre tous les cours obligatoires du plan d'études correspondant et un certain nombre de cours à option pour totaliser un nombre de crédits égal ou supérieur à celui requis par son diplôme. Un étudiant obtient les crédits d'un cours uniquement s'il réussit l'examen de ce cours en fin d'année (**Réussite**). S'il échoue, l'étudiant peut se réinscrire et passer l'examen l'année suivante. S'il échoue à la deuxième tentative, il est exclu de l'école.

Domaine des constituants

Adresse	texte
Année	entier [1950..2200]
AnnéeIns	entier [1950..2200]
AnnéeObt	entier [1950..2200]
Crédits	entier [5..10]
Equipement	texte
Filière	mot (danse, musique, théâtre)
Interrompu	mot (départ, exclusion)
LibelléDip	texte
LibelléSalle	texte
Mention	mot (sans_mention, assez_bien, bien, très_bien)
NbCrédits	entier [60..100]
NoCours	entier [1..99]
NoEnseignant	entier [1..99]
NoImmat	texte
Nom	mot
NoSalle	entier [1..20]
Pays	mot
Prénom	mot
Réussi	mot (oui, non)
SigleDip	mot
Titre	texte
Type	mot (obligatoire, optionnel)

Diagramme de classes



Relations

(une clé de la relation est indiquée par un soulignement)

Diplôme (SigleDip, LibelléDip, Filière, NbCrédits)

Cours(NoCours, Titre, Filière)

PlanEtudes(SigleDip, NoCours, Type, Crédits)

Salle(NoSalle, LibelléSalle, Equipement)

Enseignant (NoEnseignant, NomE, PrénomE, PaysE, AdresseE)

Horaire (NoCours, AnnéeH, NoEnseignant, NoSalle)

Etudiant (Nolmmat, Nom, Prénom, Pays, Adresse)

InscriptionDip(Nolmmat, AnnéeIns, SigleDip, AnnéeObt, Mention, Interrompu)

InscriptionCours(Nolmmat, NoCours, Année, Réussi)

Dépendances fonctionnelles

(les instances des relations valident ces d.f. !)

1) SigleDip -> LibelléDip, Filière, NbCrédits

2) NoCours -> Titre, Filière

3) SigleDip, NoCours -> Type, Crédits

4) NoSalle -> LibelléSalle, Equipement

5) NoEnseignant -> NomE, PrénomE, PaysE, AdresseE

6) NoCours, AnnéeH -> NoEnseignant, NoSalle

7) Nolmmat -> Nom, Prénom, Pays, Adresse

8) Nolmmat, AnnéeIns -> SigleDip, AnnéeObt, Mention, Interrompu

9) Nolmmat, SigleDip -> AnnéeIns, AnnéeObt

10) Nolmmat, NoCours, Année -> Réussi

Sémantique

Remplir les tables correspondant aux instances des relations afin d'exprimer le texte qui suit:

Dans la filière "danse" figure le diplôme "danse contemporaine". Il a le sigle DCL et nécessite 110 crédits.

Son plan d'études contient les cours obligatoires "ballet 1" (no 54), "ballet 2" (no 52), "musique contemporaine 1" (no 24), et le cours à option "art moderne" (no 10). Les deux premiers sont rattachés à la filière "danse", tandis que le troisième et le quatrième sont dans la filière "musique" et "théâtre" respectivement.

Le cours "ballet 1" est donné cette année par Mme Julia Durand (no 7) à la salle de danse (no 15) qui est équipée d'un lecteur CD.

L'étudiante japonaise Sue Suzuki (no 99-11), qui s'est inscrite au diplôme "danse contemporaine" en 1999, suit cette année les cours "ballet 2" et "art moderne".

Diplôme	<u>SigleDip</u>	LibelléDip	Filière	NbCrédits

Cours	<u>NoCours</u>	Titre	Filière

PlanEtudes	<u>SigleDip</u>	<u>NoCours</u>	Type	Crédits

Salle	<u>NoSalle</u>	LibelléSalle	Equipement

Enseignant	<u>NoEnseignant</u>	NomE	PrénomE	PaysE	AdresseE

Horaire	<u>NoCours</u>	<u>AnnéeH</u>	NoEnseignant	NoSalle

Etudiant	<u>NoImmat</u>	Nom	Prénom	Pays	Adresse

InscriptionDip	<u>NoImmat</u>	<u>AnnéeIns</u>	SigleDip	AnnéeObt	Mention	Interrompu

InscriptionCours	<u>NoImmat</u>	<u>NoCours</u>	<u>Année</u>	Réussi

Sur la modélisation

Répondre aux questions suivantes en basant vos arguments **uniquement** sur la modélisation (domaine des constituants, relations et dépendances fonctionnelles).

- Si toutes les informations étaient mises dans une seule relation RU, quelle(s) clé(s) aurait cette relation (la relation universelle)
- Est-il possible d'obtenir la relation RU, au moyen des relations du schéma par composition.?
- Expliciter **toutes** les dépendances d'inclusion à vérifier sur la modélisation (sous la forme $R[C] \subseteq S[D]$)
- Donner la requête SQL qui permet de créer la relation

- Horaire (NoCours, AnnéeH, NoEnseignant, NoSalle)
avec les clauses sur toutes les contraintes d'intégrité
- Imaginer une règle d'intégrité dont le contexte serait {Horaire, Cours} qui ne soit pas une dépendance d'inclusion.

Répondre en SQL aux questions suivantes:

- Donner la liste des diplômes par ordre croissant du nombre de crédits.
- Donner le nom et le prénom des étudiants ayant obtenu un diplôme en 1994.
- Donner le titre, la filière et le crédit des cours optionnels du diplôme "danse classique".
- Donner le nombre d'étudiants inscrits en 95 pour chaque cours.
- Donner le titre des cours qui ne sont obligatoires dans aucune filière.

Vues

Ecrire une requête SQL qui permet de créer la vue SituationActuelle. Cette vue donne pour un étudiant inscrit actuellement à un diplôme le nombre de crédits qu'il a totalisés jusqu'à présent pour ce diplôme.

Règles d'intégrité

"Un étudiant ne peut obtenir son diplôme que si la somme des crédits des cours qu'il a réussis est supérieure ou égale à celle demandée par son diplôme."

- Donner la portée de cette ri en remplissant le tableau de contexte suivant (mettre une croix si la case appartient à la portée)

primitive/ relation	insérer	supprimer	maj	maj	maj	maj	maj	maj

- Citez et expliquez quelles sont les opérations qui pourraient paraître particulièrement injustes et cruelles aux yeux des élèves ayant réussi un cours et que devrait s'interdire la direction de l'école ?
- Ecrire un trigger correspondant à une croix du tableau de contexte (réutiliser la vue situationActuelle!).

24. ELECTA

La société de sondage ELECTA est spécialisée dans l'étude des résultats concernant des votations nationales. On lui a commandé une étude sur les dernières votations françaises.

ELECTA dans une phase de nouveaux développements informatiques a entrepris l'étude d'une base de données permettant de gérer les informations nécessaires à ces analyses.

Le texte qui suit est une description du champ d'application tel qu'il apparaît à la suite d'une étude avec les analystes (les mots en style **gras** sont les constituants qui sont retenus dans la modélisation)

Extrait du rapport

La France est découpée en départements (**LibDept**) ayant un numéro (**Dept**). Les départements sont découpés en circonscriptions (**LibCirc**) ayant un numéro (**Circ**). Les bureaux de votes (**BurVot**) sont attachés aux circonscriptions.

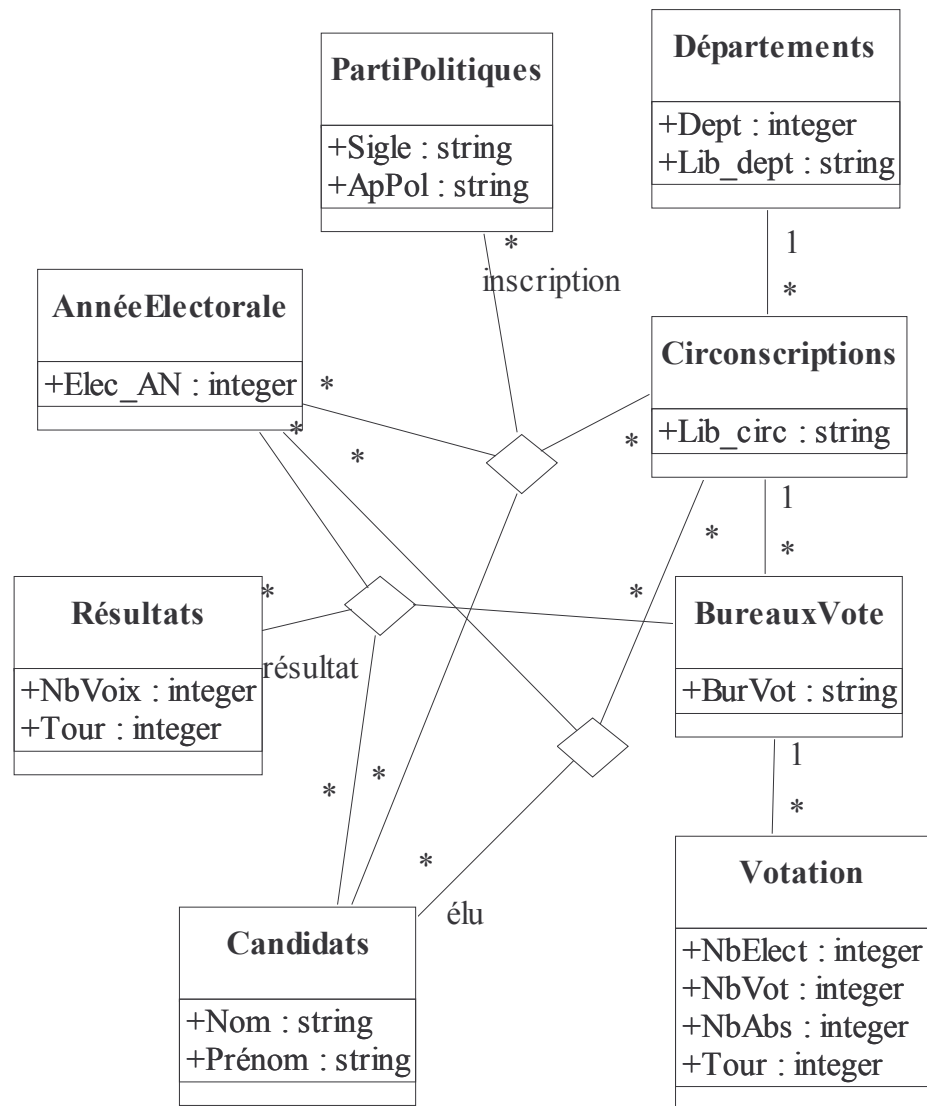
Le monde politique est découpé en différentes appellations (**ApPol**) ayant chacune un sigle (**Sig**). Les candidats (**Cand**) ont un nom (**Nom**) et un Prénom (**Prénom**). Une votation est identifiée par son année (**ElecAn**), elle a lieu en deux tours (**Tour**). On compte le nombre d'électeurs (**NbElect**), le nombre de votants (**NbVot**), le nombre d'abstentions (**NbAbs**) et le nombre de voix pour chaque candidat (**Nbvoix**). Par circonscription, un seul candidat est élu (**CandElu**)

Domaine des constituants

ApPol	texte
BurVot	entier
Cand	entier
CandElu	entier
Circ	entier
Dept	entier [1..95]
ElecAn	entier [93, 97, ...]
LibCirc	texte
LibDept	texte
NbAbs	entier
NbElect	entier
Nbvoix	entier
NbVot	entier
Nom	mot (Mignon, Durand, ...)

Prénom mot (jean, paul, anne, ...)
 Sig mot (RPR, UDF, PS, PPM, ...)
 Tour entier[1..2]

Diagramme de classes



Qui dit que la démocratie c'est simple !

Relations

BureauVote (ElecAn, Tour, BurVot, Circ, NbElect, NbVot, NbAbs)
 Circonscription (Circ, LibCirc, Dept, LibDept)
 Politique (Sig, ApPol)
 Candidat (Cand, Nom, Prénom)
 Inscription (ElecAn, Cand, Sig, Circ)

ResultatTour (ElecAn, Tour, BurVot, Cand, Nbvoix)
 ResultatElec (ElecAn, Circ, CandElu)
 Z (CandElu, Cand)

Dépendances fonctionnelles

1. ElecAn, Tour, BurVot -> NbElect, NbVot, NbAbs
2. BurVot -> Circ
3. Circ -> LibCirc, Dept
4. Dept -> LibDept
5. Sig -> ApPol
6. Cand -> Nom, Prénom
7. ElecAn, Cand -> Sig, Circ
8. ElecAn, Tour, BurVot, Cand -> Nbvoix
9. ElecAn, Circ -> CandElu
10. CandElu -> Cand

Sur la modélisation

1)

- a) Expliquez ce que signifie **Nbvoix** dans la relation **ResultatTour**, en utilisant les df pour supporter votre explication.
- b) Le calcul $100 * \text{Nbvoix} / \text{NbElect}$, pour une circonscription et un résultat de tour, représente-t'il le pourcentage de voix obtenu par ce candidat pour cette circonscription ?

2) Déterminez si la décomposition est totale

	A P P o l	B u r V o t	C a n d	C a n d E l u	C i r c	D e p t	E l e c A n	L i b C i r c	L i b D e p t	N b A b s	N b v o i x	N b V o t	N o m	P r é n o m	S i g	T o u r	N b E l e c t
BureauVote																	
Circonscrip.																	
Politique																	
Candidat																	
Inscription																	
ResultatTour																	
ResultatElec																	
Z																	

3) Donnez le chemin de composition (pour retrouver la relation universelle) auquel vous êtes arrivé

25. CONFER

Enoncé

CONFER est une association qui organise des conférences dans différents domaines, pour le compte de différents groupes de recherche. CONFER organise toute la conférence pour ces groupes depuis la distribution des « appels aux contributions » jusqu'à la location des salles. Afin de rester compétitif, le conseil d'administration de CONFER s'est interrogé sur l'opportunité de réaliser l'organisation des conférences entièrement avec les outils d'Internet.

A cette fin, ils ont engagé une procédure d'analyse de l'existant. Et ils se sont plus particulièrement concentrés sur la soumission des papiers.

Ils ont trouvé que les acteurs de cette activité "soumettre des papiers" étaient:

- les auteurs (ceux qui soumettent les papiers)
- les référés (ceux qui commentent et jugent les papiers)

Le scénario retenu est généralement le suivant:

- L'auteur reçoit un appel à soumettre un papier à une conférence (une publicité)
- L'auteur envoie une lettre d'intention de communiquer un papier
- Le comité d'organisation lui envoie des informations sur la forme de la communication (nbr de mots, format, ...)
- L'auteur envoie son papier en 4 exemplaires
- Après la date de soumission, le comité d'organisation se réunit et détermine sur la base d'une liste de référés, l'attribution des papiers aux référés.
- On envoie à chaque référé un ensemble de papiers, avec une fiche d'évaluation pour chaque papier.
- Chaque référé renvoie les fiches d'évaluation remplies avec d'éventuelles annotations sur les papiers
- Le comité scientifique se réunit pour sélectionner les papiers pour la conférence. Chaque papier est soit accepté soit refusé.
- Le comité scientifique communique son travail au comité d'organisation.
- Le comité d'organisation renvoie aux auteurs les fiches d'évaluation et les papiers annotés par les référés ainsi que la décision d'acceptation ou de refus
- Les auteurs acceptés renvoient une copie définitive de leur papier pour l'impression des actes de la conférence.

...

Questions

- 1) Décrivez ce scénario avec un graphe utilisant les symboles des « use-case »
- 2) A partir du graphe, définissez la responsabilité du comité scientifique ainsi que les objets avec lesquels il travaille.
- 3) Décrivez comment avec les outils Internet (Web, e-mail, ...) et les bases de données, il est peut-être possible de satisfaire CONFER.

26. 39,2 Le matin

Enoncé

Monsieur l'administrateur se trouvait assis derrière son bureau, un observateur extérieur aurait pu penser qu'il rêvait. En fait il pensait à la rédaction d'une étude qu'il devait envoyer la semaine prochaine aux "moyens et coordinations de l'informatique administrative" qui supervisait l'ensemble des projets informatiques.

L'administration de cette école de commerce avec ses deux mille élèves, ses cinq cents enseignants n'allait pas toujours toute seule, car les moyens en personnel étaient limités à quatre secrétaires. La seule échappatoire semblait être l'informatisation d'un certain nombre de tâches répétitives.

Le point le plus chaud de l'année était la rentrée et sa préparation. Une fois l'horaire établi, tâche qui consistait à satisfaire les contraintes d'horaires des enseignants et celles des cours du plan d'étude de chaque degré, il fallait établir deux types de documents :

- le plan des cours par classe, un pour chaque élève;
- l'horaire de l'enseignant.

Il se pencha pour examiner les documents, ils étaient identiques dans leur présentation, sous forme de grille horaire :

- Les jours étaient (LU, MA, ME, VE, SA).
- Les tranches horaire
 - 1 : 08h.00 - 08h.45
 - 2 : 08h.50 - 09h.35
 - 3 : 10h.00 - 10h.45
 - 4 : 10h.50 - 11h.35
 - 5 : 14h.00 - 14h.45
 - 6 : 14h.50 - 15h.35
 - 7 : 15h.50 - 16h.35
 - 8 : 16h.50 - 17h.35.
- Le nom des enseignants figurait sur certains documents.
- La classe était notée par un chiffre désignant le degré et une lettre (ex. 3H). Les classes avaient en moyenne 20 élèves.
- Les disciplines (souvent abrégées) : droit, comptabilité, français, anglais, dactylographie.

Si ce système informatique pouvait répondre à différents types d'interrogations, il y avait une activité qu'il fallait qu'il assume, celle dont s'occupait Mme Xeres; il s'agissait du service des remplacements. En effet, chaque jour il fallait veiller à ce que tous les cours soient donnés même en

cas de maladie d'un professeur. Et ce n'était qu'en dernier recours que l'on donnait congé aux élèves.

Cette activité l'inquiétait un peu, car d'une part Madame Xeres allait partir à la retraite et que d'autre part elle exigeait que la personne qui s'en occupe arrive avant tout le monde à 7 heures pour prendre les messages (sur le répondeur téléphonique) des maîtres qui étaient tombés malades depuis la veille. Et parmi les autres secrétaires de l'école aucune n'avait montré d'intérêt pour cette tâche qui n'était pas très complexe, mais qui demandait de l'organisation. Une fois il en avait parlé avec Mme Xeres qui lui avait déclaré :

"Voilà comment je procède :

- Le matin j'arrive, j'écoute toute la bande du répondeur et je note au fur et à mesure le nom des enseignants qui seront absents et la durée supposée de leur absence, si elle est connue.
- Ensuite je vais chercher la fiche de chaque enseignant et je repère lesquels ont des heures d'enseignement cette journée. Bien entendu, il faut se préoccuper des heures de la matinée en premier lieu.
- Après je vais consulter le fichier des remplaçants. Il y en a 200 environ, mais la difficulté est de trouver ceux qui sont disponibles pour la tranche horaire considérée car chacun suit des cours à l'Université et ils ne sont pas toujours disponibles. De plus il faut qu'ils soient capables d'enseigner la matière. Heureusement que j'ai une bonne mémoire car ce fichier n'est pas pratique; il est classé par ordre alphabétique et ne se prête pas à ce type de recherche (voir fiche en annexe).
- Ensuite, je téléphone au remplaçant pour vérifier s'il lui est possible de donner ce cours et je lui indique l'heure des cours qu'il doit remplacer, le degré de la classe et la discipline. Je lui donne aussi le nom du professeur et son numéro de téléphone pour qu'il puisse prendre des renseignements sur le programme à suivre".
- Souvent je dois téléphoner à plusieurs remplaçants car ils ne sont pas chez eux ou ils effectuent déjà des remplacements pour d'autres écoles.
- Plus tard dans la journée, je recherche des remplaçants satisfaisant les mêmes critères mais sur une plus longue durée si l'absence de l'enseignant doit se prolonger."

Oui, une "bonne mémoire" se dit l'administrateur en examinant une fiche type d'horaire de classe. Le nombre de professeurs absents était relativement faible : de 3%, mais en hiver ce chiffre pouvait monter jusqu'à 8%. En résumé, ce qu'il lui fallait, c'était un système permettant de se substituer à ce bac de cartes de remplaçants qui était très difficile à utiliser.

Horaire de classe					
Numéro de la classe : 3H					
Tranche Horaire	LU	MA	ME	VE	SA
1	Dactylo	Droit	Economie	Dactylo	Math
2	Français	Droit	Anglais	Français	Math
3	Français		Anglais	Français	Economie
4			Géographie		Economie
5	Anglais	Histoire	Compta	Allemand	
6	Anglais	Compta	Compta	Allemand	
7	Info	Compta	Sténo	Gestion	
8	Info		Sténo	Gestion	

figure 1 : fiche horaire de classe

L'administrateur commença à rédiger sa demande et son analyse.

Nous vous demandons d'effectuer l'analyse que va faire cet administrateur. Il s'agit d'analyser le champ d'application, en particulier les différents acteurs et intervenants et leurs activités, de décrire les données qui doivent être stockées, d'établir une modélisation possible de celles-ci (en utilisant UML-les cas d'utilisation, les classes et les objets) , d'envisager les architectures informatiques possibles pour supporter et faciliter ces activités.

Horaire - Professeur					
Nom: Hugo		Prénom: Victor			
Adresse: 15, avenue de France		Num.Tel. : 15.15.15			
Tranche Horaire	LU	MA	ME	VE	SA
1					
2	Français/3H			Français/3H	Français/3G
3	Français/3H			Français/3H	Français/3G
4				Histoire/3G	
5		Histoire/3H			
6		Français/3G			
7		Français/3G			
8					

figure 2: fiche horaire de professeur

Horaire - Professeur

Nom: Prénom:

Adresse: Num.Tel. :

Année immatriculation : Licence :

Faculté:

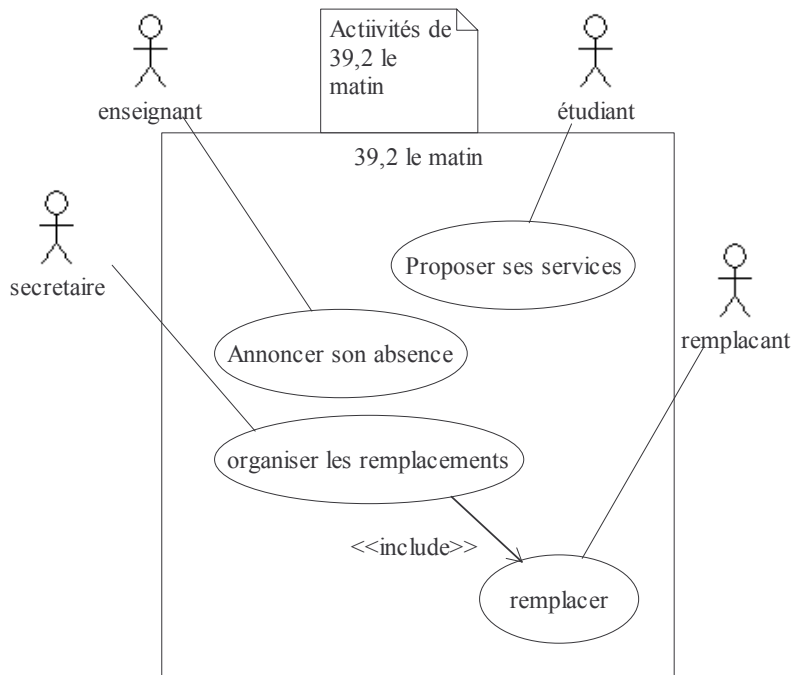
Disciplines possibles : 1)
 2)
 3)
 4)

Table de disponibilité : (mettre une croix si disponible)

Tranche Horaire	LU	MA	ME	VE	SA
1					
2					
3					
4					
5					
6					
7					
8					

figure 3: fiche du remplaçant

Diagramme des cas d'utilisation



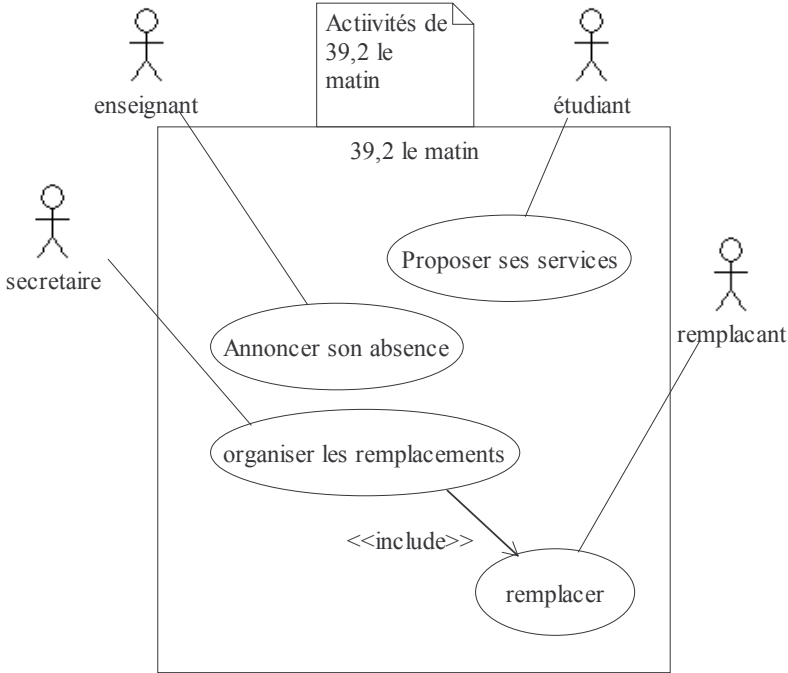


Diagramme des classes

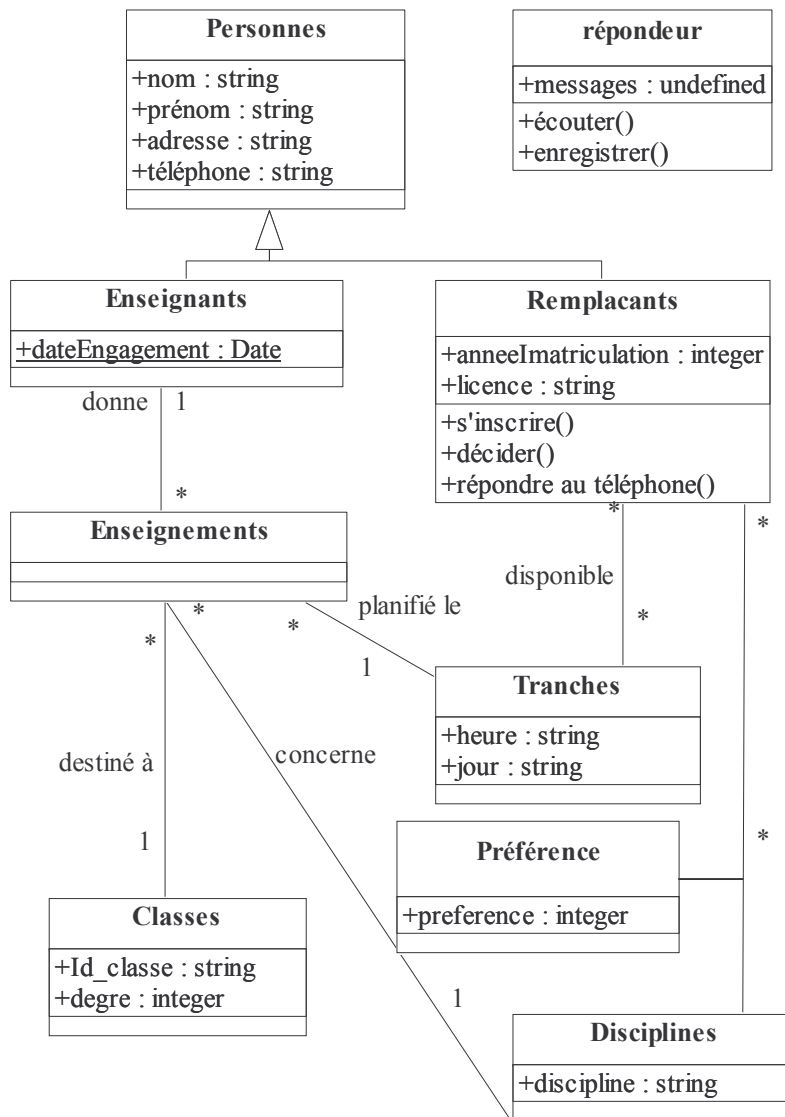


figure: 39,2 le matin

27. QCM lighth

Une petite révolution secoue l'Agence Nationale pour la Sécurité Routière (ANSR) depuis quelques mois, l'ANSR a décidé d'informatiser la génération des examens théoriques du code de la route. Les examens, déjà sous la forme de Questionnaires à Choix Multiples (QCM), seront entièrement informatisés. Une première étape consiste à :

- entrer dans une base de données, l'ensemble des questions d'examen par domaine;
- générer automatiquement des QCM;
- installer des salles d'examen informatisées.

La seconde étape prévoit :

- l'étude statistique des réponses;
- l'édition d'un logiciel grand public (accompagnant le code de la route);
- une étude sur la pédagogie.

Une pré-étude a établi les informations qui suivent :

L'enseignement est réparti en différentes catégories (CODE, PRIORITE, SECOURISME, ...) identifiées par `Id_Domaine`. A chaque domaine, on associe un libellé `Lib_Domaine`. Selon l'importance du domaine, un nombre minimal `Nbr_Min_questions` de questions est exigé pour chaque domaine.

Chaque domaine est couvert par un ensemble de questions possibles (Un QCM est un sous-ensemble de ces questions). Une question est identifiée par un numéro `Id_Question`. L'énoncé de la question est un texte `Lib_Question`. Une question est associée à un seul domaine. Un renvoi `A_revoir` précise la ou les pages qui concernent cette question. Un niveau de difficulté `Niveau_difficulté` de 1 à 10 évalue chaque question.

A chaque question est associé un ensemble de réponses possibles. Une seule réponse est correctE. Une réponse est identifiée `Id_Reponse` par A, B, C, ... (une des possibilités du QCM). L'énoncé de la réponse est un texte `Lib_Reponse`. Une réponse est associée à une seule question. Pour chaque réponse, on évalue si elle est juste (oui/non).

Les informations décrites ci-dessus permettent de décrire, pour chaque domaine, un ensemble de questions et leurs réponses. A partir de ces informations, il est possible de générer des QCM.

Un QCM est identifié par un numéro `Id_QCM`. On lui associe la date de l'examen `Date_examen` pour lequel il a été créé. Les questions du QCM sont numérotées de 1 à 100 `No_Question_QCM`. Chaque question du QCM est associée à une seule question possible d'un des domaines enseignés.

Un examen est l'ensemble des réponses apportées par une personne à un QCM. Donc chaque QCM fait l'objet d'un ensemble d'examens.

Une personne est identifiée par numéro Id_Personne. Une personne porte un prénom et un nom. Un examen est rempli par une seule personne et porte sur un seul QCM. Pour chaque question, de l'examen, on conserve les réponses de la personne. Ces dernières permettent de donner un résultat à l'examen resultat_obtenu.

Domaine des constituants

A_revoir	texte
Date_examen	date
Id_Domaine	mot (CODE, PRIORITE, SECOURISME, ...)
Id_personne	entier [1..9999999]
Id_QCM	entier [1..9999]
Id_Question	entier [1..9999]
Id_Reponse	mot (A, B,C)
Juste	mot (oui, non)
Lib_Domaine	texte
Lib_Question	texte
Lib_Réponse	texte
Nbr_Min_questions	entier [3..10]
Niveau_difficulté	entier [1..10]
nom	texte
No_Question_QCM	entier [1..100]
prenom	texte
resultat_obtenu	entier [0..100]

Relations

Domaines_Enseignés (Id_Domaine, Lib_Domaine, Nbr_Min_questions)

Questions_Possibles(Id_Question, Lib_Question, Id_Domaine, Niveau_difficulté, A_revoir)

Reponses_Possibles(Id_Question,Id_Reponse, Lib_Réponse, Juste)

QCM(Id_QCM, Date_examen)

Question_QCM(Id_QCM, No_Question_QCM, Id_Question)

Personnes(Id_personne, nom, prenom)

Examen(Id_personne, Id_QCM, resultat_obtenu)

Reponses_Examen(Id_personne, Id_QCM, No_Question_QCM, Id_Reponse)

Dépendances fonctionnelles

- 1) Id_Domaine -> Lib_Domaine, Nbr_questions
- 2) Id_Question, -> Lib_Question, Id_Domaine, Niveau_difficulté, A_revoir
- 3) Id_Question, Id_Reponse -> Lib_Réponse, Juste
- 4) Id_QCM -> Date_examen
- 5) Id_QCM, No_Question_QCM -> Id_Question
- 6) Id_personne -> nom, prenom
- 7) Id_personne, Id_QCM -> resultat_obtenu
- 8) Id_personne, Id_QCM, No_Question_QCM -> Id_Reponse

Sémantique

Remplir les tables correspondant aux instances des relations afin d'exprimer le texte qui suit:

"Gaston LeRoux (personne123) à répondu à la 33ème question de son examen QCM (144) par le choix B.

La 33ème question de ce QCM est la question (numéro 4500) qui appartient au domaine SIGNALISATION. Son énoncé est "L'interdiction de circuler est un panneau". Son niveau de difficulté est 3.

Les réponses à choix sont:

A) circulaire, avec un bord blanc et un fond rouge (faux)

B) circulaire, avec un bord rouge et un fond blanc (juste)

Un QCM comporte au minimum 10 questions sur le domaine SIGNALISATION."

Domaines_enseignés	Id_Domaine	Lib_Domaine	Nbr_Min_questions

Questions_Possibles	Id_Question	Lib_Question	Id_Domaine	Niveau_difficulté	A_revoir

Reponses_Possibles	Id_Question	Id_Reponse	Lib_Réponse	Juste

QCM	Id_QCM	Date_examen

Question_QCM	Id_QCM	No_Question_QCM	Id_Question

--	--	--

Personnes	Id_personne	nom	prenom

Examen	Id_personne	Id_QCM	resultat_obtenu

Reponses_Examen	Id_personne	Id_QCM	No_Question_QCM	Id_Reponse

Questions de modélisation

Compléter le schéma qui suit en plaçant les associations manquantes et en donnant des cardinalités en rapport avec notre énoncé.

QCM
<u>ID_QCM</u>
DATE_EXAMEN

QUESTIONS DU QCM
<u>NO_QUESTION_QCM</u>

PERSONNES
<u>ID_PERSONNE</u>
PRENOM
NOM

DOMAINES_ENSEIGNES
<u>ID_DOMAINE</u>
LIB_DOMAINE
NBR_QUESTIONS

EXAMEN
RESULTAT_EXAMEN

QUESTIONS_POSSIBLES
<u>ID_QUESTION</u>
LIB_QUESTION
NIVEAU_DIFFICULTE
A_REVOIR

REPONSES_POSSIBLES
<u>ID_REPONSE</u>
LIB_REPONSE

Répondre en SQL

- 1) Liste des domaines par ordre alphabétique
- 2) Liste des personnes (Nom et prénom) ayant 0 faute.(100 points)
- 3) Difficulté moyenne des questions possibles par domaine
- 4) Liste des questions possibles ayant aucune réponse possible juste (erreur de saisie)
- 5) Libellé des questions et des réponses justes du QCM 144 par ordre croissant.

- 6) Quel est l'énoncé de cette requête ?

```
select a.id_qcm, sum(b.niveau_difficulté)
from Questions_du_QCM a , Questions_Possibles b
where a.id_question=b.id_question
group by a.id_qcm
order by sum(b.niveau_difficulté)
```

- 7) Quel est l'énoncé de cette requête ?

```
select nom, prenom
from Personnes p, Examen e, QCM q
where p.id_personne=e.id_personne
      e.id_qcm=q.id_qcm
      and q.date_examen between '1-jan-94' and '31-dec-94'
```

- 8) Quel est l'énoncé de cette requête ?

```
select nom, prenom, count(e.id_personne)
from Personnes p, Examen e
where p.id_personne=e.id_personne
group by nom, prenom
having count(e.id_personne) >=5
```


28. METGE

(Menuiserie En Tout Genre et Ebénisterie)

METGE²³ est une entreprise de menuiserie, dont la plupart de l'activité est orientée dans l'équipement des nouveaux bâtiments. Elle fournit et pose dans les immeubles les cadres de porte, les portes, les cadres de fenêtre et les armoires. L'entreprise est divisée en quatre départements :

- le bureau technique qui établit les soumissions et qui s'occupe des nouvelles conceptions;
- l'atelier qui commande les matières premières, fabrique et prépare les éléments d'un contrat passé à METGE;
- le département chantier qui est chargé du montage et de la pose des éléments dans les immeubles;
- le département de gestion qui s'occupe de la comptabilité et de la gestion administrative de l'entreprise.

PV de la Réunion :

Dans le bureau du directeur de METGE sont réunis le Directeur (D), le Gestionnaire (G), le chef d'atelier (A), le technicien (T) et le Chef de chantier (C) (dans le dialogue qui suit nous n'utiliserons que l'initiale des personnages pour désigner celui qui parle).

D : Vous n'ignorez pas que nous avons, il y a 6 mois, pris la décision de fabriquer nous-mêmes les armoires et que nous avons effectué des investissements dans plusieurs machines spécialisées. Avant cette décision, nous achetions chez ARMO-BLOC les armoires toutes équipées que nous n'avions plus qu'à poser. Depuis, nous achetons chez AGLO SA le bois qui est prédécoupé et sur lequel nous faisons toutes les opérations nécessaires à l'obtention d'une armoire. Nous avons sous-estimé les difficultés de gestion, de coordination et de calcul que demandent ces opérations. Des erreurs de calcul, de fabrication nous ont obligés à envoyer au rebut des éléments de chez AGLO SA car ils n'avaient pas les bonnes dimensions ou bien, comme cela est arrivé sur le dernier chantier, à démonter une partie des blocs déjà posés, car le perçage pour fixer la barre de penderie était trop bas. De plus, l'énerverment et les heures supplémentaires demandées par la gestion de cette nouvelle fabrication contribuent à un mauvais climat de travail. Je comprends bien que chacun de vous fait de son mieux et je tiens à ce que la bonne humeur qui régnait dans notre entreprise revienne. M.G., que j'ai engagé il y a deux ans pour restructurer et moderniser nos outils de gestion

²³ sujet élaboré avec le professeur Jean-Claude Courbon

et d'analyse financière, peut certainement nous aider dans ce cas aussi. Il peut nous effectuer une analyse de la situation, qui nous servira ensuite comme cahier des charges pour prendre contact avec des entreprises de services informatiques. Cette façon de procéder nous avait épargné bien des peines, il y a déjà deux ans, lors de notre prise de contact avec les informaticiens "purs". M.G., je vous laisse la parole.

G : Je crois que le plus simple est de suivre le parcours d'une armoire depuis le départ jusqu'à sa pose. Où cela débute-t-il ? (G sort un bloc et prend des notes).

T : En ce qui concerne le bureau d'étude technique, nous recevons les demandes de soumission, que nous effectuons sur plan ou après visite du chantier. La soumission consiste à établir le prix détaillé de l'ensemble à poser, celui-ci varie en fonction du nombre de "boîtes" à poser.

G : Qu'entendez-vous par boîte ?

T : Une boîte est l'élément le plus simple d'un ensemble d'armoires. Nous mesurons la longueur totale de l'ensemble que nous divisons en boîtes, selon les désirs de l'architecte, par exemple une longueur de 160 cm peut être divisée en trois boîtes (60cm 50cm 50cm). Un autre point important est la situation de l'armoire par rapport au mur, nous avons quatre cas :

(le technicien sort son stylo et dessine).

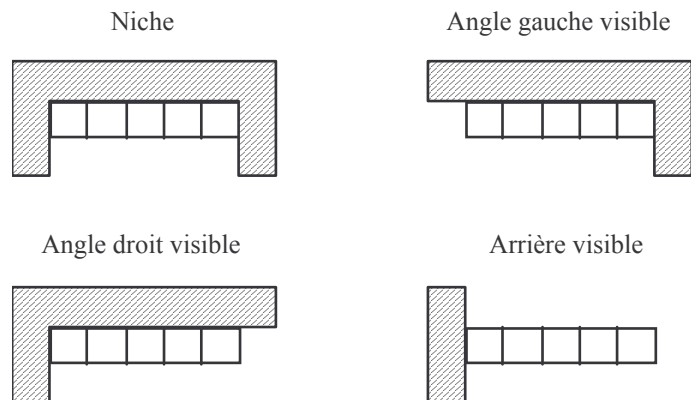


figure 1 : situation d'un bloc d'armoires

A : En effet, c'est important car lors de la commande du bois, il faut en tenir compte car les côtés visibles doivent être aussi recouverts du stratifié demandé par le client.

T : La couleur du stratifié fait partie de la soumission, car le prix change en fonction des couleurs, le blanc étant le moins cher.

G : Il y a quelque chose que je ne comprends pas, si l'on juxtapose des boîtes, il y a deux parois pour les parties qui se touchent, c'est une perte.

C : Vous avez raison, on parle de boîtes mais en fait il s'agit de boîtes sur lesquelles il manque le côté gauche, et qui ont ainsi une forme de C, sur la dernière on ajoute un panneau de terminaison pour compléter le tout.

C. dessine aussi !

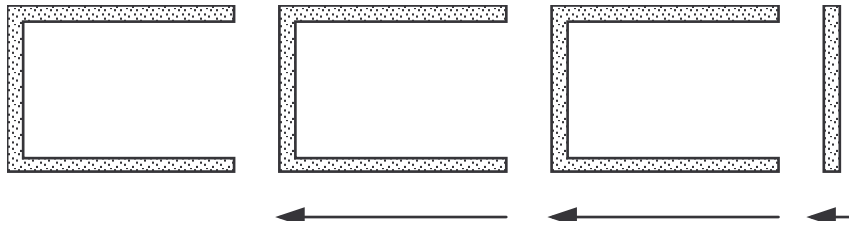


figure 2 : assemblage d'un bloc d'armoire

G : Ah oui, je vois, continuons avec la soumission.

T : Voyons ! Un ensemble de boîtes a la même hauteur, même profondeur. Mais chaque boîte a sa propre largeur et une ou deux portes. S'il n'y a qu'une porte il faut préciser si elle s'ouvre à gauche ou à droite. (Il allume une cigarette.)

Je vois que l'on a fait le tour de l'extérieur de la boîte. Maintenant, passons à son aménagement intérieur. L'architecte doit préciser le nombre de rayonnages par boîte et si éventuellement il y a une penderie. On est complet ?

C : Oui.

G : Comment s'établit le prix de la soumission ?

T : C'est assez simple, avec l'expérience, nous avons pris des temps de pose et de fabrication moyens et nous nous sommes aperçus que ceux-ci étaient en fonction du nombre de portes d'une boîte, nous comptons donc en portes.

- par porte à l'atelier 1h.30
- et par porte sur le chantier 2h.00.

G : A Fr. 52.- l'heure d'ouvrier, selon notre dernière analyse.

T : Oui, c'est cela. L'autre élément important est la surface de bois utilisée à Fr. 9,50 / m² pour le blanc et des surplus pour les autres couleurs.

A : Vous oubliez les garnitures, vis, serrures, poignées et les gonds. On compte environ Fr. 40.- de garniture par boîte.

T : Vous avez raison. Avec tous ces éléments, on calcule le prix de la soumission que l'on envoie au client. Trois cas se présentent, il accepte celle-ci, ou bien il la refuse et dans certains cas on nous demande certaines modifications, ce qui nous oblige à tout recalculer. Les soumissions comportent entre 50 et 300 boîtes.

G : Oui, c'est beaucoup de travail quand on sait que nous avons effectué 450 soumissions l'année passée et que nous avons 20% de taux d'adjudication pour des soumissions.

D : C'est, je pense, très important, de limiter les investissements en temps humain dans les soumissions, tout en conservant une précision et une compétitivité des prix sur le marché. Et en plus cela prend beaucoup de place, ces dossiers, quand on pense que certaines soumissions sont gardées deux ans.

C : Quelqu'un veut-il un café ?

G;T;A : oui, volontiers.

D : Oui, je vais aller les chercher, je crois que nous en avons terminé avec les soumissions, continuez sans moi. (Le directeur sort).

T : Oui, dès que la soumission est acceptée nous la transférons à l'atelier.

A : C'est là que nous intervenons, nous effectuons une commande à AGLO SA des différents panneaux de bois stratifié en indiquant les mesures désirées ainsi que la couleur. Dès que nous les réceptionnons, nous les calibrons, car ils sont découpés grossièrement, ensuite nous effectuons toutes les opérations de préparation à l'assemblage, c'est-à-dire :

- pose des charnières sur les portes;
- perçage des trous pour les rayonnages;
- collage des bandes de stratifié sur les bords apparents des panneaux;
- et diverses préparations concernant les dos des boîtes, les poignées ...

G : Où se situent les difficultés ?

A : Elles sont sur les calculs des différentes cotes de perçage et sur les dimensions de panneaux à commander. Les calculs sont simples, mais il ne faut pas oublier des détails comme de tenir compte de l'épaisseur des panneaux eux-mêmes. Ce travail est répétitif et ennuyeux, et malgré son importance, il arrive que l'on se trompe.

G : Etes-vous si sûr de sa simplicité ?

A : Oh oui, pour l'apprenti j'ai préparé un schéma-type, avec les différentes cotes, et les règles de calcul. Il lui suffit d'appliquer systématiquement les règles de calcul pour obtenir les bonnes dimensions.

G : Et que se passe-t-il ensuite, lorsque tout est préparé ?

C : C'est l'équipe du chantier qui travaille, elle passe au magasin chercher les garnitures :

- 8 vis d'assemblage simple par boîte
- 1 poignée par rayon
- 4 taquets par rayon
- (les tringles à habit si nécessaire).

On charge, bien sûr, les panneaux préparés sur un camion et l'on va monter le tout dans un immeuble. (Le directeur revient avec les cafés.)

G : Quels sont les problèmes qui peuvent apparaître sur le chantier ?

C : Normalement aucun si toutes les étapes précédentes se sont déroulées correctement ... Parfois, on perd du temps à reconnaître les éléments d'une boîte, il serait pratique de pouvoir étiqueter les piles, pour simplifier le montage.

D : Oui, c'est vrai. Mais je crois que les étapes en amont sont prioritaires. G, avez-vous suffisamment d'information pour esquisser un système d'information assistant la gestion de cette nouvelle fabrication ?

G : Un dernier point. Qui d'entre vous est susceptible de se servir d'un système informatisé à mettre en place ?

T : Moi, obligatoirement pour gérer toutes les soumissions et les contrats signés.

A : Nous en aurons besoin pour établir les commandes chez AGLO SA, mais aussi pour pouvoir consulter les éléments de réglage des machines.

C : Moi, il suffit d'un listing, surtout si l'étiquetage est bien fait.

D : C'est tout ?

G : Oui, je pense. Il est midi, je vais aller manger et je ferai cette analyse cet après-midi. Et je propose que nous nous réunissions à 17 heures pour en débattre.

(Les gens se disent au-revoir et partent manger. G ramasse un dessin fait par A représentant un bloc d'armoire).

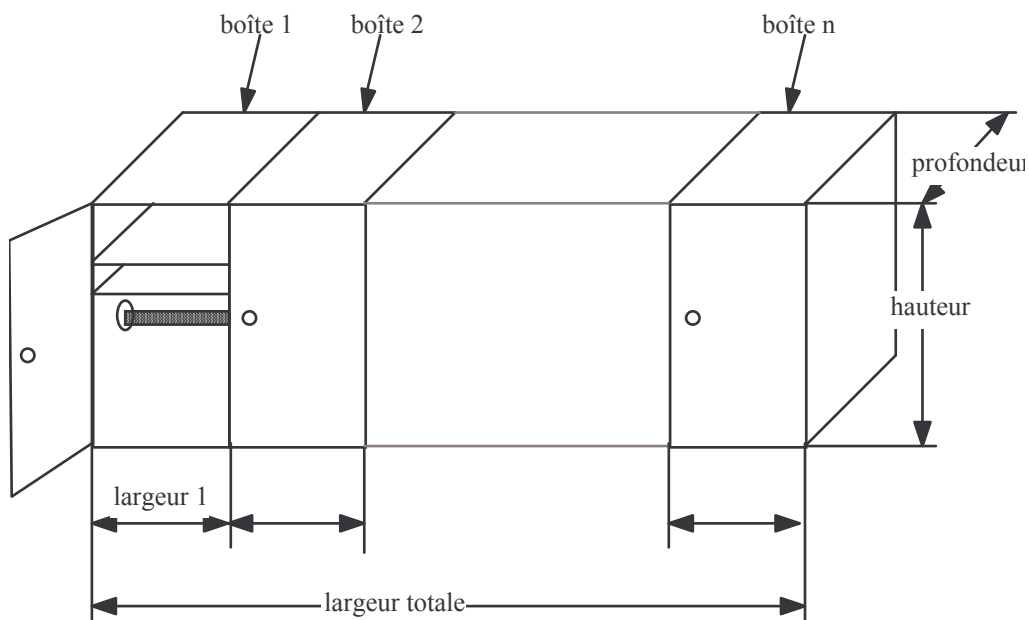
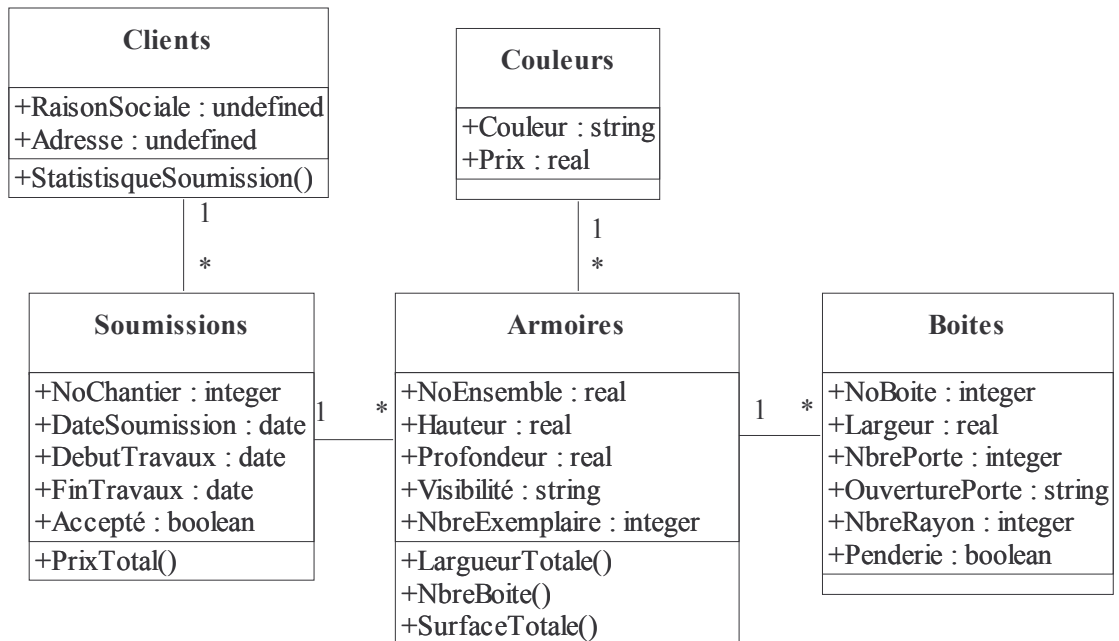


figure 3 : un bloc d'armoires

Nous vous demandons d'effectuer l'analyse que va faire ce Monsieur G., de reconstruire le parcours de cette armoire, des différents acteurs et intervenants, des tâches qu'ils effectuent, de décrire les données qui doivent être stockées, de décrire une modélisation possible de celles-ci, d'envisager les architectures informatiques possibles pour supporter et faciliter cette nouvelle fabrication.

Certaines informations sont absentes du dialogue, par exemple un client a une adresse, un chantier a une adresse, etc. Nous vous demandons de les compléter avec le "bon sens", en mentionnant "nous faisons l'hypothèse que ...". Par ailleurs, d'autres informations peuvent être inconnues de Monsieur G., et nous vous demandons de dresser une liste de questions que devra poser G. lors de la réunion à 17 heures pour affiner son analyse.

Diagramme des classes



Domaine des constituants

RaisonSociale	texte
Accepté	booléen (Oui, Non)
Adresse	entier
Couleur	mot(blanc, rouge, vert, ...)
DateSoumission	date
DebutTravaux	date
FinTravaux	date
Hauteur	réel
Largeur	réel
LargeurTotale	réel
NbrBoite	entier
NbreExemplaire	entier
NbrePorte	entier [1..2]
NbreRayon	entier
NoBoite	entier
NoChantier	réel
NoClient	entier
NoEnsemble	entier

OuverturePorte mot(Gauche, Droite, Centrale)

Penderie booleén (Oui, Non)

Prix réel

PrixTotal réel

Profondeur réel

SurfaceTotale réel

Visibilité mot(Niche, AngleGauche, ...)

Relations

(une clé de la relation est indiquée par un soulignement)

On remarque le choix d'implémenter les méthodes comme des attributs stockés.

CLIENTS (NoClient , RaisonSociale, Adresse)

COULEURS (Couleur, Prix)

SOUMISSIONS (NoChantier, NoClient, DateSoumission, DebutTravaux, FinTravaux, Accepté, PrixTotal)

ARMOIRES (NoEnsemble, NoChantier, Profondeur, Visibilité, NbreExemplaire, LargeurTotale, NbrBoite, SurfaceTotale)

BOITES (NoBoite, NoEnsemble, Largeur, NbrePorte, OuverturePorte, NbreRayon, Penderie)

Questions

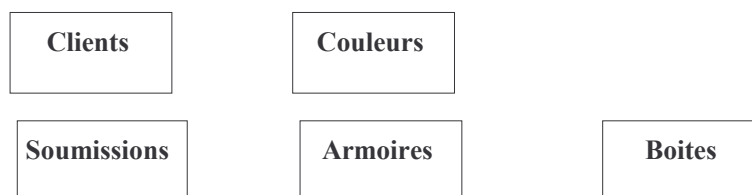
1) Donnez le diagramme des cas d'utilisation.

2) La modélisation est entièrement faite avec des associations, peut-on utiliser dans certain cas la composition si oui pourquoi.

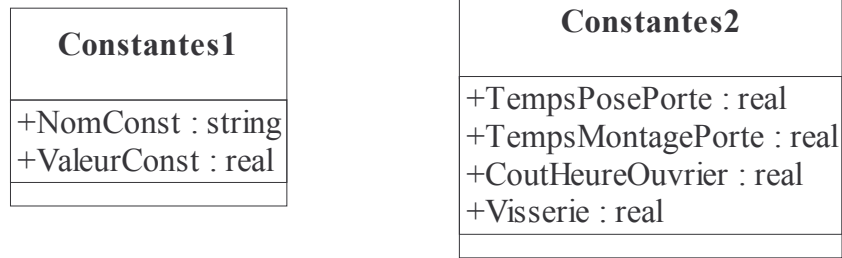
Cas où on peut utiliser la composition :

Cas où on peut pas utiliser la composition :

Complétez le diagramme suivant :

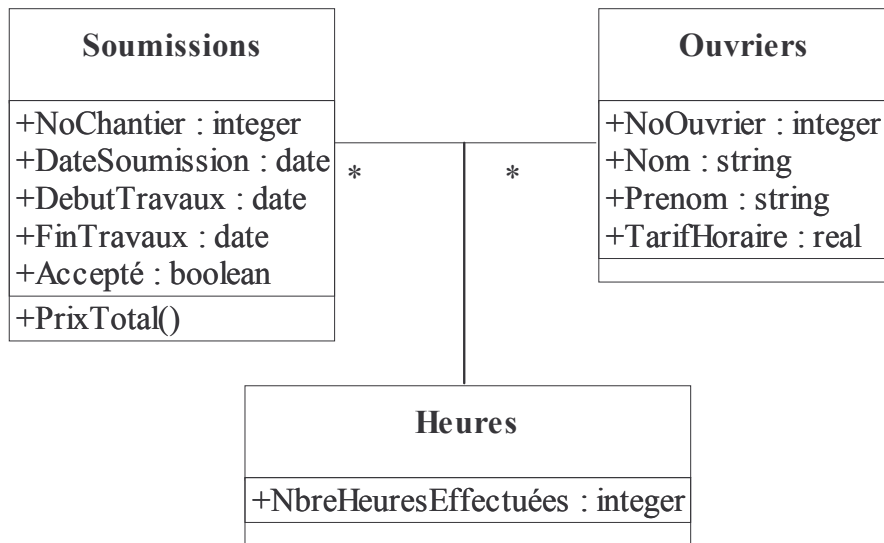


3) Pour mémoriser les constantes de calcul, le concepteur hésite entre deux modélisations de classe. Aidez le en dressant un tableau des avantages et inconvénients (pensez en termes de complexité et d'évolutivité !)



	constante1	constante2
Avantages		
Inconvénients		

4) Pour mesurer les prix de revient, il a été décidé de rajouter les éléments suivants à la modélisation. Quelles sont les relations à ajouter ?



5) Donner l'ordre SQL pour créer la table BOITES et insérer un exemple de données dans celle-ci avec un ordre SQL.

6) Répondre en Langage Algébrique si possible et en SQL aux questions suivantes:

- 1) Donnez une liste de tous les clients par ordre alphabétique
- 2) Donnez une liste des clients dont une des soumissions acceptées étaient d'un montant supérieur à un million de francs
- 3) Donnez une requête qui permette de calculer le taux d'adjudication (nbr de soumissions / nbr de celle qui sont acceptés)
- 4) Donnez une liste des armoires dont la largeur totale n'est pas égale à la somme de la largeur de ses boites.

29. ETUDE DE CAS : A B C D²⁴

P.B. plaça dans la platine une nouveauté qu'il n'avait pas encore écoutée, qui silencieusement fut avalée par le lecteur. Il imagina le rayon du laser à la recherche des "trous" du disque compact (CD). Les circuits électroniques qui reconstruisaient le signal sonore. Oui, c'était vraiment de la musique cela !

Derrière son comptoir, il regarda le magasin, minuscule peut-être, mais qui contenait déjà 9.000 CDs. Il essaya de se le rappeler 9 mois auparavant, vide quand avec son associé il était venu le visiter. Il se dit "qu'ils avaient encore une certaine naïveté", ce qu'ils voulaient c'était vendre des disques, créer un magasin de CDs où l'on pourrait écouter le disque, obtenir un renseignement, faire une réservation, discuter musique et bien entendu acheter un CD au prix le plus avantageux. En résumé, les CDs au prix des grandes surfaces avec tous les services et les aspects humains d'un petit commerçant.

Ces objectifs avaient failli tourner court. En effet, ils avaient sous estimé les charges de gestion et de manipulation exigées par ce magasin. Ces activités étaient telles qu'ils avaient de moins en moins de temps à consacrer aux clients. Il y a deux mois, ils avaient décidés d'informatiser une partie des activités de gestion du magasin. Cette ordinateur serait un peu comme le troisième associé (qui dans la situation actuelle n'aurait pas pu être rétribué).

De l'informatique, il n'avait pas de connaissances spécifiques mais les premiers contacts avec cet univers furent frustrants, il n'avait trouvé que des gens qui parlaient "Mbyte, MS/DOS, CGA, Lotus, BD, RS232, ...". Son expression favorite "C'est Martien" était tout à fait de circonstance. En effet, aucun ne s'était intéressé à son problème; tous ne voulaient que lui vendre un ordinateur. Cependant, ceci était une caricature, car il avait trouvé une personne qui avant de lui parler dans un langage hermétique l'avait écouté et s'était déplacé dans le magasin pour examiner les activités quotidiennes à informatiser.

Ce soir, il se sentait rassuré. M. Gécoute semblait avoir compris leur problème. Il s'était promené dans le magasin toute la journée, avait posé des questions, pris des notes, observé leur comportement.

L'activité du magasin était entièrement orientée vers le CD. Quand un CD arrivait, on effectuait les opérations suivantes :

- S'il est nouveau, on lui attribue un numéro et l'on crée un fiche sur carte.
- On enregistre sur la fiche le nombre d'exemplaire qui sont entrés.

²⁴ sujet élaboré avec le professeur Jean-Claude Courbon

- On place dans le présentoir la pochette du CD (avec un numéro identifiant la fiche correspondante).
- On place le disque dans un tiroir fermé à clef. (Malgré ces opérations, on enregistrerait encore quelques vols de pochettes vides !).

Lorsqu'un client achetait un CD, avec le numéro inscrit, on décrémen-tait le stock d'une unité sur la fiche et l'on recherchait le disque dans le tiroir.

Le fichier était séparé en trois bacs de cartes de couleurs différentes, indiquant un style de musique :

- Bleu -> Classique
- Jaune-> Rock - Variété
- Rouge-> Jazz

M. Gécoute avait demandé si la numérotation actuelle pourrait être modifiée afin de ne pas transporter des informations (la couleur) qui n'auraient plus leur place dans un système informatisé. P.B. avait répondu que seul le style de musique lui importait.

Ces fiches étaient la "mémoire" de l'entreprise ABCD, tout y étaient noté, mais c'était une mémoire d'amnésique car comment rechercher un titre, un auteur dans 3000 fiches ? Comment gérer les commandes, les ruptures de stock ?

En examinant une fiche, on voyait que chacune comportait, un auteur, un style, un titre, un numéro interne (pour son classement dans le magasin), un éditeur, un numéro externe (celui qui est marqué par l'éditeur) et différents paramètres de gestion tels que la quantité en stock ou en commande et son code de prix de vente.

L'estimation de la demande était le point le plus délicat de la gestion des CDs. Certains CDs allaient être vendus à plusieurs dizaines d'exemplaires sur quelques mois alors que d'autres ne seraient que le fait de la demande d'un unique client pendant plusieurs années. Il y avait une centaine de titres qu'il fallait surveiller quotidiennement. La courbe des ventes s'amorçait légèrement au départ, puis elle atteignait un plateau qui se maintenait pendant un certain temps puis c'était la chute soudaine.

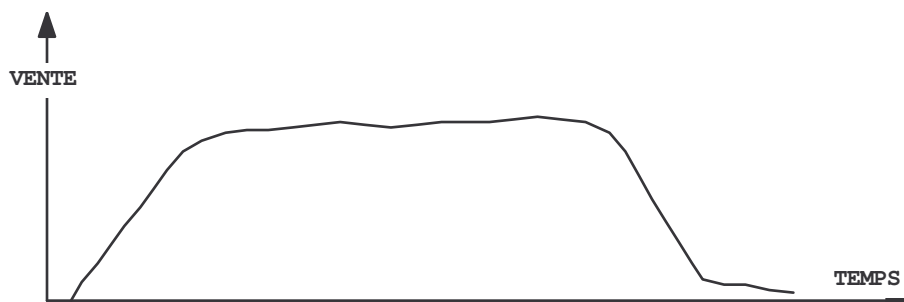


figure 1 : Ventes quotidiennes d'un titre

Il fallait donc estimer les ventes futures probables et se réapprovisionner en conséquence en évitant qu'un stock important reste invendu.

Les commandes s'effectuaient chez les distributeurs suisses des maisons d'édition des disques. Soit un représentant passait au magasin ou bien PB ou son associé téléphonait au distributeur et passait sa commande. Une commande faisait l'objet d'un bulletin de livraison et sur celui-ci pour un CD commandé on avait trois possibilités :

- Il était livré.
- Il était en rupture de stock (manque suivra) et parfois une année plus tard on recevait le CD.
- Il n'était pas disponible en Suisse.

Le cas 2) était à noter soigneusement car le reliquat d'une commande était à considérer dans le stock car il serait tôt ou tard livré. Le cas 3) pouvait faire l'objet d'une importation de l'étranger donc d'une nouvelle commande. Le cas 1) demandait une vérification assidue, car là il fallait contrôler que :

- Les CDs livrés avaient bien été commandés.
- Les quantités étaient exactes.
- La facture correspondait à ce qui avait été convenu sur les prix.

En effet, le prix d'achat des CDs était source de négociation, le prix était discuté en fonction de la quantité commandée et du chiffre d'affaire que le magasin avait réalisé avec le distributeur. D'ailleurs PB tenait à ce que l'ordinateur lui indique le chiffre d'affaire réalisé avec le distributeur. Ceci allait dans la stratégie "payer le CD au meilleur prix" et "vendre le CD au meilleur prix par rapport à la concurrence". Ces rabais de quantité suivaient une échelle. Par exemple pour 4 CDs on payait prix fort et pour 100 on avait un rabais de 30%. Ceci renforçait la nécessité d'estimer les ventes journalières, car il valait mieux commander le lot optimal en une seule fois plutôt qu'en plusieurs commandes. Finalement, un autre cas se présentait, le pré-commande. Il s'agissait de la sortie d'une nouveauté qui était annoncée plusieurs mois à l'avance et qui le même jour serait sur toute la planète, dans tous les magasins de CD. Ce cas était encore plus épineux que les autres car il fallait estimer un an à l'avance les chances de succès d'une nouveauté.

article : Gabriel Peter No .. 2519titre SO Virgin 257 587							
Date	QTE	Fournisseur	Entrée	Sortie	Stock	Client	Observation
	50	MV	1/9/86				
	49			3/9/86			
	45			4/9/86			
	44			5/9/86			
	43			7/9/86			
	42			8/9/86			
	40			10/9/86	-40		
		commande le 10/9/86			+10		
	50		11/9/86				

figure 2 : fiche pour un CD

Avant que M. Gécoute parte, PB lui avait résumé ses désirs :

- Le stock en temps réel.
- L'estimation des ventes journalières.
- Le contrôle des livraisons.
- Les indications de négociations avec les distributeurs.

M. Gécoute est rentré chez lui avec des informations similaires à celles données dans cette étude de cas.

Décrivez le travail qu'il va faire, donnez la modélisation de données. Etablissez une (ou des) propositions de système informatique avec leurs justifications.

Diagramme des cas d'utilisation

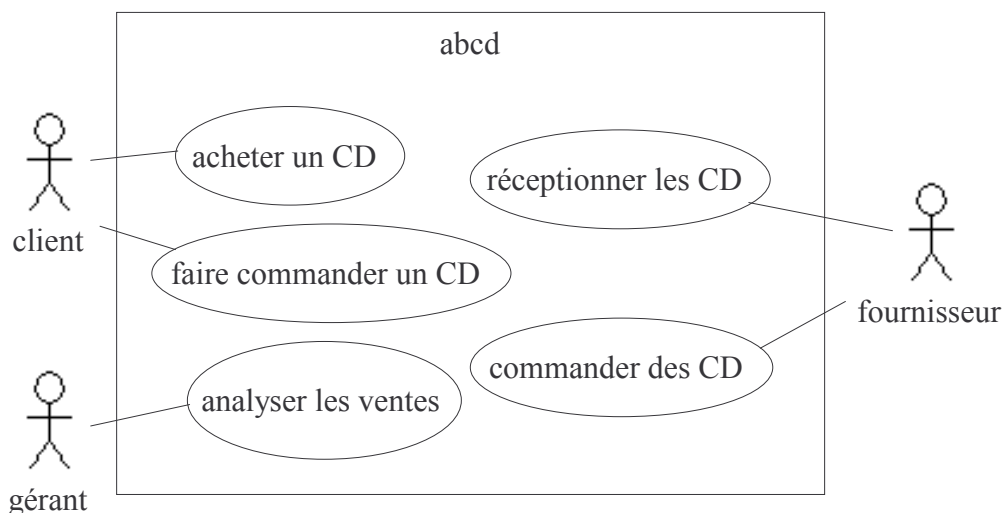
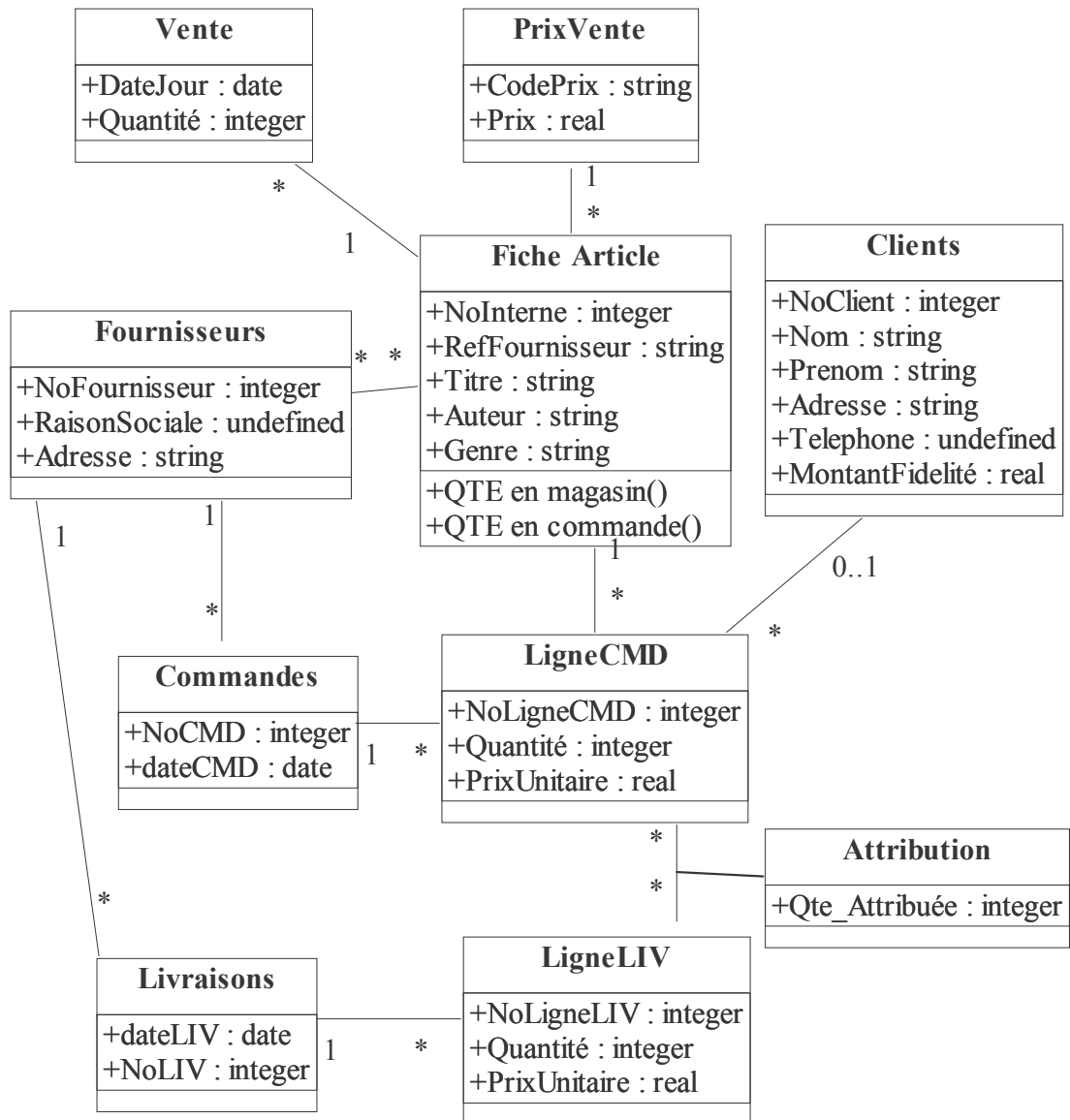


Diagramme des classes



Commentaires :

On remarquera que le suivi des livraisons n'est pas simple en effet une livraison physique peut :

- Concerner une ligne de commande et être juste quantité
- Regrouper plusieurs lignes de commande (regroupement chez le fournisseur)
- Être incomplète (nécessite plusieurs livraisons)

30. PLANS, PLANS, RATAPLAN

M. Saroule avait été engagé dans un département de l'administration publique. Il avait choisi l'administration car, depuis ses études à l'université, il aimait les organisations et la gestion de leur information, en somme les systèmes complexes. Le département auquel il appartenait était chargé du réseau routier, plus de 10.000 kilomètres de routes et de tous les ouvrages (tunnel, pont, ...). Ce département possédait un service qui archivait les plans et les documents concernant l'exécution des chantiers relatifs au réseau routier.

Le service des archives employait une dizaine de personnes qui répondaient aux différents besoins du département. Par exemple, lors de l'entretien ou de la réfection d'une route, ils fournissaient les plans nécessaires aux travaux. Si un chantier modifiait les plans existants, ils étaient chargés de mettre à jour l'état des archives. Bien entendu, autour de ces routes, d'autres informations "circulaient", celles relatives au contrôle périodique des ouvrages, au constat des dégâts dus aux intempéries et toutes les informations sur les coûts de construction et d'entretien qui n'étaient pas du ressort du service des archives.

A son entrée, M. Saroule avait perçu la lenteur et l'inefficacité du service des archives, dues entre autres au manque de personnel, à la masse considérable de documents et à la grande dimension des plans (format A0). Cette inefficacité du service se traduisait dans le reste du département par des retards et des incomplétudes dans les dossiers. Par exemple, des renseignements simples du genre: "Quel est le bureau d'ingénieur qui s'est occupé du dernier chantier de la section 5 du tronçon 14 de la route nationale 11 ?" pouvaient rester dans le service des archives sur la pile des "urgents" pendant plusieurs jours. Il s'agissait donc de questions auxquelles un système informatisé aurait pu apporter rapidement des réponses précises.

Pour faire face à ces problèmes de gestion, le département avait débloqué un crédit d'informatisation du département qui s'étalerait sur une période de cinq ans. Une petite équipe d'informaticiens serait recrutée pour la réalisation et la maintenance du système. Mais au préalable, il fallait que les besoins du département soient clairement explicités. Un groupe de travail avait donc été formé et M. Saroule devait au sein de ce groupe étudier les besoins du service des archives.

Pour mieux cerner, les données manipulées par le service des archives et afin d'illustrer son dossier, il avait fait un croquis (figure 1)

S'il avait dû commenter ce schéma, il aurait dit "Le réseau routier se répartit entre plusieurs catégories de routes: les autoroutes, les routes nationales, les routes cantonales, les routes communales. Chacune est identifiée par un nom unique (A10, RN17, RC206, ...). Chaque route est découpée en

tronçons et chaque tronçon est lui-même découpé en sections. A chaque section est associé l'ensemble des chantiers concernant cette section. Les chantiers sont repérés par une date de début et de fin. La supervision d'un chantier est exécutée par un bureau d'ingénieur. Ce dernier est décrit par un ensemble de caractéristiques (nom, adresse, ...) utilisées dans la correspondance ou dans les différents rapports exécutés par le département. Chaque chantier apporte une nouvelle version des plans de la section (un plan d'ensemble et les différents plans de détail). En ce qui concerne le rangement, chaque plan est rangé dans un local qui comporte un certain nombre d'armoires divisées elles-mêmes en plusieurs tiroirs".

Les sections avaient en moyenne une longueur de 1 kilomètre et les tronçons entre 3 et 20 kilomètres. Chaque section possédait un chantier (celui de sa création), mais certaines sections avait déjà subi une quinzaine de modifications (et donc autant de chantiers).

Un autre point important du système d'information à élaborer était celui des micro-films. En effet, le volume des archives était devenu tellement important qu'une partie des plans, ceux concernant des chantiers anciens, étaient actuellement stockés dans un autre bâtiment. Il avait été décidé de micro-filmer ces plans. Le système informatique devrait donc donner une indication sur la nature des plans (grandeur nature ou micro-filmé). Par ailleurs, le système devrait aussi sélectionner les plans à microfilmer sur des critères tels que le nombre de versions et l'ancienneté des chantiers pour une même section.

Ceci permettrait de ne conserver sur papier que les versions les plus récentes et ainsi de minimiser la surface des locaux d'archives.

La disponibilité des informations du service des archives dans l'ensemble du département était souhaitable. Chaque service devrait donc avoir accès aux données, mais seul le service des archives aurait la possibilité de faire des modifications. Ainsi les demandes les plus simples (qui s'est occupé de tel chantier ?) seraient directement satisfaites. Pour les demandes nécessitant par exemple la copie d'un plan, il faudrait envisager un moyen de communication entre les différents services du département et le service des archives.

M. Saroule resta encore tard dans son bureau pour rédiger son analyse préliminaire pour la réunion du groupe de travail.

Nous vous demandons d'effectuer l'analyse que va faire ce Monsieur Saroule, d'examiner le service des archives, les différents acteurs et intervenants, les tâches qu'ils effectuent, de décrire les données qui doivent être stockées, de décrire une modélisation possible de celles-ci, d'envisager les architectures informatiques possibles pour supporter et faciliter ce système d'information ainsi que les types d'utilisation qu'il permettra et les procédures pour les accomplir.

Certaines informations sont absentes de l'énoncé. Nous vous demandons de les compléter avec le "bon sens", en mentionnant "nous faisons l'hypothèse que ...".

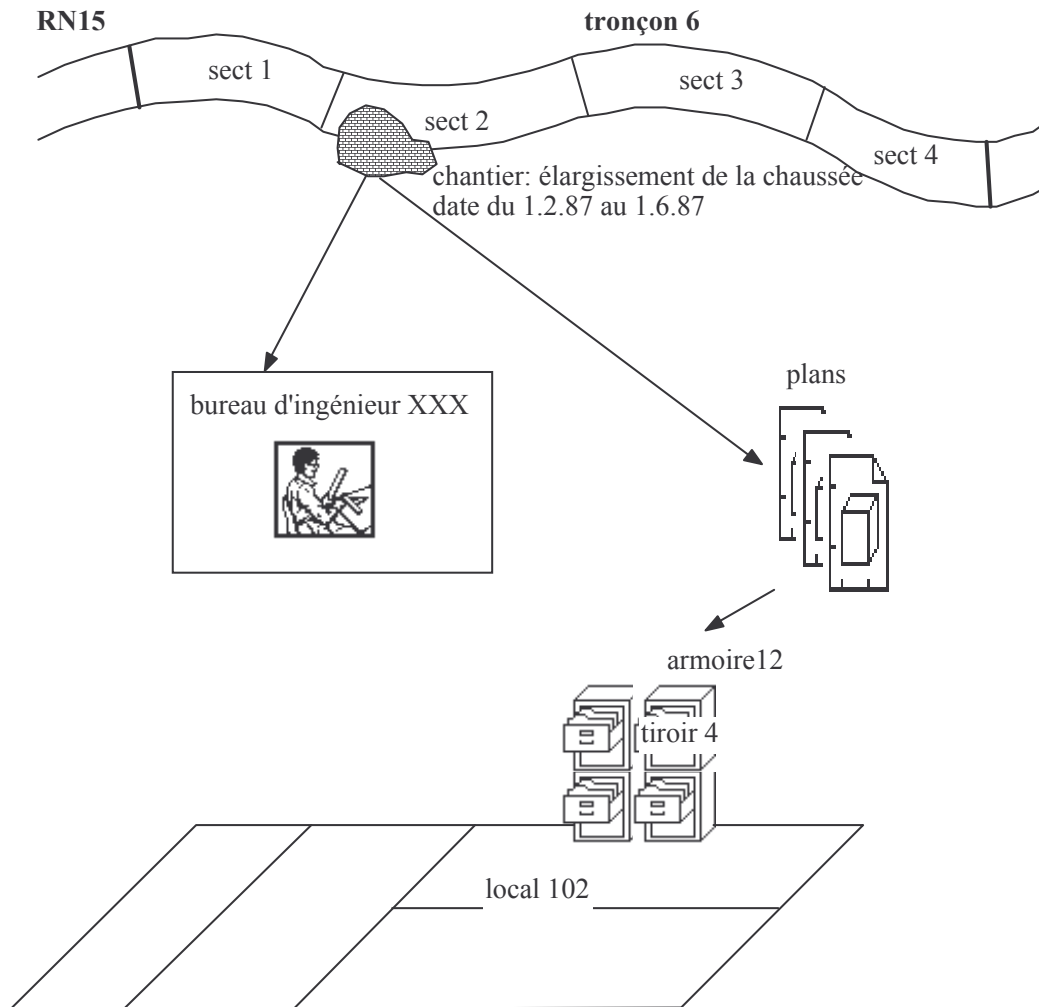


figure 1

31. BIBLIOGRAPHIE

- [ARM74] Armstrong W. W.
«Dependency structures of data base relationships» Proceeding IFIP 1974,
North Holland
- [BOE76] Boehm B.W.
«Software Engineering», IEEE, Transaction computer C-25, December 1976
- [BON99] Bonjour M. ; Falquet G. ; Guyot J. ; Le Grand A.
«Java : de l'esprit à la méthode, distribution d'applications sur Internet »
Edition Vuibert, 1999
- [BRE90] Breton P.
«Une histoire de l'informatique» Edition de la Découverte, 1990
- [CHE74] Chenique F.
Comprendre la logique moderne. Ed. Dunod 1974
- [CLO81] Clocksin W.F.; Mellish C.S.
, Springer-Verlag, 1981
- [COD70] Codd E.F.
«A relational Model of Data for Large Shared Data Banks» Communication
ACM, Vol 13, No 6, June 1970
- [DAT87] Date C.J.
A guide to the SQL standard Addison-Wesley 1987
- [DEL82] Delobel C.; Adiba M.
Bases de données et systèmes relationnels, Dunod 1982
- [FAL89] Falquet G.
Interrogation de bases de données à l'aide d'un modèle sémantique Ed.
Systèmes et Information Genève, 1989
- [FRY76] Fry J.P.; Sibley E.H.
«Evolution of data base management systems», Computing Surveys ACM, Vol
8, No 1, 1976
- [GOL72] Goldstine H.
«The computer FROM Pascal to von Neuman» Princeton University Press,
1972
- [GRA81] Gray J.N.
«the transaction concept: virtues AND limitations» Proceeding 7th
Internatioal conferenc 1981, IEEE Very Large Data base System.
- [HOT89] Hottois G.
Penser la logique. Ed. De Boeck Université 1989
- [HOU88] Houillère P.
Le schéma directeur d'un système d'information, Eyrolles, 1988

- [JAC92] I. Jacobson, M. Christerson, P. Jonsson, G. Overgard
« Object-Oriented software engineering : A use case » Addison-Wesley 1992
- [JAC94] I. Jacobson, M. Griss, A. Jacobson
« The object advantage : Business Process Reengineering with object
techonology » Addison-Wesley 1994
- [JAC97] I. Jacobson, ,M. Griss, P. Jonsson
« software Reuse : Architecture, Process and Oranization for Business
Success » Addison-Wesley 1997
- [JEN75] Jensen K.; Wirth N.
PASCAL user manual AND report, Springer Verlag, 1975
- [KOW79] Kowalski R.
Logic for problem solving, North Holland, 1979
- [LEO88] Léonard M.
Structure des bases de données. Ed. Dunod 1988
- [LIS94] Liskov B.; Wing J.
"A Behavioral Notion of Subtyping", ACM Transactions on Programming
Languages and Systems, Vol 16, No 6, November, 1994, pages 1811-1841
- [MISH92] Mishra P.; Eich M.H.
«Join processing in Relational Databases, ACM computing surveys, Vol 24, No
1, March 1992
- [NAU63] Naur P.
«Report on the algorithmic langage ALGOL 60», ACM 6, No 1, 1963
- [OMG99] Object Management Group
«Unified Modelling Language» dernière version: www.omg.org
- [PAP79] Papadimitriou C.H.
«serializability of concurrent database updates.» Journal ACM, Vol 26, No 4 ,
Oct. 1979
- [PHA90] Le Pham Ngung Huong
«Optimisation globale des contrôles d'intégrité dans une base de données»
Editions systèmes et information, 1990
- [REE50] Rees M.
«The federal computing machine program» reproduit dans «Annals of history
of computing» vol 7, No 2, 1985
- [ULL82] Ullman J. D.
Principles of Database systems Computer Science Press, 1982
- [WAR99] J. B. Warmer; A. G. Kleppe
The Object Constraint Language: Precise Modeling with UML , Addison-
Wesley, 1999
- [WIR76] Wirth Niklaus
Algorithms + Data Structures = Programs , Prentice- Hall, 1976

32. Index

- %, 148
- _, 148
- 1NF, 40, 236
- 2NF, 238
- 3NF, 239
- abstraction, 15, 60
- Access, 160
- acteur*, 50, 54
- Acteur, 62
- acycliques, 218
- agrégation, 154
- Agrégation, 37, 113
- algèbre relationnelle, 123, 257
- ALGOL 60, 96
- Algorithme, 234
 - de recherche de clés, 216
 - lié au df, 209
- alias*, 143
- all*, 146
- all**, 143, 150, 154, 260
- AND, 146
- anomalie, 185, 191, 195, 225, 243
- anomalies de mise à jour, 236
- any*, 146
- any*, 150, 158, 260
- appartenance
 - à un ensemble, 146, 147
 - à un intervalle, 146, 149
 - à une chaîne de caractères, 146
- Appartenance
 - à une chaîne de caractères, 148
- arborescence, 156
- arité*, 124
- Armstrong, 202
- asc*, 158
- association, 25, 108
 - attribué, 112
- Association binaire, 30
- associations, 22, 25
- atomicité, 176
- attributs, 24, 112
- augmentation, 202
- autojointure, 145
- avg*, 154
- axiome, 202
- axiomes, 88
- Backus-Naur-Form, 96
- B-arbres, 258
- base
 - complète, 209, 210, 221
 - irredondante, 211, 267
- base de données iMd, 93
- besoins, 50
- between, 146
- between, 149
- BETWEEN, 260
- biconditionnel, 87
- bien formée, 131
- binaires, 102
- BNF, 96
- Booch, 17
- BPR, 50, 53
- by rowid, 262
- calcul
 - des classes, 90
 - des propositions, 90
- caractère, 101
- cardinalité, 27, 31, 108
- cardinalité de la relation, 124
- cas d'utilisation, 50
- chaîne de caractères, 144
- chaînes de caractères, 86, 101
- character*, 101
- charnière, 199
- charnière, 129
- check*, 192
- chemins d'accès, 261, 263
- classe, 21, 105
- Classe, 55
- Classe d'association, 32
- classes, 20
- classification, 43
- clauses de Horn, 249
- clé, 106, 199, 215
 - primaire, 194
 - unique, 219
- Codd, 199
- colonne, 100
- column-constraint, 192
- commit
 - implicite, 180
- commit, 180
- communicable, 15
- comparaison, 146
- complétude, 203
- Complétude, 206
- comportement, 20
- comportement homogène, 86

- Composants, 54
- composition, 38, 41, 113
- concaténé, 144
- concurrence, 141
- concurrence, 173, 176, 187
- condition, 146
- Condition, 141
- conditionnel, 87
- confirmation, 180
- conjonction, 87
- CONNECT BY, 251
- Connect-Clause, 141, 156
- connecteurs logiques, 126
- conséquence logique, 202
- conséquence logique, 201
- conservation de l'information, 225
- consistance, 176
- consistance, 186
- consistant, 185
- constante non signée, 144
- constituant, 100
- constituant, 86
- contrainte, 34
- contrainte de navigabilité, 37
- contrainte entre associations, 37
- contrainte entre rôles, 36
- contraintes, 101
- Contrôleur, 62
- conversion de type, 147
- Count, 154
- coûts
 - de validation, 188
- couverture, 208
- covariance, 47
- CREATE-TABLE, 192
- CREATE-VIEW, 244
- création, 174
- création, 178
- création d'une entité, 101
- croissant, 158
- cycle de vie, 18
- d'états-transitions, 64
- date, 101
- declaration-domaine, 97
- declaration-relation, 97
- Déclenchement, 55
- décomposition
 - binaire, 230
 - d'une relation, 225
 - joiniable, 226
 - propriété, 227
- Décomposition, 204
 - d'une relation, 221
 - totale, 228
- décroissant, 158
- déduction, 249
- déduire, 202
- delete, 180
- delete-command, 180
- dénombrable, 84
- dénombrer, 154
- Dénormalisation, 272
- dépend fonctionnellement, 199
- dépendance
 - calculée, 247
 - d'inclusion, 190
 - élémentaire, 207
 - fonctionnelle, 199
- dérivation, 203
- desc, 158
- détermine, 199
- développement, 52
- diagnostic, 191
- diagramme, 55
- Diagramme d'objets, 24
- Diagramme de classes, 22
- diagrammes, 66
 - syntaxiques, 96
 - syntaxiques de SQL, 100
- différence, 124
- disjonction, 87
- displayed column, 141
- distinct, 142, 154
- distributivité, 90
- dom, 86
- domaine, 84
- domaines, 84
- doublons, 156
- doublons, 142
- durabilité, 176
- dynamiques, 187
- ebf, 131
- éléments, 64
- enjeux, 59
- ensemble
 - de constituants, 92
 - de dépendances fonctionnelles, 201
 - de données, 86
 - de primitives, 190
 - de relations, 93
 - de valeurs, 84
 - des constituants, 86
 - des domaines, 86
 - des entités, 94
 - des instances, 185
 - des ri, 185
- ensembles, 84
- entité, 20
- entité, 100
- Entité, 62
- entité r, 92
- équi-jointure, 128

- Equivalence, 209
- équivalence, 204
- équivalence des ordonnancements, 177
- et logique, 146
- état, 20
 - consistant, 176
- étoile, 200
- événement., 171
- exclusive, 181
- exists, 146
- exists, 150, 157
- Exists, 260
- expression, 144
- expressions
 - algébriques, 130, 137
 - arithmétiques, 143
 - calculées, 141
 - propositionnelles, 89
- exp-set, 148
- facteur, 144
- Factor, 144
- Factoriser, 45
- faux, 87
- fermeture*, 186, 201
- fonction
 - de regroupement, 144
 - de sélection, 127
 - propositionnelle, 90
- fonction, 144
- Forme canonique, 205
- forme normale, 40
- Forme normale Boyce-Codd, 240
- formule, 130
- FROM**, 137, 145, 243
- frontière, 15, 59
- généralisation, 43, 46, 114
- Généralisation, 41
- gestion de la concurrence, 177, 178
- grammaire d'un langage, 96
- graphique des df, 200
- GROUP BY, 154, 244
- Group-Clause, 141, 154
- hash-coding, 258
- having, 147, 154
- héritage, 41, 43
- héritage multiple, 48
- Héritage Multiple, 46
- Heuristique, 219
- identificateur de rangée, 261
- identificateurs, 97
- implémentation physique, 256
- in, 146, 260
- in, 147
- indépendance logique, 246
- index, 258, 262
- index, 263, 268
- informations redondantes, 179
- initialiser une table, 152
- insatisfaisable, 186
- insérer, 102
- insert, 178
- insert-command, 178
- insertion, 178
- instance, 104, 124
 - de la relation, 93
 - de R, 93
- Interactions, 56
- interdépendance, 274
- Interface, 62
- interface graphique, 160
- interprétations correctes, 131
- interpréter une valeur, 87
- inter-relation., 188
- interrogation, 123, 132, 225
- intersect, 156
- intra
 - relation,, 188
 - tuple, 188, 190
- invariant, 199
- invariants, 170
- irredondant, 186, 208
- isomorphisme, 90
- Jacobson, 18, 50
- Java, 106
- jointure*, 192, 259, 264, 268
 - externe, 130
 - naturelle, 140
 - naturelle, 129
 - par boucle imbriquée, 264
 - par tri et fusion, 264
- jointure*, 127, 137
- Jointure
 - externe, 151
- journal, 177
- l'arité, 30
- L'encapsulation, 22
- l'instance, 24
- langage
 - de description de modélisation, 96
 - de description des données, 96, 100
 - de manipulation de données, 178
- langage procédural, 106
- langages
 - de programmation, 84
- langages de programmation, 101
- l'évolution de l'informatique, 12
- liens**, 24, 35
- ligne, 100
- like, 146
- like, 149, 260
- lisibilité, 103
- Liskov, 44

- liste-intervalle, 97
- liste-valeur, 97
- Lock-mode, 181
- logical-factor, 146
- l'OMG, 17
- l'optimisation ; des requêtes SQL. Jusqu'à présent, nous avons fait abstraction de l'implémentation physique. i. implémentation physique; des relations et nous avons décrit le fonctionnement de la machine SQL en termes purement logiques. Les instances des relations dans un SGBD sont stockées physiquement; au schéma logique des relations est associé un schéma physique. i. schéma, 256
- machine SQL, 137, 159
- machine SQL, 138
- masque de saisie, 183
- masques de saisie, 184
- match-string, 149
- max, 154
- maximum, 154
- mécanismes de sécurité, 143
- méthode, 106
- méthodes, 20, 22
- min, 154
- minimum, 154
- minus, 156
- mise à jour**, 174, 176, 179
- modèle, 15, 105
- Modèle de déploiement**, 19, 63
- Modèle de réalisation**, 19, 63
- Modèle des cas d'utilisation**, 18, 63
- Modèle des classes**, 18, 63
- Modèle des états**, 18, 63
- Modèle des interactions**, 18, 63
- modèle Md, 104
- modélisation, 12, 97
- modélisation, 85
- modélisation Md, 93
- modélisation graphique, 132
- modélisations
 - équivalentes, 199
- modification, 225, 245
- MODIFICATION
 - DES DONNEES, 170
- moyenne, 154
- négation, 87
- nombres, 86
 - à virgule fixe, 102
 - en virgule flottante, 101
 - entiers, 102
- not, 146
- Not Null, 192
- notation, 22
- nowait, 181
- noyau du SGBD, 184
- null, 115, 146, 150, 246
- null, 151
- Null, 192
- n-uplet, 92
- objectif, 15
- objectifs, 51
- Objets, 20
- occurrence, 171
- OCL, 65
- Oid, 106
- OOSE, 17
- opérateurs
 - arithmétiques, 144
 - de comparaison, 126, 147
 - de projection, 126
 - ensemblistes, 156
 - logique, 128
 - logiques, 87, 146
- opérations
 - de comparaison, 86
 - de sélection, 85
 - ensemblistes, 141
- optimisation, 159
- optimiseur, 257
- OR, 146
- Order-Clause, 141, 158
- ordre
 - de jointure, 264
 - hiérarchique, 156
 - lexicographique, 85
- ou logique*, 124, 146
- package, 23
- paquetages, 65
- PASCAL, 183
- PASCAL, 96
- patron, 148
- performance, 256, 273
- performance, 263
- plan d'exécution;., 269
- portée d'une règle, 189
- prédicat, 88, 184
 - quantifié, 146
- prédicat, 90, 102, 146
- préservation des df, 225
- Préservation des df, 232
- primary key, 106
- Primary Key, 192
- primitive de modification, 188
- primitives de modification, 171, 174
- Principe de substitution, 44
- prior, 156
- processus, 62
 - de conception, 199
 - de modélisation, 100
 - d'interrogation, 131
- produit
 - cartésien, 84, 92, 100, 126, 141, 145, 257
- produit*, 126, 137

- cartésien, 90
- programmation, 22, 40
- programmation logique, 249
- projection, 143
 - de F, 221
- projection*, 125, 137, 141
- PROLOG, 249
- proposition, 89
 - composée, 87
 - simple, 87
- proposition, 87
- propositions
 - logiques, 84
- protocoles légaux, 176
- Pseudo transitivité, 204
- puits*, 217
- quantificateur universel, 151
- quantification
 - existentielle, 89
 - universelle, 89
- quantifié, 150
- racine, 157
- raffinements, 41
- raitement, 256
- range scan, 263
- read only, 180
- recherche des clés, 219
- recherche des invariants, 184
- récupération des erreurs, 176
- récurtivité, 250
- récurtivité, 154
- redondances logiques, 243
- réécriture, 257, 259
- réécriture de la requête, 245
- References, 192
- réflexivité, 202
- règles d'intégrité, 101, 176, 184, 225
 - en SQL, 192
- regroupement, 141, 143, 144
- regroupement, 154
- relation, 105
 - d'ordre, 85, 147
 - parent-enfant, 157
 - universelle, 247
- relation*, 124, 137
 - n-aire*, 84, 91
- relation d'extension, 57
- relation d'inclusion, 57
- relation de communication, 57
- renommer une colonne, 143
- représentations
 - logiques, 246
- reprise, 177
- requêtes SQL, 137
- restructuration, 199
- réutilisation, 45
- risque, 59
- rôles multiples, 26
- rollback, 181
- row
 - exclusive, 181
 - share, 181
- rowid, 261
- Rumbaugh, 17
- satisfaisable, 185
- saturation, 205, 215
- savepoint, 180
- scénario, 55
- Scénario, 61
- schéma
 - de relation, 201
 - physique, 256
- sécurité des données, 173
- SELECT**, 137, 243
- SELECT-command, 141
- selected-table, 145
- selected-table, 141
- sélection, 257
- sélection*, 127, 137
- sélection en SQL, 137
- sémantique, 102
 - de la relation, 92
- sémantique, 94
- semi-jointure, 129
 - gauche, 130
- sensibilité d'une primitive, 190
- séquence
 - de primitives, 176
- séquences, 64
- Set-Clause**, 141, 156
- SGBD historique, 172
- share, 181
 - row exclusive, 182
- somme, 154
- source, 217
- sous-requête, 150, 152, 178
- spécification, 50, 51
- SQL, 123
- start with, 157
- statiques, 187
- statistiques, 259
- stéréotype, 65
- stéréotypes, 23
- Stéréotypes, 53
- stockage, 256
- structure, 93
- structuré, 10
- structure de table, 100, 101, 103
- structures de données, 257
- Subquery, 152
- subquery-CREATE, 182
- substitution, 243

- sum, 154
- suppression, 174, 175
- Suppression, 180
- symboles
 - non-terminaux, 96
 - terminaux, 96
- syntaxe SQL, 96
- système de déduction, 202
- systèmes informatiques, 94
- table séquentielle, 261
- table-name.*, 143
- tables, 100
- tautologie, 90
- tautologie, 88
- terme, 144
- théorèmes, 88
- théta-jointure, 127
- tiers exclu, 90
- traductions, 118
- transaction**, 59, 171, 176, 180
 - annulée, 181
 - confirmée, 181
 - erronée, 177
 - inachevée, 176
- transaction, 186
- transaction-command, 180
- transitivité, 202
- tri, 158
- tri, 141
- trivial, 185
- type, 84, 101
 - booléen, 86
 - date, 86
 - domaine, 188
 - mot, 85
 - mot ordonné, 85
 - numérique, 86
 - texte, 85
- types numériques, 102
- typologie des domaines, 85
- ull scan table, 262
- unicité, 190
- union*, 124, 137, 156
- Union**, 204
- unique, 263
- Unique, 192
- Unique scan, 263
- update, 179
- Update, 107
- Update-Clause, 141
- Update-command, 179
- utilisateurs, 50
- valeur par défaut, 101
- validation
 - globale, 191
 - locale, 191
- valide, 185
- validité, 203
- validité des données, 86
- variable libre, 89
- variables, 20, 88
 - libres, 84
- vérification, 184
- verrouillage, 141
- verrous, 177, 178, 181
- version, 45
- vide, 148
- VIEW, 244
- vrai, 87
- vue, 115, 243, 268, 270
- vues, 107
- WHERE**, 137, 146, 154, 179, 243
- Wirth, 96
- with check option, 246

