Conception d'une structure de données dans les environnements de bases de données

Michel Leonard

THESE

Présentée à

L'UNIVERSITE Joseph FOURIER (GRENOBLE I)

pour obtenir le grade de
DOCTEUR D'ETAT ES-SCIENCES
spécialité :
"Informatique"

par

Michel LEONARD

ooooo

Conception d'une structure de données dans les environnements de bases de données

ooooo

Thèse soutenue le 9 mai 1988 devant la commission d'examen.

C. BENZAKEN	**Président**
J.L. HAINAUT	**Rapporteur**
J. KOULOUMDJAN	**Rapporteur**
C. DELOBEL	**Directeur de thèse**
M. ADIBA	
F. BODARD	

Laboratoire de Génie Informatique

UNIVERSITE Joseph FOURIER (GRENOBLE I)

Président de l'Université :
M. PAYAN Jean Jacques

Année Universitaire 1987 - 1988

MEMBRES DU CORPS ENSEIGNANT DE SCIENCES ET DE GEOGRAPHIE

PROFESSEURS DE 1ère Classe

ARNAUD Paul	Chimie Organique
ARVIEU ROBERT	Physique Nucléaire I.S.N.
AUBERT Guy	Physique C.N.R.S
AURIAULT Jean-Louis	Mécanique
AYANT Yves	Physique Approfondie
BARBIER Marie-Jeanne	Electrochimie
BARJON Robert	Physique Nucléaire ISN
BARNOUD Fernand	Biochimie Macromoléculaire Végétale
BARRA Jean-René	Statistiques-Mathématiques Appliquées
BECKER Pierre	Physique
BEGUIN Claude	Chimie Organique
BELORISKY Elie	Physique
BENZAKEN Claude	Mathématiques Pures
BERARD Pierre	Mathématiques Pures
BERNARD Alain	Mathématiques Pures
BERTRANDIAS Françoise	Mathématiques Pures
BERTRANDIAS Jean-Paul	Mathématiques Pures
BILLET Jean	Géographie
BOELHER Jean-Paul	Mécanique
BONNIER Jane Marie	Chimie Générale
BOUCHEZ Robert	Physique Nucléaire ISN
BRAVARD Yves	Géographie
CARLIER Georges	Biologie Végétale
CAUQUIS Georges	Chimie Organique
CHARDON Michel	Géographie
CHIBON Pierre	Biologie Animale
COHEN ADDAD Jean-Pierre	Physique
COLIN DE VERDIERE Yves	Mathématiques Pures
CYROT Michel	Physique du Solide
DEBELMAS Jacques	Géologie Générale
DEGRANGE Charles	Zoologie
DEMAILLY Jean-Pierre	Mathématiques Pures
DENEUVILLE Alain	Physique
DEPORTES Charles	Chimie Minérale
DOLIQUE Jean-Michel	Physique des Plasmas
DOUCE Roland	Physiologie Végétale
DUCROS Pierre	Cristallographie
FONTAINE Jean-Marc	Mathématiques Pures
GAGNAIRE Didier	Chimie Physique
GERMAIN Jean-Pierre	Mécanique,
GIRAUD Pierre	Géologie
HICTER Pierre	Chimie
IDELMAN Simon	Physiologie Animale
JANIN Bernard	Géographie
JOLY Jean-René	Mathématiques Pures
KAHANE André, détaché	Physique
KAHANE Josette	Physique
KRAKOWIAK Sacha	Mathématiques Appliquées

LAJZEROWICZ Jeanine	Physique
LAJZEROWICZ Joseph	Physique
LAURENT Pierre-Jean	Mathématiques Appliquées
LEBRETON Alain	Mathématiques Appliquées
DE LEIRIS Joël	Biologie
LHOMME Jean	Chimie
LLIBOUTRY Louis	Géophysique
LOISEAUX Jean-Marie	Sciences Nucléaires I.S.N.
LUNA Domingo	Mathématiques Pures
MACHE Régis	Physiologie Végétale
MASCLE Georges	Géologie
MAYNARD Roger	Physique du Solide
OMONT Alain	Astrophysique
OZENDA Paul	Botanique (Biologie Végétale)
PAYAN Jean-Jacques	Mathématiques Pures
PEBAY-PEYROULA Jean-Claude	Physique
PERRIER Guy	Géophysique
PIERRARD Jean-Marie	Mécanique
PIERRE Jean-Louis	Chimie Organique
RENARD Michel	Thermodynamique
RINAUDO Marguerite	Chimie CERMAV
ROSSI André	Biologie
SAXOD Raymond	Biologie Animale
SENGEL Philippe	Biologie Animale
SERGERAERT Francis	Mathématiques Pures
SOUCHIER Bernard	Biologie
SOUTIF Michel	Physique
STUTZ Pierre	Mécanique
TRILLING Laurent	Mathématiques Appliquées
VALENTIN Jacques	Physique Nucléaire I.S.N.
VAN CUTSEM Bernard	Mathématiques Appliquées
VIALON Pierre	Géologie

PROFESSEURS de 2ème Classe

ADIBA Michel	Mathématiques Pures
ANTOINE Pierre	Géologie
ARMAND Gilbert	Géographie
BARET Paul	Chimie
BLANCHI J.Pierre	STAPS
BLUM Jacques	Mathématiques Appliquées
BOITET Christian	Mathématiques Appliquées
BORNAREL Jean	Physique
BRUANDET J.François	Physique
BRUGAL Gérard	Biologie
BRUN Gilbert	Biologie
CASTAING Bernard	Physique
CERFF Rudiger	Biologie
CHIARAMELLA Yves	Mathématiques Appliquées
COURT Jean	Chimie
DUFRESNOY Alain	Mathématiques Pures
GASPARD François	Physique
GAUTRON René	Chimie
GENIES Eugène	Chimie
GIDON Maurice	Géologie
GIGNOUX Claude	Sciences Nucléaires
GILLARD Roland	Mathématiques Pures
GIORNI Alain	Sciences Nucléaires
GONZALEZ SPRINBERG Gérardo	Mathématiques Pures
GUIGO Maryse	Géographie
GUMUCHAIN Hervé	Géographie
GUITTON Jacques	Chimie

HACQUES Gérard	Mathématiques Appliquées
HERBIN Jacky	Géographie
HERAULT Jeanny	Physique
JARDON Pierre	Chimie
JOSELEAU Jean-Paul	Biochimie
KERCKHOVE Claude	Géologie
LONGEQUEUE Nicole	Sciences Nucléaires I.S.N.
LUCAS Robert	Physique
MANDARON Paul	Biologie
MARTINEZ Francis	Mathématiques Appliquées
NEMOZ Alain	Thermodymamique CNRS - CRTBT
OUDET Bruno	Mathématiques Appliquées
PECHER Arnaud	Géologie
PELMONT Jean	Biochimie
PERRIN Claude	Sciences Nucléaires I.S.N.
PFISTER Jean-Claude	Physique du Solide
PIBOULE Michel	Géologie
RAYNAUD Hervé	Mathématiques Appliquées
RICHARD Jean-Marc	Physique
RIEDTMANN Christine	Mathématiques Pures
ROBERT Gilles	Mathématiques Pures
ROBERT Jean-Bernard	Chimie Physique
SARROT-REYNAULD Jean	Géologie
SAYETAT Françoise	Physique
SERVE Denis	Chimie
STOECKEL Frédéric	Physique
SCHOLL Pierre-Claude	Mathématiques Appliquées
SUBRA Robert	Chimie
VALLADE Marcel	Physique
VIDAL Michel	Chimie Organique
VIVIAN Robert	Géographie
VOTTERO Philippe	Chimie

MEMBRES DU CORPS ENSEIGNANT DE L' IUT 1

PROFESSEURS de 1ère Classe

BUISSON Roger	Physique IUT 1
DODU Jacques	Mécanique Appliquée IUT 1
NEGRE Robert	Génie Civil IUT 1
NOUGARET Marcel	Automatique IUT 1
PERARD Jacques	EEA. IUT 1

PROFESSEURS de 2ème classe

BOUTHINON Michel	EEA. IUT 1
CHAMBON René	Génie Mécanique IUT 1
CHEHIKIAN Alain	EEA. IUT 1
CHENAVAS Jean	Physique IUT 1
CHOUTEAU Gérard	Physique IUT 1
CONTE René	Physique IUT 1
GOSSE Jean-Pierre	EEA.IUT 1
GROS Yves	Physique IUT 1
KUHN Gérard, (Détaché)	Physique IUT 1
MAZUER Jean	Physique IUT 1
MICHOULIER Jean	Physique IUT 1
MONLLOR Christian	EEA.IUT 1
PEFFEN René	Métallurgie IUT 1
PERRAUD Robert	Chimie IUT 1
PIERRE Gérard	Chimie IUT 1
TERRIEZ Jean-Michel	Génie Mécanique IUT 1
TOUZAIN Philippe	Chimie IUT 1
VINCENDON Marc	Chimie IUT 1

PROFESSEURS DE PHARMACIE

AGNIUS-DELORD Claudine	Physique	Faculté La Tronche
ALARY Josette	Chimie Analytique	Faculté La Tronche
BERIEL Hélène	Physiologie et Pharmacologie	Faculté La Tronche
CUSSAC Max	Chimie Therapeutique	Faculté La Tronche
DEMENGE Pierre	Pharmacodynamie	Faculté La Tronche
FAVIER Alain	Biochimie	C.H.R.G.
JEANNIN Charles	Pharmacie Galénique	Faculté Meylan
LATURAZE Jean	Biochimie	Faculté La Tronche
LUU DUC Cuong	Chimie Générale	Faculté La Tronche
MARIOTTE Anne-Marie	Pharmacognosie	Faculté La Tronche
MARZIN Daniel	Toxicologie	Faculté Meylan
RENAUDET Jacqueline	Bactériologie	Faculté La Tronche
ROCHAT Jacques	Hygiéne et Hydrologie	Faculté La Tronche
SEIGLE-MURANDI Françoise	Botanique et Cryptogamie	Faculté Meylan
VERAIN Alice	Pharmacie Galénique	Faculté Meylan

MEMBRES DU CORPS ENSEIGNANT DE MEDECINE

PROFESSEURS CLASSE EXEPTIONNELLE ET 1ère CLASSE

AMBLARD Pierre	Dermatologie	C.H.R.G.
AMBROISE-THOMAS Pierre	Parasitologie	C.H.R.G.
BEAUDOING André	Pédiatrie-Puericulture	C.H.R.G.
BEZEZ Henri	Orthopédie-Traumatologie	Hopital SUD
BONNET Jean-Louis	Ophtalmologie	C.H.R.G.
BOUCHET Yves	Anatomie	Faculté La Merci
	Chirurgie Générale et Digestive	C.H.R.G.
BUTEL Jean	Orthopédie-Traumatologie	C.H.R.G.
CHAMBAZ Edmond	Biochimie	C.H.R.G.
CHAMPETIER Jean	Anatomie-Topographique et Appliquée	C.H.R.G.
CHARACHON Robert	O.R.L.	C.H.R.G.
COLOMB Maurice	Immunologie	Hopital sud
COUDERC Pierre	Anatomie-Pathologique	C.H.R.G.
DELORMAS Pierre	Pneumophtisiologie	C.H.R.G.
DENIS Bernard	Cardiologie	C.H.R.G.
GAVEND Michel	Pharmacologie	Faculté La Merci
HOLLARD Daniel	Hématologie	C.H.R.G.
LATREILLE René	Chirurgie Thoracique et Cardiovasculaire	C.H.R.G.
LE NOC Pierre	Bactériologie-Virologie	C.H.R.G.
MALINAS Yves	Gynécologie et Obstétrique	C.H.R.G.
MALLION Jean-Michel	Médecine du Travail	C.H.R.G.
MICOUD Max	Clinique Médicale et Maladies Infectieuses	C.H.R.G.
MOURIQUAND Claude	Histologie	Faculté La Merci
PARAMELLE Bernard	Pneumologie	C.H.R.G.
PERRET Jean	Neurologie	C.H.R.G.
RACHAIL Michel	Hépato-Gastro-Entérologie	C.H.R.G.
DE ROUGEMONT Jacques	Neurochirurgie	C.H.R.G.
SARRAZIN Roger	Clinique Chirurgicale	C.H.R.G.
STIEGLITZ Paul	Anestésiologie	C.H.R.G.
TANCHE Maurice	Physiologie	Faculté La Merci
VIGNAIS Pierre	Biochimie	Faculté La Merci

PROFESSEURS 2ème CLASSE

BACHELOT Yvan	Endocrinologie	C.H.R.G.
BARGE Michel	Neurochirurgie	C.H.R.G.
BENABID Alim Louis	Biophysique	Faculté La Merci
BENSA Jean-Claude	Immunologie	Hopital Sud
BERNARD Pierre	Gynécologie-Obstétrique	C.H.R.G.
BESSARD Germain	Pharmacologie	ABIDJAN
BOLLA Michel	Radiothérapie	C.H.R.G.
BOST Michel	Pédiatrie	C.H.R.G.
BOUCHARLAT Jacques	Psychiatrie Adultes	Hopital Sud
BRAMBILLA Christian	Pneumologie	C.H.R.G.
CHIROSSEL Jean-Paul	Anatomie-Neurochirurgie	C.H.R.G.
COMET Michel	Biophysique	Faculté La Merci
CONTAMIN Charles	Chirurgie Thoracique et Cardiovasculaire	C.H.R.G.
CORDONNIER Daniel	Néphrologie	C.H.R.G.
COULOMB Max	Radiologie	C.H.R.G.
CROUZET Guy	Radiologie	C.H.R.G.
DEBRU Jean-Luc	Médecine Interne et Toxicologie	C.H.R.G.
DEMONGEOT Jacques	Biostatistiques et Informatique Médicale	Faculté La Merci
DUPRE Alain	Chirurgie Générale	C.H.R.G.
DYON Jean-François	Chirurgie Infantile	C.H.R.G.
ETERRADOSSI Jacqueline	Physiologie	Faculté La Merci
FAURE Claude	Anatomie et Organogénèse	C.H.R.G.
FAURE Gilbert	Urologie	C.H.R.G.
FOURNET Jacques	Hépato-Gastro-Entérologie	C.H.R.G.
FRANCO Alain	Médecine Interne	C.H.R.G.
GIRARDET Pierre	Anesthésiologie	C.H.R.G.
GUIDICELLI Henri	Chirurgie Générale et Vasculaire	C.H.R.G.
GUIGNIER Michel	Thérapeutique et Réanimation Médicale	C.H.R.G.
HADJIAN Arthur	Biochimie	Faculté La Merci
HALIMI Serge	Endocrinologie et Maladies Métaboliques	C.H.R.G.
HOSTEIN Jean	Hépato-Gastro-Entérologie	C.H.R.G.
HUGONOT Robert	Médecine Interne	C.H.R.G.
JALBERT Pierre	Histologie-Cytogénétique	C.H.R.G.
JUNIEN-LAVILLAUROY Claude	O.R.L.	C.H.R.G.
KOLODIE Lucien	Hématologie Biologique	C.H.R.G.
LETOUBLON Christian	Chirurgie Générale	C.H.R.G.
MACHECOURT Jacques	Cardiologie et Maladies Vasculaires	C.H.R.G.
MAGNIN Robert	Hygiène	C.H.R.G.
MASSOT Christian	Médecine Interne	C.H.R.G.
MOUILLON Michel	Ophtalmologie	C.H.R.G.
PELLAT Jacques	Neurologie	C.H.R.G.
PHELIP Xavier	Rhumatologie	C.H.R.G.
RACINET Claude	Gynécologie-Obstétrique	Hopital Sud
RAMBAUD Pierre	Pédiatrie	C.H.R.G.
RAPHAEL Bernard	Stomatologie	C.H.R.G.
SCHAERER René	Cancérologie	C.H.R.G.
SEIGNEURIN Jean-Marie	Bactériologie-Virologie	Faculté La Merci
SELE Bernard	Cytogénétique	Faculté La Merci
SOTTO Jean-Jacques	Hématologie	C.H.R.G.
STOEBNER Pierre	Anatomie Pathologique	C.H.R.G.
VROUSOS Constantin	Radiothérapie	C.H.R.G.

RÉSUMÉ

Les régles d'intégrité considérées comme des invariants, deviennent une aide pour le travail de modélisation. En plus des traditionnelles dépendances, dépendances fonctionnelles et dépendances de composition, sont introduites des dépendances d'inclusion particulières qui sont très utiles pour l'analyse de structures de données cycliques.

L'étude des structures de données cycliques conduit à proposer la notion de décomposition étanche d'une relation qui permet à un utilisateur d'avoir des réponses cohérentes et complètes à des questions posées simplement. Cette étude conduit à étendre les résultats connus de décomposition d'une relation à l'aide de dépendances fonctionnelles et à proposer de nouveaux mécanismes à rajouter aux fonctionnalités traditionnelles des sytèmes de gestion de bases de données.

Le cadre général de cette thèse est de permettre la prise en compte de bases de données ayant des structures complexes, de rendre plus efficace le processus de leur conception et de leur réalisation, de les faire évoluer et d'améliorer leur compréhension autant par leurs utilisateurs que par leurs responsables. Plusieurs prototypes ont été réalisés s'appuyant sur ces propositions.

MOTS CLÉS

bases de données, modèles de données, conception, décomposition, règles d'intégrité, cycle fonctionnel, structure d'une base de données, étanchéité d'une décomposition.

AVANT-PROPOS

1. Une nouvelle matière d'études

Notre objectif est de travailler une nouvelle matière que nous ne sommes pas les premiers à découvrir. Au fur et à mesure que les bases de données deviennent de plus en plus nombreuses dans le monde des organisations et des entreprises, il nous semble que cette matière va devenir de plus en plus reconnue.

Cette matière n'est pas une partie traditionnelle de la matière informatique. Son sujet principal concerne l'adéquation d'outils informatiques développés à l'aide de systèmes de gestion de bases de données, à un ou plusieurs processus de gestion d'une organisation. Cette matière n'est pas non plus une partie traditionnelle de la matière gestion d'organisations car son autre sujet principal est la réalisation de bases de données dans un environnement organisationnel. Cette matière nous apparaît plus comme une articulation entre la matière informatique et la matière gestion d'organisations. Elle est pour nous comme le passage obligé entre d'une part l'analyse d'un processus de gestion et son informatisation à l'aide d'une base de données et d'autre part l'analyse de l'adéquation d'une base de données à un processus de gestion. Nous appelons cette nouvelle matière l'*information structurée* et nous présentons une étude de ses propriétés pour faciliter les échanges entre les compétences informatiques et les compétences de gestion. Ainsi l'essence même de cette matière est pour nous celle de fournir un espace intellectuel commun aux deux matières qui lui sont connexes. En particulier elle doit permettre l'expression des résultats d'analyse d'un processus de gestion dans des termes facilement traduisibles en termes de spécifications informatiques et de réalisations informatiques, et inversement, elle doit permettre une facile compréhension du fonctionnement des applications informatiques travaillant sur une base de données par les acteurs du processus de gestion ainsi informatisé.

Nous allons limiter l'étude de cette matière aux bases nécessaires à la conception de structures de bases de données, bien que nous portions un grand intérêt aux travaux relatifs à la conception des traitements (ROLLAND78; BODART-PIGNEUR83; WASSERMAN83; GUYOT 86).

2. Plan commenté

La première partie concerne la modélisation d'un champ d'application. Après avoir introduit les concepts classiques du modèle relationnel de données (CODD70), nous présentons les règles d'intégrité et différentes variétés de dépendances. Notamment nous introduisons les dépendances de référence et les dépendances relatives qui sont à notre connaissance originales et qui sont essentielles pour l'étude de structures de données admettant des cycles. Nous tentons de montrer que l'utilité des règles d'intégrité ne se limite pas à assurer la cohérence de la base de données mais qu'elles sont des aides efficaces au difficile processus de modélisation d'un champ d'application qui comprend son observation, son analyse et l'expression des résultats d'analyse dans des termes précis. Cette partie se termine sur l'importance du dictionnaire de données qui va contenir la modélisation.

La seconde partie suppose que la modélisation obtenue précédemment puisse former le noyau de la structure de la base de données finale. Dans le cas de modélisations complexes, la troisième partie montrera qu'il est nécessaire de travailler la première modélisation et de la transformer dans une modélisation équivalente pour que les contrôles d'intégrité puissent être vérifiées efficacement.

Cette troisième partie étant nettement plus complexe, nous présentons d'abord, au cours de cette seconde partie, la transformation d'une modélisation relationnelle de données dans une structure informatique de données.

Au début de cette partie, nous montrons pouquoi à notre avis, le modèle relationnel de données ne peut pas être retenu comme un modèle informatique de stockage de données. Et nous présentons un tel modèle que nous appelons graphe de chemin d'accès (GCA). Dans cette seconde partie, nous fournissons une approche algorithmique de transformation d'une modélisation relationnelle de données dans un graphe de chemins d'accès puis dans une structure informatique de données. Nous sommes

amenés à fournir une représentation graphique équivalente d'un ensemble de relations que nous appelons graphe de relation (GR). Cette représentation est beaucoup plus simple que celle souvent utilisée des hypergraphes (BERGE70), et elle peut être considérée comme un hypergraphe particulier.

Dans la troisième partie, "Décomposition d'une relation", nous étudions la qualité de la modélisation relationnelle initiale par rapport à l'utilisation de la future base de données.

La notion essentielle ici est celle de décomposition d'une relation.
Il s'agit de remplacer une relation R par un ensemble de relations (R_1... R_i...R_n) telle que potentiellement toute information stockée dans R puisse être stockée dans (R_1... R_i...R_n) et réciproquement.

Comme nous l'avons montré dans la première partie, ce sont les dépendances qui vont permettre des décompositions de relations. L'étude des décompositions nous a demandé l'étude de l'extension et inversement de la projection de règles d'intégrité. Ensuite nous présentons les décompositions franches (la validité des règles d'intégrité au niveau de chacune des relations R_1... R_i...R_n suffit à garantir la validité au niveau de R) et les décompositions étanches (un utilisateur qui veut extraire des données de la base peut considérer que les n relations sont indépendantes et ainsi oublier l'existence de R).

Nous montrons, dans un cas spécial, comment il est possible de construire des décompositions franches et comment il est parfois impossible d'obtenir une décomposition franche et étanche.

Cette situation nous conduit à revenir sur une solution avancée communément (BERNSTEIN76) comme une solution satisfaisante d'obtention d'une structure efficace de base de données.
Nous sommes amenés à introduire de nouveaux mécanismes qui devraient faire partie des systèmes de gestion de bases de données et qui permettraient de rendre artificiellement étanches certaines décompositions.

La présence de ces mécanismes nous permet d'avancer une autre solution d'obtention d'une structure efficace de bases de données, qui nous semble posséder entre autres l'avantage suivant sur la solution précédente : celui de placer le concepteur de base de données devant des choix qui peuvent s'exprimer seulement en des termes d'informations et non en des termes techniques.

La quatrième partie est une partie de perspectives:

- elle met l'accent sur la nécessité de disposer de modèles de données plus intelligents, non seulement pour la phase de modélisation, mais également pour l'implantation informatique. Nous citons alors notre tentative concrétisée par le SGBD ÉCRINS (LÉONARD-GALLAND-JUNET-TSCHOPP85) et des tentatives d'autres équipes (par exemple TIGRE (VELEZ-LOPEZ87));

- elle met l'accent sur la nécessité d'augmenter les fonctions d'un SGBD pour que l'implantation d'applications informatiques utilisant des SGBD soit efficace. Les mécanismes mis en évidence dans la troisième partie sont englobés dans d'autres mécanismes tous aussi nécessaires et nous montrons par la démonstration d'un théorème en quoi tous ces mécanismes seraient très utiles.

3. Objectif fondamental

Nous proposons l'étude de points que nous considérons comme fondamentaux dans la conception d'une structure d'une base de données. Nous pensons qu'ils seront utiles pour tout concepteur de bases de données notamment lorsque celles-ci deviendront de plus en plus complexes.

Par contre nous ne proposons aucune méthode (qui proposerait différentes étapes de conception et les rangerait dans un ordre chronologique en précisant comment passer de l'une à l'autre dans un contexte forcément itératif), aucun outil, aucune méthodologie (qui devrait englober non seulement des aspects de conduite de projets informatiques mais également des aspects cognitifs liés aux processus de modélisation, un processus d'intégration des outils informatiques à l'organisation, un processus de suivi de cette intégration, un autre de réorganisation et de restructuration, des aspects de dynamique des groupes des personnes concernées par la base de données).

Comme notamment (BENCI-ROLLAND79; FLORY82; DELOBEL-ADIBA82; GALACSI86; HAINAUT86), nous allons essayer de contribuer à la mise en place d'une *base* de connaissances pour la conception et réalisation de bases de données où les utilisateurs auront plaisir à déposer leurs données et à revenir les chercher.

Nous souhaitons remercier:

le professeur C.BENZAKEN de nous faire l'honneur de présider le jury de notre thèse;

le professeur M. DELOBEL d'avoir été le directeur de notre thèse;

les professeurs J-L HAINAUT et J. KOULOUMDJAN d'avoir bien voulu être les rapporteurs de notre thèse;

les professeurs M. ADIBA et F.BODART de faire partie du jury de notre thèse;

les chercheurs de l'équipe de bases de données de l'Université de Genève pour tout le travail fait ensemble et pour toute leur attitude chaleureuse compétente et efficace;

les chercheurs du LGI de Grenoble pour toute l'ambiance sympathique et scientifique qu'ils nous ont fait partager;

Mmes E. KOHL et L. NOEL, M. P-A. MARTI pour leurs compétences bienveillantes à l'égard de notre manuscrit et de nos disquettes;

Marilyne, Marion, Gabrielle, Gilbert, Daniel, Sylvia, Véronique, Marc, Réginald, Jef,...

M. Léonard

ABRÉVIATIONS UTILISÉES

Si R désigne une relation alors:

R^+	désigne l'ensemble des constituants de R.
$\|\|R\|\|$	désigne le prédicat de R.
r	désigne une entité de R.
+R	désigne une instance de R, c'est-à-dire un ensemble. d'entités de R.
KR	désigne une clé de R.
KR^*	désigne l'ensemble des constituants de R qui. appartiennent à au moins une clé de R.

Si dans le contexte d'un paragraphe aucune confusion n'est à craindre avec d'autres relations alors: r'r''r_i désignent également des entités de R.

Si A désigne un constituant, alors: a désigne une valeur de son domaine; si aucune confusion n'est à craindre avec d'autres constituants, alors a'a''a_i désignent également d'autres valeurs de A.

Si D est une décomposition alors: D^+ désigne l'ensemble des constituants des relations de D.

Si f est une dépendance fonctionnelle alors:

f^+ désigne l'ensemble des constituants de f,
g(f) désigne la partie gauche de f,
d(f) désigne la partie droite de f.

dc	:	dépendance de composition.
dd	:	dépendance de dimensionnement.
dec	:	décomposition.
df	:	dépendance fonctionnelle.
GCA	:	graphe de chemins d'accès.
GCA_b	:	graphe de chemins d'accès brut.
GR	:	graphe de relation.
ri	:	règle d'intégrité.
SGBD	:	système de gestion de bases de données.

Notations:

fex	:	fin exemple.
*	:	fin de démonstration.

TABLE DES MATIÈRES

DEUXIÈME PARTIE
TRANSFORMATIONS

PREMIÈRE PARTIE

MODÉLISATION DE DONNÉES

1

UN MODÈLE RELATIONNEL DE DONNÉES

1.1. Concepts du modèle relationnel de données

1.1.1. Domaine

Un *domaine* est un ensemble D non vide. Pour exprimer que l'élément a appartient à D, nous écrivons: $a \in D$. Nous appelons les éléments d'un domaine, ses *valeurs* ou ses *objets*. Un domaine D est *composé* s'il est défini à l'aide de plusieurs domaines $D_1 ... D_n$. Son ensemble est le produit cartésien des ensembles $D_1 ... D_n$ diminué éventuellement d'éléments vérifiant une condition précise.

Le *type* d'un domaine indique les opérations qui sont définies sur ses éléments; en voici une liste non exhaustive.
Un domaine est de type *texte* si aucune opération n'est autorisée.
Il est de type *mot* si les opérations de comparaison "égalité" et "différence" y sont définies; un cas particulier du type mot est le type *mot ordonné*: alors, en plus des opérations précédentes, les opérations de comparaison "inférieur", "supérieur", y sont définies; si la relation d'ordre n'est pas reconnue comme évidente, on doit la définir (par exemple: la note 4 est-elle une meilleure note que 5 pour un examen du code de la route en France?
Il est de type *numérique* si en plus des opérations précédentes, les opéra-

tions d'addition et de soustraction y sont définies. Plutôt que numérique, qui peut ne pas être assez précis, on peut caractériser le type par entier, entier positif, décimal, réel, etc.
Il est de type *booléen* s'il ne contient que deux valeurs et si les opérations de l'algèbre de Boole (*non, et , ou*) y sont définies.
Il est de type *monôme* s'il est composé de plusieurs domaines de type booléen.
Pour des domaines de type numérique, et éventuellement de type mot et mot ordonné, il peut être nécessaire de préciser l'*unité* relative aux valeurs du champ (km, kg, g, mn, ...).
Pour *définir* un domaine, on peut préciser les valeurs qu'il contient par leur énumération dans le cas de domaines de type mot ou mot ordonné ou booléen, ou par des intervalles de définition dans le cas des domaines de type numérique ou mot ordonné, ou par un formalisme mathématique ou algorithmique. On doit toujours fournir le type d'un domaine.

Cas particulier:
Il existe des domaines utilisés très fréquemment dans les applications de gestion et dont il est inutile d'en donner la définition à chaque fois. En voici quelques exemples:

dom NOJOUR	:	mot ordonné croissant (1,31).
dom MOIS	:	mot (janvier, février, mars, avril, mai, juin, juillet, août, septembre,octobre, novembre, décembre).
dom ANNÉE	:	mot ordonné croissant (1900, -).
dom DATE	:	mot ordonné composé de NOJOUR MOIS ANNÉE moins "jours farfelus".

"jours farfelus" est le résultat d'un algorithme qui fournit l'ensemble des jours farfelus comme par exemple le mardi 30 février 1985.

dom HEURE	:	mot ordonné croissant (0,23).
dom MINUTE	:	mot ordonné croissant (1,59).
dom HORAIRE	:	mot ordonné composé de HEURE MINUTE.

1.1.2. Constituant (ou attribut)

Un *constituant* est une classe de données qui ont toutes un comportement homogène dans la base de données; leur signification réside seulement dans leur appartenance à cette classe.

Chaque constituant est associé à un et un seul domaine.

Une donnée appartenant à la classe d'un constituant est représentée dans la base de données par l'une des valeurs du domaine de ce constituant.

Un constituant est *composé* si son domaine est composé ou s'il est défini à partir de plusieurs constituants; alors son domaine est un sous-ensemble du produit cartésien des domaines de ces constituants; ce sous-ensemble est déterminé par un processus algorithmique (éventuellement ce sous-ensemble est égal au produit cartésien lui-même).

Notation:
Soit c une valeur du constituant C. Si le constituant C est composé des constituants $C_1...C_n$, alors les notations $c.C_1$ et $c(C_1)$ désignent la valeur prise par c pour le constituant C_1. Maintenant si le constituant C admet un domaine composé des domaines $D_1...D_n$ alors $c.D_1$ désigne la valeur prise par la valeur c pour le domaine D_1.

1.1.3. Relation

Une *relation* n-aire R est définie sur le produit cartésien des domaines de *n* constituants formant l'ensemble R^+ des constituants de R. Elle est définie par un prédicat noté ||R|| dont les variables libres (c'est-à-dire qui ne sont quantifiées par aucun des deux quantificateurs $\exists$ $\forall$) correspondent aux constituants de R^+ et prennent leurs valeurs dans les domaines de ces constituants.

Un *n-uplet* de R est un élément du produit cartésien des domaines des *n* constituants de R.
Une *entité* r de la relation R est un n-uplet de R qui valide le prédicat:
|| R || (r) = vrai.

Remarque:
Le fait qu'un n-uplet du produit cartésien soit ou ne soit pas une entité de la relation R ne relève d'aucun processus algorithmique: c'est l'utilisateur qui en prend la responsabilité par le fait qu'il demande ou ne demande pas le stockage de ce n-uplet dans la base. Voilà la différence fondamentale pour nous entre les concepts de relation et de constituant composé.

Notation:
Si r est une entité de R (ou un n-uplet du produit cartésien PR) $r.A_i$ ou $r(A_i)$ ou $r[A_i]$ désigne la valeur prise par r pour le constituant A_i. Si X^+ désigne un ensemble de constituants de R^+ , $r.X^+$ désigne les valeurs prises par r pour les différents constituants de X^+ .

1.1.4. Valeurs inconnues

Au moment de la définition d'une relation R, il s'agit de préciser pour chaque constituant si les entités de R peuvent prendre la valeur inconnue pour lui.
Une entité r de R est *claire* si pour tout constituant A de R, r ne prend pas la valeur inconnue. Elle est *obscure* s'il existe au moins un constituant pour lequel elle prend une valeur inconnue.

1.1.5. Instance

Une *instance* d'une relation R est un ensemble d'entités de cette relation que nous notons iR (et éventuellement plus simplement R s'il n'y a pas d'ambiguïté possible avec la relation R).
Il existe une instance particulière de R, celle qui regroupe toutes les entités claires de R qui ont été stockées dans la base pendant toute sa durée de vie. Nous l'appelons la *fermeture* de R. Du point de vue formel, elle est équivalente au prédicat de R puisqu'elle contient l'ensemble de tous les n-uplets de R qui valident ce prédicat. Nous la notons simplement R. Nous nous intéresserons seulement aux instances d'une relation formée d'entités claires.
Nous notons: $r \in R$ pour indiquer que le n-uplet r du produit cartésien des domaines des constituants de R est une entité de R.
Une *instance* d'une base de données formée des relations $R_1...R_n$ est un ensemble d'instances de relation, une pour chacune des relations $R_1...R_n$.

1.1.6. Clé

Une *clé* d'une relation R est un ensemble minimal de constituants de R tels que si deux entités de R prennent les mêmes valeurs pour ces constituants dans une même instance, elles sont identiques.
Une clé K de R est *obligatoire* si pour toute entité r de R et pour tout constituant A de K, r.A doit toujours être claire.
Nous supposerons que toute relation admet au moins une clé éventuellement formée de tous les constituants de la relation. Elle peut admettre plusieurs clés. Nous donnerons une définition plus formelle de la notion de clé dans le § 2.4.3.
Ainsi, une clé va permettre de distinguer entre elles les entités d'une même relation R suivant les valeurs qu'elles prennent pour les constituants formant cette clé.

Il se dégage une règle implicite: toute relation doit avoir au moins une clé obligatoire. Si la relation R admet une seule clé K alors elle est obligatoire et aucune entité de R ne doit prendre la valeur inconnue pour un constituant de la clé K.

Une clé permet d'identifier les entités d'une relation et c'est en ce sens que nous préfèrons le mot *identifiant* d'une relation à la place du mot clé, comme le propose (HAINAUT86).

1.1.7. Relation d'ordre définie sur les relations

$R < S$ si et seulement si la fermeture de R est incluse dans celle de S. C'est une relation d'ordre partiel. L'égalité de deux relations provient de l'égalité de leurs prédicats et donc de leurs fermetures.

1.2. Opérations définies sur un ensemble de relations

Les démonstrations des propriétés de ce chapitre se trouvent en annexe.

1.2.1. Opérations usuelles définies sur un ensemble de relations

1.2.1.1. Produit et composition de deux relations

Le produit de deux relations R et S est une nouvelle relation T telle que $\|T\| = \|R\| \wedge \|S\|$ et $T^+ = R^+ \parallel S^+$:
tous les constituants de R et de S sont marqués dans T^+ par le nom de leur relation d'origine: ainsi si A est un constituant de R et de S , il donne lieu à deux constituants de T: A de R et A de S.
Nous notons: T = R**S.

La composition de deux relations R et S est une nouvelle relation T telle que $T^+ = R^+ \cup S^+$ (union des ensembles de constituants de R^+ et de S^+) et $\|T\| = \|R\| \wedge \|S\|$.
Nous notons: T = R*S.

Remarque:

que T soit obtenue par produit ou par composition des deux relations R et S, toute entité t de T vérifie que $t.R^+$ est une entité de R et $t.S^+$ est une entité de S.

La *charnière* de R et de S est l'ensemble des constituants communs à R et à S; nous la notons $RS^+ = R^+ \wedge S^+$.

Si RS^+ est vide, il n'y a aucune différence entre le produit et la composition. Sinon, toute entité de T obtenue par composition vérifie: $t.RS^+ = r.RS^+ = s.RS^+$.

1.2.1.2. Somme de deux relations

La somme de deux relations R et S donne une nouvelle relation T telle que $T^+ = R^+ \cup S^+$ et $\|T\| = \|R\| \cup \|S\|$.
Nous notons: T = R + S.
Ainsi toute entité t de T vérifie que:
$t.R^+$ est une entité de R ou $t.S^+$ est une entité de S.
Comme l'opération d'union d'ensembles et celle du ou logique de prédicats sont des opérations associatives et commutatives, l'opération de somme de relations est une opération associative et commutative.

1.2.1.3. Complément d'une relation

Le complément de la relation R est une nouvelle relation S telle que
$S^+ = R^+$ et $\forall s \in S$ $s.R^+$ n'est pas une entité de R.
Nous notons: S = non R.

1.2.1.4. Projection d'une relation

La projection d'une relation R sur un ensemble de constituants X n'est définie que si X est inclus dans R^+ . Dans ce cas, c'est une nouvelle relation S qui est définie sur l'ensemble des constituants $S^+ = X$ et dont le prédicat $\|S\|$ vérifie: $\forall s \in S \; \exists r \in R$ telle que r.X = s.
Nous disons que s est la projection de r sur X^+ .
Nous notons: S = R[X].

1.2.1.5. Sélection d'une relation

La sélection d'une relation R par un prédicat P n'est définie que si les

variables de P sont des constituants de R ($P^+ \subset R^+$) et que si elles sont libres. Cette opération construit une nouvelle relation T telle que son ensemble de constituants est celui de R: $T^+ = R^+$ et que son prédicat vérifie $\|T\| = \|R\| \wedge P$.
Nous notons: T = (*P)R.

Propriété:
Au prédicat P, il est possible d'associer une relation PR telle que
$PR^+ = P^+$ et $\|PR\| = P$.
Alors T = (*P)R = PR*R.

Cette propriété nous permet de ne pas considérer l'opération de sélection dans l'étude suivante des propriétés des opérations relationnelles.

1.2.1.6. P-produit de deux relations

Le P-produit de deux relations R et S est défini à l'aide d'un prédicat P dont les variables P^+ sont des constituants de R ou de S et sont libres. C'est une nouvelle relation T telle que
$T^+ = R^+ \| S^+$ et $\|T\| = \|R\| \wedge \|S\| \wedge P$.
Nous notons: T = R (**P) S.

*Propriété (**P):*

R (**P) S = (*P) (R**S).

En effet, ces deux relations sont définies sur les mêmes constituants et ont le même prédicat.
Cette propriété nous permet de ne pas considérer le P-produit dans l'étude suivante des propriétés des opérations relationnelles.

Il est également possible de définir la *P-composition* de deux relations.

1.2.2. Relations formellement remarquables

U^+ est un ensemble de constituants qui contient tous les autres ensembles.
La *relation 1* est définie sur U^+ et son prédicat est toujours vérifié; la relation 1_X désigne la projection de 1 sur l'ensemble des constituants X.
La *relation 0* est définie sur U^+ et son prédicat n'est jamais vérifié; la relation 0_X désigne la projection de 0 sur X.
L'*élargissement* d'une relation R est une nouvelle relation notée R^e définie sur U^+ tel que $R^e = 1*R$. On dit que R^e est la *relation élargie* de R.

1.2.2.1. Propriétés de l'élargissement

(e1) $R^e[R^+] = R$.
(e11) $1[X] = 1_X$ et $(1_X)^e = 1$.
(e12) $0[X] = 0_X$ et $(0_X)^e = 0$.
(e2) Si $X \subset R^+$ alors $R^e[X] = R[X]$.
(e21) Si $X \wedge R^+ = \varnothing$ alors $R^e[X] = 1_X$.
(e3) $(R*S)^e = R^e*S^e$.
(e31) $(R^e*S^e) = R^e*S = R*S^e$.
(e4) $(R+S)^e = R^e+S^e$.
(e41) $(R^e+S^e) = R^e+S = R+S^e$.
(e5) $(\text{non } R)^e = \text{non}(R^e)$.
(e6) $((*P)R)^e = (*P)(R^e)$.
(e7) Si $R^+=S^+$ alors $(R=S) \Leftrightarrow (R^e=S^e)$.

1.2.2.2. Algèbre de Boole de relations

Soit (U) l'ensemble des relations définies sur U^+ ou sur un sous-ensemble de U^+ et soit (U^e) l'ensemble des relations élargies de relations de (U). La fermeture $(U^e)^*$ de (U^e) désigne l'ensemble des relations qui peuvent être construites à partir des relations de (U^e) à l'aide des opérations somme, composition, complément.

Théorème (e8):

Les opérations somme et composition de relations forment une algèbre de Boole définie sur $(U^e)^*$ car:

- les opérations *et, ou* définies sur les prédicats sont associatives, commutatives et doublement distributives l'une par rapport à l'autre: donc il en est de même pour la somme et la composition de deux relations de $(U^e)^*$;
- l'idempotence de la somme et de la composition de deux relations de $(U^e)^*$ provient de l'idempotence des opérations *et* et *ou* définies sur les prédicats;
- l'élément neutre de la somme est la relation 0 et celui de la composition est la relation 1;
- non R est le complément de la relation R de (U^e) car l'on peut facilement vérifier que:

$$\text{non } R + R = 1,$$
$$\text{non } R * R = 0.$$

Corollaire (e9):

Voici la liste des propriétés remarquables d'une algèbre de Boole qui s'appliquent à l'algèbre de Boole des relations de $(U^e)^*$ comme un cas particulier:

$1 + R = 1.$
$0 * R = 0.$
$R + (R * S) = R.$
$R + (\text{non } R * S) = R + S.$
$R = R * S$ est équivalent à $R + S = S.$
Le complément d'une relation est unique.
Les éléments neutres sont uniques.
$\text{non } (\text{non } R) = R.$
$\text{non } (R + S) = (\text{non } R) * (\text{non } S).$
$\text{non } (R * S) = (\text{non } R) + (\text{non } S).$

1.2.2.3. Opération focus

L'opération focus permet autant la projection d'une relation R sur un ensemble de constituants X inclus dans R^+, que l'élargissement de la relation R à un ensemble de constituants Y contenant R^+.
De manière générale, elle définit une nouvelle relation: focus (R/X) à partir d'une relation R et d'un ensemble de constituants X inclus dans U^+ de la manière suivante:

$$\text{focus}(R/X) = R^e[X].$$

Propriétés de l'opération focus:

(f1) focus $(R/U^+) = R^e$ (élargissement);
si $X \subset R^+$ alors focus $(R/X) = R[X]$ (projection).
(f2) focus $(1_x/Y) = 1_y$ (avec $Y \subset U^+$).
focus $(0_x/Y) = 0_y$.
(f3) focus $(R*S/X) \subset$ focus $(R/X) *$ focus (S/X);
si X contient la charnière $R^+ \wedge S^+$ alors
focus $(R*S/X) =$ focus $(R/X) *$ focus (S/X).
(f4) focus $(R+S/X) =$ focus $(R/X) +$ focus (S/X).
(f5) non focus $(R/X) \subset$ focus (non R/X);
si X contient R^+ alors non focus $(R/X) =$ focus (non R/X).
(f6) Si $Y \subset X \subset R^+$ alors
focus (focus $(R/X)/Y) =$ focus$(R/Y) = R[Y]$.
Si X contient R^+ alors focus (focus$(R/X)/Y) =$ focus (R/Y).

1.2.3. Propriétés des opérations usuelles relationnelles

Soit (U) l'ensemble des relations définies sur U^+ ou sur un sous-ensemble de U^+ et soit (U)* la fermeture de (U) c'est-à-dire l'ensemble des relations qui peuvent être construites à partir des relations (U) à l'aide des opérations somme, composition, complément et projection.

1.2.3.1. Propriétés des opérations somme, composition, complément

(01) La somme et la composition de relations sont des opérations idempotentes, commutatives, associatives et doublement distributives l'une par rapport à l'autre. Ces propriétés dérivent directement des propriétés correspondantes des opérations *ou* et *et* définies sur les prédicats.

(02) Si $X = R^+$ alors non $R + R = 1_X$,
non $R * R = 0_X$,
$R * 0_X = 0_X$,
$R + 0_X = R * 1_X = R$,
$R + 1_X = 1_X$.

(03) $R + (R*S) = R$.
$R + (\text{non } R*S) = R + S$.

(04) non (non R) = R.
non (R+S) = non R * non S.
non (R*S) = non R + non S.

Les démonstrations des propriétés sont fournies en annexe; elles s'appuient sur l'algèbre de Boole des relations définies sur U^+ et sur la propriété (e7).

1.2.3.2. Propriétés des opérations somme, composition et complément par rapport à l'opération de projection

(p1) Si $Y \subset X \subset R^+$ alors (R[X])[Y] = R[Y].
(p2) Si $X \subset R^+ \wedge S^+$ alors $(R*S)[X] \subset R[X] * S[X]$;
si X est égal à la charnière de R et de S, alors il y a égalité.
(p3) Si $X \subset R^+ \wedge S^+$ alors (R+S)[X] = R[X]+S[X].
(p4) Si $X \subset R^+$ alors non $R[X] \subset$ (non R) [X].

Ces propriétés sont des corollaires des propriétés de l'opération focus (f3,f4,f5,f6).

1.2.3.3. Propriétés de l'opération de sélection

(s1) (*P) ((*Q)R) = (*(P∧Q))R.
(s2) Si P^+ est inclus dans X, (*P) (R[X]) = ((*P)R)[X].
(s3) ((*P)R) + ((*Q)R) = (*(P+Q))R.
(s4) Si P^+ est inclus dans R^+, alors (*P) (R*S) = ((*P)R) *S.
(s5) Si P^+ est inclus dans R^+ et dans S^+,
alors (*P) (R+S) = ((*P)R) + ((*P)S).
Ces propriétés découlent de la définition de l'opération de sélection (s1 et s3) et des propriétés (s, p2, associativité de la composition, distributivité de la somme et de la composition).

1.2.4. Expression relationnelle

Une expression relationnelle est construite à l'aide de relations, des opérateurs relationnels précédents, des prédicats pour les opérations de sélection, d'ensembles de constituants et de crochets pour l'opération de projection, et de parenthèses.

Une expression relationnelle *bien formée* se définit récursivement de la manière suivante:
une relation est une expression relationnelle bien formée;
soit X un ensemble de constituants et soit E_1 et E_2 deux expressions relationnelles bien formées et soit P_1 un prédicat, alors les expressions (E_1), non E_1, $E_1[X]$, $(*P_1)E_1$, $E_1 * E_2$, $E_1 + E_2$, $E_1 ** E_2$ sont bien formées.
Elles sont *bien construites* seulement si $X \subset E_1^+$ et $P_1^+ \subset E_1^+$.

Nous adoptons la convention suivante afin d'alléger l'écriture des expressions relationnelles:
a) Les opérateurs unaires sont plus prioritaires que les opérateurs binaires: non R * $T[X^+]$ signifie (non R) * ($T[X^+]$).
b) La projection est plus prioritaire que le complément et la sélection:
non $R[X^+]$ signifie non ($R[X^+]$).
c) Le complément et la sélection ont le même rang de priorité: l'opérateur écrit le plus proche de la relation est le plus prioritaire:
non (*P)R signifie non ((*P)R).
d) La composition est plus prioritaire que le produit:
R*S**T signifie (R*S)**T.
e) Le produit est plus prioritaire que la somme.
f) Les règles précédentes étant respectées, les opérations binaires sont effectuées de gauche à droite.

1.3. Opérations usuelles définies sur un ensemble d'instances

Le *produit* de deux instances iR et iS de deux relations R et S construit un ensemble de n-uplets N définis sur $T^+ = R^+ \parallel S^+$, tel que:
$\forall n \in N \quad n.R^+ \in iR$ et $n.S^+ \in iS$,
et $\forall r \in iR \; \forall s \in iS \; \exists n \in N$ tel que $n.R^+ = r$ et $n.S^+ = s$.
Nous notons: $N = iR ** iS$.
Il est facile de prouver que N est une instance de la relation $T = R**S$.

La *composition* de deux instances iR et iS, contenant des entités claires, des relations R et S construit un ensemble de n-uplets N définis sur $T^+ = R^+ \cup S^+$, tel que:
$\forall n \in N \quad n.R^+ \in iR$ et $n.S^+ \in iS$
et $\forall r \in iR, \forall s \in iS$ telles que $r.R^+ \wedge S^+ = s.R^+ \wedge S^+$, alors $\exists n \in N$ tel que $n.R^+ = r$ et $n.S^+ = s$.
Nous notons: $N = iR * iS$.
Il est facile de prouver que N est une instance de la relation $T = R*S$.

La *projection* d'une instance de iR de la relation R sur un ensemble de constituants S^+, n'est définie que si S^+ est inclus dans R^+; elle construit un ensemble de n-uplets N définis sur S^+ tel que:

$\forall n \in N, \exists r \in iR$ tel que $r.S^+ = n$
et $\forall r \in iR, \exists n \in N$ tel que $r.S^+ = n$.

Nous notons: $N = iR[S^+]$.
Il est facile de prouver que N est une instance de $S = R[S^+]$.

La *sélection* d'une instance iR de la relation R par un prédicat P n'est définie que si les variables de P sont des constituants de R ($P^+ \subset R^+$) et que si elles sont libres. Elle construit un nouvel ensemble de n-uplets N définis sur R^+ qui est le sous-ensemble des entités de iR validant P.
Nous notons: $N = (*P)\, iR$.
Il est facile de prouver que N est une instance de la relation $(*P)\, R$.

1.4. Synthèse: dictionnaire de données

1.4.1. Pertinence du modèle relationnel de données

Ce chapitre a pour but de présenter les principaux concepts du modèle relationnel de données le plus simple. Nous en avons fourni des définitions formelles et nous avons montré certaines de leurs propriétés mathématiques. Finalement nous allons les illustrer à l'aide d'un exemple concret. A ce stade, nous souhaitons en guise de synthèse, montrer la pertinence d'une telle approche qui est déjà perceptible dès ce premier chapitre.

a) Cette pertinence concerne pour nous le nécessaire langage commun entre d'une part des gestionnaires très intéressés par l'utilisation des bases de données mais connaissant relativement peu les langages de la technique informatique, et d'autre part des informaticiens, spécialistes des bases de données mais ne comprenant pas bien toutes les finesses, parfois très importantes, des processus réels de gestion qui leur apparaissent parfois même comme incohérents, car ils ne sont pas assez formés aux enjeux de l'implantation d'une base de données dans une organisation.
Le modèle relationnel de données fournit pour les données, la possibilité d'un tel langage commun. Nous avons pu constater qu'il est facilement compréhensible par des gestionnaires quand nous n'avons pas essayé de leur enseigner formellement les concepts, mais au contraire quand nous avons discuté avec eux de la modélisation relationnelle que nous avons obtenue dans leur domaine. Ce langage commun ne nous apparaît pas comme étant le modèle relationnel lui-même, mais comme étant la modélisation relationnelle elle-même.
Le modèle relationnel de données apparaît alors comme une *superstructure* (certains l'appellent méta-structure) connue des spécialistes de bases de données, qui permet à ces spécialistes un dialogue avec des gestionnaires.

b) Un autre point fondamental concerne d'une part la conception d'une base de données et d'autre part son suivi. Une phase très importante dans la conception d'une base de données est relative à l'observation de toutes les informations qui doivent être prises en compte dans la base de données. Cette phase que nous appelons modélisation des données du

champ d'application, met les concepteurs généralement en face d'une multitude d'informations d'intérêt très variable. Il s'agit pour eux d'arriver rapidement à les classer et à les ranger. Le modèle relationnel de données leur propose de classer ces informations en termes de relations, de constituants, de domaines, d'entités, de valeurs d'un domaine, de règles d'intégrité (chapitre suivant) ou d'informations inintéressantes. Il nous apparaît qu'il existe deux caractéristiques fondamentales de cette phase:
l'une reconnaît qu'une modélisation n'atteint sa stabilité qu'après de nombreuses réflexions et de nombreuses observations; bien souvent au cours de cette phase le concepteur peut s'apercevoir que telle relation doit être supprimée, que telle autre doit être créée, qu'il faut supprimer (ou rajouter) tel constituant de telle relation... Il s'agit pour le concepteur d'avoir une aide pour tenir clairement à jour sa modélisation.
L'autre constate que la modélisation d'une base de données un peu complexe ne requiert pas le travail d'une personne, mais d'une équipe.

Autant pour augmenter l'efficacité que pour diminuer des tensions "nerveuses" à l'intérieur de l'équipe, il nous apparaît comme indispensable que les personnes de l'équipe disposent non seulement d'un langage commun mais également d'une aide logistique pour que le dossier de modélisation de l'équipe soit tenu clairement à jour de manière complète, c'est-à-dire en tenant compte des travaux de chaque membre de l'équipe.

La première réponse à ces deux caractéristiques de la phase de modélisation est la mise en place d'un *dictionnaire de données* dont la superstructure est fournie par les concepts du modèle relationnel de données. Ce dictionnaire de données pourra être stocké comme une petite base de données et ainsi fournir toutes les facilités de mises à jour de la modélisation en cours.
Ce dictionnaire de données qui contient la modélisation n'est pas seulement nécessaire au moment de la conception de la base. C'est lui qui détient en fait toute la sémantique des données contenues dans la base. Aussi, si au cours du suivi de la base de données, il faut modifier les informations de la base de données (en supprimer ou en rajouter) pour répondre à des modifications de l'environnement de la base, est-ce en consultant et en modifiant soigneusement ce dictionnaire que les responsables pourront restructurer convenablement la base de données.

c) Un dernier point fondamental de la pertinence du modèle relationnel est la relative facilité de la transcription des résultats d'une modélisation de données en des termes informatiques. Ainsi ce n'est pas pour nous une

simple coïncidence si tous les développements des nouveaux systèmes de gestion de bases de données (SGBD) depuis 1974 sont partis des concepts de ce modèle relationnel.

Nous pensons que le modèle relationnel de données a permis la mise en évidence de concepts clairs et pertinents d'architecture de ces nouveaux SGBD appelés relationnels. Notamment ils ont un *dictionnaire de données* qui recueille la structure de la base de données. Ce dictionnaire de données est géré lui-même comme une petite base de données et souvent peut être interrogé à l'aide du même langage d'interrogation que celui de la base de données elle-même.

C'est cette pertinence du modèle relationnel de données qui va nous permettre au cours de ce livre, de présenter les aspects principaux de la cohésion nécessaire entre d'une part une modélisation de données et d'autre part une structure informatique de données qu'elle soit prise en compte ou non par un SGBD relationnel ou non.

Nous allons maintenant expliciter l'architecture d'un dictionnaire de données dans ses grandes lignes et concrétiser en même temps le rôle que nous attribuons au modèle relationnel de données, celui d'être la structure d'un tel dictionnaire.
Dans la dernière partie nous montrons comment il est possible d'enrichir ce modèle relationnel de données pour permettre la prise en compte de manière encore plus aisée de modélisations et de structures complexes de bases de données.

1.4.2. Règles du modèle relationnel

Le modèle relationnel de données considère l'existence de trois concepts fondamentaux: relation, constituant et domaine. Nous disons qu'une *catégorie* est soit une relation soit un constituant soit un domaine.

Une modélisation des données d'un champ d'application construite à partir du modèle relationnel va définir une structure de données en termes de ces trois catégories: elle comporte donc une liste de relations, de constituants et de domaines qui fournissent tous les *types* des données du champ d'application. Ensuite tout objet du champ d'application est repéré comme une donnée de l'un de ces types.

Il existe ainsi en fait trois niveaux:

1) celui des catégories,
2) celui des types,
3) celui des données.

Ces trois niveaux sont communs autant à l'analyse d'un champ d'application et au processus de modélisation qu'à l'architecture d'un système de gestion de bases de données relationnelles et aux bases de données relationnelles.
Par exemple pour une base de données relationnelle, le niveau 2) correspond à la structure logique des données et le niveau 3) au stockage des données.

Le niveau 1) apparaît alors comme la *superstructure* d'une modélisation de données, d'un SGBD ou d'une base de données.

Le *dictionnaire de données* tant d'une modélisation de données que d'un SGBD ou d'une base de données correspond aux niveaux 1 et 2; il a pour but de stocker de manière ordonnée toutes les informations d'un champ d'application relatives à des données et de fournir un moyen d'accès facile à ces informations. Un dictionnaire de données est donc une petite base de données dont la structure est justement la superstructure évoquée précédemment et dont les données sont tous les types des données du champ d'application.

Cette superstructure peut être décrite à l'aide du graphe suivant (ABRIAL74) où:
- chaque noeud du graphe est associé à une catégorie;
- une arête orientée du noeud N_1 vers le noeud N_2 signifie qu'un type de la catégorie N_1 peut être associé à un ou plusieurs types de la catégorie N_2.
Ainsi par exemple un type de la catégorie "relation", c'est-à-dire une relation, peut être associé à un ou plusieurs types de la catégorie "constituant" c'est-à-dire à un ou plusieurs constituants.

Il y a deux paramètres associés à chaque arête, les cardinalités minimale et maximale, c'est-à-dire respectivement les nombres minimal et maximal de types de la catégorie de N_2 qui peuvent être associés à un type de la catégorie de N_1.
(Nous avons l'habitude d'indiquer par une seule flèche le fait que la cardinalité maximale est égale à 1, par deux flèches le fait contraire.)

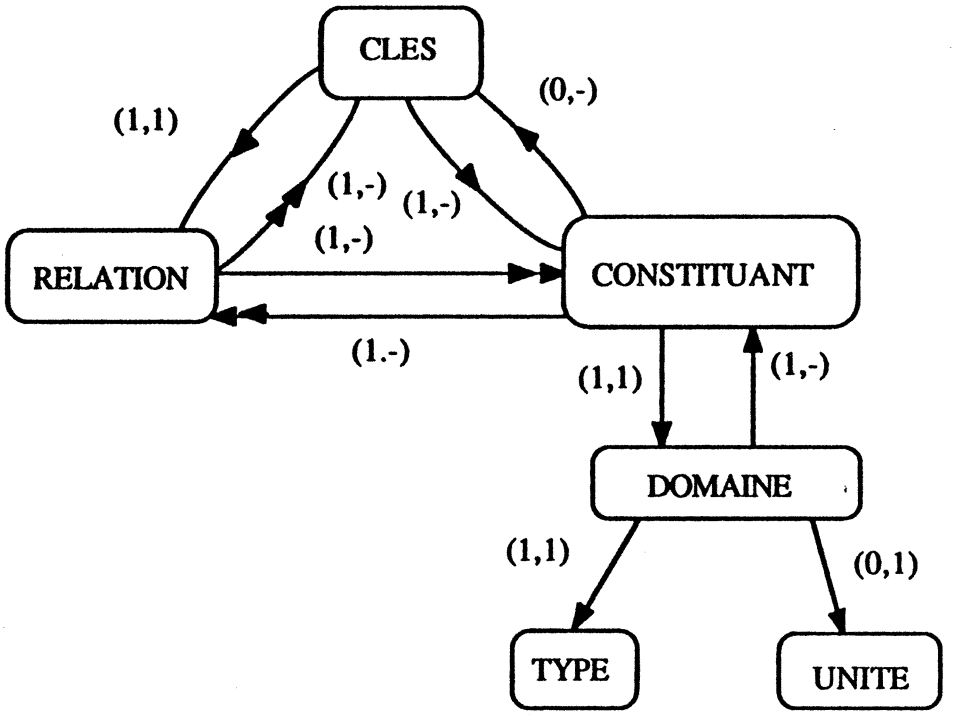

Figure 1.1.: Modèle relationnel de données

Voici les règles implicites que contient la représentation graphique:

(m1) une relation doit être définie sur au moins un constituant;
(m2) un constituant doit être un constituant d'au moins une relation;
(m3) une clé est une clé d'une seule relation;
(m4) une relation a toujours au moins une clé (éventuellement formée sur l'ensemble des constituants de la relation);
(m5) une clé est un ensemble formé d'au moins un constituant;
(m6) un constituant peut n'appartenir à aucune clé ou bien appartenir à plusieurs clés;
(m7) un constituant a un et un seul domaine;
(m8) un domaine est le domaine d'au moins un constituant mais plusieurs constituants peuvent avoir le même domaine;
(m9) un domaine a un et un seul type;
(m10) un domaine peut être associé à au plus une unité.

Les règles m1, m4, m5, m7, m9, m10 sont des règles implicites du modèle relationnel de données.
Les règles m2, m8 doivent être vérifiées pour que la modélisation soit *bien formée*: elles indiquent qu'il n'y a lieu de conserver un constituant (respectivement un domaine) dans une modélisation que s'il est associé à au moins une relation (resp. un constituant).

La règle m3 indique que le dictionnaire ne permet pas la prise en compte de relations ayant une même clé.

En plus de ces règles décrites dans la représentation graphique, il y a les règles suivantes:
(m20) les constituants formant la clé d'une relation sont des constituants de cette relation;
(m21) un constituant dont le domaine est de type texte n'appartient à aucune clé;
(m22) il existe au moins une clé obligatoire pour chaque relation.

La règle m20 est une règle implicite du modèle relationnel de données. La règle m21 provient de la définition d'une part d'une clé et d'autre part du type texte d'un domaine: si AB sont deux constituants formant une clé de la relation R, cela signifie qu'il ne peut exister deux entités r et r' de R ayant mêmes valeurs pour AB. Il faut donc pouvoir savoir si r.AB et r.'AB sont identiques ou non. Donc les domaines de A et de B ne doivent pas être de type texte.

La règle m22 oblige toute entité d'une relation à prendre des valeurs non inconnues pour les constituants formant une clé obligatoire; ainsi est-on assuré que toutes les entités d'une relation sont distinctes les unes des autres.

Toutes ces règles m1...m21 sont vérifiables automatiquement lors de la création du dictionnaire ou lors de sa mise à jour. Nous allons en présenter deux autres qui par contre ne peuvent être vérifiées que par les responsables de la base de données au moment de la modélisation.

(m30) Le fait qu'une clé K_i d'une relation R_i contienne une clé K_j d'une relation R_j, signifie que toute entité r_i de R_i telle que $r_i.K_j$ est clair, ne peut exister dans la base que s'il existe l'entité r_j de R_j telle que $r_j.K_j=r_i.K_j$.
L'inclusion de clés entre relations exprime implicitement pour nous, une dépendance existentielle entre les entités de ces relations; ainsi la suppression de l'entité r_j doit entraîner la suppression des entités r_i de R_i telles que $r_i.K_j=r_j.K_j$ si K_i est une clé obligatoire, ou bien une modification de r_i telle que $r_i.K_j$ devient obscur sinon.

(m31) Si deux relations R_i et R_j admettent des constituants en commun, alors il est possible de composer ces deux relations et d'obtenir ainsi une troisième relation qui a un sens pour les responsables de la base de données.

Les responsables de la base de données doivent vérifier ces dernières règles et éventuellement modifier le contenu du dictionnaire pour que ces règles soient respectées. L'exemple du paragraphe suivant nous permettra d'illustrer cette vérification.

1.4.3. Conclusion

Pour terminer, nous voudrions insister sur le nom même du modèle relationnel de donnés. Ce nom induit pour nous le fait qu'une donnée n'a de sens qu'*en relation* avec une autre. Ainsi la définition d'une relation, d'un constituant ou d'un domaine n'a de sens que par rapport à d'autres relations, constituants ou domaines et rarement en lui-même. Il est relativement aisé de constater que pour un champ d'application donné, il n'y a pas *une* modélisation correcte; généralement il en existe plusieurs. L'important est que chaque élément d'une modélisation soit défini de manière cohérente par rapport aux définitions des autres éléments.

Une modélisation de données peut être considérée comme un espace de noms qui sont tous en relation sémantique avec d'autres noms; ces relations sémantiques sont exprimées explicitement dans la modélisation. En ce sens une modélisation fournit un "*univers du discours*" (TARDIEU-ROCHFELD-COLLETTI83); en effet, la modélisation des données d'un champ d'application en termes relationnels fournit comme un langage dans lequel tout discours concernant la base de données doit être exprimé. Ce langage est formé d'un vocabulaire comprenant les noms des relations, des constituants et des valeurs de domaines principalement; ses phrases sont des phrases toutes faites, les prédicats des relations, ou de nouvelles phrases qui peuvent être construites à partir des premières à l'aide des opérateurs relationnels.

Il faut considérer que pour un utilisateur de la base de données cette modélisation de données devient la *réalité* de la base de données: il ne peut pas exprimer d'autres phrases que celles contenues dans l'univers du discours. Induit par une certaine déformation, cet univers du discours devient la *réalité* du champ d'application lui-même pour un utilisateur non averti: là réside à notre sens un grand danger, car ce comportement renverse la déformation en son contraire. En effet, c'est la modélisation de données qui n'est qu'une réalité partielle et déformée du champ d'application.

Nous tentons au cours de l'exercice suivant d'illustrer ce point important à nos yeux, en posant des questions sur la signification de la modélisation.

1.5. Exemple: atelier de production

1.5.1. Enoncé

Cette *modélisation* concerne un atelier de production (dont toute ressemblance avec un atelier réel serait fortuite) organisé ainsi.
-Il peut fabriquer une grande variété de produits.
-L'unité temporelle est l'heure.
-La fabrication d'unités d'un produit une fois lancée, l'est pour au moins une heure et conduit à la fabrication d'un nombre minimal d'unités de ce produit (QMINFAB) qui représente le nombre d'unités fabriquées en une heure. La fabrication se fait ainsi par *lots* de produit d'au moins QMINFAB unités. La fabrication d'un lot se fait dans une seule journée.

-La fabrication de lots d'un même produit nécessite toujours les mêmes matières premières dans les mêmes quantités, ces dernières étant proportionnelles au nombre d'unités du produit qui sont fabriquées.

-Les lots d'un même produit peuvent être fabriqués par n'importe laquelle des machines ayant une fonction particulière; toutes ces machines ont les mêmes performances. La durée de fabrication d'un lot est proportionnelle au nombre d'unités du lot. Un lot est fabriqué par la même machine, et plusieurs lots de produits différents ou du même produit peuvent être fabriqués en parallèle sur des machines différentes.
Une machine peut fabriquer un seul lot dans une heure donnée.

-Un ouvrier peut conduire plusieurs machines: cela dépend seulement de ses compétences techniques. S'il a la compétence de conduire une machine d'une telle fonction, alors il peut conduire toutes les machines ayant la même fonction. Il y a généralement plusieurs ouvriers qui ont la compétence de conduire une même machine.

-Un et un seul ouvrier est affecté à la conduite d'une machine pendant une heure donnée et un ouvrier ne peut conduire qu'une machine à la fois pendant une heure donnée.

La modélisation proposée est destinée à la réalisation d'un outil informatique permettant l'établissement de l'emploi du temps de manière journalière à partir de l'ensemble des lots à fabriquer ce jour; il concerne bien sûr les seuls ouvriers actifs et les seules machines disponibles; il est établi en fonction des quantités de matières premières actuellement en stock.

1.5.2. Relations et constituants

OUVRIER(NOV NOMOV).

Prédicat:	le numéro d'ouvrier (NOV) permet de distinguer entre eux les ouvriers; il lui est associé le nom de l'ouvrier.	
NOV	numéro d'ouvrier	mot entier (1,100).
NOMOV	nom d'ouvrier	texte.

ACTIVITÉ(NOV NH ACTIF).

Prédicat:	l'ouvrier de numéro (NOV) est actif à l'heure donnée (NH) s'il n'a pas pris congé à cette heure-là; il est inactif sinon.	
ACTIF	état d'activité	mot (actif, inactif).
NH	tranche horaire de travail	mot entier (1,8).

MACHINE(NMH FONCTION).

Prédicat:	le numéro de machine (NMH) permet de distinguer les machines entre elles; chaque machine a une seule fonction (FONCTION).	
FONCTION	fonction d'une machine	mot.
NMH	numéro de machine	mot.

DISP-MACHINE(NMH NH DISPMH).

Prédicat:	la machine de numéro (NMH) à une heure donnée est soit en panne, soit en révision (dans ces deux cas elle ne peut être utilisée pour la fabrication de lots), soit disponible (dans ce dernier cas elle est ou n'est pas utilisée pour la fabrication de lots).	
DISPMH	état de disponibilité d'une machine	mot (disponible, en panne, en révision).

PRODUIT(NP QMINFAB).

Prédicat:	le numéro d'un produit (NP) permet de le distinguer de tous les autres produits. A un numéro de produit correspond une seule quantité d'unités de ce produit: c'est la quantité minimale de tout lot de ce produit (QMINFAB).	
NP	numéro de produit	mot entier positif.
QMINFAB	quantité minimale d'unités d'un lot	numérique entier positif.

LOT(NP NLOT NBH NMH).

Prédicat:	le numéro d'un lot (NLOT) permet de le distinguer de tous les lots d'un même produit (NP). La fabrication de ce lot est prévue en tant d'heures (NBH) par la machine (NMH).	
NLOT	numéro de lot	mot entier positif.
NBH	nombre d'heures	numérique entier (1,8).

M-P(NMP QMP).

Prédicat:	le numéro de matière première NMP permet de distinguer les matières premières entre elles. A chaque matière première correspond la quantité en stock de cette matière première (QMP).	
NMP	numéro de matière	mot entier positif.
QMP	quantité de matière première	numérique entier positif.

APPR(NP NMP QMINMP).

Prédicat:	à un numéro de produit (NP) et à un numéro de matière première (NMP) correspond la quantité (QMINMP) de cette matière première nécessaire à la fabrication du lot minimal de ce produit (APPR pour approvisionnement).	
QMINMP	quantité minimale de matière première	numérique entier positif.

COMPÉTENCE(NOV NMH).

Prédicat:	l'ouvrier de tel numéro (NOV) a la compétence de conduire la machine de tel numéro (NMH).

UTILISATION(NP FONCTION).

Prédicat:	le produit de tel numéro (NP) peut être fabriqué par les seules machines de la fonction (FONCTION).

EMPLOI-DU-TEMPS(NH NP NLOT NOV NMH).

Prédicat:	pour ce jour, à l'heure (NH) il est prévu que le lot (NLOT) du produit (NP) soit fabriqué par la machine de numéro (NMH) conduite par l'ouvrier de numéro (NOV).

1.5.3. Exercices relatifs à cette modélisation

Déterminer les clés de chaque relation.
Indiquer si les affirmations suivantes sont compatibles avec la modélisation. Justifier les réponses.
a) Il existe un produit dont la fabrication nécessite une seule matière

première.
b) On peut prévoir dans l'emploi du temps que la fabrication d'un lot dure seulement 1 heure 35 minutes.
c) Un ouvrier peut être remplacé par un autre ouvrier pour la fabrication d'un même lot dans une heure donnée.
d) Il y a au plus 5 machines pour une même fonction.
e) Des unités d'un même lot peuvent être fabriquées ensemble.

Nous donnons nos solutions au § 1.5.5.

1.5.4. Illustration des différents concepts du modèle relationnel

1.5.4.1. Relation, entité, clé

Une entité de la relation DISP-MACHINE est un élément du produit cartésien des domaines des constituants NMH,NH,DISPMH.
(1, 1, disponible) est un tel élément; il correspond à une entité seulement s'il valide le prédicat de la relation, c'est-à-dire seulement si un utilisateur affirme que la machine de numéro 1 est disponible au cours de la première heure.

Une instance d'une relation est un ensemble d'entités de la relation.

Une clé de la relation DISP-MACHINE est un ensemble minimal de ses constituants tels que si deux entités de DISP-MACHINE prennent les mêmes valeurs pour ces constituants dans une même instance, ces deux entités sont identiques. Si NMH est une clé alors il ne sera pas possible de considérer la disponibilité d'une machine au cours de plusieurs heures dans une même instance !! Bien sûr ceci n'est pas acceptable.
Si NH est une clé de DISP-MACHINE alors la disponibilité d'une seule machine pour une heure donnée pourra être stockée dans une instance !! Bien sûr ceci n'est pas acceptable.
Par contre à une valeur de NMH NH doit correspondre une seule entité dans toute instance de DISP-MACHINE car il y correspond un seul état de disponibilité. Comme cet ensemble de constituants est minimal - car ni NH ni NMH ne vérifie cette propriété -, NMH NH est une clé de DISP-MACHINE.
C'est la seule. Nous laissons le soin au lecteur de le comprendre. L'exercice du chapitre suivant fournira une démarche complète pour déterminer les clés des relations.

1.5.4.2. Projection d'une relation, charnière de deux relations, composition de deux relations

La projection de la relation EMPLOI-DU-TEMPS sur les constituants NH NP NLOT conduit à une nouvelle relation dont le prédicat peut s'exprimer ainsi:
à l'heure (NH) il est prévu la fabrication du lot (NLOT) du produit (NP).

La charnière entre les relations de DISP-MACHINE et de EMPLOI-DU-TEMPS est formée de l'ensemble des constituants communs de ces deux relations: NMH NH.
La composition de ces deux relations va fournir une nouvelle relation construite sur les constituants: NMH NH NP NLOT NOV DISPMH, et dont le prédicat peut se formuler ainsi:
à l'heure (NH) il est prévu la fabrication du lot (NLOT) du produit (NP) par une machine (NMH) qui est conduite par un ouvrier (NOV) et dont l'état de disponibilité à cette heure (NH) est (DISPMH).

1.5.4.3. Clé obligatoire

NOV NH forme une clé de EMPLOI-DU-TEMPS; elle n'est pas la seule car NP NLOT NH et NH NMH sont aussi des clés de EMPLOI-DU-TEMPS. Mais elle peut en plus être déclarée comme obligatoire: alors toute entité de toute instance de EMPLOI-DU-TEMPS doit prendre des valeurs claires pour les constituants NOV NH.

1.5.5. Solutions des exercices

1.5.5.1. Clés des relations

Dans le chapitre suivant, nous étudions cet aspect de manière plus intensive.

clés de	OUVRIER:	NOV,
	ACTIVITE:	NOV NH,
	MACHINE:	NMH,
	DISP-MACHINE:	NMH NH,
	PRODUIT:	NP,
	LOT:	NP NLOT,
	M-P:	NMP,
	APPR:	NP NMP,
	COMPETENCE:	NOV NMH,

UTILISATION: NP,
EMPLOI-DU-TEMPS : NH NOV,
NH NMH,
NH NP NLOT.

NOMOV ne peut pas être une clé de la relation OUVRIER: son domaine est en effet déclaré comme ayant un type texte. Deux valeurs de NOMOV ne peuvent pas être comparées et donc NOMOV ne peut appartenir à aucune clé.

1.5.5.2. Réponses aux questions sur la modélisation

Par les questions posées au § 1.5.3, nous essayons de sensibiliser le lecteur à la distance séparant le "monde réel" de celui de base de données obtenue à partir de la modélisation proposée. Voici nos réponses:

a) Le fait qu'un produit ne nécessite qu'une seule matière première pour sa fabrication peut être pris en compte dans la future base de données: en effet, la modélisation prévoit qu'un produit peut être associé de manière générale, avec plusieurs matières premières pour sa fabrication (clé de la relation APPR est NP NMP).
b) Il est possible que dans la "réalité" la fabrication d'un lot ne dure que 1 heure 35 minutes. Mais la modélisation actuelle ne compte qu'en heures! Aussi ce fait ne pourra-t-il être pris en compte dans la future base avec cette même précision: la fabrication de ce lot va durer deux heures pour la base !
c) Dans la "réalité", si l'on peut admettre le remplacement d'un ouvrier par un autre au cours d'une heure donnée pour la fabrication d'un lot donné, il faut bien comprendre qu'au niveau de la base avec la modélisation actuelle, on pourra également remplacer un ouvrier par un second: mais alors ce second ouvrier sera considéré par la base comme ayant fait tout le travail pendant toute l'heure considérée !! En effet, NH NP NLOT est une clé de EMPLOI-DU-TEMPS et ainsi à une heure et à un lot on ne peut associer qu'un ouvrier (NOV).
d) La modélisation prévoit qu'une fonction peut être remplie par plusieurs machines (relation MACHINE). Si une fonction peut être remplie seulement par au plus cinq machines, ce fait n'est pas écrit dans la modélisation actuelle. Il s'exprime à l'aide d'une règle d'intégrité comme nous le présentons dans le chapitre suivant.
e) Il est possible que des unités d'un même lot puissent être fabriquées en même temps. Mais "l'univers du discours" de la modélisation ne contient pas le concept d'unité de lot. Aussi ce fait n'est-il pas pris en compte dans la future base de données.

1.6. Annexe

1.6.1. Démonstrations des propriétés de l'élargissement

(e1) $R^e[R^+] = R$.
$\forall r \in R$, il suffit de considérer n'importe quelle valeur de chaque constituant de U^+-R^+ pour former avec r une entité re de R^e telle que $re.R^+ = r$; ainsi R est inclus dans $R^e[R^+]$.
$\forall s \in R^e[R^+]$, il existe une entité re de R^e telle que $s = re.R^+$ (définition de la projection). Si re est une entité de R^e alors $re.R^+$ est une entité de R (définition de l'élargissement) et ainsi s est une entité de R.

(e11) $1[X] = 1_X$ et $(1_X)^e = 1$.
(e12) $0[X] = 0_X$ et $(0_X)^e = 0$.
Les démonstrations de ces deux propriétés sont immédiates.

(e2) Si $X \subset R^+$ alors $R^e[X] = R[X]$.
$\forall x \in R^e[X]$, il existe une entité re de R^e telle que re.X=x d'après la définition de la projection; il existe une entité r de R telle que $re.R^+ = r$ d'après la définition de l'élargissement. Comme X est inclus dans R^+, $re.X = (re.R^+).X = r.X$; ainsi il existe une entité r de R telle que x=r.X. $R^e[X]$ est inclus dans R[X].
$\forall x \in R[X]$, il existe une entité r de R telle que x=r.X. Par l'élargissement de R, il existe une entité re de R^e telle que $re.R^+ = r$. Comme X est inclus dans R^+, $x = r.X = (re.R^+).X = re.X$. Ainsi x appartient à $R^e[X]$ et R[X] est inclus dans $R^e[X]$.

(e21) Si $X \wedge R^+ = \varnothing$ alors $R^e[X] = 1_X$.
$\forall x \in X$ $\exists r \in R$ et $\exists re \in R^e$ telle que $re.R^+ = r$ et re.X=x; donc x appartient à $R^e[X]$. La réciproque est immédiate.

(e3) $(R*S)^e = R^e * S^e$.
$\forall rse \in (R*S)^e$ $\exists rs \in R*S$ telle que $rse.R^+ \cup S^+ = rs$.
D'après la définition du produit, $\exists r \in R$ et $\exists s \in S$ telle que
$rs.R^+ = r$ et $rs.S^+ = s$.
Il est possible de former un n-uplet trs sur $R^+ \cup S^+$ telle que $trs.R^+ = r$ et $trs.S^+ = s$ et ensuite d'élargir ce n-uplet à U^+ et d'obtenir ainsi un n-uplet trse formé sur U^+; $trse.R^+ = r$ et $trse.S^+ = s$ par construction.
Ainsi, trse appartient à R^e et à S^e et donc à $R^e * S^e$.

Réciproquement:
$\forall rse \in R^e * S^e$, $rse \in R^e$ et $rse \in S^e$ car R^e et S^e sont des relations définies sur le même ensemble de constituants U^+. Ainsi $r=rse.R^+$ et $s=rse.S^+$ sont des entités respectivement de R et de S telles que:
$r.R^+ \wedge S^+ = s.R^+ \wedge S^+ = rse.R^+ \wedge S^+$.
Donc $rse.R^+ \cup S^+$ est une entité de R*S et rse une entité de $(R*S)^e$.

(e31) $R^e * S^e = R^e * S = R * S^e$.
La démonstration suit le même raisonnement que le précédent.

(e4) $(R+S)^e = R^e + S^e$.
La démonstration suit le même raisonnement que le précédent.

(e41) $R^e+S^e = R^e + S = R + S^e$.
$\forall rse \in R^e+S^e$, $rse \in R^e$ ou $rse \in S^e$;
donc $rse \in R^e$ ou $rse.S^+ \in S$ et ainsi $rse \in R^e+S$.
Réciproquement: $\forall rse \in R^e+S$, $rse \in R^e$ ou $rse.S^+ \in S$;
donc $rse \in R^e$ ou $rse \in S^e$; finalement $rse \in R^e + S^e$.

(e5) $(non\ R)^e = non\ (R^e)$.
$\forall re \in (non\ R)^e$, $r=re.R^+$ est une entité de non R et n'est donc pas une entité de R; re n'est donc pas une entité de R^e, c'est une entité de non(R^e).
Réciproquement:
$\forall re \in non\ (R)^e$, re n'est pas une entité de R^e et ainsi $r=re.R^+$ n'est pas une entité de R. r est une entité de non R et re une entité de $(non\ R)^e$.

(e6) $((*P)R)^e = (*P)\ R^e$.
Il existe une relation PR telle que:
$PR^+ = P^+$ et $||PR||=P$ et $(*P)R = PR * R$.

$((*P)R)^e = (PR*R)^e = PR^e * R^e$ (d'après e3)
$= PR*R^e$ (d'après e31)
$= (*P)R^e$.

(e7) Si $R^+= S^+$ alors $(R=S) \Leftrightarrow (R^e = S^e)$.
Si $R \Rightarrow S$ alors $\forall re \in R^e$, re.X est une entité de R donc de S;
re est une entité de S^e; $R^e \Rightarrow S^e$.
Par symétrie on conclut que si R = S alors $R^e = S^e$.
Réciproquement:
si $R^e \Rightarrow S^e$, alors $\forall r \in R\ \exists re \in R^e$ telle que re.X=r;
comme re est également une entité de S^e et que $R^+=S^+$, $re.S^+=r$ est une entité de S; ainsi $R \Rightarrow S$.
Par symétrie on conclut que si $R^e=S^e$ et $R^+=S^+$ alors R=S.

1.6.2. Démonstrations des propriétés de l'opération focus

(f1a) focus $(R/U^+) = R^e$
car focus $(R/U^+) = R^e[U^+] = R^e$.

(f1b) Si $X \subset R^+$ alors focus $(R/X) = R[X]$
car focus $(R/X) = R^e[X] = R[X] = R[X]$ (d'après e2).

(f2) focus $(1_X/Y)=(1_X)^e[Y] = 1[Y] = 1_Y$ (d'après e11).
focus $(0_X/Y)=(0_X)^e[Y] = 0[Y] = 0_Y$ (d'après e12).

(f3) focus $(R*S/X) \subset$ focus $(R/X) *$ focus (S/X).
Si X contient la charnière RS^+ alors
focus $(R*S/X) =$ focus $(R/X) *$ focus (S/X).

focus $(R*S/X) = (R*S)^e [X] = (R^e*S^e) [X]$ (d'après e3).
$\forall rsx \in (R^e*S^e)[X]$ $\exists$ rse$\in (R^e*S^e)$ telle que rse.X = rsx.
Donc $\exists$ rse$\in R^e$ et $\exists$ rse$\in S^e$ telle que rse.X = rsx.
Donc $\exists$ rse$\in R^e$ telle que rse.X = rsx, et $\exists$ rse$\in S^e$ telle que rse.X = rsx;
finalement rsx est une entité de $R^e[X] * S^e[X]$.

Réciproquement (dans le cas où $RS^+ \subset X$):
$\forall rsx \in R^e[X]*S^e[X]$, rsx$\in R^e[X]$ et rsx$\in S^e[X]$,
$\exists$ re$\in R^e$ telle que re.X = rsx et $\exists$ se$\in S^e$ telle que se.X = rsx.
Donc $\exists$ r$\in$R telle que re.R^+= re et $\exists$ s$\in$S telle que se.S^+= s.

Comme $RS^+ \subset X$, re.RS^+ = rsx.RS^+ = se.RS^+ =s.RS^+ =r.RS^+.
Donc $\exists$ rs$\in$R*S telle que rs.R^+ =r et rs.S^+ = s;
alors re.$R^+ \cup S^+$ =rs et ainsi re est une entité de $(R*S)^e$.
Par conséquent rsx est une entité de $(R*S)^e[X]$.

(f4) focus $(R+ S/X) =$ focus $(R/X) +$ focus (S/X).
focus $(R+S/X) = (R+S)^e[X] = (R^e+S^e)[X]$.
La démonstration suit le même raisonnement que le précédent.

(f5) non focus $(R/X) \subset$ focus (non R/X).
Si X contient R^+ alors il y a l'égalité.

non focus $(R/X) =$ non $(R^e[X])$.
$\forall x \in$ non$(R^e[X])$ alors $\forall$reeR^e re.X$\neq$x ;

donc $\forall$ re' n-uplet formé sur U^+ telle que re'.X=x, re'$\in$ non R^e:
x appartient à (non R^e)[X].
Ainsi non focus (R/X) $\subset$ focus (non R/X) (d'après e5).

Réciproquement ($R^+ \subset X$):
focus (non R/X) = (non R^e)[X] = (non R^e)[X].
$\forall x \in$ (non R^e)[X] et $\forall re \in U$ telle que re.X = x.
On sait qu'il existe au moins une telle entité re de non R^e par définition de la projection.
Mais s'il existe re' dans R^e telle que re'.X=x, alors re'.R^+ appartient à R; comme R^+ est inclus dans X, re'.R^+ =re.R^+ et ainsi re n'appartient pas à non R^e.
Donc $\forall re \in U$ telle que re.X=x, re$\in$ non R^e.
Donc x$\in$ non (R^e[X]) et focus (non R/X) $\subset$ non focus (R/X).

(f6a) Si $Y \subset X \subset R^+$ alors focus (focus (R/X)/Y)= focus (R/Y)=R[Y].
La démonstration est simple.

(f6b) Si $R^+ \subset X$ alors focus (focus(R/X)/Y) = focus (R/Y).
focus(focus(R/X)/Y) = (focus(R/X))^e[Y] = (R^e[X])^e[Y].
$\forall re \in (R^e[X])^e$ x=re.X $\in$ R^e[X] et $\exists re' \in R^e$ telle que re'.X=x;
donc re'.$R^+ \in$ R.
Comme $R^+ \subset$ X, re'.R^+=re.R^+ est donc une entité de R.
re est une entité de R^e.

Réciproquement:
$\forall re \in R^e$ re.$R^+ \in$ R.
x = re.X est une entité de R^e[X] par construction.
Ainsi re est une entité de (R^e[X])^e.
Ainsi (R^e[X])^e = R^e.
focus (focus (R/X)/Y) = R^e[Y] = focus (R/Y).

1.6.3. Démonstrations des propriétés des opérations somme, produit, complément, projection

(02) On applique systématiquement la propriété (e7). X = R^+.

(non R + R)^e	= (non R)^e + R^e	(e4)
	= non (R^e) + R^e	(e5)
	= 1	(e8).

$(\text{non } R * R)^e$	$= (\text{non } R)^e * R^e$	(e3)
	$= \text{non } (R^e) * R^e$	(e5)
	$= 0$	(e8).
$(R * 0_x)^e$	$= R^e * (0_x)^e$	(e3)
	$= R^e * 0$	(e12)
	$= 0$	(e8).
$(R + 0_x)^e$	$= R^e + (0_x)^e = R^e + 0 = R^e.$	
$(R * 1_x)^e$	$= R^e * (1_x)^e = R^e * 1 = R^e.$	
$(R + 1_x)^e$	$= R^e + (1_x)^e = R^e + 1 = 1.$	

(03) $R + (R*S) = R.$
Cette propriété découle directement des propriétés (e7, e3, e4, e5, e9).

$(R + (\text{non } R*S))^e$	$= R^e + (\text{non } R*S)^e$	(e4)
	$= R^e + (\text{non } R)^e * S^e$	(e3)
	$= R^e + \text{non } R^e * S^e$	(e5)
	$= R^e$	(e9).

2

RÈGLES D'INTÉGRITÉ

2.1. Introduction[1]

Une règle d'intégrité est un invariant du champ d'application qui concerne n'importe lequel de ses états ou n'importe lequel de ses changements d'états. Traduit en termes de bases de données et appliqué aux seules données (à l'exclusion des traitements effectués sur la base), ce concept de règle d'intégrité devient une propriété qu'il est possible d'exprimer à partir des relations de la modélisation, et qui doit toujours être vérifiée à l'aide d'un processus algorithmique par chaque état de la base de données ou par chaque modification apportée à la base de données.

Au cours de cette difficile phase de modélisation d'un champ d'application, le travail d'analyse et de formalisation de ses invariants est d'une remarquable efficacité et pertinence pour au moins deux raisons. La première a son origine dans le fait que tout invariant relatif aux données est une propriété tellement remarquable du champ d'application qu'elle doit pouvoir être décrite dans la modélisation. Si ce n'est pas possible, les concepteurs doivent compléter la modélisation. Toute modélisation finale de données est *complète* par rapport aux invariants si elle permet la description de tous les invariants relatifs aux données.

La seconde raison provient de la position même des concepteurs qui analysent un champ d'application. Leur tâche principale est d'observer et

---

[1] *Note aux lecteurs*:
Une lecture rapide comprend les paragraphes 2.1., 2.2., 2.7., 2.8., 2.9. et 2.10.

de synthétiser leurs observations dans une modélisation obéissant au formalisme du modèle relationnel de données par exemple. Ils sont ainsi dans une position d'écoute attentive de toutes les informations concernant le champ d'application. Ils sont confrontés à la difficulté de rester dans cette position un peu passive dirons-nous: "les informations leur sont apportées et ils en font une modélisation", ou au contraire de prendre une position plus "active" et ainsi de susciter les informations qui leur sont nécessaires pour établir la modélisation de données. Les concepteurs en position "active" rencontrent aussitôt un problème majeur: comment rechercher les informations pertinentes; comment mieux comprendre le champ d'application ? Là encore la recherche des invariants fournit une aide précieuse aux concepteurs de la base de données. Nous allons montrer comment toute modélisation relationnelle même (et surtout) inachevée contient des situations remarquables qui sont susceptibles de renfermer des invariants. Ces situations suggèrent aux concepteurs des réflexions pertinentes sur l'éventuelle existence d'invariants dans cette partie de la modélisation: s'il y a un invariant, il faut pouvoir l'exprimer et si la modélisation ne le permet pas, il faut la compléter. Par contre s'il n'y a aucun invariant, le concepteur doit s'assurer que la modélisation est bien correcte.malgré une situation propice à l'existence d'un invariant.

De plus, face aux dangers de la position "active" dans une démarche d'analyse, qui sont dus aux abus conscients ou non que les concepteurs peuvent manifester, il est important de remarquer que ces situations contenues dans une modélisation doivent être étudiées par les concepteurs, qu'ils aient adopté une position plus ou moins passive, ou plus ou moins active.
Nous qualifions la modélisation de *valide par rapport aux invariants* quand toutes ces situations ont été analysées.

Une modélisation de données doit contenir non seulement les relations, constituants et domaines comme nous l'avons écrit au chapitre précédent mais également les règles d'intégrité. Celles-ci ont un rôle très important à jouer dans l'utilisation de la base de données: celui d'assurer la cohérence de la base de données. C'est leur rôle habituel. De quoi s'agit-il? Les entités d'une relation sont des n-uplets de valeurs validant le prédicat de la relation. Mais cette "validation" est de la seule responsabilité d'un utilisateur à son terminal ou d'un programme travaillant sur la base. C'est lui qui décide si oui ou non tel n-uplet est une entité. Les risques de fausses manoeuvres n'étant pas toujours minimes, les concepteurs et les utilisateurs souhaitent contracter quelques assurances contre ces risques. Ce sont les invariants du champ d'application décrits en termes de règles d'intégrité de la base de données

qui vont servir d'assurances: toute modification apportée à la base devra valider ces règles d'intégrité pour être acceptée. Ainsi a-t-on une assurance de détecter certaines erreurs et donc d'éviter certaines incohérences à la base de données.
Le problème qui se pose alors aux concepteurs est un problème classique de toute entreprise. Faut-il prendre toutes les assurances possibles quel qu'en soit le prix, et donc valider toutes les règles d'intégrité, ou bien faut-il se contenter des plus importantes ? Dans le second cas certaines règles d'intégrité (et donc certains invariants) ne sont pas retenues généralement pour une combinaison des deux raisons suivantes:
- l'assurance fournie par leur validation couvre un risque minime;
- le prix de l'assurance fournie par leur validation est beaucoup trop élevé, c'est-à-dire les coûts informatiques, pertes d'efficacité, pertes de temps, pertes de ressources, etc. sont trop élevés par rapport aux risques encourus.

Une *base de données* est dite *cohérente* (par rapport aux règles d'intégrité) si toutes les instances de ses relations valident toutes les règles d'intégrité retenues pour assurer cette cohérence. A chaque règle d'intégrité il faut associer sa *portée* c'est-à-dire l'ensemble des primitives de mises à jour qui réclament sa validation pour que la base reste cohérente.
Enfin même si les concepteurs ont pris toutes les assurances en retenant toutes les règles d'intégrité, il reste néanmoins impossible de garantir la cohérence absolue de la base de données de manière algorithmique puisque justement les prédicats des relations ne peuvent être validés par l'exécution d'un algorithme; les utilisateurs de la base de données auront toujours leur responsabilité engagée dans la cohérence de la base de données.

2.2. Règle d'intégrité

2.2.1. Définition d'une règle d'intégrité

Une *règle d'intégrité* (ri) est une condition qui est définie par rapport soit à une relation (règle d'intégrité intra-relation) soit à plusieurs relations (ri inter-relation), et dont le test de validation peut être effectué de manière algorithmique sur les instances des relations concernées.

Le *contexte* d'une règle d'intégrité désigne les relations sur lesquelles la règle d'intégrité est définie.

La *condition* d'une règle d'intégrité doit être vérifiée pour tout état de la base (règle d'intégrité statique) ou pour tout changement d'état de la base (règle de comportement), par les instances des relations du contexte.
Une règle d'intégrité statique s'applique à chaque entité d'une instance d'une relation (règle d'intégrité statique individuelle) ou à une instance d'une relation (règle d'intégrité statique ensembliste).
Une règle de comportement est relative à une opération de modification d'une ou plusieurs entités d'une instance d'une relation: créer, supprimer, interroger une entité ou un ensemble d'entités, mettre à jour la valeur prise par une entité pour un constituant. Elle exprime une condition logique entre l'état de la base avant la modification et l'état de la base après la modification.

La *portée* d'une règle d'intégrité désigne l'ensemble des primitives de modification de relations de base, qui doivent contenir un algorithme de validation de la règle d'intégrité, pour qu'un état de la base cohérent soit transformé par cette primitive en un autre état cohérent.

La *réponse* d'une règle d'intégrité ri indique les actions à entreprendre en cas de non-validation de ri. Il s'agit de spécifier les actions à entreprendre lorsque l'une des primitives de la portée ayant été déclenchée, l'algorithme de validation s'aperçoit que l'exécution complète de la primitive provoquerait une incohérence relative à ri. Une réponse très commune est le refus de l'exécution de la primitive. Mais cette réponse peut être plus compliquée comme celle qui peut entraîner une succession de suppressions en cascade.

L'*expression* d'une règle d'intégrité peut prendre la forme d'une expression mathématique (exemple: dépendance fonctionnelle, dépendance de composition que nous présentons dans un paragraphe ultérieur, comparaison d'ensembles d'entités, automate, expression prédicative dont les variables sont des relations ou des constituants formant le contexte de la règle d'intégrité et prennent leurs valeurs dans l'ensemble des entités ou données stockées dans la base relatives à ces relations et ces constituants), d'une expression algorithmique terminée par une condition.

2.2.2. Exemples

Nous allons présenter des exemples d'une règle d'intégrité statique individuelle, d'une autre ensembliste et d'une règle de comportement. Le paragraphe (§ 2.10.) traite d'un exemple complet qui contient une grande variété de règles d'intégrité.

Soit la relation JOUEUR (NOMJOUEUR ÉTAT-CIVIL ÉQUIPE CAPITAINE) dont le prédicat est le suivant:
"tout joueur a un nom (NOMJOUEUR), un état civil (ÉTAT-CIVIL), appartient à une seule équipe (ÉQUIPE) et est ou non capitaine de cette équipe (CAPITAINE)".
Nous supposons que la clé de cette relation est NOMJOUEUR. NOMJOUEUR a un domaine de type mot; ÉTAT-CIVIL a un domaine de type mot, qui contient les valeurs: inconnu, célibataire, marié(e), divorcé(e), veuf(ve). CAPITAINE a un domaine de type booléen où la valeur "vrai" signifie que le joueur est capitaine, et "faux" le contraire.

2.2.2.1. Exemple de règle d'intégrité statique individuelle

"Un joueur célibataire ne peut pas être capitaine". C'est une ri individuelle. Son contexte est la relation JOUEUR. Elle peut s'exprimer ainsi:
$\forall$ j $\in$ JOUEUR[ÉTAT-CIVIL=célibataire] j.CAPITAINE=faux.
Sa portée est l'ensemble des primitives suivantes:

créer JOUEUR,
mise à jour JOUEUR[ÉTAT-CIVIL],
mise à jour JOUEUR[CAPITAINE].

2.2.2.2. Exemple de règle d'intégrité statique ensembliste

"Toute équipe a au moins un capitaine".
Son contexte est la relation JOUEUR. Elle s'exprime ainsi:
$\forall$ eq $\in$ ÉQUIPE
$\exists$ j $\in$ JOUEUR[ÉQUIPE=eq et CAPITAINE = vrai].

C'est une ri ensembliste. Voici sa portée:

mise à jour JOUEUR[CAPITAINE],
mise à jour JOUEUR[ÉQUIPE],
supprimer JOUEUR.

La création d'un nouveau joueur d'une équipe ne réclame pas en général la validation de cette ri: l'équipe existant déjà, il existe un capitaine. Mais si "créer JOUEUR" n'appartient pas à la portée, la base de données risque d'être incohérente: en effet, lors de la création des entités de JOUEUR relatives à cette équipe, on pourrait créer une équipe sans son capitaine. Il ne serait pas très précis pour ce seul cas d'introduire "créer JOUEUR" dans la portée. Mieux vaut définir une nouvelle primitive créer ÉQUIPE qui permettra de créer les entités de JOUEUR d'une nouvelle équipe, et insérer cette nouvelle primitive dans la portée.

Remarque:
c'est l'étude des règles d'intégrité qui vient d'enrichir la modélisation en conduisant le concepteur à créer cette primitive créer ÉQUIPE.

2.2.2.3. Exemple de règles de comportement

Considérons l'état civil d'un joueur. Il est bien connu qu'une personne mariée ne peut pas repasser à l'état-civil célibataire. Ainsi le changement d'état-civil d'un joueur obéit à un automate qui peut se représenter graphiquement de la manière suivante (cf. BODART-PIGNEUR83):

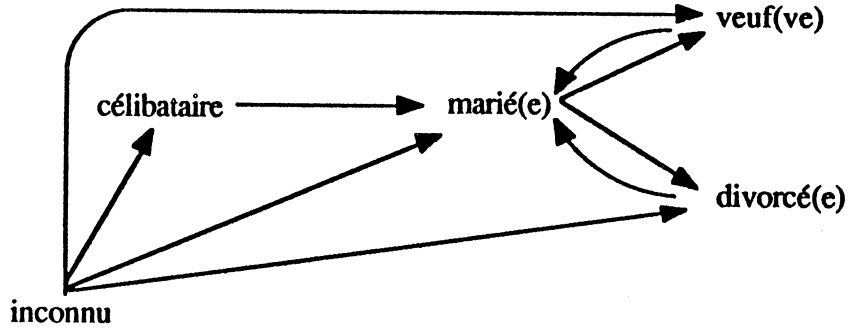

Figure 2.1.: Exemple de règle de comportement.

Les noeuds représentent les différentes valeurs du domaine et les seuls changements autorisés sont ceux matérialisés par les arcs.

Une étude plus approfondie fait apparaître que cet automate ne permet pas de changer l'une des valeurs claires (célibataire ... veuf(ve)) dans la valeur inconnue. Si l'application réclame cette possibilité alors il faut introduire une nouvelle valeur inconnue pour chaque valeur claire. En effet, la seule valeur inconnue déjà en place ne suffit pas car elle permettrait pour un joueur la succession suivante de changements d'état civil: marié(e) - inconnu - célibataire.

Ainsi c'est l'étude approfondie de cette règle d'intégrité qui enrichit la modélisation car elle conduit à compléter la modélisation par 4 nouvelles valeurs: 4 valeurs inconnues associées à chaque valeur claire.

Le contexte de cette règle d'intégrité est la relation JOUEUR, sa portée: mettre à jour ÉTAT-CIVIL de JOUEUR.

2.2.3. Définitions d'une structure de relation et d'une collection de données

Nous apportons deux définitions supplémentaires relatives au concept de relation, qui précisent les liens structurels existants entre une relation et une règle d'intégrité.
Une *structure d'une relation* R ou *schéma de relation* R est formée de la définition de cette relation et de l'ensemble des règles d'intégrité I(R) qui l'admettent comme contexte.
Une *collection de données* de la relation R est une instance de R qui valide toutes les règles d'intégrité statiques contenues dans I(R).

2.2.4. Relation d'équivalence et relation d'ordre

Soit deux règles d'intégrité ri_1 et ri_2 ayant le même contexte: $R_1...R_n$ (si n=1, les ri sont des ri intra-relation; sinon elles sont des ri inter-relation).

ri_1 et ri_2 sont *équivalentes* si tout ensemble d'instances ($iR_1...iR_n$) qui valide l'une, valide l'autre, et inversement.
ri_1 est *plus contraignante que* ri_2 si tout ensemble d'instances ($iR_1...iR_n$) qui valide ri_1, valide ri_2: on note $ri_1 \leq ri_2$.
ri_1 est *triviale* (ou égale à 1) si tout ensemble d'instances ($iR_1...iR_n$) valide ri_1.
ri_1 *fait le vide* (ou est égale à 0) si aucun ensemble d'instances ($iR_1...iR_n$) ne la valide.

La première relation est une relation d'équivalence, la seconde est une relation d'ordre partiel.

2.3. Introduction aux dépendances

Ce terme générique regroupe toute une classe de règles d'intégrité statiques ensemblistes que la communauté scientifique a jugé remarquables si nous prenons comme référence la quantité de travaux scientifiques de qualité qui en traitent. Pour nous, leur importance est due au fait qu'elles sont un point de convergence de nombreux thèmes liés aux structures de données.

Un premier thème considère que les dépendances permettent d'exprimer

une classe d'invariants qui sont essentiels pour comprendre une modélisation. Nous tâchons d'illustrer ce point de vue à l'aide de l'exemple du paragraphe (§ 2.10.).

Un autre thème traite du rôle des dépendances dans l'obtention de modélisations équivalentes du point de vue sémantique mais conduisant à des structures de bases de données plus ou moins efficaces pour la validation de ces dépendances.

Un troisième thème concerne le rôle fondamental des dépendances dans l'utilisation cohérente d'une base de données tant lors d'interrogation ou d'extraction de données que lors de modifications apportées à une base de données.

Un quatrième thème est lié à la difficile phase de modélisation ou au difficile problème de l'évolution d'une base de données: les dépendances nous apparaissent comme un guide précieux pour la compréhension d'une modélisation, sa critique et éventuellement sa modification. Il nous semble qu'elles amènent les concepteurs à se poser d'excellentes questions en premier lieu sur la modélisation et donc en second lieu sur la fidélité de cette modélisation au champ d'application.
Enfin un cinquième thème est leur lien très étroit avec le problème de la redondance d'informations dans une base de données.

Nous allons aborder ces thèmes autant dans les paragraphes suivants que dans la troisième partie "Décomposition d'une relation". En effet, les concepts de dépendances et de décomposition de relations sont très étroitement mêlés. Dans ce chapitre, nous allons présenter les différentes dépendances, leurs propriétés, le concept de décomposition d'une relation et les propriétés d'une décomposition qui se déduisent directement de celles des dépendances de composition. Ces propriétés nous permettent de montrer l'intérêt des dépendances dans le processus de modélisation d'un champ d'application (1^{e}, 3^{e} et 4^{e} thèmes précédents). Nous laissons l'étude des 2^{e} et 5^{e} thèmes à un chapitre ultérieur (Décomposition d'une relation, troisième partie).

Remarque:
notre but n'étant pas la théorie des dépendances, nous renvoyons le lecteur aux ouvrages cités en référence et notamment à ceux de (ULLMANN80, DELOBEL-ADIBA82, MAIER83) pour une étude plus approfondie.

2.4. Dépendances fonctionnelles et de dimensionnement

2.4.1. Dépendances fonctionnelles

Les dépendances fonctionnelles (df) sont les premières dépendances qui furent introduites dans le domaine des bases de données (CODD70).

DÉFINITIONS

Une relation R est munie de la dépendance fonctionnelle (df) notée $X \rightarrow C$ si et seulement si:
-X est un sous-ensemble de R^+ et C un constituant de R,
-pour toute valeur x de X il existe une valeur c de C tel que pour toute entité r de R, si r.X = x alors r.C = c.
La df $X \rightarrow A$ est dite *triviale* si A est un constituant de X.

Exemple:
soit la relation EXAMEN (ÉTUDIANT COURS NOTE DIPLOME PROFESSEUR) qui associe les cours d'un diplôme aux étudiants inscrits à ce diplôme et au professeur chargé de ce cours; chaque étudiant reçoit une note pour chaque cours qu'il a suivi.
ÉTUDIANT COURS $\rightarrow$ NOTE signifie que dans EXAMEN, un étudiant reçoit une seule note par cours.
COURS $\rightarrow$ PROFESSEUR signifie que dans EXAMEN, un seul professeur est chargé d'un cours.

Notation: la df f: $X \rightarrow A$ admet comme partie gauche $g(f) = X$ et comme partie droite $d(f) = A$.

2.4.2. Dépendances de dimensionnement

Pour généraliser les df, nous dégageons deux aspects importants de la définition des df:
- à toute valeur x de X correspond *au plus* une valeur c dans R;
- la valeur c correspondant à x est connue indépendamment des entités r de R qui prennent la valeur x pour X.

Nous généralisons le premier aspect en introduisant un nouveau type de dépendances: les dépendances de dimensionnement.
Nous généralisons le second aspect en introduisant les dépendances relatives et les dépendances de référence (§ 2.6.3 et § 2.6.4).

DÉFINITION

Une relation R est munie de la *dépendance de dimensionnement* (dd) notée X → n → C où X est un sous ensemble de R^+, C un élément de R^+, n un entier positif si
à toute valeur x de X correspondent au plus n valeurs c de C dans R;
si n est égal à 1, la dd est une simple df.
Une dd est une règle d'intégrité statique ensembliste. Nous allons illustrer cette nouvelle dépendance à l'aide de l'exemple suivant:

Exemple DD:
il concerne des contrats d'une société de services informatiques. Pour simplifier nous considérons une seule relation CONTRAT définie de la manière suivante:

CONTRAT (NOCONTRAT EMPLOYÉ CLIENT THÈME
DURÉE DATE DÉBUT RESPONSABILITÉ) avec comme prédicat:
un numéro de contrat (NOCONTRAT) permet de distinguer tous les contrats entre eux. Un contrat est rempli par au plus 10 employés (EMPLOYÉ) et chaque employé a une responsabilité particulière (RESPONSABILITÉ) pour chaque contrat auquel il participe. Un contrat est passé avec un seul client (CLIENT) pour un travail concernant un seul thème (THÈME) qui doit être réalisé en tant de mois (DURÉE) à partir de telle date (DATEDÉBUT).
Le domaine de THÈME est formé des nombreux thèmes de l'informatique comme par exemple bases de données, bureautique, système expert, système d'information ...

Voici certaines dépendances fonctionnelles:
NOCONTRAT → CLIENT THÈME DURÉE DATEDÉBUT
NOCONTRAT EMPLOYÉ → RESPONSABILITÉ.
Comme à un contrat on peut associer au plus 10 employés, il y a la dépendance de dimensionnement suivante:
NOCONTRAT → 10 → EMPLOYÉ.

2.4.3. Propriétés des df

W X Y Z désignent des sous-ensembles de constituants de R^+. Il est relativement facile de démontrer les propriétés suivantes:
(df1) Réflexivité:
si X est inclus dans Y alors Y → X est une df de R. Nous la qualifions de *triviale* car X est inclus dans Y.

(df2) Augmentation:
si $X \rightarrow Y$ est une df de R et si $Z \subset W$ alors $X\,W \rightarrow Y\,Z$ est une df de R.
(df3) Transitivité:
si $X \rightarrow Y$ et $Y \rightarrow Z$ sont des df de R alors $X \rightarrow Z$ en est une autre.
(df4) Pseudo-transitivité:
si $X \rightarrow Y$ et $Y\,W \rightarrow Z$ sont des df de R alors $X\,W \rightarrow Z$ en est une autre.
(df5) Union:
si $X \rightarrow Y$ et $X \rightarrow Z$ sont des df de R alors $X \rightarrow Y\,Z$ en est une autre.
(df6) Décomposition:
si $X \rightarrow Y$ est une df et si Z est inclus dans Y alors $X \rightarrow Z$ est une df.

RÉSULTATS FONDAMENTAUX (df)

Chacune de ces propriétés peut être considérée comme une règle de dérivation. En effet à partir d'un ensemble F de dépendances fonctionnelles de R, il est possible de lui rajouter de nouvelles dépendances fonctionnelles par application de ces propriétés mathématiques. Celles-ci apparaissent alors dans leur rôle de "règles de dérivation" qui permettent de dériver de nouvelles df de R à partir de F . Comme le nombre de constituants de R est fini, l'ensemble de toutes les df que l'on peut dériver de F par applications répétées des règles de dérivation conduit à un ensemble fini unique de dépendances fonctionnelles de R appelé la *fermeture* F** de F. L'utilisation des seules règles de dérivation df1, df2, df3 suffit pour obtenir la fermeture F** de F.

Les règles de dérivation df1, df2, df3 constituent un système valide et complet (ARMSTRONG74) (BEERI FAGIN HOWARD77) c'est-à-dire: d'une part il est valide car elles permettent de dériver effectivement de nouvelles dépendances fonctionnelles de R à partir de F; d'autre part il est complet car si toutes les instances de R validant les df de F valident en plus une df particulière f1, alors f1 appartient à la fermeture F**.

Propriété:

si f et h sont deux dépendances fonctionnelles vérifiant:
$d(f) = d(h)$ et $g(f) \subseteq g(h)$ alors f est dite supérieure à h: $f > h$ et cette relation est une relation d'ordre.

Définitions et propriétés:

La *base complète* F* de F est l'ensemble des dépendances fonctionnelles maximales de F** par rapport à la relation d'ordre précédente: ces dépendances fonctionnelles sont dites *élémentaires.*

G, ensemble de dépendances fonctionnelles, est une *couverture* de F si $G^{**} = F^{**}$; c'est une *base* de F si G est inclus dans F^* (ou égal à F^*).
Une couverture C de F est *irredondante* si aucune dépendance fonctionnelle f de C n'appartient à $(C\text{-}f)^{**}$. Une *base irredondante* de F est une couverture de F qui est irredondante et qui de plus est une base de F.

Exemple:
F = (AB → C; A → DE; D → B; C → A)
La fermeture F^{**} contient A → B (transitivité de A → D et D → B) et aussi A → C (pseudo-transitivité A → B et AB → C).
Donc A → C est supérieure à AB → C et AB → C n'est pas élémentaire.
F^* = (A → C; A → DEB; D → B; C → A; C → DEB)

Voici une base irredondante de F:
$F1^+$ = (A → C; A → DE; D → B; C → A)

Remarque:
La fermeture F^{**} de F contient toutes les couvertures et bases de F; en particulier elle contient F^* et F elle-même qui est une couverture de F.

Clé d'une relation:
Les df permettent de donner une définition plus formelle de la notion de clé d'une relation que celle du chapitre précédent. En annexe nous donnons un exemple d'algorithme d'obtention de toutes les clés d'une relation.

K est une clé d'une relation R si et seulement si
K est inclus dans R^+;
pour tout constituant A de R, K → A est une df de R;
pour tout sous-ensemble K' de K, il existe un constituant B de R tel que K' → B n'est pas une df de R.

Dépendance fonctionnelle intrinsèque:
La df X → A de la relation R est une df intrinsèque de R si X est une clé de R.

2.4.4. Propriétés des dd

Nous laissons le soin au lecteur de démontrer que les dépendances de dimensionnement vérifient également les propriétés de réflexivité, d'augmentation de transitivité, de pseudo-transitivité, d'union et de décomposition définies comme dans le cas des df.

Nous faisons simplement remarquer que si A → n → B et B → m → C sont deux dd, alors par transitivité on peut en déduire A → n m → C.

Les dd vérifient de plus une autre propriété d'augmentation:

Augmentation (bis): si A → n → B est une dd, alors A → p → B en est une autre pour tout p > n.

Clé multiple d'une relation:
KM est une clé multiple de R si et seulement si:
-KM est inclus dans R^+;
-pour tout constituant A de R, KM → n → A est une dd de R;
-pour tout sous-ensemble KM' de KM, il existe un constituant B de R tel que KM' → n → B n'est pas une dd de R.

Dans l'exemple DD relatif à CONTRAT, la relation CONTRAT admet une clé: NOCONTRAT EMPLOYÉ et une clé multiple NOCONTRAT: en effet NOCONTRAT → CLIENT THÈME DURÉE DATEDÉBUT est une df, NOCONTRAT → 10 → EMPLOYÉ est une dd; d'autre part par pseudo-transitivité, on obtient:
NOCONTRAT → 10 → RESPONSABILITÉ.

2.5. Dépendances de composition et décomposition

Les notions de dépendances de composition définies sur une relation et de décomposition d'une relation sont du point de vue formelle très voisines l'une de l'autre. C'est la raison pour laquelle nous les présentons toutes les deux dans ce paragraphe en indiquant leurs différences. La notion de décomposition est tellement importante qu'elle fera l'objet de la troisième partie.

Propriété de base (dc):
pour toute relation R, et pour tout ensemble d'ensembles de constituants R_i^+ de R tel que:

$$R^+ = \bigcup_{k=1}^{n} R_k^+ \text{ alors: } R \subseteq \overset{n}{\underset{k=1}{*}} R[R_k^+],$$

et pour toute instance iR: $iR \subseteq \overset{n}{\underset{k=1}{*}} iR[R_k^+]$.

Les preuves sont immédiates.

Exemple: voici trois instances des relations R, R_1, R_2:

R(ABC)	R_1=R[AB]	R_2=R[AC]
$a_1\ b_1\ c_1$	$a_1\ b_1$	$a_1\ c_1$
$a_1\ b_2\ c_1$	$a_1\ b_2$	$a_1\ c_2$
$a_1\ b_1\ c_2$		

iR ne contient pas $(a_1\ b_2\ c_2)$ qui appartient à $iR_1 * iR_2$. Si elles sont les fermetures des relations, alors R n'admet pas $(a_1\ b_2\ c_2)$ comme entité, qui est pourtant une entité de $R_1 * R_2$.

2.5.1. Définitions et premières propriétés

DÉFINITION

Une *décomposition* D d'une relation R est un ensemble de relations R_1 $R_2...R_k...R_n$ vérifiant:

a) $$R_k = R[R_k^+]$$

b) $$R = \overset{n}{\underset{k=1}{*}} R[R_k^+].$$

D^+ désigne l'ensemble des constituants des relations R_k: $D^+ = R^+$.
D est une décomposition *totale* de R. S'il existe une relation RR dont R est une projection, alors D est une décomposition *partielle* de RR.

Remarque:
La propriété b) implique la propriété: c) $\overset{n}{\underset{k=1}{\cup}} R_k^+ = R^+$.

Notation:
Dorénavant D_k D D' désignent implicitement des décompositions respectivement des relations R_k R et R'.

Propriété (d):

Si un ensemble de relations $(R_1...R_k...R_n)$ vérifie les propriétés a) et c) alors la propriété suivante est toujours vérifiée:

$$R \subseteq \overset{n}{\underset{k=1}{*}} R_k.$$

Preuve:
Pour toute instance iR de R,

$$\forall r \in R \; \exists r_k \in iR_k = iR[R_k^+]: r.R_k^+ = r_i$$

La composition des entités $(r_1...r_k...r_n)$ existe donc, elle est égale à r d'après la propriété c) et appartient à: $\overset{n}{\underset{k=1}{*}} iR[R_k^+]$.

DÉFINITION

Si l'ensemble des relations $R_1...R_k...R_n$ est une décomposition D de la relation R, les entités $(r_1...r_k...r_n)$, une par relation de la décomposition, sont *compatibles entre elles* s'il existe une entité r de R telle que $r.R_k^+ = r_k$. Les instances $iR_1...iR_k...iR_n$ sont *compatibles entre elles* si

$$\left(\overset{n}{\underset{k=1}{*}} (iR_k) \right) [R_j^+] = iR_j \quad (\forall j \in (1,n)).$$

PROPRIÉTÉ DE COMPLÉTUDE

Si les relations $R_1...R_k...R_n$ forment une décomposition D de la relation R, alors pour toute entité r_i, il existe un ensemble d'entités compatibles entre elles $(r_1 r_2...r_k...r_n)$, la contenant, et pour toute instance iR_k, il existe un ensemble d'instances compatibles entre elles $(iR_1 iR_2...iR_n)$ la contenant.

Preuve:
Comme R_k est une projection de R, il existe pour chaque instance iR_k une instance iR de R telle que $iR[R_k^+] = iR_k$. Les instances compatibles iR_j de iR_k sont formées par les projections de iR sur R_j^+.
Pour toute entité r_k de iR_k, on obtient ainsi un ensemble d'entités compatibles.

DÉFINITION

Soit les relations $R_k = R[R_k^+]$. R admet la *dépendance de composition* (dc) notée $*(R_j)$ (j=1...n) ou $(R_1...R_n)$ si et seulement si pour toute instance iR de R validant cette règle d'intégrité on a:

$$iR = \overset{n}{\underset{j=1}{*}} iR[R_j^+].$$

La dc est totale pour R; elle est partielle pour toute relation dont R est une projection.

Propriété (dco):

Si R admet la dc $(R_1...R_n)$, alors $\{R_1...R_n\}$ forme une décomposition de R.
Si R se décompose suivant D: $\{R_1...R_n\}$ alors toute instance de R est contenue dans une instance appelée *complète* de R pour D qui vérifie la dc $(R_1...R_n)$.

Preuve:
Toutes les instances de R valident dc, et en particulier la fermeture de R: R se décompose suivant D.
Soit iR une instance de R; $iR[R_j^+]$ est une instance de R_j car $R_j = R[R_j^+]$.

$iR' = \overset{n}{\underset{j=1}{*}} iR[R_j^+]$ est une instance de R car $R = \overset{n}{\underset{j=1}{*}} R[R_j^+]$.

D'après la propriété de la base $iR \subseteq iR'$.
iR' est une instance complète de R pour D car elle valide la dc $(R_1...R_n)$.
*

Nous qualifions d'*associées* une décomposition D de R et une dc de R qui sont formées sur les mêmes relations.

Corollaire:

Un système valide d'inférences pour les dc l'est également pour les décompositions associées.

Preuve:

Soit une relation R qui se décompose suivant D_1 $D_2...D_p$. Sa fermeture valide les dépendances de composition associées d_1 $d_2...d_p$. Par la règle d'inférence, on peut en déduire qu'elle valide la dc d formée sur $R_1...R_j...R_n$. Comme il s'agit de la fermeture de R, on a:

$$R = \overset{n}{\underset{j=1}{*}} R[R_j^+]$$

autant au niveau des fermetures qu'au niveau des relations. R admet donc $R_1...R_j..R_n$ comme décomposition.

Remarque:

Ce corollaire nous permet de présenter en même temps des règles valides

d'inférences des dc qui le sont également pour les décompositions.
Différences entre une dc et une décomposition de R:

Si la relation R(ABC) se décompose suivant $R_1 = R[AB]$ et $R_2 = R[AC]$, la base de données peut contenir les deux instances suivantes:

iR_1		iR_2	
a	b	a	c
a'	b'	a"	b"

L'instance iR associée à cet état de la base contient une seule entité (ab) et $iR[AB] \subseteq iR_1$, et $iR[AC] \subseteq iR_2$.

Nous considérons qu'on ne connaît pas encore une entité (a' c') de R_2 et une entité (a" b") de R_1, mais que ces entités seront connues dans un état futur de la base. Ainsi la fermeture de R et les fermetures de R_1 et de R_2 vérifieront la dc (R_1 R_2).

L'instance iR formée des entités: a b c, a b'c', a b'c, n'est pas complète: pour la rendre complète il faut lui rajouter a b c'.
Par contre iR ne valide pas la dc (R_1 R_2).

Une autre différence entre une décomposition de R et une dc de R se trouve dans leur rôle dans la définition d'une base de données. Au niveau de la définition d'une structure d'une base de données, nous allons chercher à trouver des décompositions d'une relation R et en choisir une parmi elles: c'est l'objet de la troisième partie. Mais les entités de R ne sont pas réellement stockées dans la base: on peut les recalculer à partir des entités des relations R_k de la décomposition qui, elles, sont stockées.
Par contre au niveau de l'implantation de la base de données, une dc doit être validée comme toute règle d'intégrité.

DÉFINITION

Une décomposition (resp. une dc) est *incluse* dans une autre si et seulement si l'ensemble des relations formant la première, est inclus dans l'ensemble des relations formant la seconde.

Notation:
si D_1 est une décomposition partielle de RR,
si D_2 en est une autre,
si l'ensemble des relations de ($D_1 \cup D_2$) forme une autre décomposition, alors celle-ci est notée D_1 D_2.

Nous utilisons la même convention pour les dc.

2.5.2. Propriétés des dc et des décompositions

Toutes les propriétés de ce paragraphe et du suivant permettent de construire à partir d'une décomposition D (resp. d'une dc d): $(R_1...R_k...R_n)$ de la relation R:
-une nouvelle décomposition D' (resp. dc d') incluse dans la première: propriétés de projection, de réflexivité généralisée, réduction, réduction généralisée;
-une nouvelle décomposition D' (resp. dc d') qui contient la première: propriétés d'extension, d'augmentation;
-une nouvelle décomposition (resp. dc) à l'aide d'une autre décomposition (resp. dc): propriétés de regroupement, substitution, élargissement.

Du point de vue pratique, les premières propriétés vont servir à dépouiller une décomposition ou une dc, de constituants ou de relations: nous les utiliserons notamment pour étudier les seuls relations et constituants intervenant dans une structure cyclique de bases de données.

Les autres propriétés vont servir à déterminer les espaces sémantiques les plus vastes possibles dans une base de données (§ 2.10.2.).

(dc1) La même dc peut s'exprimer à l'aide de n'importe quelle permutation des relations la composant.

(dc2) Extension:
si $S^+ \subseteq R^+$ et si $S = R[S^+]$ alors $(R_1...R_n\ S)$ forme une dc de R.

(dc3) Augmentation:
si $(R_1*...R_k*...R_p)\ [S^+] = S$ $(p < n)$ alors $(R_1...R_k...R_p...R_n\ S)$ forme une dc de R .

(dc4) Projection:
si S^+ est inclus dans R^+ et si chaque constituant de R^+-S^+ appartient à une seule relation R_k alors
$(R_1[R_1^+ \cap S^+]...R_k\ [R_k^+ \cap S^+]...R_n\ [R_n^+ \cap S^+])$ est une dc de R $[S^+]$.
Toutes les charnières de la première dc sont préservées dans la deuxième.

(dc5) Regroupement:
si $(R_1...R_k...R_{n-1}\ S_1...Sj...S_p)$ forme une dc de R
si $(S_1...Sj...S_p)$ forme une dc de R_n

alors (R_1...R_k...R_{n-1} R_n) forme une dc de R.
(dc6) Substitution:
si (R_1...R_k...R_{n-1} R_n) forme une dc de R,
et si (S_1...S_j...S_p) forme une dc de R_n,
alors (R_1...R_k...R_{n-1} S_1...S_j...S_p) forme une autre dc de R.

2.5.3. Propriétés structurelles

(dc7) Réflexivité généralisée:
si (R_2*...R_j*...R_m) [R_1^+] = R_1 (m < n) alors (R_2...R_j...R_m...R_n) est une autre dc de R.

DÉFINITION

Une relation R_1 est une *extrémité* de la dc d si et seulement si pour l'ensemble de constituants Chmax de R vérifiant:

$$\text{Chmax} = \bigcup_{k=2}^{n} (R_k^+ \cap R_1^+) : \exists\, k \in (2..n) \text{ tel que } R_k^+ \cap R_1^+ = \text{Chmax}.$$

Si R_1 est une extrémité de d, elle l'est également de la décomposition (R_1...R_k...R_n).

Exemple:
-Soit la décomposition de R: D = {R_1(ABC) R_2(BCD) R_3(CDE)};
R_1 est une extrémité de D (Chmax = BC) ainsi que R_3(Chmax = CD).
-Soit une décomposition de R':
D' = {R_1(ABC) R_2(BCD) R_3(CDE) R_4(ACG)}.
R_1 n'est pas une extrémité de D' car Chmax = ABC et aucune autre relation de D' ne contient Chmax.

(dc8) Réduction:
Si R_1 est une extrémité de d, (R_2...R_n) est une dc partielle d' de R.
d' est obtenue par *réduction* de d.

DÉFINITIONS

Une décomposition est qualifiée de *linéaire* si l'application successive de la réduction conduit à une décomposition réduite à une relation. Elle est qualifiée de *cyclique* sinon.
Une décomposition D_1 de R_1 est une *extrémité* d'une décomposition D_2 de R_2 si elle y est incluse, et si R_1 est une extrémité de la décomposition obtenue par regroupement dans D_2 de D_1 en R_1 (propriété dc 5).
Ces définitions s'appliquent également aux dc.

Exemple:
Soit la décomposition de R: D = {R_{11}(AB) R_{12}(AC) R_2(BCD) R_3(CDE)}.
Soit la décomposition de R_1 = R[ABC]: D_1 = {R_{11}(AB) R_{12}(AC)}.
D_1 est une extrémité de D (Chmax = BC).

Soit la décomposition de R': D' = {R_{11}(AB) R_{12}(AC) R_2(BCD) R_3(CDE)}.
Soit la décomposition de R_1 = R'[ABC]: D_1 = {R_{11}(AB) R_{12}(AC)}.
D_1 n'est pas une extrémité de D (Chmax = ABC).

(dc9) Réduction généralisée:
Si la dc d' est une extrémité de la dc d alors l'ensemble des relations de d qui n'appartiennent pas à d' forme une dc partielle.

(dc10) Élargissement:
Soit d_1 une dc de R_1, d_2 une de R_2 et (R_1, R_2) une de R_{12} alors d_1 d_2 est une dc de R_{12}.

2.5.4. Propriétés des df et des dc

(df-dc1) Soit une dc (R_1 R_2) de R et une df AB → C de R telles que A et B sont des constituants de R_1 et C est un constituant de R_2.
Soit Ch^+ l'ensemble des constituants communs de R_1 et de R_2.
Alors Ch^+ → C est une df de R.

Preuve:
Pour toute instance iR de R validant la dc (R_1 R_2) et AB → C, et pour toutes entités r, r' de iR telles que r. Ch^+ = r'.Ch^+ = ch, on peut alors construire les entités suivantes:

entités	A	B	Ch^+	C	
r	a	b	ch	c	r ∈ R
r'	a'	b'	ch	c'	r' ∈ R
r.R_1^+ = r_1	a	b	ch		r_1 ∈ R_1
r.R_2^+ = r_2			ch	c	r_2 ∈ R_2
r'.R_1^+ = r_1'	a	b	ch		r_1' ∈ R_1
r'.R_2^+ = r_2'			ch	c'	r_2' ∈ R_2
r_1*r_2' = r_{12}	a	b	ch	c'	r_{12} ∈ R

A cause de la df AB → C, c = c' et r_{12} = r.

(df-dc2) Si la relation R(ABC) est munie de la df $A \rightarrow B$ alors (R[AB] R[AC]) est une dc de R.

Preuve:
Pour toute instance iR de R validant $A \rightarrow B$ alors $\forall r \in iR\ [AB] * iR\ [AC]$, on peut construire les entités suivantes:

entités	A	B	C	
r	a	b	c	
r_1	a	b		$r_1 \in$ iR[AB]
r_2	a		c	$r_2 \in$ iR[AC]
rab	a	b	c'	rab $\in$ iR car $r_1 \in$ iR [AB]
rac	a	b'	c	rac $\in$ iR car $r_2 \in$ iR[AC]

A cause de la df $A \rightarrow B$, b = b' et ainsi rac = r. r est une entité de iR.

2.5.5. Dépendances arborescentes

DÉFINITION

Il existe un cas particulier important de dépendance de composition: les dépendances arborescentes (da) (DELOBEL73; DELOBEL-LÉONARD74). Une *dépendance arborescente* est une dépendance de composition où 2 à 2 les relations formant la dc ont les mêmes constituants en commun qui forment la racine de la dépendance arborescente.

Voici un exemple d'une dc qui est une da:
(R_1(ABC) R_2(AF) R_3(ADE)); sa racine est A et elle peut également s'écrire comme une dépendance arborecente A: BC/F/DE.

Les propriétés des da se déduisent de celles des dc, comme par exemple:
(dc1) A: F/BC/DE;
(dc4) A: F/B/D et également A: BC/DE;
(dc5) A: FBC/DE;
(dc6) si l'on sait en plus que A: B/C alors on peut obtenir: A: B/C/F/DE;
(dc8) les da sont des dc linéaires;
(dc-df1) si $F \rightarrow E$ est une df alors $A \rightarrow E$ en est une;
si $AF \rightarrow E$ est une df alors $A \rightarrow E$ en est une.
Les dépendances arborescentes ont d'autres propriétés comme celle qui permet de déduire de la da considérée, celle-ci: da1) AE: BC/F/D.

2.5.6. Les dépendances multivaluées

Introduites par (FAGIN77), elles sont un cas particulier des da. La *dépendance multivaluée* AB →→ C est définie dans R si et seulement si ABC sont des sous-ensembles de constituants de R^+ et si AB: C/D est une da, D étant un sous-ensemble de constituants de R tel que D = R^+ - ABC.

Cette notion de dépendance multivaluée ne doit surtout pas être confondue avec la notion de dépendance de dimensionnement
Ainsi dans l'exemple DD, la dd NOCONTRAT → 10 → EMPLOYÉ est définie dans la relation CONTRAT: elle signifie qu'à un contrat on peut associer au plus 10 employés.
La dépendance multivaluée NOCONTRAT→→EMPLOYÉ signifierait l'existence de la da suivante:
NOCONTRAT:EMPLOYÉ / CLIENT THÈME DURÉE DATE DÉBUT RESPONSABILITÉ.
En particulier on pourrait en déduire la da suivante par projection (dc4):
NOCONTRAT: EMPLOYÉ / RESPONSABILITÉ.
Elle signifierait que tout employé engagé dans un contrat assume toutes les responsabilités prévues pour ce contrat!
On pourrait déduire de cette da et de la df:
NOCONTRAT EMPLOYÉ → RESPONSABILITÉ,
la df suivante par application de la propriété (dc-df1):
NOCONTRAT → RESPONSABILITÉ.
Tous les employés engagés dans un même contrat aurait en fait la même responsabilité. Cette conclusion n'est pas compatible avec l'application décrite.
Cette application admet la dd: NOCONTRAT → 10 → EMPLOYÉ,
mais n'admet pas la dépendance multivaluée:
NOCONTRAT →→ EMPLOYÉ.

De plus cette notion de dépendance multivaluée ne doit surtout pas être confondue avec une autre notion qui s'exprime à l'aide des dépendances d'inclusion (présentées au paragraphe suivant).
Nous illustrons cette différence avec le même exemple que précédemment: nous supposons, en plus, qu'un employé ne peut être engagé dans un contrat que s'il est compétent dans le thème du contrat.

Cette règle ne se modélise pas par la dépendance multivaluée suivante: EMPLOYÉ →→ THÈME dans la relation CONTRAT. En effet cette dépendance multivaluée s'écrit comme la da suivante:
EMPLOYÉ:THÈME / NOCONTRAT CLIENT DURÉE DATE DÉBUT RESPONSABILITÉ.

Celle-ci pourrait se projeter suivant la da suivante:
EMPLOYÉ:THÈME / NOCONTRAT.

A partir de la df NOCONTRAT → THÈME, on pourrait en déduire (propriété dc-df1) la df: EMPLOYÉ → THÈME. Un employé ne pourrait plus être compétent que pour un seul thème! La dépendance multivaluée EMPLOYÉ →→ THÈME n'est donc pas définie dans CONTRAT.

La situation invoquée dans cet exemple demande la création de la relation COMPÉTENCE (EMPLOYÉ THÈME), dont le prédicat est évident, et l'introduction de la règle d'intégrité suivante:
CONTRAT[EMPLOYÉ THÈME] ⊆ COMPÉTENCE.
Il faut une grande prudence pour utiliser les dépendances multivaluées: à notre avis, mieux vaut utiliser les dépendances arborescentes qui leur sont équivalentes mais qui explicitent dans leur formulation l'ensemble des constituants concernés.

2.6. Dépendances d'inclusion

Voici la forme la plus générale d'une dépendance d'inclusion:
RR ⊆ RS où RR et RS sont deux relations.
Elle signifie que toute entité de RR est également une entité de RS.
Elle réclame bien sûr que $RR^+ = RS^+$.

De telles dépendances ont fait l'objet de relativement peu d'études (CASANOVA-FAGIN-PAPADIMITRIOU82; MITCHEL83; ABITEBOUL85; HUONG87).
Les différents types de dépendances d'inclusion proviennent de la nature des relations RR et RS: ces relations peuvent en effet désigner des relations de la modélisation ou bien des relations obtenues par la composition de relations de la modélisation suivie éventuellement d'une projection. Ces dépendances sont généralisables à toutes les expressions relationnelles.

Exemple (laboratoire de recherches):

Un chercheur identifié par son nom (NOMCH) appartient à un laboratoire de recherches identifié par son sigle (LABO). Un chercheur écrit plusieurs articles identifiés par leurs numéros (NOART). Les domaines (DOM) de chacun de ses articles doivent être des domaines de recherches de son laboratoire. Voici les relations:

ARTICLE(NOMCH NOART) de clé: NOMCH NOART,
LABORATOIRE(NOMCH LABO) de clé: NOMCH,
DOM-ART(NOART DOM) de clé: NOART DOM,
DOM-LABO(LABO DOM) de clé: LABO DOM.

Voici une dépendance existentielle:
ARTICLE[NOMCH] $\subseteq$ LABORATOIRE [NOMCH].

Voici une dépendance d'inclusion:
(ARTICLE * DOM-ART) [NOMCH DOM] $\subseteq$
(LABORATOIRE * DOM-LABO) [NOMCH DOM].

2.6.1. Dépendances existentielles

RR et RS sont des projections de simples relations. La dépendance d'inclusion s'appelle alors dépendance existentielle. Le cas le plus important de dépendances existentielles est celui où l'une des relations de la modélisation R_k admet comme constituants, des constituants formant une clé KR_j d'une relation R_j: généralement il y a alors la dépendance existentielle suivante: $R_k[KR_j] \subseteq R_j[KR_j]$.

2.6.2. Dépendances d'inclusion particulières

Nous allons nous intéresser principalement aux dépendances d'inclusion admettant pour RR ou pour RS, une relation de la modélisation éventuellement projetée sur l'une ou l'autre de ses clés. Cette limitation conduit à la création de nouvelles relations: nous pensons que cette méthode est pertinente pour la compréhension de la modélisation.

Ainsi dans l'exemple précédent nous allons créer une nouvelle relation formée sur NOMCH DOM qui se définit comme:

(LABORATOIRE*DOM-LABO) [NOMCH DOM],
ou bien comme: (ARTICLE*DOM-ART) [NOMCH DOM],
ou bien avec un prédicat inférieur au premier et supérieur au second.

Dans cet exemple, une association entre un chercheur *ch* et un domaine de recherches *d* a comme sens "naturel" que *d* est un des domaines de recherches de *ch*. C'est le prédicat que nous choisissons pour la nouvelle relation que nous appelons COMPÉTENCE.

La précédente dépendance d'inclusion se transforme en deux dépendances d'inclusion:
(ARTICLE * DOM-ART) [NOMCH DOM] $\subseteq$ COMPÉTENCE et
COMPÉTENCE $\subseteq$ (LABORATOIRE * DOM-LABO) [NOMCH DOM].

Ces dépendances d'inclusion sont d'un type particulier; nous appelons la première, *dépendance de référence* et la seconde, *dépendance relative*. Nous qualifierons cette nouvelle relation COMPÉTENCE de *multifonctionnelle* (§ 2.10.1.).

2.6.3. Dépendances de référence (dr)

Soit deux décompositions D = (R_1...R_n) de R et D' = (R_1'...R_m') de R'.
La dépendance de référence D -: D' n'existe que si $R'^+ \subseteq R^+$.
Elle signifie que: $R[R'^+] \subseteq R'$.

Propriétés des dépendances de référence:

(dr1) Réflexivité:
Si D' est une décomposition partielle de D alors D -: D'.

Preuve:
D est une décomposition de R, D' une décomposition de R_n.
D = (R_1...R_k...R_{n-1} S_1...S_j...S_p) et D' = (S_1...S_j...S_p).
D'après la propriété de regroupement (dc5), (R_1...R_k...R_{n-1} R_n) forme une décomposition de R. Ainsi $R[R_n^+] = R_n$ et D -: D'.

(dr2) Union:
Si D -: D_1 et D -: D_2 sont deux dépendances de référence et si D_1 D_2 est une décomposition partielle, alors D -: D_1 D_2.
La démonstration est immédiate.

(dr3) Transitivité:
Si D -: D' et D' -: D" alors D -: D".
La démonstration est immédiate.

(dr4) Pseudo-transitivité:
Si D_1 -: D_2 et D_2 D_3 -: D_4 sont deux dépendances de référence et si D_1 D_3 est une décomposition, alors D_1 D_3 -: D_4 est une nouvelle dépendance de référence.

La preuve se déduit des trois premières propriétés.

(**dr5**) Décomposition:
Si D_{22} est une décomposition partielle de D_2, D_1 -: D_2 implique D_1 -: D_{22}.

Preuve:
Comme D_{22} est une décomposition partielle de D_2, on a D_2 -: D_{22} (réflexivité). Par transitivité on obtient donc D_1 -: D_{22}.

(**dr6**) Projection:
Si D_{11} est une décomposition partielle de D_1 et si $D_2^+ \subseteq D_{11}^+$, alors D_1 -: D_2 implique D_{11} -: D_2.

Preuve:
$\forall r_2 \in R_{11}[D_2^+]$ $\exists r_{11} \in R_{11}$ telle que $r_{11}.D_2^+ = r_2$.
Comme D_{11} est une décomposition partielle de D_1 et à cause de la propriété de complétude, il existe une entité r_1 de R_1 telle que $r_1.R_{11}^+ = r_{11}$. A cause de la dépendance de référence D_1 -: D_2, il existe une entité r_2' de D_2 telle que $r_1.D_2^+ = r_2'$.
Comme $r_1.R_2^+ = r_{11}.R_2^+$, $r_2 = r_2'$.

2.6.4. Dépendances relatives (div)

Soit deux décompositions $D = (R_1...R_n)$ de R et $D' = (R_1'...R_m')$ de R'. La dépendance relative D +: D' n'existe que si $R'^+ \subseteq R^+$. Elle signifie que: $R' \subseteq R[R'^+]$.

Propriétés des dépendances relatives:

(**div1**) Réflexivité:
Si D' est une décomposition partielle de D alors D +: D'.

Preuve:
$D = (R_1...R_{n-1}\ S_1...S_j...S_p)$: décomposition de R.
$D_n = (S_1...S_j...S_p)$: décomposition de R_n.
Par la propriété de regroupement (dc5) on obtient:
$R[R_n^+] = R_n$. Ainsi D +: D'.

(**div2**) Isotonie
Si D_1 +: D_{11} et D_2 +: D_{22},
et si (D_{11} D_{22}) forment des décompositions,
et si la charnière $(R_1\ R_2)^+$ est incluse (ou égale) dans celle de $(R_{11}\ R_{22})^+$,
alors $D_1\ D_2$ +: $D_{11}\ D_{22}$ est une autre dépendance relative.

Preuve:

$\forall r_{1122} \in R_{1122}$, $\exists r_{11}$ de R_{11} et r_{22} de R_{22} telles que:
$r_{11} = r_{1122}.R_{11}^+$ et $r_{22} = r_{1122}.R_{22}^+$,
car $(D_{11} \cup D_{22})$ est une décomposition de R_{1122}.
A cause des deux dépendances relatives, il existe r_1 de R_1 et r_2 de R_2 telles que: $r_1.R_{11}^+ = r_{11}$ et $r_2.R_{22}^+ = r_{22}$.
r_1 et r_2 sont compatibles car la charnière de R_1 et de R_2 est incluse dans celle de R_{11} et de R_{22}. $r_{12} = (r_1 * r_2)$ est une entité de R_{12} car $(D_1 \cup D_2)$ en est une décomposition; r_{12} vérifie par construction $r_{12}.R_{1122}^+ = r_{1122}$.

Exemple:
Voici un exemple où la propriété d'inclusion des charnières n'est pas vérifiée:

D_1: S(ABE) D_{11}: V(AB)
D_2: T(ACE) D_{22}: U(AC).

S(ABE)	V(AB)	T(ACE)	U(AC)
a b e	a b	a c e	a c
a b'e'	a b'	a c'e'	a c'

S*T
a b c e
a b'c'e'

V*U
a b c
a b'c
a b c'
a b'c'

Bien que $V \subseteq S[AB]$ et $U \subseteq T[AC]$,
V*U n'est pas contenue dans (S*T) [ABC].

(div3) Transitivité:
Si D_1 +: D_2 et D_2 +: D_3 sont deux dépendances relatives alors D_1 +: D_3 en est une également.
La preuve est immédiate.

(div4) Pseudo-transitivité:
Si D_1 +: D_2 et D_2 D_3 +: D_4,
si D_1 D_3 est une décomposition,
si $(R_1\ R_3)^+$ est inclus (ou égale) dans $(R_2\ R_3)^+$,
alors D_1 D_3 +: D_4.

Preuve:
Il suffit d'appliquer la propriété d'isotonie aux dépendances relatives D_1 +: D_2 et D_3 +: D_3 pour obtenir D_1 D_3 +: D_2 D_3;
par transitivité on obtient D_1 D_3 +: D_4.

Exemple où la condition relative aux charnières n'est pas vérifiée:
Soit les dépendances relatives suivantes:
R_1 (ABC) +: R_2 (AC),
(R_2 (AC) R_3 (BCE)) +: R_4 (AE),
$(R_1\ R_3)^+$ = BC et $(R_2\ R_3)^+$ = C,
$(R_1\ R_3)^+$ n'est pas incluse dans $(R_2\ R_3)^+$.

R_1	R_2	R_3	R_4
a b c	a c	b c e	a e
a b c'	a'c'	b c'e'	a'e'
a'b c			

R_1*R_3
a b c e
a b c'e'
a'b c e

(R_1*R_3) [R_4^+] ne contient pas R_4.

(div5) Décomposition:
D_{22} est une décomposition partielle de D_2.
Si D_1 +: D_2 alors D_1 +: D_{22}.

Preuve: par réflexivité on a D_2 +: D_{22} et par transitivité D_1 +: D_{22}.

(div6) Projection:
D_{11} est une décomposition partielle de D_1 et D_2^+ est inclus dans D_{11}^+.
Si D_1 +: D_2 alors D_{11} +: D_2.
La preuve est immédiate.

2.6.5. Dépendances de composition-projection

Soit deux décompositions D = (R_1...R_n) et D' = (R_1'...R_m'). La dépendance de composition-projection D =:D' n'existe que si $R'^+ \subseteq R^+$.
Elle signifie que: R' = R[R'+].
Une dépendance de composition-projection est un cas particulier de dépendance relative et de dépendance de référence.

Propriétés des dépendances de composition-projection:
elles découlent des propriétés des dépendances relatives et de référence.

(dcp1) Réflexivité:
Si D' est une décomposition partielle D alors D =: D'.

(dcp2) Isotonie:
Si D_1 =: D_{11} et D_2 =: D_{22},
si (D_1 D_2) et (D_{11} D_{22}) forment des décompositions,
si la charnière $(R_1 R_2)^+$ est incluse dans $(R_{11} R_{22})^+$ ou lui est égale,
alors D_1 D_2 =: D_{11} D_{22}.

(dcp3) Transitivité:
Si D_1 =: D_2 et D_2 =: D_3 alors D_1 =: D_3.
La preuve est immédiate.

(dcp4) Pseudo-transitivité:
Si D_1 =: D_2 et D_2 D_3 =: D_4 sont deux dépendances de composition-projection,
si D_1 D_3 est une décomposition,
si $(R_1 R_3)^+$ est incluse dans $(R_2 R_3)^+$,
alors D_1 D_3 =: D_4.

(dcp5) Décomposition:
D_{22} étant une décomposition partielle de D_2,
si D_1 =: D_2 alors D_1 =: D_{22}.

(dcp6) Projection:
D_{11} étant une décomposition partielle de D_1 et $D_2^+ \subseteq D_{11}^+$,
si D_1=:D_2 alors D_{11}=:D_2.

2.7. Cycles de relations

Voici une représentation graphique simple d'un ensemble de relations à l'aide d'un réseau de noeuds et d'étoiles:

- à chaque constituant on associe un noeud
- à chaque relation on associe une étoile
- une arête relie un noeud et une étoile si et seulement si le constituant du noeud appartient à la relation de l'étoile.

DÉFINITIONS

Dans la représentation graphique d'un ensemble de relations, un *cycle* (cf:la définition d'un cycle dans un hypergraphe de BERGE70) est une séquence de la forme: (a_{11} S_1 a_{12} S_2 a_{23} S_3 ... a_{n-1n} S_n a_{nn}) où:

- chaque aij est un constituant de la relation S_i et de la relation S_j,
- tous les a_{ij} sont distincts 2 à 2,
- tous les S_j sont distincts 2 à 2,
- $a_{nn} = a_{11}$,
- $n > 1$.

Les relations (S_1 S_2...S_n) forment alors un *cycle de relations.*

Soit S^+ l'ensemble des constituants des noeuds appartenant à tous les cycles construits à partir des relations (S_1 S_2...S_n).
Les relations $R_k = S_i [S_i^+ \cap S^+]$ forment un *cycle propre de relations.*

Exemple:

S_1(ABDX), S_2(BCDY), S_3(CAZ) forment un cycle de relations.

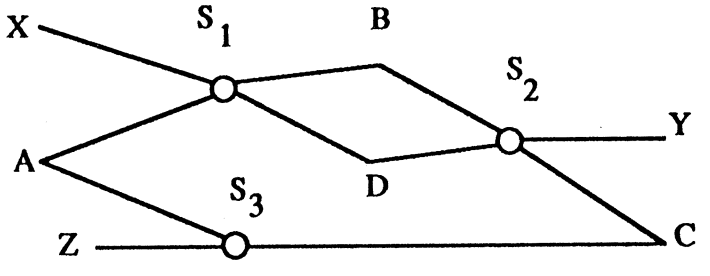

Figure 2.2 Exemple de cycles

Cette représentation graphique contient les deux cycles suivants:
(A S_1 B S_2 C S_3 A) et (A S_1 D B S_2 C S_3 A).

Les relations S_1 S_2 S_3 forment un cycle de relations dont voici l'ensemble de constituants S^+ = ABDC. Les relations S_1[ABD], S_2[BCD], S_3[CA] forment un cycle propre de relations.

Il existe un autre cycle: (B S_2 D S_1 B): les relations S_1 S_2 forment un autre cycle de relations.
fex.

Les cycles de relations sont très importants dans l'étude d'une modélisation car ils montrent l'existence de plusieurs associations possibles entre des mêmes constituants. Ainsi dans l'exemple précédent, une association entre les constituants ABD peut être obtenue de deux manières: S_1[ABD] et (S_2 * S_3)[ABD].

Ces deux associations sont-elles *indépendantes*? Ou bien existe-t-il une règle d'intégrité ?

Pour nous, le concepteur a la responsabilité de répondre à une telle question et nous allons montrer comment les dépendances d'inclusion le lui permettent.

THÉORÈME (di)

Si $R_1,R_2...R_n$ forment un cycle propre de relations et s'il existe la dépendance de référence $R_1R_2...R_{n-1}$ -: R_n, alors il existe toutes les dépendances relatives suivantes:
si $(R_1...R_{k-1}\ R_{k+1}...R_n)$ forme une décomposition,
alors $\forall\, i \in (2,...,n-1)$ $\quad R_1...R_{k-1}R_{k+1}...R_n$ +: R_k,
et si $(R_2...R_n)$ forme une décomposition, alors $\quad R_2...R_{n-1}R_n$ +: R_1.

Preuve:
$\forall\, r_1 \in R_1\ \exists\, (r_2...r_{n-1}) \in R_2*...R_{n-1}$ compatibles entre elles avec r_1 pour former une entité $r = (r_1*r_2...*r_{n-1})$ de $(R_1*...R_{n-1})$ d'après la propriété de complétude appliquée à la décomposition $(R_1...R_{n-1})$.

Comme $R_1\ R_2...R_{n-1}$ -: R_n, il existe une entité r_n de R_n telle que $r.R_n^+ = r_n$. Par construction, $(r_2..r_i..r_n)$ sont compatibles (pour i = 2...n) et il existe $r' = r_2*...*r_n$ entité de $R_2*...*R_n$ et $r'.R_1^+ = r_1$.

Donc $R_2...R_n$ +: R_1 si $(R_2...R_n)$ forme une décomposition.

Corollaire (di):
Si $R_1,R_2...R_n$ forment un cycle propre de relations et si les dépendances de référence suivantes sont vérifiées:

a) $\quad R_1R_2...R_{n-1}$ -: R_n

et b) $\quad R_2...R_{n-1}\ R_n$ -: R_1

alors les dépendances de composition-projection suivantes le sont également:

c) $\quad R_iR_2...R_{n-1}$ =: R_n

et d) $\quad R_2...R_{n-1}\ R_n$ =: R_1.

Preuve:
par le théorème précédent on déduit de a): e) $R_2 \ldots R_{n-1}\ R_n +: R_1$;
de b) et de e) on déduit d) par définition de la dépendance de composition-projection.

Face à un cycle, le concepteur cherche à mettre en évidence d'abord toutes les dépendances de composition-projection et les dépendances de référence; s'il en trouve au moins une, son travail est terminé d'après le théorème (di). Sinon, il cherche à mettre en évidence les dépendances relatives.

Il détermine ainsi toutes les dépendances d'inclusion et l'interdépendance des entités des relations du cycle entre elles.

Les dépendances d'inclusion ne sont pas les seules règles d'intégrité qui peuvent exister entre les relations formant un cycle: bien sûr! Mais elles permettent une étude très sérieuse d'un problème difficile.

L'exemple CONCOURS à la fin de ce chapitre montre une application du théorème précédent.

Théorème (v):

$(R_1\ R_2 \ldots R_n)$ formant une décomposition D de R,
si $(R_2 \ldots R_n)$ est une autre décomposition de R alors il existe la dépendance de composition-projection: $R_2 \ldots R_n =: R_1$.
La réciproque est vérifiée.

Preuve:
a) Comme $R_2 * \ldots * R_n = R$ et que $R[R_1^+] = R_1$ puisque D est une décomposition, $R_2 \ldots R_n =: R_1$ est vérifiée.
b) La réciproque découle de la propriété de réflexivité généralisée (dc7).

Corollaire (v):

Si $R_2 \ldots R_n$ -: R_1 ou $R_2 \ldots R_n +: R_1$ existe, alors les relations $R_1 R_2 \ldots R_n$ ne peuvent former une décomposition de la relation R; R se décompose par contre en $(R_2 \ldots R_n)$.

Ce corollaire est très utile dans la détermination des contextes: les relations précédentes $R_1 R_2 \ldots R_n$ ne formant pas une décomposition d'une relation, ne peuvent pas appartenir à un même contexte.

2.8. Autres règles d'intégrité

Les règles d'intégrité peuvent prendre des formes très diverses et *il n'est pas possible de fournir un canevas exhaustif* permettant de trouver toutes les règles d'intégrité d'une modélisation. Néanmoins, nous allons proposer un canevas qui permet au concepteur de se poser de bonnes questions pour approfondir sa connaissance du champ d'application et de mettre en évidence des règles d'intégrité. Ce canevas s'applique à chaque espace l'un après l'autre.

a) *Cas des relations ayant plusieurs clés*

Soit une relation R admettant plusieurs clés K_1, K_2 ...
Soit une relation S telle que S^+ contient une ou plusieurs clés de R.
Il s'agit de savoir si la (ou les) clé(s) de R contenue(s) dans S^+ est bien choisie pour assurer les associations entre les entités de S et celles de R.

Ainsi dans l'exemple CONCOURS (§ 2.10.4.) de la fin de ce chapitre, il y a notamment deux relations
RÉSULTATS (JURÉ NOTE TITRE-EXPOSÉ) et
SUJETS (TITRE-EXPOSÉ CANDIDAT THÈME DEMI-JOURNÉE
NOTE-FINALE).
Rapidement dit, ces relations signifient qu'un candidat passant un oral dans un thème donné, présente un exposé lors d'une demi-journée devant les jurés qui chacun le note; il obtient une note finale pour cet exposé.

RÉSULTATS admet une clé:	JURÉ TITRE-EXPOSÉ,
SUJETS admet trois clés:	TITRE-EXPOSÉ,
	CANDIDAT THÈME,
	CANDIDAT DEMI-JOURNÉE.

Dans la modélisation retenue se cache une règle d'intégrité un peu sournoise: toute note de juré ne peut être stockée que si le titre de l'exposé est connu.
Comme la relation SUJETS a plusieurs clés, on aurait pu imaginer une autre relation à la place de RÉSULTATS :

RES (JURÉ NOTE CANDIDAT THÈME),
ou encore RES (JURÉ NOTE CANDIDAT DEMI-JOURNÉE).

Si l'on choisit RES de préférence à RÉSULTATS, c'est maintenant le candidat et le thème qui sont nécessaires pour stocker la note d'un juré.

b) *Constituants appartenant à plusieurs relations ayant un domaine commun*

Le concepteur doit être averti qu'une telle situation est analogue à celle d'un cycle de relation. Dans la représentation graphique des relations composant la modélisation il suffit par convention de réunir les noeuds de tous les constituants ayant un domaine commun à l'étoile d'une même et nouvelle relation qui a pour nom le nom du domaine. Par cette opération, on crée généralement un cycle de relations qui pose le problème pertinent de l'indépendance des associations à l'intérieur de ce cycle.

Dans l'exemple CONCOURS (§ 2.10.4.), nous présentons un exemple d'une telle situation avec les constituants NOTE et NOTE-FINALE.
Les constituants exprimant des dates, des intervalles de temps sont des exemples fréquents de tels constituants.

c) *Constituants exprimant des mesures*

Souvent des règles d'intégrité concernent des sommes de ces mesures.
Par exemple, la somme des montants des lignes de commande (relation LIGNE-DE-COMMANDE) doit être égale au montant de la commande (relation COMMANDE) dans le cas des relations suivantes:
COMMANDE (NOCDE MONTANT NBLGN):
une commande est désignée par un numéro (NOCDE); on lui associe son montant (MONTANT) et le nombre de lignes de commande la composant;
LIGNE-DE-COMMANDE(NOCDE NPRODUIT MONTANTLGN):
il y a une ligne de commande par produit pour chaque commande et on y écrit la valeur de la quantité du produit commandé (MONTANTLGN).

d) *Constituants exprimant des dénombrements*

Dans l'exemple précédent, le nombre de lignes de commandes (NBLGN) de la relation COMMANDE pour un produit est égal au nombre de lignes de commandes (relation LIGNE-DE-COMMANDE) de cette commande.

e) *Constituants exprimant un état ou un critère de classement*

Généralement, il existe des règles d'intégrité exprimant des conditions pour que telle entité ait une valeur particulière pour un tel constituant.
Ainsi, dans l'exemple de l'atelier de production, la modélisation a mis en évidence les relations DISP-MACHINE et EMPLOI-DU-TEMPS.
DISP-MACHINE est formée par les constituants NMH (numéro de ma-

chine), NH (numéro d'une heure dans la semaine), DISPMH (disponibilité de la machine qui admet comme valeurs "disponible", "occupée", "en révision").
EMPLOI-DU-TEMPS est formée notamment par les constituants NP (numéro du produit concerné), NLOT (numéro du lot), NH et NMH, et associe à un lot, la machine qui va le produire et l'heure de production.
Le constituant DISPMH exprime un état et il entraîne la règle d'intégrité suivante: aucun lot ne peut être produit par une machine en révision.

2.9. Tableau des portées des ri

2.9.1. Portée de chaque ri

Il s'agit de déterminer la portée de chaque ri: c'est-à-dire d'indiquer quelles sont les primitives de modifications qui doivent faire l'objet d'une validation. Ces primitives concernent chaque relation de base: *créer* une entité (*c*), *supprimer* une entité (*s*), *mettre à jour* la valeur prise par une entité pour un constituant (*maj*), interroger des entités (*i*).

Les règles d'intégrité suivantes sont *supposées* validées:

-les dépendances fonctionnelles intrinsèques d'une relation de base;
-les domaines des constituants: la valeur prise par une entité pour un constituant appartient au domaine du constituant;
-les dépendances existentielles entre deux relations R et S quand une clé KS de l'une (S) est incluse dans l'ensemble des constituants de l'autre: au niveau des instances stockées en même temps dans la base, on suppose que la condition $iR[KS] \subseteq iS[KS]$ est toujours vérifiée.

Nous ne spécifions donc pas la portée de ces règles d'intégrité.

2.9.2. Tableau des portées

Nous allons construire un tableau des portées des règles d'intégrité: chaque ligne k correspond à une règle d'intégrité ri_k. Chaque colonne j correspond à une relation R_j de la modélisation. Chaque case du tableau à l'intersection de la ligne k et de la colonne j contient les primitives de modification de la relation R_j qui doivent faire l'objet d'un algorithme de validation de ri_k.

Nous donnons dans le paragraphe 2.10.4, le tableau des portées des règles d'intégrité de l'exemple CONCOURS.

Un tel tableau permet ensuite le réalisation de ces différentes primitives et plus tard la programmation d'applications sans le souci de la validation des ri si cette programmation utilise justement ces primitives. Un autre avantage de l'utilisation d'un tel tableau se trouve lors du changement de règles d'intégrité en fonction de l'évolution de l'environnement de la base de données: qu'une règle d'intégrité soit modifiée, supprimée ou introduite, on sait exactement quelles sont les primitives à modifier; de plus les programmes d'application ne devront pas être modifiés dans la plupart des cas: si l'action à entreprendre en cas de non validation d'une règle d'intégrité dépend du contexte du programme d'application, celui-ci peut subir des modifications si l'on change la ri.
Nous appelons une telle manière de réaliser l'ensemble des programmes d'application d'une base de données, *l'architecture normale* d'une base de données.

2.10. Conclusions

Nous allons maintenant montrer comment l'étude des règles d'intégrité s'insère dans le processus de modélisation et comment, à notre avis, elle aide le concepteur à mieux comprendre le champ d'application.
Le processus de modélisation a déjà mis en évidence plusieurs relations que nous appelons les *relations de base*. Le concepteur peut poursuivre son travail par l'analyse de ces relations de base à l'aide notamment des différentes dépendances, et ainsi enrichir la modélisation.

D'abord nous présentons deux aspects très importants dans l'analyse d'une modélisation:
- l'*indépendance* de certaines données par rapport à d'autres, et en corollaire l'interdépendance de certaines données: il va s'exprimer à l'aide des dépendances d'inclusion;
- la détermination des espaces formés par un ensemble de relations de base; chacun forme une décomposition d'une relation appelée *espace*. Nous montrons à l'aide d'un exemple l'importance de cette notion.

Ensuite nous présentons un canevas de travail au concepteur pour analyser une modélisation et éventuellement la modifier et l'enrichir. Nous finissons par la présentation d'un *exemple complet* CONCOURS.

2.10.1. Indépendance: relation multifonctionnelle

Nous allons nous servir de la relation CONTRAT (NOCONTRAT EMPLOYÉ CLIENT THÈME DURÉE DATE DÉBUT RESPONSABILITÉ) de l'exemple DD du § 2.4.2.
Un contrat est passé pour un travail concernant un seul thème.

CONTRAT [EMPLOYÉ THÈME] permet d'associer un employé *e* à un thème *t* si cet employé *e* est engagé dans au moins un contrat dont le thème de travail est *t*. En poussant plus loin l'analyse, on peut s'apercevoir qu'un employé ne peut travailler dans un contrat que s'il est compétent dans le thème du contrat. Les compétences d'un employé sont définies *indépendamment* des contrats auxquels il participe.
Il existe donc une nouvelle relation COMPÉTENCE (EMPLOYÉ THÈME) qui associe un employé et un thème s'il est compétent pour ce thème, et une nouvelle règle d'intégrité:
CONTRAT -: COMPÉTENCE.

La méthode sousjacente à cet exemple propose au concepteur l'analyse de chaque relation de base R: si les entités associées à une projection de R sur certains de ses constituants, par exemple A et B, peuvent être connues *indépendamment* des entités de R, alors il est important de les considérer comme entités d'une nouvelle relation S(AB) vérifiant R -: S.

Nous qualifions une telle nouvelle relation de *multifonctionnelle.*

Bien sûr cette étude se généralise aux relations obtenues par composition de relations de base.
Ces relations multifonctionnelles généralisent en fait la notion de dépendance fonctionnelle comme nous l'avons mentionné dans le § 2.4.2. précédent. Notamment elles peuvent se mettre en évidence rapidement dans le cas de l'existence de la df: A → B.

Par exemple à partir de la relation APPROVISIONNEMENT (PRODUIT NOM-FOURNISSEUR ADRESSE-du-FOURNISSEUR)
avec les df: PRODUIT → NOM-FOURNISSEUR
NOM-FOURNISSEUR → ADRESSE-du-FOURNISSEUR,
on peut facilement construire la relation: FOURNISSEUR (NOM-FOURNISSEUR ADRESSE-du-FOURNISSEUR).

Remarque:
Cet aspect d'indépendance ne s'exprime pas à l'aide de dépendances multivaluées comme nous l'avons déjà mentionné au § 2.5.6.

2.10.2. Contextes et espaces

Pour introduire cet aspect, nous partons d'un constat simple: les systèmes de bases de données relationnelles permettent aux utilisateurs d'interroger eux-mêmes la base de données à l'aide d'un langage relativement simple. Ce langage leur permet notamment de composer plusieurs relations de base et d'obtenir une nouvelle relation RC. Si du point de vue formel le prédicat de cette relation est parfaitement défini, du point de vue *opératoire*, les utilisateurs donnent un sens à l'association des constituants formant RC: le *piège* est bien en place dès que le sens opératoire ne correspond pas au prédicat de RC. Nous illustrons tout de suite ce point de vue à l'aide de l'exemple ci-dessous.

Exemple:
Voici un exemple d'un tel piège tendu aux utilisateurs par la structure de base de données suivante:

	en abrégé:
ACCUSATION (NOPROCÈS NOMACCUSÉ)	(NP NAC)
PLAIDOIERIE (NOPROCÈS NOMAVOCAT)	(NP NAV)
DÉFENSE (NOMAVOCAT NOMACCUSÉ)	(NAV NAC).

ACCUSATION associe un numéro de procès et un nom d'accusé si celui-ci a été accusé au cours de ce procès;
PLAIDOIERIE associe un numéro de procès et un nom d'avocat si celui-ci a plaidé au cours du procès;
DÉFENSE associe un nom d'avocat et un nom d'accusé si cet avocat a défendu cet accusé.

On veut trouver les noms de tous les avocats qui ont défendu bouc-émissaire au cours du procès de numéro 969.
Cette question est relative à la relation PROCÈS (NP NAC NAV) ayant pour prédicat: au cours du procès de numéro np, la personne de nom nac est accusée et l'avocat de nom nav plaide pour la défendre.

L'utilisateur maîtrisant les langages relationnels d'interrogation de bases de données peut formuler la question de l'une des manières suivantes:

(1) (ACCUSATION*PLAIDOIERIE) [NAC=bouc-émissaire,NP=969],
(2) (PLAIDOIERIE*DÉFENSE) [NAC=bouc-émissaire,NP=969],
(3) (DÉFENSE*ACCUSATION) [NAC=bouc-émissaire,NP=969],
(4) (DÉFENSE*ACCUSATION*PLAIDOIERIE)
[NAC=bouc-émissaire,NP=969].

Que penser de ces formulations ? Sont-elles équivalentes ? Sont-elles correctes ? Chacune d'elles sous-entend en fait une décomposition de la relation PROCÈS.

Ainsi la formulation (1) sous-entend que (ACCUSATION PLAIDOIERIE) est une décomposition de PROCÈS; or s'il y a plusieurs accusés dans un même procès, les avocats ne défendent généralement pas tous les accusés mais au contraire chaque avocat défend généralement un seul accusé. Donc (ACCUSATION PLAIDOIERIE) n'est pas une décomposition de PROCÈS et la formulation (1) est incorrecte.

Voici un exemple de faits réels qui va nous montrer qu'*aucune des formulations précédentes de la question n'est correcte.*

PROCÈS

NP	NAC	NAV
969	bouc-émissaire	assurancetousrisques
969	lucky-luke	dalton
001	bouc-émissaire	perdudavance
969	idéefixe	perdudavance

Les instances qui sont stockées dans la base de données sont donc:

ACCUSATION	NP	NAC
	969	bouc-émissaire
	969	lucky-luke
	001	bouc-émissaire
	969	idéefixe

PLAIDOIERIE	NP	NAV
	969	assurancetousrisques
	969	dalton
	001	perdudavance
	969	perdudavance

DÉFENSE	NAV	NAC
	assurancetousrisques	bouc-émissaire
	dalton	lucky-luke
	perdudavance	bouc-émissaire
	perdudavance	idéefixe

A partir des précédentes formulations, on trouve en particulier les réponses erronées suivantes:

dans (1)	969	bouc-émissaire	dalton
dans (2)	969	bouc-émissaire	perdudavance
dans (3)	969	bouc-émissaire	perdudavance
dans (4)	969	bouc-émissaire	perdudavance.

La relation PROCÈS n'est pas décomposable comme le montre l'exemple.

Si effectivement les utilisateurs de la base souhaitent connaître les avocats qui ont défendu tel accusé au cours de tel procès, le concepteur doit introduire la relation PROCÈS dans la modélisation.
fex.

Une *relation opératoire* est une relation dont l'ensemble des constituants est l'union de constituants de plusieurs relations de base et dont le prédicat a un sens connu non ambigü "naturel" pour les utilisateurs.

Pour déterminer si un prédicat a un sens connu, non ambigu et naturel pour les utilisateurs, il suffit de les interroger en leur fournissant l'ensemble des constituants de la relation et en leur demandant le sens qu'ils donnent à leur association; si les réponses convergent remarquablement, la relation sousjacente est considérée comme opératoire.

Nous pensons que le travail de modélisation doit fournir une liste des relations opératoires dont les utilisateurs ont besoin. La modélisation sera complète par rapport à cet ensemble si chaque relation opératoire peut s'exprimer par une expression relationnelle de relations de base.

Un *contexte* est une relation opératoire RO obtenue par composition de certaines relations de base dont l'ensemble forme une décomposition de RO.

Un *espace* ou une relation espace RE d'une base de données est un contexte qui est maximal dans le sens où aucun ensemble de relations de base ne peut être rajouté à celui qui forme RE pour obtenir un nouveau champ.

Un *espace partiel* ou un *contexte* d'une relation espace RE correspond à une décomposition partielle de RE.

Le piège que tend la structure de données précédente provient du fait que la relation PROCÈS qui est une relation opératoire n'est pas une relation espace. La modélisation n'est pas complète. Cette structure contient trois relations espaces: ACCUSATION, PLAIDOIERIE et DÉFENSE.

C'est ce type de piège que nous tentons de *démasquer* et nous pensons qu'il est de la responsabilité des concepteurs de bases de données de *déminer* une structure de données de tels pièges. Pour cela ils doivent mettre en évidence les relations espaces et les contextes d'une structure de données. A l'intérieur de celles-ci les utilisateurs peuvent formuler leurs questions sans courir autant de risques qu'auparavant.

L'établissement d'un espace RE ou d'un contexte C demande au concepteur de bien respecter la propriété $RE[R_k^+] = R_k$ ou $C[R_k^+] = R$. Il peut s'aider des propriétés de (df-dc2), extension (dc2) augmentation (dc3) substitution (dc6). Mais l'établissement d'un espace ne peut se résumer à une simple application de propriétés mathématiques: il demande au concepteur un travail de réflexion sur les informations pour déterminer si la relation est opératoire ou non.

2.10.3. Canevas d'analyse d'une modélisation

Ce canevas n'a pas la prétention d'être exhaustif.

a) Chercher toutes les df définies dans chaque relation de base et déterminer leurs clés.
Chercher toutes les relations multifonctionnelles de chaque relation de base, et trouver leurs clés.
Rajouter les relations multifonctionnelles aux relations de base.
b) Pour chaque cycle de relations, examiner s'il existe une interdépendance entre les associations. Déterminer notamment les dépendances de références, les dépendances relatives et les dépendances de composition-projection.
c) Mettre en évidence les contextes et les espaces.
d) Chercher les df définies sur plusieurs relations de base.
Chercher les relations multifonctionnelles qui ne sont pas définies à partir d'une seule relation de base et trouver leurs clés.
Rajouter ces nouvelles relations aux relations de base.

Tant qu'il y a de nouveaux cycles de relations, recommencer les points b) c) d).

Nous montrons dans l'exemple CONCOURS comment le concepteur peut procéder.

2.10.4. Exemple CONCOURS

Il s'agit de l'organisation d'un concours où toutes les épreuves sont des oraux. Ce concours porte sur plusieurs thèmes.
Chaque thème est associé à un barème qui sert de pondération.
Une épreuve d'un thème se déroule dans une salle lors d'une demi-journée et plusieurs candidats sont examinés au cours d'une même épreuve.
Chaque candidat donne un exposé pour passer l'épreuve d'un thème devant un certain nombre de jurés tous compétents dans ce thème.
Chaque juré donne une note pour chaque exposé qu'il écoute.
La note finale du candidat pour un thème est la moyenne des notes des jurés.
NOTE et NOTE-FINALE ont le même domaine qui est de type mot ordonné. Les domaines des autres constituants sont de type mot.
Voici la modélisation déjà obtenue:

LISTE-THÈMES(THÈME BARÈME).
A chaque thème on associe son barème.

PROGRAMME(THÈME DEMI-JOURNÉE SALLE)
Le programme p = (th, dj, s) désigne une épreuve du thème th se déroulant lors de la demi-journée dj dans la salle s.

SUJETS(TITRE-EXPOSÉ THÈME CANDIDAT DEMI-JOURNÉE NOTE-FINALE).
L'entité de SUJETS s = (ti, th, c, dj, nf) désigne l'exposé de titre ti, que le candidat c a donné pour passer l'épreuve du thème th, lors de la demi-journée dj; nf en donne la note finale.

RÉSULTATS(JURÉ TITRE-EXPOSÉ NOTE).
L'entité de RÉSULTATS r = (j, ti, n) désigne la note n donnée par le juré j pour l'exposé de titre ti.

Voici le réseau de relations que nous obtenons auquel nous avons rajouté la relation NOTE (car NOTE et NOTE-FINALE ont le même domaine) et qui contient un cycle relatif aux relations PROGRAMME SUJETS et un autre relatif aux relations NOTE RÉSULTATS SUJETS.

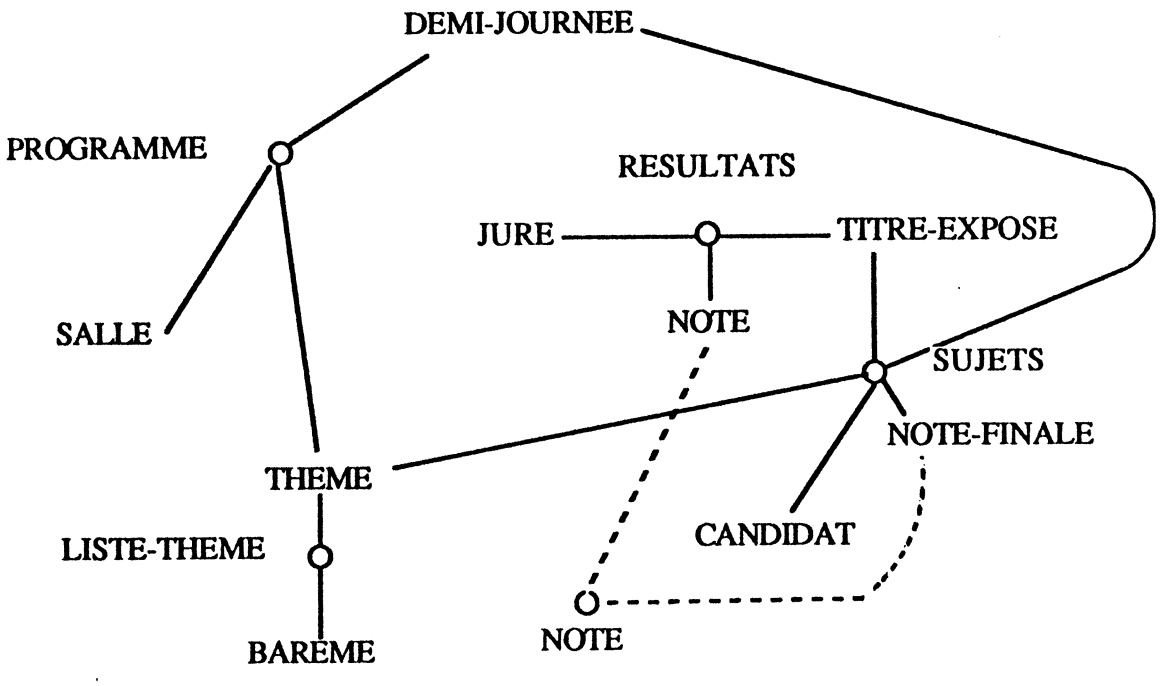

Figure 2.3. CONCOURS (1)

2.10.4.1. Poursuite du travail de modélisation

a) Recherche des df et des clés des relations

Le concepteur recherche toutes les dépendances fonctionnelles d'une relation: dans ce but il forme tous les schémas de df qu'il est possible de générer à l'aide des constituants de la relation; pour chacun d'eux il doit décider si oui ou non il correspond à une df définie sur la relation en analysant le champ d'application.
Ce travail qui peut être fastidieux est allégé par le fait qu'il s'agit simplement de trouver une base de l'ensemble des df: ainsi si AB → C est une df de la relation, point n'est besoin de rechercher si ABD → C en est une également. De même si C → D et D → E sont deux df définies sur la relation, on sait alors que C → E l'est également: le concepteur n'a aucun effort pour la trouver.
En considérant une relation R nous écrivons C → D pour signifier que la df C → D est définie sur R, et nous écrivons C non → D pour signifier que la df C → D n'est pas définie sur R.

Cette partie peut facilement faire l'objet d'une aide informatisée. Nous l'illustrons à l'aide de questions.
Le concepteur en déduit les clés des relations.

LISTE-THÈMES(THÈME BARÈME).
A un thème, peut-on associer plusieurs barèmes ?
Réponse: non.
A un barème peut-on associer plusieurs thèmes ?
Réponse: oui.
LISTE-THÈMES a une seule clé: THÈME

PROGRAMME(THÈME DEMI-JOURNÉE SALLE).
1.Les épreuves d'un thème se déroulent-t-elles toujours dans une même salle ?
Réponse: non.
2.Les épreuves d'un thème se déroulent-t-elles toujours pendant une même demi-journée ?
Réponse:non.
3.Les épreuves d'un thème se déroulent-t-elles toujours dans une même salle pendant une demi-journée ?
Réponse: oui.
4.Une salle est-elle le lieu d'épreuves d'un thème lors d'au plus une demi-journée ?
Réponse: non.
5.Une demi-journée est-elle consacrée aux épreuves d'un seul thème ?
Réponse: non.
6.Pendant une demi-journée, les épreuves se déroulent-elles toutes dans une même salle ?
Réponse: non.
7.Les épreuves qui ont lieu dans une même salle pendant une même demi-journée, sont-elles d'un même thème ?
Réponse: oui.
8.Une salle est-elle le lieu d'épreuves d'un seul thème ?
Réponse: non

Ces réponses permettent de déterminer l'ensemble de df suivant:

1. THÈME	non	→	SALLE,
2. THÈME	non	→	DEMI-JOURNÉE,
3. THÈME DEMI-JOURNÉE		→	SALLE,
4. THÈME SALLE	non	→	DEMI-JOURNÉE,
5. DEMI-JOURNÉE	non	→	THÈME,
6. DEMI-JOURNÉE	non	→	SALLE,
7. DEMI-JOURNÉE SALLE		→	THÈME,
8. SALLE	non	→	THÈME.

Les clés de PROGRAMME sont donc: THÈME DEMI-JOURNÉE,
et SALLE DEMI-JOURNÉE.

SUJETS(TITRE-EXPOSÉ THÈME CANDIDAT DEMI-JOURNÉE NOTE-FINALE).

1. Un candidat peut-il faire plusieurs exposés pour passer l'épreuve d'un thème ?
   Réponse: non.
2. Existe-t-il plusieurs candidats par thème ?
   Réponse: oui.
3. Un candidat passe-t-il des épreuves de plusieurs thèmes ?
   Réponse: oui.
4. Un candidat peut-il donner le même titre d'exposé pour des épreuves de thèmes différents ?
   Réponse: non.
5. Un titre d'exposé peut-il être donné par plusieurs candidats dans le cadre d'une épreuve ?
   Réponse: non.
6. Un titre d'exposé peut-il être donné dans le cadre de thèmes distincts ?
   Réponse: non
7. Un candidat reçoit-il plusieurs notes finales par thème ?
   Réponse: non.
8. Pour un thème, y a-t-il plusieurs notes finales ?
   Réponse: oui.
9. Chaque candidat reçoit-il la même note finale pour tous les thèmes dont il passe l'épreuve ?
   Réponse: non.
10. Au cours d'une même demi-journée, y a-t-il plusieurs candidats qui peuvent passer l'épreuve d'un thème ?
   Réponse: oui.
11. Un exposé peut-il se dérouler sur plusieurs demi-journées ?
   Réponse: non.
12. Un candidat peut-il passer plusieurs épreuves au cours d'une même demi-journée ?
   Réponse: non.

Ces réponses permettent d'établir l'ensemble suivant de df:

1. CANDIDAT THÈME		→	TITRE-EXPOSÉ,
2. THÈME	non	→	CANDIDAT,
3. CANDIDAT	non	→	THÈME,
4. CANDIDAT TITRE-EXPOSÉ		→	THÈME,
5. TITRE-EXPOSÉ THÈME		→	CANDIDAT,
6. TITRE-EXPOSÉ		→	THÈME,
7. CANDIDAT, THÈME		→	NOTE-FINALE,

8. THÈME	non	→	NOTE-FINALE,
9. CANDIDAT	non	→	NOTE-FINALE,
10. THÈME DEMI-JOURNÉE	non	→	CANDIDAT,
11. TITRE-EXPOSÉ		→	DEMI-JOURNÉE,
12. CANDIDAT DEMI-JOURNÉE		→	THÈME.

Les clés de la relation SUJETS sont: TITRE-EXPOSÉ,
THÈME CANDIDAT,
CANDIDAT DEMI-JOURNÉE.

RÉSULTATS(JURÉ TITRE-EXPOSÉ NOTE).

Un exposé peut-il recevoir plusieurs notes ?
Réponse: oui.
Un juré peut-il mettre plusieurs notes ?
Réponse: oui.
Un juré peut-il mettre plusieurs notes au même exposé ?
Réponse: non.
Un juré peut-il mettre une seule fois une même note ?
Réponse: non.

La clé de RÉSULTATS est donc: JURÉ TITRE-EXPOSÉ.

Pour les relations admettant plusieurs clés, le concepteur indique au moins une clé obligatoire. Si THÈME DEMI-JOURNÉE est une clé obligatoire de PROGRAMME, cela signifie que les valeurs prises par toute entité de PROGRAMME pour THÈME et DEMI-JOURNÉE doivent être connues.

b) Recherche des relations multifonctionnelles (rmf)

Ainsi si dans PROGRAMME il existe la rmf (THÈME SALLE), alors elle signifie que les épreuves d'un thème se déroulent dans une liste de salles prédéfinie et qu'une salle accueille les épreuves d'une liste de thèmes prédéfinie.
Nous supposons que ce n'est pas le cas pour l'organisation de ce concours. Sinon il aurait fallu créer cette nouvelle relation (THÈME SALLE).

Si à partir SUJETS il existe la rmf (CANDIDAT DEMI-JOURNÉE) alors elle signifie qu'un candidat passe ses épreuves dans les demi-journées appartenant à une liste établie indépendamment d'autres informations: nous supposons que tel n'est pas le cas. De même nous supposons que la rmf (THÈME DEMI-JOURNÉE) n'existe pas.

S'il existe la rmf (THÈME CANDIDAT) définie dans SUJETS, elle signifie qu'indépendamment des titres d'exposés, des notes finales, des demi-journées, on puisse établir la liste des candidats qui doivent passer une épreuve dans ce thème: nous supposons que tel est le cas.
Il faut une relation formée sur CANDIDAT THÈME; celle-ci existe déjà car CANDIDAT THÈME est une clé de la relation SUJETS.
Une manière de prendre en compte ce fait est de rendre cette clé de SUJETS obligatoire. Ainsi la création d'une entité de SUJETS revient justement à créer une association entre un thème et un candidat, sans éventuellement connaître le titre de l'exposé, la note et la demi-journée qui pourront lui être adjoints ultérieurement.

c) Analyse de cycles

Notre proposition ne fournit aucun raisonnement permettant de conduire le concepteur à la découverte de la ri sousjacente à un cycle quand elle existe. Elle le place simplement devant une situation remarquable, source potentielle de ri.

L'analyse du cycle de relations PROGRAMME SUJETS fait apparaître la dépendance existentielle qui est supposée validée (§ 2.9.1.):
PROGRAMME[THÈME DEMI-JOURNÉE] +: SUJETS[THÈME DEMI-JOURNÉE].

L'analyse du cycle NOTE RÉSULTATS TITRE-EXPOSÉ SUJETS NOTE-FINALE conduit à la ri suivante: la note finale d'un exposé est la moyenne des notes données pour cet exposé par les jurés. Elle s'exprime de manière algorithmique ainsi:
(RI1) $\forall$ s $\in$ SUJETS[NOTE-FINALE $\neq$ Nil]
s.NOTE-FINALE = moyenne(RÉSULTATS[NOTE] [TITRE-EXPOSÉ = s.TITRE-EXPOSÉ]).

d) Mise en évidence des espaces

Le concepteur met en évidence les différents espaces en dégageant les dépendances de composition. Il s'y prend par tâtonnement.
Nous tentons de trouver un espace EXAMEN dont une décomposition est formée de l'ensemble des quatre relations de base. Le prédicat de cet espace se déduit facilement des prédicats de ces relations: un juré (j) attribue une note (n) à un exposé (te) d'un candidat (c) au cours d'une 1/2 journée (j) dans une salle (s) pour l'épreuve d'un thème (th) ayant un barème (b), le candidat ayant la note finale (nf) pour cette épreuve.

La projection de EXAMEN sur l'ensemble des constituants de chacune des quatre relations de base est bien identique à cette relation de base.
Dans cette relation sont définies les dépendances fonctionnelles définies dans chacune des quatre relations de base. On obtient par applications successives de la propriété (df-dc2):
EXAMEN = LISTE-THÈMES * EXAMEN [EXAMEN+ - BARÈME]
=LISTE-THÈMES * PROGRAMME * EXAMEN [EXAMEN+ - (BARÈME SALLE)]
= LISTE-THÈMES * PROGRAMME * SUJETS * RÉSULTATS.
EXAMEN est donc bien un espace admettant l'ensemble des relations de base comme décomposition.

Par application de (dc8) et comme LISTE-THÈMES est une extrémité, on obtient les espaces partiels suivants:
C_1 = PROGRAMME * SUJETS * RÉSULTATS,
C_2 = SUJETS * RÉSULTATS,
C_3 = LISTE-THÈMES * PROGRAMME * SUJETS,
C_4 = LISTE-THÈMES * PROGRAMME.

e) DF non intrinsèques des espaces

Le concepteur étudie les df de l'espace EXAMEN qui n'ont pas encore été mises en évidence et tente de découvrir d'éventuelles relations multifonctionnelles définies à partir d'EXAMEN.
Le concepteur doit les rechercher dans des relations obtenues par projection de EXAMEN sur des constituants appartenant à plusieurs relations de base.

EXAMEN [CANDIDAT SALLE].
Si une rmf existe, alors chaque candidat est associé à une liste de salles indépendamment des autres informations.
Nous supposons que ceci est incorrect.

EXAMEN [JURÉ SALLE].
Le même raisonnement nous conduit à ne pas retenir une rmf (JURÉ SALLE).

EXAMEN [JURÉ THÈME].
La rmf JURY(JURÉ THÈME) existe dans EXAMEN: elle associe un juré à ses thèmes de compétence. Elle signifie qu'un juré ne peut noter des épreuves que pour ses thèmes de compétence.
Cette ri peut s'écrire alors suivant la dépendance de référence suivante:
RÉSULTATS SUJETS -: JURY.

EXAMEN [JURÉ CANDIDAT].
La rmf JC(JURÉ CANDIDAT) associe un juré et un candidat si ce candidat passe une épreuve dans un des thèmes de compétence du juré.
Dans ce cas il existe la dépendance de référence:
RÉSULTATS SUJETS -: JC.
On peut remarquer qu'il existe la dépendance de composition-projection:
SUJETS[THÈME CANDIDAT] JURY =: JC.
Ainsi JC apparaît comme redondante.

EXAMEN [JURÉ DEMI-JOURNÉE].
La rmf DISPONIBILITÉ (JURÉ DEMI-JOURNÉE) se définit à partir de EXAMEN: elle permet d'associer un juré (j) à une demi-journée (dj) où il est disponible. Elle signifie qu'un juré ne peut assister aux épreuves d'examens que lors des demi-journées où il est disponible.
Cette ri s'écrit alors comme une dépendance de référence:
RÉSULTATS SUJETS -: DISPONIBILITÉ.

Voici le nouveau réseau de relations que nous obtenons:

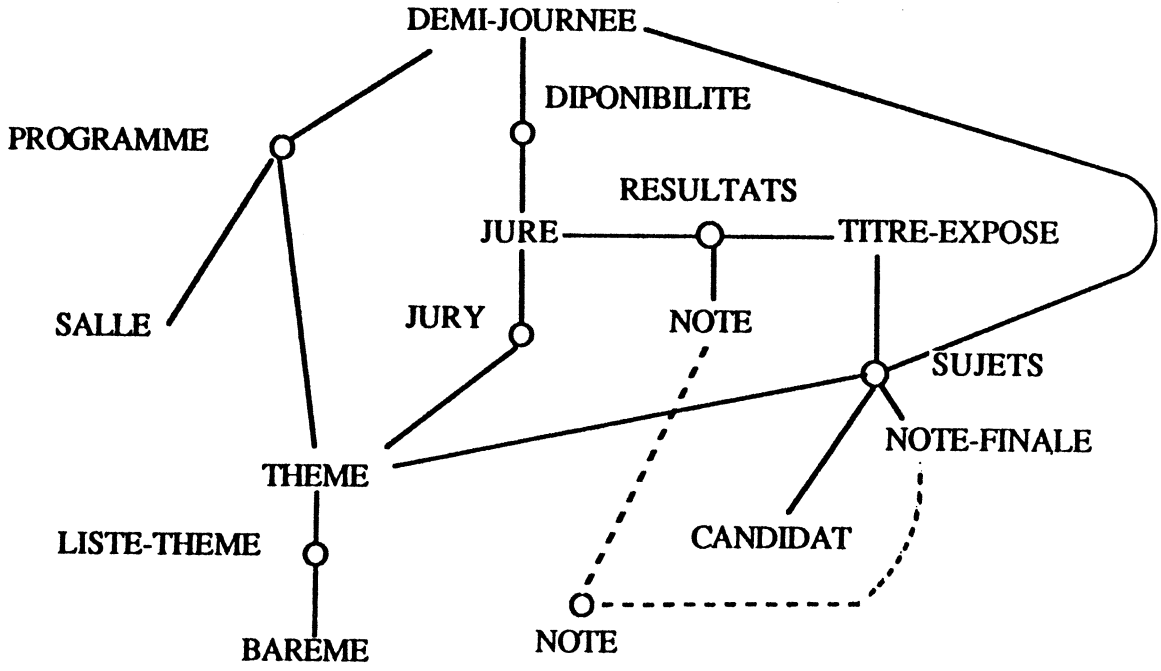

Figure 2.4. CONCOURS (2)

Neuf nouveaux cycles de relations sont introduits:
(c1) JURY-SUJETS-RÉSULTATS;

(c2) DISPONIBILITÉ-SUJETS-RÉSULTATS;
(c3) DISPONIBILITÉ-PROGRAMME-JURY;
(c4) DISPONIBILITÉ-SUJETS-JURY;
(c5) DISPONIBILITÉ-SUJETS-RÉSULTATS-PROGRAMME;
(c6) RÉSULTATS-PROGRAMME-SUJETS;
(c7) JURY-DISPONIBILITÉ-SUJETS-PROGRAMME;
(c8) JURY-DISPONIBILITÉ-SUJETS-PROGRAMME-RÉSULTATS.

f) Recherche de nouveaux espaces

Comme de nouvelles relations et de nouveaux cycles viennent d'être introduits au cours de la phase précédente, le concepteur va retenir les nouveaux espaces suivants:
EXAMEN = PROGRAMME * LISTE-THÈMES * SUJETS * RÉSULTATS;
POSSIBILITÉS = DISPONIBILITÉ * JURY;
POSSIBLE-HORAIRE = DISPONIBILITÉ * RÉSULTATS.

g) Analyse des nouveaux cycles de relations

(c1) Par construction EXAMEN -: JURY.
Comme SUJETS RÉSULTATS est un contexte, on obtient:
SUJETS RÉSULTATS -: JURY (propriété dr6).
Le théorème (di) nous indique que les dépendances relatives éventuelles liées à ce cycle se déduisent de cette dépendance de référence et qu'il ne peut y avoir d'autres dépendances de référence.

(c2) Par construction EXAMEN -: DISPONIBILITÉ.
On obtient de la même façon:
SUJETS RÉSULTATS -: DISPONIBILITÉ.

(c3) JURY * PROGRAMME fournit les jurés potentiels et demi-journées potentielles pour faire passer les épreuves d'un thème.
JURY * DISPONIBILITÉ fournit les demi-journées possibles d'un juré et ses thèmes des compétences.
DISPONIBILITÉ * PROGRAMME fournit les jurés disponibles et les thèmes traités pour une demi-journée.
La dernière relation ne peut être considérée comme un espace partiel.

JURY DISPONIBILITÉ +: PROGRAMME [DEMI-JOURNÉE THÈME] est une règle d'intégrité; c'est la seule dépendance d'inclusion liée à ce cycle.

(c4) De PROGRAMME [DEMI-JOURNÉE THÈME]
+: SUJETS [DEMI-JOURNÉE THÈME]
et de DISPONIBILITÉ JURY
+: PROGRAMME [DEMI-JOURNÉE THÈME],
on peut déduire par transitivité:
DISPONIBILITÉ JURY +: SUJETS [DEMI-JOURNÉE THÈME].

(c5) De SUJETS RÉSULTATS -: DISPONIBILITÉ, on obtient par augmentation (propriétés de réflexivité (dr1) et de transitivité (dr3)):
PROGRAMME SUJETS RÉSULTATS -: DISPONIBILITÉ.

(c6) De la même façon on obtient la dr suivante:
PROGRAMME SUJETS RÉSULTATS -: JURY.

Les cycles (c7) et (c8) n'apportent aucune nouvelle dépendance d'inclusion. Les seuls contextes possibles sont en effet formés de: PROGRAMME SUJETS RÉSULTATS et de DISPONIBILITÉ JURY. On ne peut à partir d'eux construire de nouvelles dépendances de référence ou relatives.

2.10.4.2. Portées des différentes règles d'intégrité

(RMF1) Relation multifonctionnelle (CANDIDAT THÈME).
La ri sousjacente est imprimée dans la structure car THÈME CANDIDAT est une clé obligatoire de SUJETS.

(RMF2) RÉSULTATS SUJETS -: JURY:
portée: c(RÉSULTATS), c(SUJETS), s(JURY).

(RMF3) RÉSULTATS SUJETS -: DISPONIBILITÉ:
portée: c(RÉSULTATS), c(SUJETS), s(DISPONIBILITÉ).

(DIV1) JURY DISPONIBILITÉ +: PROGRAMME:
portée c(PROGRAMME), s(JURY), s(DISPONIBILITÉ).

(RI1) Cette ri concerne le calcul de la note finale.
La portée: maj NOTE (RÉSULTATS), maj NOTE-FINALE (SUJETS): interdite, s(RÉSULTATS), c(RÉSULTATS), suppose que:
- lors de la création d'une entité de SUJETS, l'entité prend la valeur indéterminée pour NOTE-FINALE.
- de plus, il existe une *seule primitive*: calcul d'une note finale qui met à jour la note-finale d'un sujet.

Ces résultats peuvent être récapitulés dans le tableau suivant: les colonnes correspondent aux règles d'intégrité, les lignes aux relations. On indique à l'intersection d'une ligne et d'une colonne, les noms des primitives de la relation de la ligne, dont l'éxécution demande la validation de la ri de la colonne.

Tableau des portées des ri:

RELATIONS \ RI	RMF2	RMF3	DIV1	RI1
LISTE-THÈMES				
PROGRAMMES			c	
SUJETS	c	c		majNOTE-FINALE
JURY	s		s	
RÉSULTATS	c	c		c s majNOTE
DISPONIBILITÉ		s	s	

Note:
nous invitons le lecteur à poursuivre l'exemple en considérant maintenant la df: JURÉ DEMI-JOURNÉE → THÈME et le fait qu'un exposé se passe devant tous les jurés compétents pour le thème de l'exposé.

2.10.5. Conclusions

Nous espérons avoir convaincu le lecteur que les règles d'intégrité ne sont pas seulement intéressantes parce qu'elles permettent d'assurer une certaine cohérence de la base de données: dans ce simple rôle, elles jouent le rôle d'assurance que le concepteur et (ou) l'utilisateur contracte pour se prémunir de quelques fausses manoeuvres. Mais il serait bien dommage de limiter leur rôle à cette seule fonction.

Exprimant des invariants du champ d'application, elles sont très utiles au concepteur lors du difficile processus de modélisation du champ d'application: nous avons présenté dans ce chapitre des situations remarquables qu'un concepteur attentif peut déceler dans la modélisation qu'il construit. Il s'agit alors pour lui de dénicher la ou les règles d'intégrité qui peuvent être sousjacentes à de telles situations. Comme analyste d'un champ d'application, il n'est plus seulement dans une position "passive" d'écoute des informations que les utilisateurs veulent bien lui confier, mais également dans une position "active" où ces situations remarquables lui permettent de poser des questions pertinentes pour la compréhension du champ d'application et la réalisation de la base de données. Non seule-

ment la recherche des règles d'intégrité permet au concepteur-analyste de prendre cette position "active", mais elle lui fournit un guide pour ne pas succomber à la tentation de l'activisme.

C'est ainsi que l'étude de ces situations remarquables peut conduire le concepteur à modifier sa modélisation: dans l'exemple CONCOURS nous avons été amenés à construire deux nouvelles relations.

Si la recherche des règles d'intégrité est très importante dans le processus de modélisation, elle l'est également pour l'architecture générale de la programmation des applications: nous avons mis en évidence *l'architecture normale* d'une base de données qui concerne la définition et la réalisation de primitives de modifications de la base de données validant toutes les règles d'intégrité. Les programmes d'application s'écrivent alors à l'aide de ces primitives sans souci de validation de règles d'intégrité et le plus souvent résistent aux modifications des règles d'intégrité pouvant intervenir au cours du temps.

Dans la troisième partie, nous montrons comment les règles d'intégrité et notamment les dépendances fonctionnelles ont un rôle important dans la détermination d'une structure de base de données de qualité; notamment il s'agit alors d'éviter la redondance d'informations.

2.11. Annexe

2.11.1. Algorithme de clés simplifié

Il existe plusieurs algorithmes de recherche des clés d'une relation comme ceux de (FADOUS-FORSYTH75), (LUCCHESI-ORSBORN76), (KUNDU85).
Nous donnons ici le principe de la recherche de clés d'une relation.

Soit une relation R muni d'un ensemble de dépendances fonctionnelles F.

L'*écoulement* d'un ensemble de constituants X dans F, que nous notons écoulement (X), est l'ensemble des constituants A de R tels qu'il existe la df $X \rightarrow A$ dans la fermeture de F**.

La définition d'une clé d'une relation de R peut alors s'exprimer ainsi: un ensemble K de constituants de R est une clé de R si seulement si: écoulement(K) = R^+, et $\forall\ K' \subseteq K$ écoulement(K') $\neq R^+$.

L'algorithme de recherche de clés d'une relation examine l'ensemble des parties de R^+ et calcule l'écoulement de chacune d'elles:
s'il n'est pas égal à R^+, elle ne peut former une clé;
par contre s'il est égal à R^+, elle est une clé seulement si elle ne contient aucune partie dont l'écoulement vaut R^+.

Voici la description de cet algorithme non optimisé; il utilise les fonctions Parties(X), écoulement(X), min(XYZ) qui sont décrites après. Nous avons écrit ces algorithmes en utilisant les opérations ensemblistes et l'opération d'affectation d'un élément à un ensemble ($:\in$) (ABRIAL74).

Clés(R,F) contient à la fin l'ensemble des clés de la relation R munie de l'ensemble de df F. C désigne une variable pouvant contenir un ensemble d'ensembles de constituants de R.

```
C := ∅
pour chaque K ∈ Parties(R+) faire:
        si écoulement(K) = R+ alors K :∈ C finsi
refaire
Clés := min(C).
```

Parties(X) fournit l'ensemble de toutes les parties de X. P est une variable pouvant contenir un ensemble d'ensembles de constituants de R.

```
P:= ∅
{∅ } :∈ P    (un élément de P est l'ensemble vide)
pour chaque A ∈ X faire
        pour chaque Y ∈ P faire
                Y ∪ {A} :∈ P
        refaire
refaire
Parties(X) := P.
```

écoulement(X) calcule l'écoulement de X dans F.
é désigne une variable pouvant contenir un ensemble de constituants de R.

```
é := X
nouveau := vrai   (nouveau est un booléen qui garde la valeur faux
dès qu'aucun nouveau constituant ne peut être ajouté à l'écoulement de X)
tant que nouveau faire
        nouveau:=faux
        pour chaque f ∈ F faire
                si g(f) ⊆ é alors d(f) :∈ é; nouveau:=vrai finsi
        refaire
refaire
écoulement(X) := é.
```

min(XYZ) fournit les ensembles non vides minimaux par rapport à l'inclusion, contenus dans XYZ. Les ensembles de XYZ sont rangés suivant une séquence.
post(X,XYZ) désigne la séquence des ensembles qui suivent X dans XYZ.
mi est une variable pouvant contenir un ensemble d'ensembles de constituants de R.

```
mi := ∅
pour chaque X ∈ XYZ dans l'ordre de la séquence faire
        ok:=vrai     (ok est un booléen qui gardera la valeur vrai si
                      X est minimal)
        pour chaque Y ∈ post(X,XYZ) et tant que ok=vrai faire
                si X ⊃ Y alors ok:=faux finsi
        refaire
        si ok=vrai alors X :∈ mi finsi
refaire
min(XYZ) := mi.
```

2.11.2. Optimisation de l'algorithme Clés

2.11.2.1. Sources et puits

Nous ne voulons pas présenter ici des optimisations sophistiquées de l'algorithme précédent; simplement nous utilisons une propriété liée aux df pour restreindre l'ensemble des Parties de R^+ à considérer.
Un constituant est une *source* dans F s'il n'appartient à aucune partie droite de df de F;
il est un *puits* dans F s'il n'appartient à aucune partie gauche de df de F et s'il n'est pas source.

On peut démontrer que toute clé de R contient toutes les sources de R et qu'elle ne contient aucun puits.

L'ensemble des parties de R^+ à considérer se réduit alors aux seules parties de R^+ contenant toutes les sources et ne contenant aucun puits.

2.11.2.2. Extension de la définition d'un puits

Il est possible d'étendre la notion de puits de la manière récursive suivante:
un constituant de R est un *puits* dans F s'il n'est pas une source et si:
il n'appartient à aucune partie gauche de df de F,
ou toutes les df dont il est un des constituants de la partie gauche, admettent des puits comme partie droite.

L'ensemble des puits devient plus grand et ainsi l'ensemble des parties de R^+ à considérer moins grand.

L'ensemble des sources et celui des puits dépend de la couverture de F: ils sont maximaux si l'on utilise la base complète de F.

2.11.3. Exemple

Nous prenons l'exemple CONCOURS du § 2.10.4.

2.11.3.1. Relation PROGRAMME

PROGRAMME(THÈME DEMI-JOURNÉE SALLE)
Le programme p = (th, dj, s) désigne une épreuve du thème th se déroulant lors de la demi-journée dj dans la salle s.

Voici la liste des df retenues:
THÈME DEMI-JOURNÉE → SALLE,
DEMI-JOURNÉE SALLE → THÈME.

DEMI-JOURNÉE est une source; il n'y a pas de puits.
L'écoulement de DEMI-JOURNÉE ne contenant que DEMI-JOURNÉE, les clés de PROGRAMME sont donc: THÈME DEMI-JOURNÉE,
et SALLE DEMI-JOURNÉE.

2.11.3.2. Relation SUJETS

SUJETS(TITRE-EXPOSÉ THÈME CANDIDAT DEMI-JOURNÉE NOTE-FINALE).
L'entité de SUJETS s = (ti, th, c, dj, nf) désigne l'exposé de titre ti, que le candidat c a donné pour passer l'épreuve du thème th, lors de la demi-journée dj; nf en donne la note finale.

Voici la liste des df retenues:

1. CANDIDAT THÈME → TITRE-EXPOSÉ,
2. CANDIDAT TITRE-EXPOSÉ → THÈME,
3. TITRE-EXPOSÉ THÈME → CANDIDAT,
4. TITRE-EXPOSÉ → THÈME,
5. CANDIDAT THÈME → NOTE-FINALE,
6. TITRE-EXPOSÉ → DEMI-JOURNÉE,
7. CANDIDAT DEMI-JOURNÉE → THÈME.

Il n'y a pas de sources; NOTE-FINALE est un puits et ne peut donc appartenir à aucune clé.
Il s'agit de considérer les parties de l'ensemble de constituants suivant: {TITRE-EXPOSÉ THÈME CANDIDAT DEMI-JOURNÉE}.

L'écoulement de TITRE-EXPOSÉ contient d'abord THÈME, DEMI-JOURNÉE (4,6); puis il contient CANDIDAT (3); puis il contient NOTE-FINALE (5). Il contient donc tous les constituants de SUJETS. TITRE-EXPOSÉ étant formé d'un seul constituant, il est donc une clé de SUJETS.

Les écoulements de THÈME, de CANDIDAT, et de DEMI-JOURNÉE ne contiennent pas d'autres constituants: ils ne forment pas de clés.

L'écoulement de CANDIDAT THÈME contient TITRE-EXPOSÉ (1): il contient donc tous les constituants de SUJETS. C'est une clé car ni CANDIDAT ni THÈME seul ne forme une clé.

Celui de CANDIDAT DEMI-JOURNÉE contient THÈME CANDIDAT (7): il contient donc tous les constituants de SUJETS. C'est une clé car ni CANDIDAT ni DEMI-JOURNÉE seul ne forme une clé.

Celui de THÈME DEMI-JOURNÉE ne contient aucun autre constituant. Ce n'est pas une clé.
Si on lui ajoute un autre constituant, alors on obtient un ensemble de constituants qui contient une clé. On ne peut donc trouver d'autres clés de SUJETS.

Voici donc clés de la relation SUJETS: TITRE-EXPOSÉ,
CANDIDAT THÈME,
CANDIDAT DEMI-JOURNÉE.

DEUXIÈME PARTIE

TRANSFORMATIONS

3

INTRODUCTION ET GRAPHE DE CHEMINS D'ACCÈS

L'objectif d'un processus de conception d'une base de données est de fournir à un système de gestion de bases de données, les informations qui lui sont nécessaires pour gérer cette base de données. Très souvent le problème de conception peut être posé par rapport à un système de gestion de bases de données bien particulier. Bien entendu, en tant qu'universitaire nous allons présenter des résultats qui seront indépendants d'un SGBD voire même de l'utilisation d'un SGBD. Notre attitude provient d'un paradigme, pour nous fondamental: un système de gestion de base de données n'a pas une finalité en lui-même; il se trouve au contraire comme une articulation entre le monde de l'utilisation des données qu'il gère et le monde informatique dans lequel ces données sont stockées. Aussi prendre comme référence un SGBD particulier, c'est s'interdire la critique de l'articulation qu'il réalise et par conséquent enfermer l'étude de la conception de bases de données dans la logique d'un SGBD. Nous souhaitons au contraire mettre en évidence des concepts ou des mécanismes qui nous semblent fondamentaux à cette articulation, et à partir d'eux critiquer l'articulation actuellement proposée par les SGBD pour tenter de découvrir les problèmes essentiels liés à la conception d'une structure logique de données.

Dans cette seconde partie, nous allons présenter une heuristique qui permet de transformer une modélisation relationnelle dans une structure de bases de données. Comme notre approche cherche à être la plus indépendante possible des SGBD disponibles, nous allons définir un modèle abstrait de SGBD pour la conception d'une structure de données:

ce modèle que nous appelons *graphe de chemins d'accès* GCA, prend en compte tous les aspects de la plupart des SGBD concernant la définition d'une structure de données.

Nous présentons dans cette 2nde Partie les différentes difficultés rencontrées au cours du processus de conception de bases de données, une fois effectué le choix de la modélisation. Nous distinguons trois niveaux:

a) transformation d'une modélisation relationnelle de données dans un graphe de chemins d'accès particulier que nous appelons graphe de chemins d'accès brut et que nous notons GCA_b pour les différencier des autres GCA;
b) choix d'un graphe de chemins d'accès parmi la classe des GCA centrée autour du GCA_b obtenu précédemment;
c) transformation du GCA choisi dans une structure informatique de données avec la spécification de toutes les méthodes informatiques d'accès et de stockage choisies.

Avant de rentrer dans le vif du sujet, il nous semble nécessaire de répondre aux deux questions suivantes dans l'Introduction de cette seconde Partie:

- Pourquoi la modélisation de données retenue au cours de la phase étudiée dans la 1ère partie ne fournit-elle pas le résultat final du processus de conception ?

- Quelles sont les conditions à respecter pour que la structure de données finale soit fidèle à la modélisation initiale de données? Comment peut-on définir cette notion de fidélité ?

Ensuite nous introduisons les GCA pour terminer l'introduction de cette seconde partie.

3.1. Choix d'un modèle informatique de données

Nous entendons par modèle informatique de données un modèle de données qui permet de représenter le stockage des données dans la base de données. Nous allons illustrer cette notion très simple à l'aide d'un exemple en utilisant le modèle relationnel de données comme modèle informatique de données.

3.1.1. Exemple

COURS(*NOMCOURS* NBHCOURS)
Cette relation associe un nom d'un cours à son nombre d'heures de cours par semaine; la clé de COURS est NOMCOURS.

CONFÉRENCIER(*NOMCONF* ORGANISME)
Cette relation associe un nom d'un conférencier à l'organisme dans lequel il travaille; sa clé est NOMCONF.

CONFÉRENCE(*NOMCONF NOMCOURS* NBSEM)
Un conférencier (NOMCONF) est associé à un cours (NOMCOURS) car il donne une conférence dans ce cours sur tant de semaines (NBSEM). La clé de CONFÉRENCE est formée de NOMCONF NOMCOURS.

PLANNING(*SALLE NOH NOJOUR* NOMCONF NOMCOURS)
Une salle (SALLE) est réservée pour une conférence (NOMCONF NOMCOURS) pour telle heure (NOH) de tel jour (NOJOUR) de la semaine.
Une clé de PLANNING est formée de SALLE NOH NOJ.

3.1.2. Le modèle relationnel de données utilisé comme modèle informatique de données

Si le modèle relationnel de données sert de modèle informatique de données, alors la base de données précédente est implantée très simplement de la manière suivante: à chaque relation correspond une zone de stockage que nous appelons, pourquoi pas ? fichier; à chaque entité d'une relation correspond un enregistrement du fichier.
Voici un exemple d'entités de ces relations:

COURS	(NOMCOURS	NBHCOURS)
	Base de données	3
	Génie logiciel	3
	Architecture	3
	Graphique et communication	3
	Système expert	3
	Système d'information	3
	Compilation	3

Transformation de structures

CONFÉRENCIER (NOMCONF ORGANISME)

	Delobel	PARIS-ORSAY
	Pellegrini	GENÈVE
	Harms	GENÈVE
	Adiba	GRENOBLE
	Benzaken	GRENOBLE
	Krakowiak	GRENOBLE
	Levrat	GENÈVE
	Courbon	GENÈVE
	Scholl	GRENOBLE
	Mossière	GRENOBLE
	Bodart	NAMUR
	Hainaut	NAMUR
	Thalmann	GENÈVE
	Sifakis	GRENOBLE
	Chiaramella	GRENOBLE
	Kouloumdjan	LYON

CONFÉRENCE (NOMCONF NOMCOURS NBSEM)

Adiba	Base de données	14
Delobel	Base de données	14
Courbon	Système d'Information	14
Harms	Architecture	14
Krakowiak	Architecture	14
Hainaut	Base de données	14
Pellegrini	Système expert	28
Levrat	Compilation	14
Thalmann	Système d'Information	14
Bodart	Système d'Information	14
Mossière	Génie logiciel	28
Scholl	Programmation	28
Sifakis	Génie logiciel	28
Chiaramella	Système d'Information	14
Kouloumdjan	Base de données	14

PLANNING (SALLE NOH NOJOUR NOMCONF NOM-COURS)

148	1	1	Levrat	Compilation
002	1	2	Bodart	Système d'Information
B110	1	1	Courbon	Système d'Information
148	2	1	Levrat	Compilation

Le très grand avantage de cette méthode de stockage est d'éviter complètement les pointeurs pour stocker les associations entre plusieurs entités de relations différentes.

Ainsi pour obtenir l'organisme du conférencier travaillant dans la salle B110 le lundi (NOJOUR = 1) à 8 heures (NOH = 1) il suffit d'aller rechercher son nom dans PLANNING (Courbon) et de rechercher l'entité correspondante dans CONFÉRENCIER pour obtenir GENÈVE.

Cette association a pu être réalisée par simple consultation des valeurs prises par les entités et par l'utilisation du mécanisme d'accès par clé qui permet d'atteindre une entité par la valeur qu'elle prend pour une clé.

L'élimination des pointeurs est sans aucun doute une source d'allègement de la base de données, d'une augmentation de son efficacité et de sa sûreté. Aussi le modèle relationnel de données est-il un sérieux candidat pour être un modèle informatique de données !

3.1.3. Vers un modèle informatique de données

Et pourtant le modèle relationnel de données ne peut pas être un modèle informatique de données pertinent. Il est incomplet pour cet usage. C'est notre avis que nous justifions en considérant plusieurs situations remarquables à l'aide de l'exemple précédent.

a) Si l'on cherche l'organisme du conférencier de nom Chiaramella, le SGBD ne peut trouver l'entité correspondante que par un parcours séquentiel des entités de CONFÉRENCIER.

b) Si l'on supprime le cours Bases de Données, alors il ne peut plus y avoir de conférences relatives à ce cours et de salles affectées à une conférence relative à ce cours. Pour exécuter ces deux opérations, le SGBD doit pouvoir déterminer les entités de CONFÉRENCE et de PLANNING prenant la valeur Bases de Données pour NOMCOURS. Avec la modélisation actuelle considérée comme modélisation informatique des données, le SGBD doit parcourir en séquentiel toutes les entités CONFÉRENCE et de PLANNING.

c) Si l'on supprime une entité de CONFÉRENCE (par exemple (Levrat Compilation 14)), le SGBD doit libérer les salles réservées pour cette conférence (SALLE = 148 NOH = 1 NOJ = 1) et (SALLE = 148 NOH = 2 NOJ = 1). Le SGBD ne peut les trouver toutes que par un parcours séquentiel des entités de PLANNING !

d) Si l'on recherche toutes les associations entre un conférencier et un cours, on obtient la réponse complète par l'interrogation de CONFÉRENCE seulement si le SGBD fonctionne bien comme nous venons de le décrire en c); sinon il faudrait aller chercher la réponse également dans PLANNING !

e) Ces trois situations remarquables et fréquentes nous interdisent de choisir le modèle relationnel de données comme modèle informatique de données. Il nous apparaît nécessaire d'avoir un modèle informatique de données qui prend en compte les chemins d'accès. Ainsi dans l'exemple précédent, nous allons considérer les chemins d'accès permettant d'associer à une entité de CONFÉRENCE toutes les entités de PLANNING relatives à cette conférence: le SGBD pourra alors facilement exécuter le traitement envisagé en c). Plus tard, nous étudions comment implanter ces chemins d'accès: par des index, des pointeurs, des tableaux pour obtenir la structure informatique. Notamment nous supposons qu'il existe un dispositif informatique (*mécanisme d'accès par clé c'est à dire par identifiant*) qui permet de trouver l'entité d'une relation prenant une valeur particulière pour la clé. Ce dispositif peut être hash-code, B-arbre, séquentiel indexé... Nous appelons ce modèle informatique de données le *graphe de chemins d'accès* (*GCA*). Il est fortement inspiré des travaux de (ABRIAL74).

Voilà les raisons qui nous conduisent à transformer une modélisation relationnelle de données dans un graphe de chemins d'accès et ensuite dans une structure informatique de données.

3.2. Fidélité d'une structure informatique

Pour que la structure informatique finale soit fidèle à la modélisation initiale, de telles transformations doivent respecter des conditions qui proviennent de la signification de la modélisation initiale:

a) Celle-ci a été établie par l'observation du champ d'application et elle met en évidence les données qui doivent pouvoir être stockées dans la base de données sous forme d'entités de relations de la modélisation. La structure informatique finale doit permettre le stockage de ces entités et seulement de celles-ci. Nous appelons cette condition la *préservation du contenu.*

b) Une modélisation contient des relations de base à partir desquelles nous pouvons définir des relations espaces. Implicitement - et c'est notre interprétation - une modélisation réclame que les entités des relations de base et des relations espaces soient facilement accessibles dans la base de données finale. Nous appelons cette condition la *préservation de l'accès.*
c) Avec le chapitre de la décomposition, nous avons montré qu'en général il existe plusieurs décompositions d'une même relation. Aussi la modélisation qui est transformée dans une structure informatique de données, a-t-elle été choisie parmi d'autres, généralement parce qu'elle permet une validation efficace des règles d'intégrité. La structure informatique finale doit préserver cette qualité. Nous appelons cette condition la *préservation de l'efficacité.*

Si le processus de transformation d'une modélisation de données dans une structure informatique de données remplit ces trois conditions, nous le qualifions alors de fidèle et nous disons que la *structure finale est fidèle à la modélisation initiale.* Au fur et à mesure de notre présentation de ce processus de transformation nous préciserons ces critères.

3.3. Graphe de chemins d'accès

Nous utilisons la notion de graphe de chemins d'accès comme outil abstrait permettant de représenter aussi bien une décomposition d'une relation espace qu'une structure informatique de données. Notre approche est similaire à celle de (HAINAUT86).

3.3.1. Rappels de définitions de la théorie des graphes

Comme nous allons travailler avec des graphes, nous allons brièvement rappeler les définitions des concepts de la théorie des graphes que nous allons utiliser fréquemment. Ces définitions sont extraites de (BERGE70), sauf celles de bassin et d'écoulement.
Un graphe G = (X,U) est le couple constitué par:
un ensemble de noeuds $X = \{ x_1 x_2 ... x_n \}$,
une famille $U = \{ u_1 u_2 ... u_m \}$ d'éléments du produit cartésien XxX, où un élément (x,y) peut apparaître plusieurs fois.
Un graphe est *simple* si un élément de la famille U n'y apparaît qu'une seule fois.

Transformation de structures

Si le graphe n'est pas orienté, un élément u = (x,y) de la famille U est appelé une *arête* dont x et y sont les extrémités.
Si le graphe est orienté, un élément u = (x,y) de la famille U est appelé un *arc* dont x est l'extrémité initiale et dont y est l'extrémité terminale. Un arc est une arête particulière. Deux arcs (ou arêtes) sont *adjacents* s'ils ont au moins une extrémité commune.

Une *chaîne* est une séquence ch = (u_1, u_2,...,u_q) d'arcs (ou d'arêtes) de G telle que chaque arc de la séquence a une extrémité en commun avec l'arc précédent, et l'autre extrémité en commun avec l'arc suivant.
On dit alors que *ch est une chaîne reliant N_1*, l'extrémité de u_1 qui n'est pas commune avec u_2 *et N_q* l'extrémité de u_q qui n'est pas commune avec l'arête précédente. N_1 et N_q sont les extrémités de la chaîne.
Une chaîne est *simple* si elle ne rencontre pas deux fois le même noeud; elle est *élémentaire* si elle n'utilise pas deux fois le même arc.

Un *chemin* ch = (u_1, u_2,...,u_p) est une chaîne formée d'arcs, où pour tout arc u_i (avec i < p) l'extrémité terminale de u_i coïncide avec l'extrémité initiale de u_{i+1}.
N_1 étant l'extrémité initiale de u_1, Np l'extrémité terminale de u_p, on dit alors que c'est un *chemin de N_1 à N_p*.. N_1 et Np sont respectivement les extrémités initiale et terminale du chemin.

Un *cycle* est une chaîne ch = (u_1, u_2,...,u_p) telle que:
la même arête (arc) ne figure pas deux fois dans la séquence,
les deux noeuds aux extrémités de la chaîne coïncident.
Un *circuit* est un cycle qui est formé par un chemin.

Deux chaînes sont *différentes* si elles ne contiennent aucune arête en commun.

Dans un graphe G(N,U), un *bassin de N_1* est un ensemble d'éléments de U: E = {u_1, u_2,...,u_n} tels que:
ces éléments sont des arêtes,
et l'une d'entre elles admet N_1 comme extrémité,
et pour chaque extrémité de ces arêtes N_i, il existe une chaîne reliant N_1 et N_i, formée d'éléments de E.
Dans un graphe G(N,U), un *écoulement du noeud N_1* est un ensemble d'éléments de U: E = {u_1, u_2,...,u_n} tels que:
ces éléments sont des arcs,
et l'un d'entre eux admet N_1 comme extrémité initiale,
et pour chaque extrémité N_i initiale ou terminale de chacun de ces arcs, il existe un chemin de N_1 à N_i formé d'éléments de E.

C'est une forme restreinte des écoulements étudiés dans (REYNAUD75). Les extrémités des éléments d'un bassin de N_1 (ou d'un écoulement de N_1) sont *les noeuds du bassin (ou de l'écoulement).*

3.3.2. Définitions liées à un graphe de chemins d'accès

Un *graphe de chemins d'accès* GCA(NO,U,REL,Fa,f,g,h,i,j) est un graphe défini de la manière suivante:

NO	l'ensemble des noeuds du graphe;
$U \subseteq NOxNO$	l'ensemble des arcs;
REL	l'ensemble de relations;
Fa	l'ensemble de chemins d'accès;
$f:NO \rightarrow REL$	f est une application partout définie et injective;
$g:U \rightarrow REL$	g est une application partout définie;
$h:U \rightarrow Fa$	h est une application partout définie et bijective;
$i:Fa \rightarrow NxNxN$	N désigne l'ensemble contenant les entiers et une valeur inconnue notée -; i est une application partout définie;
$j:NO \rightarrow \{ 0,1 \}$	un noeud no est un noeud d'entrée si j(no) = 1.

La condition suivante est vérifiée: $f(NO) \cup g(NO) = REL$.

Remarque:
le terme *chemin d'accès* utilisé notamment dans (ABRIAL74) désigne un *arc* d'un GCA dans la terminologie de la théorie des graphes.

Définitions supplémentaires:
Une *relation de noeud* RN est une relation telle qu'il existe un noeud no du GCA vérifiant RN = f(no). Une *relation d'arc* RA est une relation telle qu'il existe un arc a vérifiant: RA = g(a).
Il existe au plus deux arcs a et a' appartenant à $g^{-1}(RA)$: ils sont alors qualifiés de *réciproques.* Les *chemins d'accès* de deux arcs réciproques a et a': h(a) et h(a') sont également qualifiés de *réciproques*: l'extrémité initiale de l'un est l'extrémité terminale de l'autre et réciproquement.

La relation d'un chemin d'accès ch est la relation $g(h^{-1}(ch))$.

Une relation RE est une *relation d'entrée* si et seulement si il existe un noeud no vérifiant: h(no) = RE et j(no) = 1.

Interprétation d'un GCA:

Un arc a_{ij} dirigé du noeud de R_i au noeud de R_j correspond à un chemin d'accès $f_{ij} = h(a_{ij})$ et signifie qu'à toute entité de R_i on peut faire correspondre un ensemble d'entités de R_j appelé $f_{ij}(r_i)$. A chaque arc a_{ij} est associé une relation $g(a_{ij}) = R_{ij}(K_i K_j)$ (relation d'arc) formée sur une clé K_i de R_i et une clé K_j de R_j.

Chaque arc est renseigné à l'aide de l'application i qui lui associe trois paramètres:
cardmin fournit le nombre minimal d'entités de $f_{ij}(r_i)$ pour toute entité r_i de R_i (cardinalité minimale);
cardmax en fournit le nombre maximal (cardinalité maximale);
card80 fournit le nombre minimal tel que pour 80% des entités r_i de R_i la cardinalité de $f_{ij}(r_i)$ est inférieure à card80. Card80 n'est intéressant que si le cardmax est supérieur à 1 ou inconnu. L'exemple suivant illustre son utilité.
Si pour un chemin d'accès dirigé de la relation R_i à la relation R_j la cardinalité maximale vaut 1, alors la clé de la relation R_{ij} est une clé K_i de R_i et il existe la dépendance fonctionnelle $K_i \rightarrow K_j$; si de plus la cardinalité minimale vaut 1, alors il existe une dépendance existentielle entre R_i et R_j.

Si la relation RE est une relation d'entrée, ceci signifie que la base de données finale contiendra un dispositif informatique qui permettra d'atteindre directement les entités de RE par les valeurs qu'elles prennent pour les constituants d'une clé de RE.

Définition d'un graphe de chemins d'accès brut:

Un *graphe de chemins d'accès brut* GCA_b est un graphe de chemins d'accès où tous les noeuds sont des noeuds d'entrée et où tous les arcs admettent leurs arcs réciproques.

De plus toutes les relations de noeud d'un GCA_b doivent avoir des clés distinctes les unes des autres.

3.3.3. Exemple d'un GCA

Le graphe de chemins d'accès suivant est un graphe de chemins d'accès brut.

Graphe de chemins d'accès

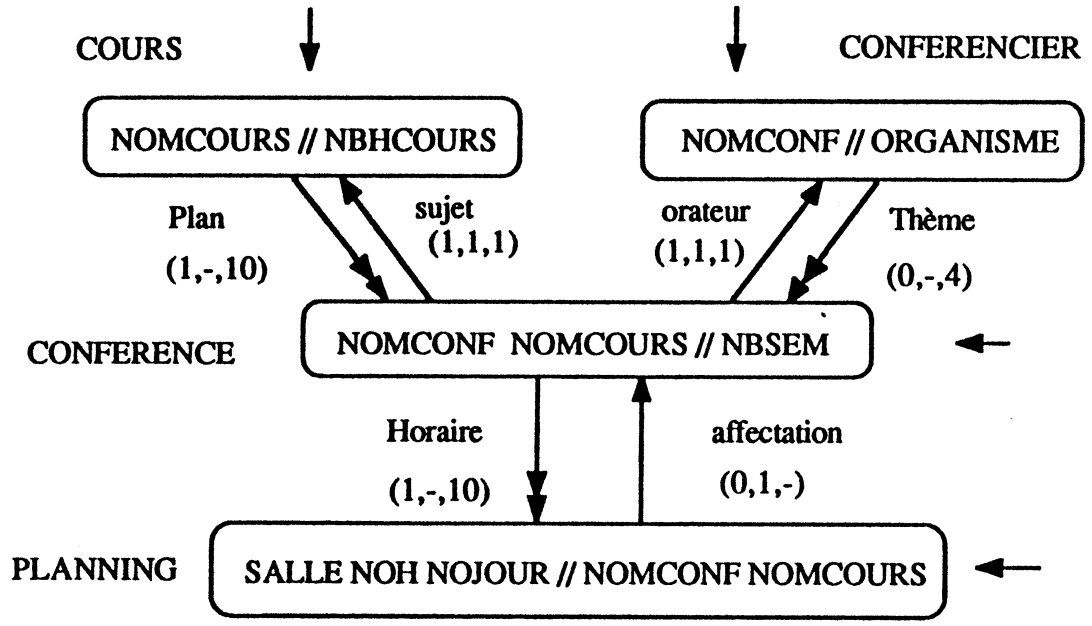

Figure 3.1. Exemple de graphe de chemins d'accès

Conventions graphiques:

Un arc correspondant à un chemin d'accès de cardmax égal à 1 est marqué d'une flèche, alors que si le cardmax est supérieur à 1, il est marqué de deux flèches.
Les noeuds d'entrée sont indiqués par des flèches.

Si la relation d'un noeud admet plusieurs clés, comme par exemple R (ABCX) de clés AB et AC, nous la notons de la manière suivante dans le graphe de chemins d'accès:

AB/AC//X

Les clés sont séparées les unes des autres par une barre et elles sont séparées des constituants de R n'appartenant à aucune clé par une double barre.

Commentaires:

Thème est un chemin d'accès qui permet d'associer un conférencier aux conférences qu'il donne; d'après les paramètres de thème, on sait qu'un conférencier peut ne donner aucune conférence (cardmin = 0), et que 80% de conférenciers donnent au plus quatre conférences (card80 = 4). Le nombre maximal de conférences d'un conférencier est inconnu. Bien sûr ce paramètre (card80) peut être une estimation grossière. Il permet d'implanter Thème à l'aide d'un tableau de dimension 4 et d'une zone de débordement qui ne sera pas utilisée dans 80% des cas.

Orateur et thème sont deux chemins d'accès réciproques et ils sont associés à une relation formée sur NOMCONF et NOMCOURS.
Lorsqu'un GCA contient deux chemins d'accès réciproques f_{ij} et f_{ji}, il contient de la redondance: les entités de la relation d'arc R_{ij} sont en effet prises en compte deux fois, une fois suivant f_{ij} et l'autre fois suivant f_{ji}. Ainsi le fait que Bodart donne une conférence pour le cours Système d'Information, sera d'une part pris en compte à l'aide du chemin d'accès Thème et d'autre part à l'aide du chemin d'accès orateur.

3.4. Graphe de relation

Un graphe de relation est une simplification d'un graphe de chemins d'accès brut. Cette simplification remplace les arcs réciproques par une seule arête (ou arc) et ne fait pas de différences entre les relations d'entrée et les autres. Elle va nous permettre de séparer les nombreux problèmes à résoudre lors de la transformation d'une modélisation en un graphe de chemins d'accès, en deux parties:

- ceux qui concernent des aspects conceptuels et qui sont résolus lors de la transformation de la modélisation dans un graphe de relation,

- et ceux qui concernent les facilités d'accès aux données qu'il faut prévoir pour les futurs traitements et les futures questions, et qui sont résolus lors de la transformation d'un graphe de relation dans un graphe de chemins d'accès.

3.4.1. Définition

Un *graphe de relation* GR(NR,UR,RELR,fr,gr,kr) est un graphe défini de la manière suivante:

NR:	l'ensemble des noeuds;
$UR \subseteq NR \times NR$	l'ensemble d'arêtes ou d'arcs;
RELR	l'ensemble de relations;
$fr: NR \rightarrow RELR$	fr est une application partout définie et injective;
$gr: UR \rightarrow RELR$	gr est une application partout définie et injective;
$kr: UR \rightarrow \{0,1\}$	kr est une application partout définie;
	u est un arc si et seulement si kr(u) = 0.

La condition suivante est vérifiée: $fr(NR) \cup gr(NR) = RELR$.

Interprétation d'un graphe de relation:

S'il existe un arc a dirigé du noeud N_1 vers le noeud N_2, alors les relations $R_1 = fr(N_1)$ et $R_1 = fr(N_2)$ sont telles que la df $KR_1 \rightarrow KR_2$ existe: gr(a) est formée sur les constituants clés des deux relations, et admet comme clés les clés de R_1.

S'il existe une arête a' entre deux noeuds N_1 et N_2, alors la relation gr(a') est formée sur les constituants clés de $R_1 = fr(N_1)$ et de $R_2 = fr(N_2)$, et ses clés ne sont ni celles de R_1 ni celles de R_2

De telles relations gr(a) ou gr(a') sont des *relations d'arête*. Les autres relations de REL sont des *relation de noeud.*

Définition d'un graphe orienté de relation:

Un *graphe orienté de relation* est un graphe de relation qui ne contient que des arcs. Ainsi tout élément de UR vérifie alors que kr(u) = 0.

3.4.2. Transformation d'un graphe de relation dans un graphe de chemins d'accès brut et réciproquement

Voici la transformation d'un graphe de chemins d'accès brut GCA_b(NO,U,REL,Fa,f,g,h,i,j) dans un graphe de relation GR(NR,UR,RELR,fr,gr,kr):
NR = NO,
RELR = REL,
fr = f.

A toute paire (a,a') d'arcs réciproques de U correspond un élément (ur) de UR tel que:
gr(ur) = g(a) (qui est égal à g(a'));

kr(ur) = 1 si les paramètres cardmax de a et de a' sont supérieurs à 1 (ur n'est pas alors un arc);
kr(ur) = 0 sinon (ur est un arc);

ur = a si ur n'est pas un arc ou si le paramètre cardmax de a est inférieur ou égal à 1;
a' sinon.

Transformation de structures

Voici la transformation réciproque d'un GR dans un GCA_b:

NO = NR,
REL = RELR,
f = fr.

A tout élément ur = (N_1,N_2) de UR est associée une paire d'arcs réciproques de U: a = (N_1,N_2) et a' = (N_2,N_1) tels que:
g(a) = gr(ur); g(a') = gr(ur).

Si kr(ur) = 0 alors les paramètres cardmax de h(a) et de h(a') sont supérieurs à 1; sinon le paramètre cardmax de h(a) est égal à 1 et celui de h(a') est supérieur à 1. Le concepteur a la charge de préciser les paramètres cardmin, cardmax, card80.

Pour tous les noeuds no de NO, j(no) = 1.

Exemple:
Voici le graphe de relation correspondant au graphe de chemins d'accès brut de l'exemple précédent:

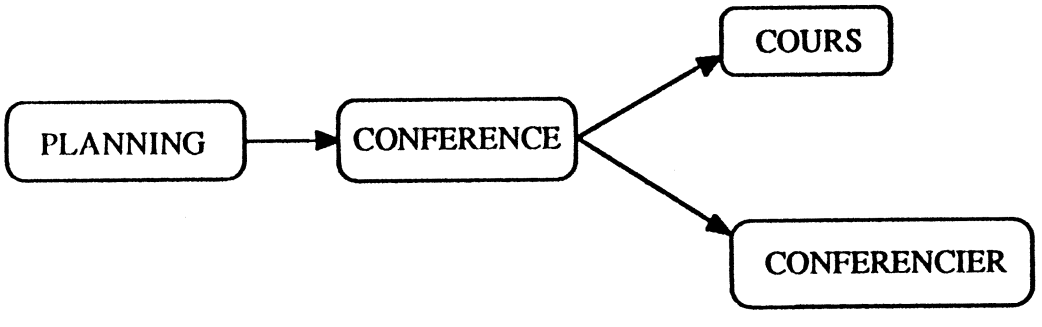

Figure 3.2. Exemple de graphe de relation

3.4.3. Cycle fonctionnel dans un graphe de relation

Un cycle fonctionnel dans un graphe de relation est formé par deux chemins dont les extrémités sont identiques et qui n'admettent aucun arc en commun. L'extrémité initiale de ces chemins est le noeud *source* du cycle fonctionnel; l'extrémité terminale de ces chemins est le noeud *puits* du cycle fonctionnel.

Exemple de cycle fonctionnel:

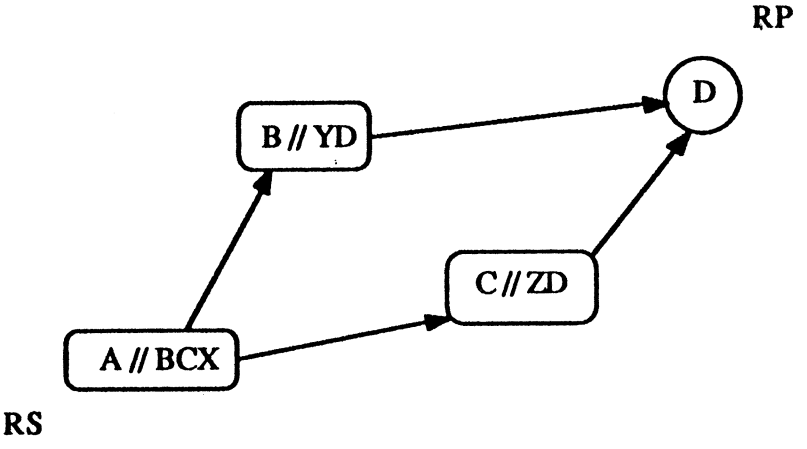

Figure 3.3. Exemple de cycle fonctionnel

L'étude des cycles fonctionnels nous semble très importante (§ 8.). Elle est liée à la découverte de contradictions potentielles dans la base de données: une entité de la source doit être reliée à la même entité du puits à travers les deux chemins du cycle.
Dans l'exemple, **a** doit être associé au même **d** à travers **b** et **c**. L'algorithme qui transforme une décomposition dans un graphe de relation (§ 4.) va les mettre en évidence.

3.5. Composition de relations implantée

La composition de relations $R_1,R_2,...,R_n$ est implantée dans un graphe de chemins d'accès GCA(NO,U.Fa,f,g,h,i,j) si et seulement si:

les relations $R_1,R_2,...,R_n$ sont des relations de REL,
et il existe un écoulement E dans GCA dont tous les éléments u vérifient:
g(u) est une des relations R_i ($i \in 1...n$),
et pour chaque relation R_i
soit il existe un élément u de E tel que $g(u) = R_i$,
soit il existe un noeud no de E tel que $f(no) = R_i$.

La composition de relations $R_1,R_2,...,R_n$ est implantée dans un graphe de relation GR(NR,UR,RELR,fr,gr,kr) si et seulement si:

Transformation de structures

les relations $R_1,R_2,...,R_n$ sont des relations de RELR,
et il existe un bassin E dans GR dont tous les éléments u vérifient:
gr(u) est une des relations R_i ($i \in 1...n$),
et pour chaque relation R_i:
soit il existe un élément u de E tel que $gr(u) = R_i$,
soit il existe un noeud nr de E tel que $fr(nr) = R_i$.

Interprétation:

Si la composition de relations $R_1,R_2,...,R_n$ est implantée dans un GCA, il existe alors un écoulement E de source, par exemple le noeud NS.
A partir de la relation RS associée à ce noeud, il est possible d'exécuter l'algorithme suivant pour obtenir les entités de la composition des relations $R_1,R_2,...,R_n$: pour chaque entité rs de RS, parcourir les chemins d'accès associés aux éléments de E et ainsi obtenir toutes les entités des relations $R_1,R_2,...,R_n$ compatibles avec rs.

Il existe ainsi un algorithme simple et efficace d'obtention de entités de la composition des relations $R_1,R_2,...,R_n$ dans le GCA. Cet algorithme sera applicable dans la future base de données construite à partir de ce GCA. C'est en ce sens que nous disons que cette composition est implantée.

Propriété:
Soit un graphe de relation GR et un graphe de chemins d'accès brut GCA_b qui se transforment l'un dans l'autre suivant la description du paragraphe 3.4.2. Si une décomposition de relations est implantée dans l'un, elle l'est dans l'autre.

La preuve est immédiate.

Un espace RE est dit *implanté* dans un GCA ou dans un GR si la composition des relations $\{R_1...R_n\}$ qui la forment, est implantée dans ce GCA ou dans ce GR.

Dans le § 5. nous transformons une décomposition $\{R_1,R_2,...,R_n\}$ d'une relation R dans une structure informatique de données dans laquelle la composition des relations $R_1,R_2,...,R_n$ est implantée.

Exemple:
Soit les relations $R_1(AX)$, $R_{12}(AB)$, $R_2(BY)$, $R_{23}(BC)$ et $R_3(CZ)$ et nous examinons la composition des relations: $ER = R_1*R_{12}*R_2*R_{23}*R_3$

pour les trois GCA suivants:

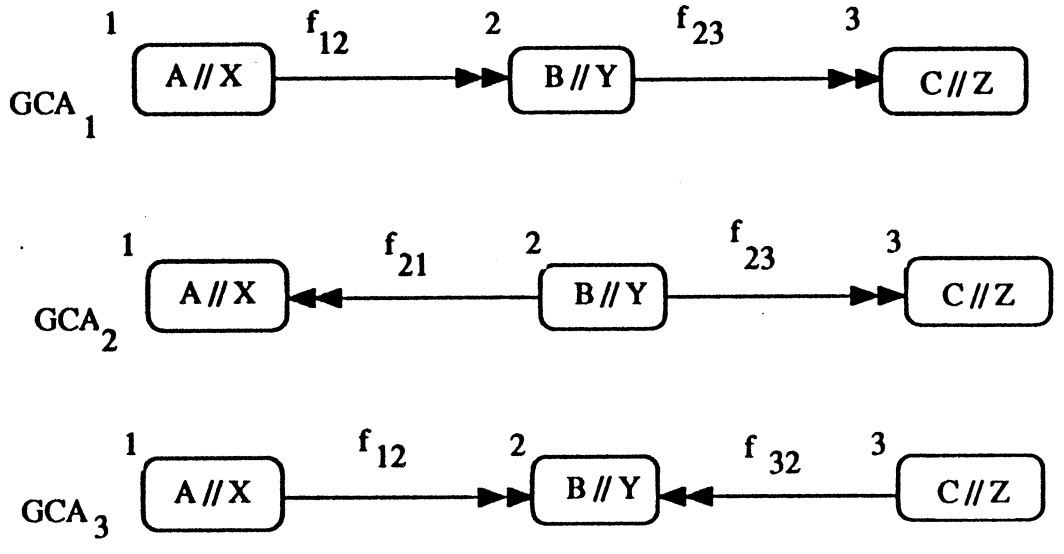

Figure 3.4. GCA et composition de relations

Cette composition est implantée dans GCA_1 (écoulement f_{12} f_{23} partant du noeud 1) et dans GCA_2 (écoulement f_{21} f_{23} partant du noeud 2). Par contre elle ne l'est pas dans GCA_3.

3.6. Plan de la 2ème partie

Voici les différents niveaux du processus de conception de bases de données que nous proposons:

- transformation d'une modélisation relationnelle initiale de données dans un graphe de relation et puis dans le graphe de chemins d'accès brut correspondant;
- choix d'un graphe de chemins d'accès parmi ceux qu'il est possible d'obtenir à partir du GCA brut précédent;
- détermination d'une structure informatique de données en indiquant la méthode informatique choisie pour l'implantation de chaque chemin d'accès.

De telles transformations garantissent la fidélité de la structure informatique finale à la modélisation initiale si:
- les relations de noeud et d'arête du GCA final recouvrent les relations de la modélisation initiale; ainsi elles préservent le contenu de la base;

- les relations espaces sont implantées dans le GCA final; ainsi elles préservent l'accès aux entités;
- les règles d'intégrité intégrées à la modélisation, comme les dépendances fonctionnelles intrinsèques aux relations, les dépendances de composition définies sur les relations, peuvent être facilement validées dans le contexte de la structure informatique finale.

La troisième condition qui est liée à la préservation de l'efficacité, fait l'objet d'un travail particulier au cours de l'obtention du graphe de relation.

4

TRANSFORMATION D'UNE DÉCOMPOSITION DANS UN GRAPHE DE RELATION

Il existe plusieurs représentations graphiques d'un ensemble de relations formant une décomposition (ZANIOLO76, MAIER83) notamment à partir des hypergraphes (BERGE70). Nous allons en proposer une autre que nous avons appelée *graphe de relation* dans le chapitre précédent.

Cette représentation n'est qu'une étape intermédiaire vers l'obtention d'une structure informatique de données; elle va permettre de poser les problèmes informatiques pas à pas. La transformation d'une décomposition compacte D dans un graphe de relation GR sera une transformation fidèle dans le sens du paragraphe 2 du chapitre précédent. Des situations complexes pourront ainsi être étudiées dans la 3ème partie.

Nous allons présenter l'algorithme qui assure la transformation d'une décomposition compacte dans un graphe de relation dans le premier paragraphe d'une manière plutôt informelle; nous l'illustrons à l'aide d'un exemple d'un chapitre précédent; nous le décrivons formellement en annexe et nous montrons à l'aide d'exemples simples l'utilité des différentes parties de l'algorithme.

Dans le second paragraphe de ce chapitre nous invitons le concepteur à critiquer le graphe de relation obtenu par l'algorithme, en s'interrogeant sur l'utilité de toutes les compositions de relations implantées dans le graphe de relation.

4.1. Graphe de relation d'une décomposition

4.1.1. Présentation de l'algorithme

Nous présentons l'algorithme de transformation d'une décomposition D d'une relation espace RE dans un graphe de relation GR, sans tous les détails qui se trouvent dans l'annexe. Il prolonge nos travaux (LÉONARD83) et ceux de (LUONG86). Il se compose de 6 étapes:

E1. Décomposition compacte d'une décomposition initiale:
la première étape transforme la décomposition initiale D dans une décomposition **compacte** (définition § 4.4.1.) où:
toutes les relations ayant une clé commune sont regroupées en une seule, et à chaque charnière entre les relations correspond une relation.

La seconde condition devient complexe dans certaines situations qui sont étudiées dans l'annexe.

E2. Noeuds du graphe de relation:
chaque relation R de la décomposition compacte est associée à un noeud de GR.

E3. Arcs du graphe de relation:
si une relation R_i contient dans ses constituants une clé de la relation R_j, alors il doit exister dans GR un chemin partant du noeud de R_i et allant au noeud de R_j. $INF(R_i)$ désigne l'ensemble de ces relations R_j pour la relation R_i. Si nous construisons un arc du noeud de R_i au noeud de R_j pour chaque relation R_j de $INF(R_i)$, nous allons en construire certains qui pourraient se déduire d'autres par transitivité.

Nous qualifions d'inutiles les arcs qui relieraient la relation R_i à la relation R_j s'il existe une relation R_k de $INF(R_i)$ dont l'ensemble des constituants clés contient celui de R_j; $INUTILE(R_i)$ est l'ensembles de telles relations R_j. $MINFD(R_i)$ qui est égal à $INF(R_i)$ - $INUTILE(R_i)$, donne l'ensemble des relations de $INF(R_i)$ qui sont reliées à R_i par un arc partant de R_i.

Dans l'annexe nous expliquons qu'il existe d'autres possibilités de définir les arcs inutiles et nous indiquons pourquoi nous avons retenu celle-ci.

E4. Nettoyage de noeuds:
c'est une simple question de détail que nous traitons dans l'annexe.

E5. Affiner les relations associées aux noeuds:
il s'agit éventuellement de supprimer certains constituants des relations; nous décrivons cette étape en annexe.

E6. Détermination des arêtes (B-arêtes):
soit un noeud N_{ij} (relation R_{ij}) relié seulement à deux noeuds N_i (R_i) et N_j (R_j) par des arcs partant de N_{ij}, dont la relation est formée de l'ensemble des constituants clés de R_i et de R_j et dont les clés sont formées chacune par l'union d'une clé de R_i et d'une de R_j;
il est remplacé par une arête reliant les noeuds N_i et N_j; les arcs reliant N_{ij} à N_i et à N_j sont supprimés.

Remarque:

Si l'on veut trouver seulement **un graphe orienté de relation** à partir d'une décomposition, il suffit de ne pas effectuer l'étape E6.

4.1.2. Exemple: Atelier de fabrication

4.1.2.1. Décomposition compacte

Voici les relations qui forment une décomposition de la relation ATELIER (§ 1.5.):

OUVRIER(NOV NOMOV).	Clé:	NOV.
ACTIVITÉ(NOV NH ACTIF).	Clé:	NOV NH.
MACHINE(NMH FONCTION).	Clé:	NMH.
DISP-MACHINE(NMH NH DISPMH).	Clé:	NMH NH.
PRODUIT(NP QMINFAB).	Clé:	NP.
LOT(NP NLOT NBH NMH).	Clé:	NP NLOT.
M-P(NMP QMP).	Clé:	NMP.
APPR(NP NMP QMINMP).	Clé:	NP NMP.
COMPÉTENCE(NOV NMH).	Clé:	NOV NMH.
UTILISATION(NP FONCTION).	Clé:	NP.

EMPLOI-DU-TEMPS(NH NP NLOT NOV NMH)
Clés: NH NP NLOT,
NH NMH,
NH NOV.

Cette décomposition n'est pas compacte car il existe des relations ayant une même clé:
d'une part EMPLOI-DU-TEMPS et ACTIVITÉ (clé NOV NH),
et EMPLOI-DU-TEMPS et DISP-MACHINE (clé NMH NH),
d'autre part UTILISATION et PRODUIT (clé NP).

Ces relations doivent être regroupées respectivement en:
EMPLOI-DU-TEMPS'(NH NP NLOT NOV NMH ACTIF DISPMH),
et PRODUIT'(NP QMINFAB FONCTION).

Cette décomposition n'est pas encore compacte car il existe une charnière FONCTION entre les relations PRODUIT' et MACHINE, qui n'est clé d'aucune relation de la décomposition: il faut créer une nouvelle relation FONCTION admettant un seul constituant FONCTION pour rendre la décomposition compacte.

Une charnière complexe est celle des relations EMPLOI-DU-TEMPS' et LOT: NP NLOT NMH. La clé de cette charnière est NP NLOT; comme celle-ci est clé d'une relation de la décomposition (LOT), il n'y a pas lieu de créer une nouvelle relation.
Mais il ne faut pas s'arrêter ici; il faut considérer les constituants de la charnière n'appartenant pas à la clé de la charnière NMH (voir § 4.4.1.) et refaire la même étude. Ici il n'y a pas lieu de créer une nouvelle relation car NMH est clé d'une relation (MACHINE).

Toutes les autres charnières sont des clés d'autres relations: par exemple la charnière de APPR et de LOT est NP qui est clé de PRODUIT'.

Cette manière de procéder conduit à mettre en évidence les cycles fonctionnels contenus dans la décomposition et sources potentielles de contradictions (§ 8.).

La décomposition formée par les relations OUVRIER EMPLOI-DU-TEMPS' MACHINE PRODUIT' LOT M-P APPR COMPÉTENCE FONCTION est compacte.

4.1.2.2. Arcs et noeuds du GR (étapes E2, E3)

INF(EMPLOI-DU-TEMPS') = {OUVRIER MACHINE PRODUIT' LOT COMPÉTENCE},
INUTILE(EMPLOI-DU-TEMPS') = {OUVRIER MACHINE PRODUIT'}
MINFD(EMPLOI-DU-TEMPS') = {LOT COMPÉTENCE}.

Les clés des relations OUVRIER MACHINE PRODUIT' LOT COMPÉTENCE sont des constituants de la relation EMPLOI-DU-TEMPS' et ainsi ces relations forment INF(EMPLOI-DU-TEMPS'). OUVRIER appartient à INUTILE(EMPLOI-DU-TEMPS') car sa clé NOV est contenue dans celle de COMPÉTENCE.

Ainsi un ouvrier est associé à une machine dans EMPLOI-DU-TEMPS' seulement s'il est compétent pour la conduire. Cette règle provient du fait que les premières relations forment une décomposition de ATELIER: ainsi EMPLOI-DU-TEMPS'[NOV NMH]=COMPÉTENCE.

INF(OUVRIER) = Ø.
INF(COMPÉTENCE) = {OUVRIER MACHINE},
    INUTILE(COMPÉTENCE) = Ø,
    MINFD(COMPÉTENCE) = {OUVRIER MACHINE}.
INF(MACHINE) = {FONCTION},
    INUTILE(MACHINE) = Ø,
    MINFD(MACHINE) = {FONCTION}.
INF(LOT) = {PRODUIT' MACHINE},
    INUTILE(LOT) = Ø,
    MINFD(LOT) = {PRODUIT' MACHINE}.
INF(PRODUIT') = {FONCTION},
    INUTILE(PRODUIT') = Ø,
    MINFD(PRODUIT') = {FONCTION}.
INF(APPR) = {M-P PRODUIT'},
    INUTILE(APPR) = Ø,
    MINFD(APPR) = {M-P PRODUIT'}.
INF(M-P) = Ø.
INF(FONCTION) = Ø.

Le graphe de relation se trouve dans la figure 4.1. de la page suivante.

La sixième étape ne modifie pas ce graphe de relation. Aucun noeud ne peut être transformé dans une arête. APPR contient un constituant QMINMP qui n'est pas un constituant clé, et COMPÉTENCE est relié à trois noeuds.

Remarque:

le cycle fonctionnel entre EMPLOI-DU-TEMPS' LOT COMPÉTENCE et MACHINE est relatif aux contradictions possibles entre la machine affectée à un lot à une heure donnée dans EMPLOI-DU-TEMPS' et la machine qui doit fabriquer ce lot d'après LOT.

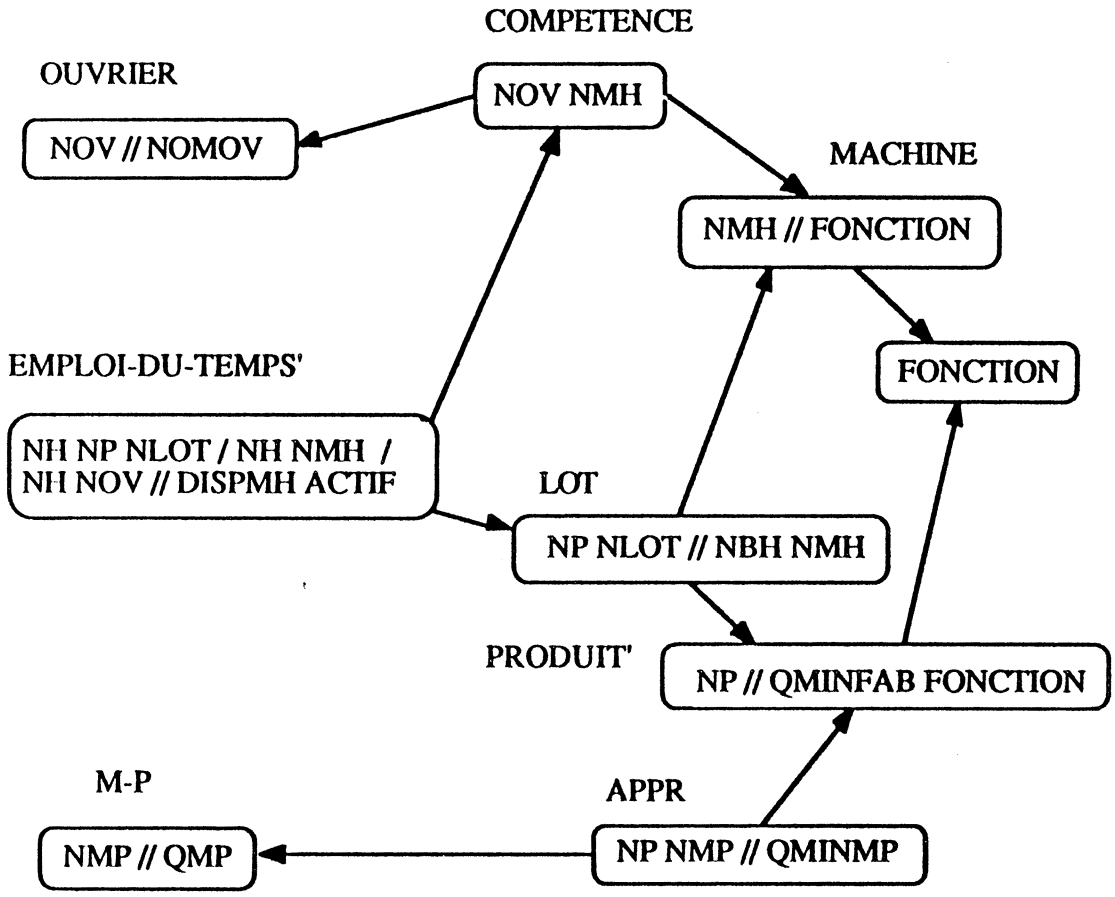

Figure 4.1. Atelier de production

4.2. Utilité des charnières

Avec l'utilisation d'une décomposition compacte on a construit systématiquement toutes les charnières possibles entre les relations. Néanmoins certaines d'entre elles peuvent être inutiles car elles ne sont jamais utilisées pour faire la composition de deux instances. En voici un cas qui nous semble très important.

Ces charnières correspondent à un défaut dans la modélisation initiale: deux constituants portent le même nom alors qu'en fait ils désignent des informations sémantiquement différentes même si elles sont représentées par des constituants ayant le même domaine.

Exemple :

LIVRAISON(*NLIVRAISON* DATE NOCOMMANDE)
Une livraison est connue par son numéro; on enregistre la date à laquelle elle a eu lieu, et la commande qui en est à l'origine.

COMMANDE(*NOCOMMANDE* DATE NOCLIENT)
Une commande est connue par son numéro; on enregistre la date de sa réception et le numéro du client qui l'a passée.

Voici le graphe de relation obtenu par l'algorithme précédent:

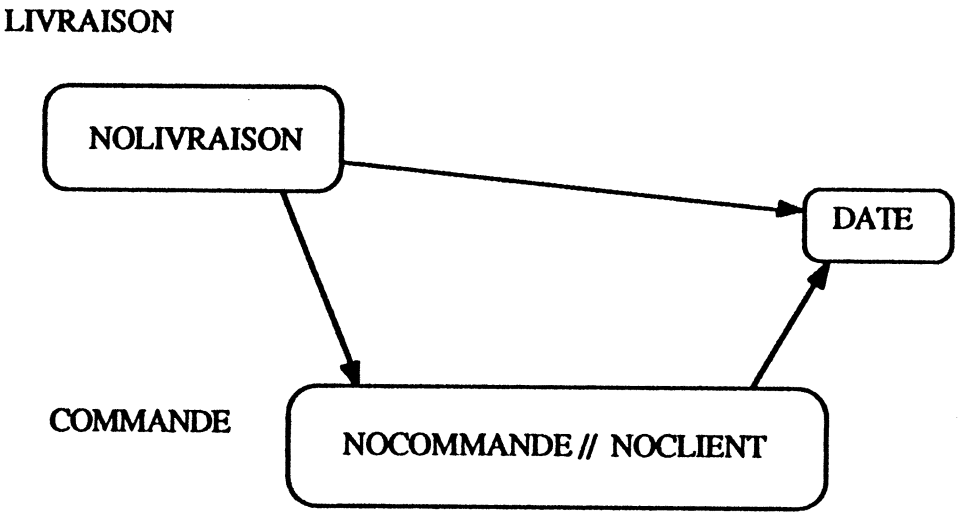

Figure 4.2. Charnière

En fait, la charnière DATE est totalement inutile car elle ne permet aucune composition: il s'agit dans LIVRAISON de la date de livraison et dans COMMANDE de la date de commande.

C'est au concepteur de la base de s'apercevoir que cette charnière est inutile, que la modélisation initiale peut conduire à des erreurs (quelle est la signification de DATE quand on fait LIVRAISON * COMMANDE ?); il propose rapidement la modification suivante:

LIVRAISON(NLIVRAISON DATE-LIVRAISON NOCOMMANDE);
COMMANDE(NOCOMMANDE DATE-COMMANDE NOCLIENT);

DATE-LIVRAISON et DATE-COMMANDE ont le même domaine: DATE.

4.3. Conclusions

La conception de bases de données demande:
- d'une part la transformation du graphe de relation obtenu dans une structure informatique de données: c'est l'objet des chapitres suivants de cette seconde partie;
- d'autre part le choix de la décomposition compacte initiale: dans la troisième partie, nous préconisons l'utilisation d'une décomposition compacte qui garantit l'équivalence de toutes les clés de chacune de ses relations: nous l'appelons décomposition *complètement homogène.*

Pour ne pas alourdir cette seconde partie, nous anticipons les résultats théoriques du (§ 8.10.); ils nous permettent de garantir que la transformation d'une décomposition compacte D d'une relation RE dans un graphe de relation GR à l'aide de notre algorithme:
- préserve le contenu de la base: les relations de D se retrouvent toutes dans les relations que l'on peut extraire de GR;
- préserve l'accès: si D est homogène, nous montrerons que GR est connexe et la composition des relations de D est implantée dans GR;
- préserve l'efficacité: les df intrinsèques aux relations de D sont également intrinsèques aux relations de RELR du GR. Leur validation sera aussi efficace dans le cas de D que dans celui de GR.
Ainsi le graphe de relation GR est *fidèle* à la décomposition D dans le sens du chapitre précédent (§ 3.2.).

4.4. Annexe

Nous présentons maintenant notre algorithme de transformation d'une décomposition D d'une relation espace RE dans un graphe de relation GR (NR, UR, RELR, fr, gr, kr).

4.4.1. Décomposition compacte

Définition:
Une décomposition $D = (R_1, R_2,...,R_n)$ d'une relation RE est *compacte* si et seulement si:
- il n'existe pas deux relations R_i et R_j telles que leurs clés K_i, K_j sont équivalentes c'est-à-dire vérifient $K_i \rightarrow K_j$ et $K_j \rightarrow K_i$,
- et pour toute paire de relations R_i et R_j distinctes de D:
si $R_{ij}^+ = R_i^+ \wedge R_j^+$ n'est pas vide, il existe une relation de D: R' qui admet

les mêmes clés que la relation R_{ij} = RE[R_{ij}^+] et qui contient les constituants de R_{ij}^+ (R' pouvant être égale à R_{ij});
si $RR_{ij}^+ = R_{ij}^+ - KR_{ij}^+$ n'est pas vide, alors il existe une relation de D: RR' qui admet les mêmes clés que la relation RR_{ij} = RE[RR_{ij}^+] et qui contient les constituants de RR_{ij}^+ (RR' pouvant être égale à RR_{ij}).

Interprétation:
les relations R' et RR' sont relatives à la charnière des relations R_i et R_j; leurs présences dans la décomposition compacte vont permettre la composition des relations R_i et R_j est possible dans la base de données finale.

Exemples:
(1.1) Soit une relation **RE**(ABC) qui se décompose suivant R_1(*B*A) et R_2(*C*A) de clés respectives B et C. La mise en place du noeud charnière R_{12}(A) permet à la relation RE d'être implantée dans le graphe de relation qui met en évidence un cycle fonctionnel:

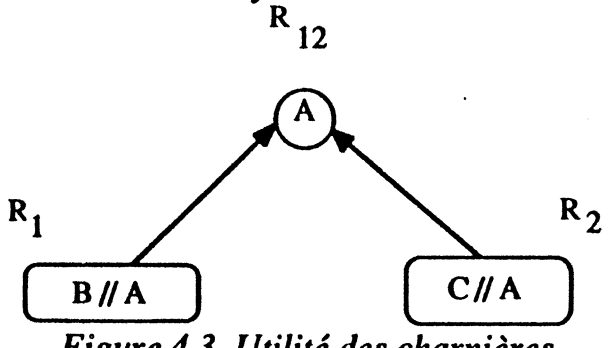

Figure 4.3. Utilité des charnières

(1.2) Soit la relation **RE**(SABC) qui se décompose suivant les relations: R_1(*S*AB), R_2(*A*C), R_3(*B*C) qui admettent chacune une seule clé formée par les constituants en italique.
Pour rendre compacte cette décomposition, il faut lui adjoindre la relation R_4(C). Voici le graphe de relation:

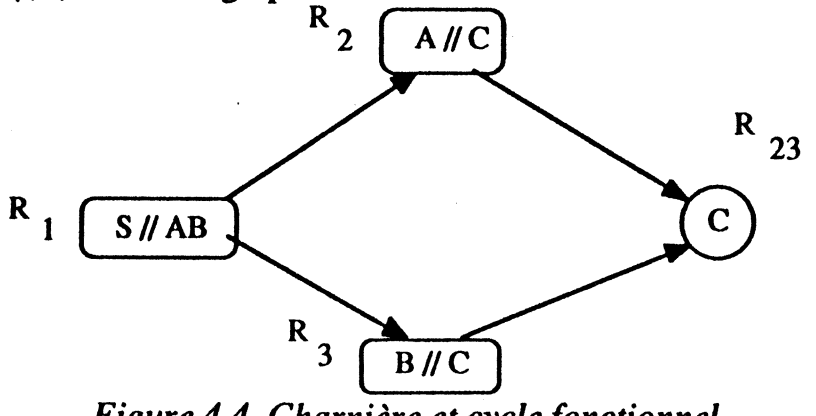

Figure 4.4. Charnière et cycle fonctionnel

(1.3) Soit une autre relation RE(ABCX) qui se décompose suivant:
R_1(ABCX) qui admet deux clés: AB et BC
et R_2(CA) qui admet C comme seule clé.

Pour rendre compacte cette décomposition, il faut lui adjoindre la relation R_3(A). En effet la charnière entre R_1 et R_2 est formée de CA dont la clé est C; il reste A qui n'est la clé d'aucune relation et qui donne lieu à cette nouvelle relation. L'introduction de cette nouvelle relation fait apparaître un cycle dans le GR: le § 8.9.Propriété 6 montre l'importance de ce cycle fonctionnel; il fait apparaître des risques de contradictions entre R_1[CA] et R_2(CA).

Voici le graphe de relation:

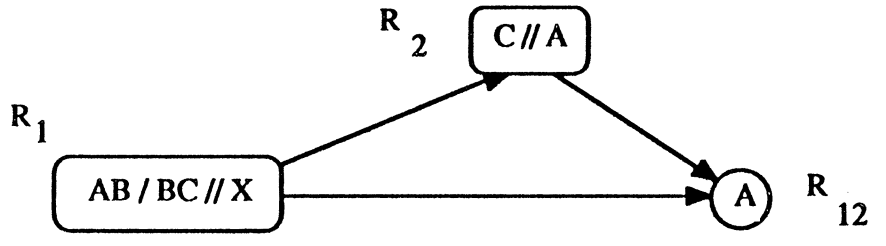

Figure 4.5. Charnière et cycle fonctionnel(2)

4.4.2. Noeuds du graphe de relation

RELR du graphe de relation GR est égal à l'ensemble des relations formant cette décomposition compacte.
A chaque relation R de RELR, on associe un noeud NO et fr(O) = R.

4.4.3. Arcs du graphe de relation

Pour chaque R_i associée à un noeud, déterminer:
$INF(R_i) = \{R_j, R_i^+ \supset KR_j^+\}$,
$INUTILE(R_i) = \{R_j \in INF(R_i), \exists R_h \in INF(R_i)$ telle que $KR_h^+ \supset KR_j^+\}$
$MINFD(R_i) = INF(R_i) - INUTILE(R_i)$ (étape E3.1.);

Pour chaque R_j de $MINFD(R_i)$ on construit un arc orienté du noeud de R_i au noeud de R_j: (étape E3.2.); et on crée une nouvelle relation R' de RELR: $R' = RE[KR_i^+ \cup KR_j^+]$ (étape E3.3.).

On obtient ainsi un nouvel arc u de GR:
$u = (fr^{-1}(R_i), (fr^{-1}(R_j)) \in UR$ et $kr(u) = 1$ et $gr(u) = R'$.

4.4.3.1. Différentes solutions possibles

Soit la relation RE(SABXYZ) qui se décompose suivant:
R_1(*S*AB), R_2(*AB*X), R_3(*A*Y) et R_4(*B*Z) de clés respectives S, AB, A et B.
INF(R_1) = {R_2, R_3, R_4}.
R_3 n'appartient pas à MINFD(R_1) car il existe R_2 dans INF(R_1) telle que KR_2^+ (=AB) contient KR_3^+ (=A).

Voici le graphe de relation GR que l'on obtient finalement:

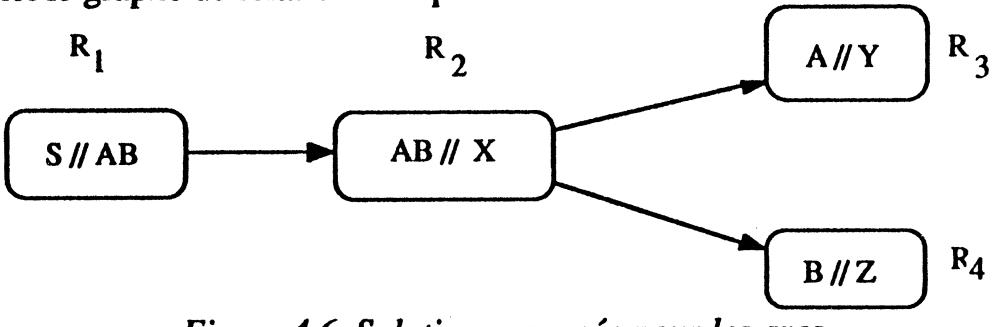

Figure 4.6. Solution proposée pour les arcs

C'est notre proposition. Elle a un avantage: iRE[AB] = iR_2[AB]; il suffit de consulter les entités de iR_2 pour obtenir toutes les entités de iRE[AB]. Elle a un inconvénient. Si l'on veut stocker dans la base le fait (sa), on ne pourra le faire que si l'on connaît (sb) et on stockera alors (sab).
Ce problème rentre dans un cadre plus général que nous étudions en proposant le mécanisme complémentaire d'association dans (§ 10.3.).

Il est possible d'imaginer deux autres solutions.

La première correspond au graphe de relation GR_1 suivant:

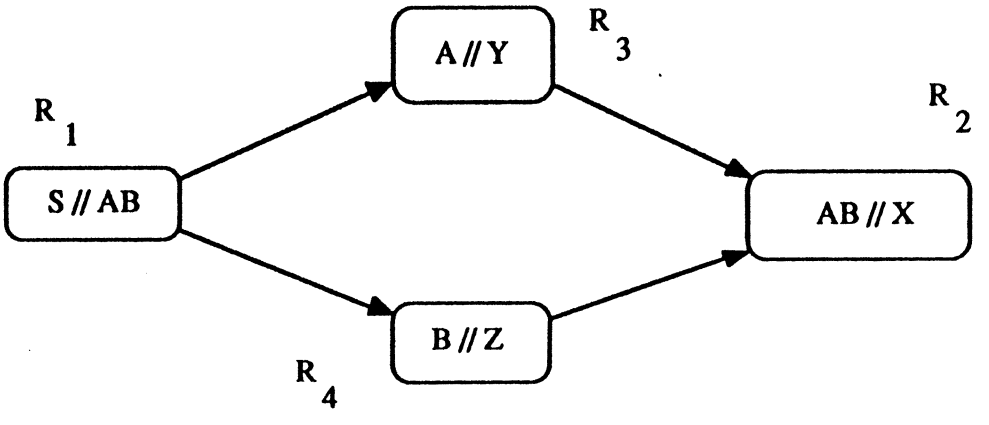

Figure 4.7. Autre solution(1) pour les arcs

La seconde correspond au graphe de relation GR_2 suivant:

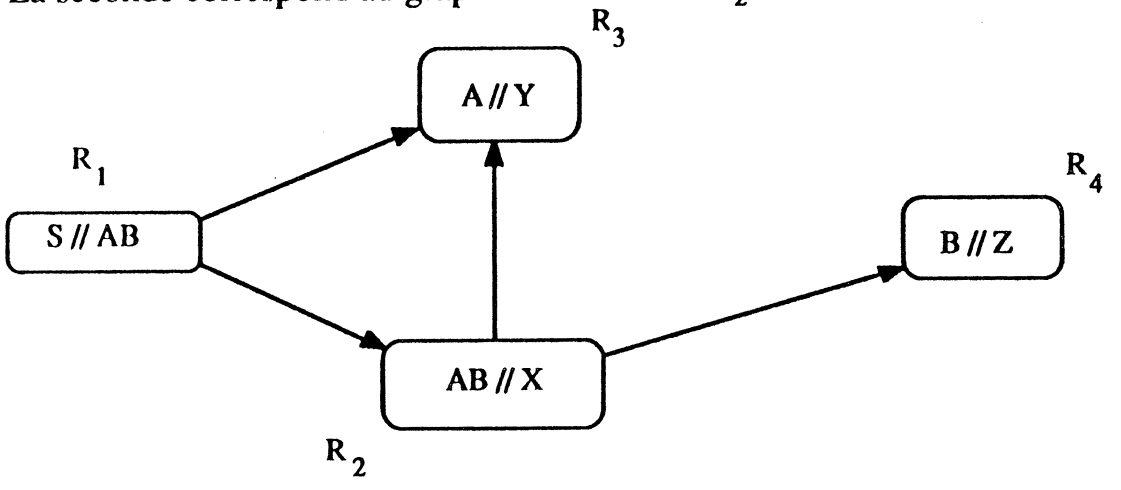

Figure 4.8. Autre solution(2) pour les arcs

La première solution a pour nous un inconvénient majeur:
$iRE[AB] = iR_1[AB] \cup iR_2[AB]$. Il faut utiliser un algorithme complexe pour obtenir une réponse complète à une question relative à iRE[AB].
La seconde solution contient de la redondance sur RE[SA].
Ces inconvénients nous font préférer la première solution correspondant au graphe GR.

4.4.3.2. Exemple illustrant la solution que nous proposons

Soit la relation RE(ABCXYZ) qui se décompose de manière compacte suivant les relations $R_1(ABCX)$, $R_2(BCY)$, $R_3(CZ)$ de clés respectives ABC, BC et C. $INF(R_1) = \{R_2, R_3\}$.
R_3 n'appartient pas à $MINFD(R_1)$ car l'ensemble des constituants clés de R_2 (=BC) contient celui de R_3 (=C). $MINFD(R_1) = \{R_2\}$.
Voici le graphe de relation que l'on obtient finalement:

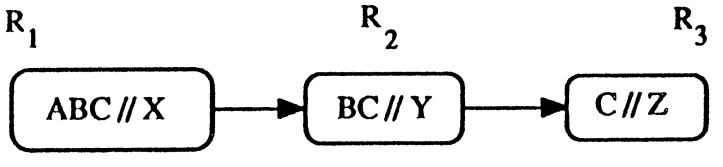

Figure 4.9. Arcs et inclusion de clés(1)

L'arc reliant R_1 à R_3 nous apparaît comme inutile car si une entité r_1 existe, alors il doit exister une entité r_2 telle que $r_2.BC=r_1.BC$; si r_2 existe alors il doit exister une entité r_3 telle que $r_2.C=r_3.C$.
Ainsi si r_1 existe, il existe r_3 telle que $r_1.C=r_3.C$.
L'arc reliant R_1 à R_3 apparaît alors comme déductible des deux autres.

4.4.3.3. Autre exemple

Soit la relation RE(ABCDEXYZ) qui se décompose suivant les relations:
R_1(ABCDEX) de clés CDE et AE, R_2(ACDY) de clés AD et CD,
R_3(ABCZ) de clés AB et C.
INF(R_1) = {R_2, R_3}.

R_3 appartient à MINFD(R_1) car l'ensemble des constituants clés de R_2 (=ACD) ne contient pas celui de R_3 (=ABC). MINFD(R_1) = {R_2, R_3}.
Voici le graphe de relation que l'on obtient finalement:

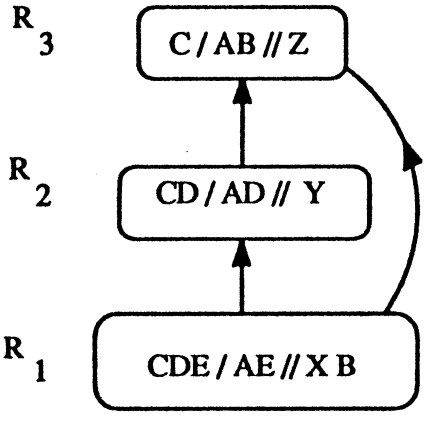

Figure 4.10. Arcs et inclusion de clés(2)

Ici l'arc reliant R_1 à R_3 nous apparaît comme utile car il montre le cycle fonctionnel sousjacent à la décomposition et un problème de contradiction relatif à la validation de la df C → AB par exemple (§ 8.).

4.4.4. Nettoyage de noeuds

On supprime tout noeud N_i de GR tel que
- sa relation R_i admet une clé unique KR_i égale à R_i^+,
- et N_i est relié à un seul noeud N_j par une arête dirigée de N_j vers N_i.

Elle permet d'éliminer des noeuds inutiles qui ont pu être construits au cours de la deuxième étape et qui proviennent des charnières entre relations: elle n'a qu'un intérêt pratique. Voici un exemple de son utilisation:

Soit la relation RE(SABCDXYZ) qui se décompose de manière compacte comme suit:
R_1(SABCD X) de clé S, R_2(ABCDZ) de clés AD et BD,
R_3(ABC Y) de clés A et BC, et R_{12} = RE[C].

Voici le graphe de relation que l'on obtient à la fin de la troisième étape:

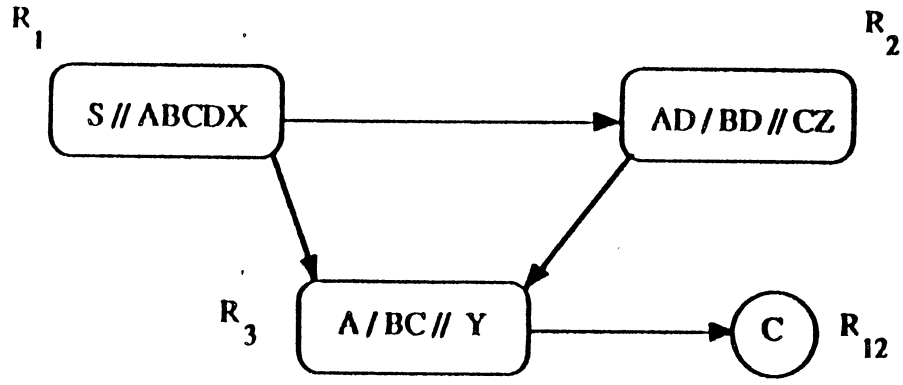

Figure 4.11. Noeud inutile

Par application de la quatrième étape le noeud de R_{12} est supprimé ainsi que l'arête reliant R_3 à R_{12}.

4.4.5. Affiner les relations associées aux noeuds

(5.1) Pour chaque noeud Ni (de R_i):
pour chaque $R_j \in MINFD(R_i)$,
$KR_j'^+ := KR_j^+ - \{ A \in (KR_i^+ \cap KR_j^+)\}$

(5.2) $R_i'^+ := R_i^+ - \{\cup KR_j'^+\}$ (pour $\forall R_j \in MINFD(R_i)$).

Cette nouvelle relation $R_i' = RE[R_i'^+]$ est associée au noeud Ni.
Bien sûr tous les constituants clés doivent être conservés.
Nous allons montrer l'intérêt de cette étape à l'aide de l'exemple précédent. Voici comment l'algorithme de la cinquième étape transforme le graphe de relation précédent:

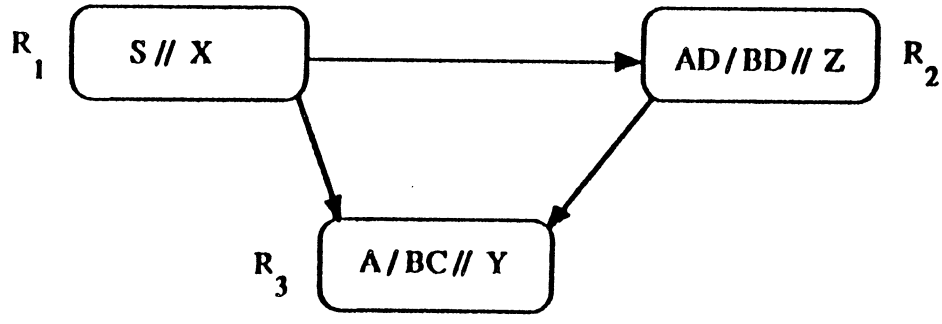

Figure 4.12. Affinement d'une relation

Les constituants ABCD ont disparu du noeud de R_1 car si l'on veut stocker une entité (sabcdx), il suffit de stocker une entité r_1 = (sx) dans iR_1, r_2 = (abd-) dans iR_2, r_3 = (abc-) dans iR_3 et d'associer r_1 à r_2 et r_1 à r_3.
Nous verrons dans le chapitre 5 comment implanter de telles associations. Une possibilité parmi d'autres est justement de stocker l'entité (sabcdx).

La cinquième étape préserve ainsi les choix possibles d'implantation de ces associations.

4.4.6. Détermination des arêtes (B-arêtes)

Soit un noeud N_{ij} (relation R_{ij}) relié seulement à deux noeuds N_i (R_i) et N_j (R_j) par deux arcs, u = (N_{ij},N_i) et u' = (N_{ij},N_j), et qui vérifie:
$KR_{ij}^+ = R_{ij}^+$ et $KR_{ij}^+ = KR_i^+ \cup KR_j^+$,
et toute clé de R_{ij} est formée de l'union d'une clé de R_i et d'une clé de R_j.

Il est remplacé par une arête u" = (N_i,N_j), kr(u") = 0 et gr(u") = R_{ij} reliant les noeuds N_i et N_j; les arcs reliant N_{ij} à N_i et à N_j sont supprimés.

Voici un exemple qui montre le rôle de cette étape.

Soit la relation **RE(ABXY)** qui se décompose de manière compacte suivant les relations:

R_1(*AB*) de clé AB, **R_2(*A*X)** de clé A, **R_3(*B*Y)** de clé B.

Voici le graphe de relation obtenu après les cinq premières étapes:

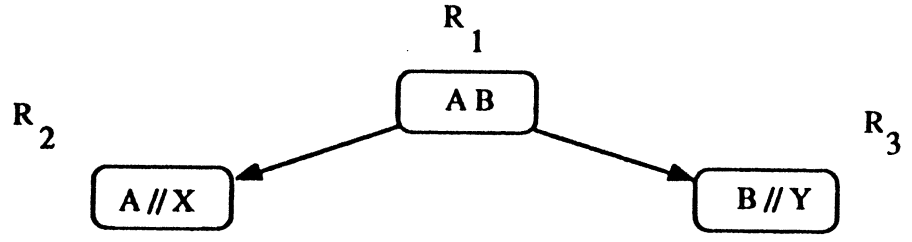

Figure 4.13. Situation de B-arête

La sixième étape considère qu'en fait une entité de R_1 permet de faire une association entre une entité de R_2 et une entité de R_3.
Une manière d'implanter une telle association (comme nous l'étudions

dans le chapitre 5) est justement de stocker les entités de R_1. Mais il existe d'autres possibilités.
Aussi pour préserver les choix d'implantation, cette sixième étape transforme-t-elle ce graphe de relation en:

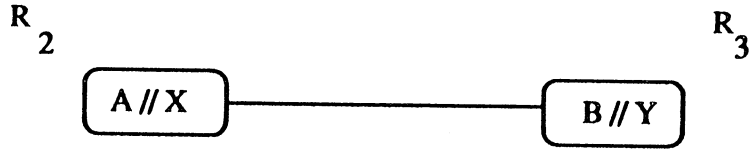

Figure 4.14. B-arête

5

D'UN GRAPHE DE RELATION À UNE STRUCTURE INFORMATIQUE DE DONNÉES

Le but de ce chapitre est de proposer la transformation d'un graphe de relation dans une structure informatique de données.
Nous allons d'abord le transformer dans un graphe de chemins d'accès (GCA) brut à l'aide de l'algorithme très simple décrit précédemment. Il s'agit ensuite d'analyser ce graphe de chemins d'accès à l'aide des résultats des analyses des traitements. Les problèmes abordés alors sont notamment de cette nature:
a) tous les chemins d'accès sont-ils utiles ?
b) quels sont les index qu'il faut rajouter?
c) quelle est la redondance qu'il est important d'introduire?
d) quelles sont les relations associées à des noeuds qu'il faut mieux décomposer en plusieurs relations de mêmes clés ou de clés équivalentes?

Nous dégageons des résultats importants de l'analyse des traitements pour la définition d'une structure informatique à partir d'un GCA brut.

Les réponses à ces questions dépendent également du système de gestion de bases de données (SGBD) utilisé; certains implantent tous les chemins d'accès réciproques automatiquement. Alors le concepteur n'a pas à se soucier de la question a). Ces SGBD travaillent directement à partir du graphe de chemins d'accès brut.
Le concepteur précise les noms des chemins d'accès finalement retenus et les paramètres associés.

Ensuite il s'agit de transformer le GCA choisi dans une structure informatique écrite dans les termes du SGBD choisi. Nous proposons une étude plus générale indépendante d'un SGBD qui donne des éléments de réponse à la question suivante:
e) comment implanter les chemins d'accès ?
Certains SGBD imposent leur propre méthode de stockage des chemins d'accès et alors le concepteur de la base de données n'a pas à se soucier de cette question.

Même si certaines de ces réflexions peuvent apparaître inutiles pour certains SGBD, elles peuvent au moins conduire à estimer la qualité de l'intelligence artificielle mise dans les SGBD.

Cette étude n'est pas exhaustive notamment à cause de la grande variété des méthodes d'implantation possibles. (HAINAUT86) (FINKELSTEIN SCHKOLNICK TIBERIO88) présentent des études plus détaillées de cette partie. Elle propose une transformation fidèle au sens du § 3.2. du graphe de relation initial dans une structure informatique de données.

5.1. Choix d'un graphe de chemins d'accès

Toute la démarche précédente (§ 4.) a conduit à l'élaboration d'un graphe de chemins d'accès brut. Nous allons maintenant tenter de fournir un guide à un concepteur pour choisir la structure informatique de données la mieux adaptée à la base de données dont il a la responsabilité. Comme nous l'avons déjà remarqué, certains SGBD ne donnent que peu de choix au concepteur à ce niveau. Sans prendre position sur le bien fondé d'une telle approche ou de l'approche contraire (laisser au concepteur tous les choix possibles), nous continuons notre travail sans nous soucier de cette réalité, et sans pour autant la mésestimer, mais simplement pour tenter d'éclairer les problèmes liés à cette étape.

Pour nous les choix possibles du concepteur à ce niveau concernent:

- la possibilité de ne pas implanter des chemins d'accès réciproques correspondant à la relation T:

l'étude de l'inutilité de l'un des deux chemins d'accès réciproques trouve son intérêt dans la redondance occasionnée par f_1 et f_2; pour que l'entité

(ab) de T soit stockée dans la base, il faut que $a \in f_2(b)$ et $b \in f_1(a)$. Toute mise à jour d'une entité de T devra être répercutée sur ces deux chemins d'accès. Aussi autant pour économiser de la place que pour des soucis d'efficacité, est-il avantageux d'envisager de ne pas retenir l'un ou l'autre de ces chemins d'accès.
Si un chemin d'accès est monovalué, et compte tenu de la facilité d'implanter un tel chemin d'accès, il nous apparaît comme tatillon de s'intéresser à son inutilité éventuelle (aux cas particuliers près qui bien sûr confirmeront la règle !!). Reste le cas des chemins d'accès multivalués. L'étude du paragraphe suivant cerne les conditions dans lesquelles on peut les considérer comme inutiles.

MAIS, dans une frénésie toujours possible de suppressions de chemins d'accès multivalués, le concepteur doit veiller à maintenir un écoulement dans le GCA pour chaque espace, pour préserver l'accès aux entités de l'espace. Ainsi le GCA reste fidèle (§ 3.2.) à la modélisation initiale.

- le choix des relations d'entrée:

le fait que la relation R est une relation d'entrée signifie qu'il sera possible d'atteindre directement une entité r de iR en connaissant la valeur qu'elle prend pour une clé. Généralement pour rendre possible un tel accès, il faut utiliser des techniques plus ou moins sophistiquées (DELOBEL-ADIBA82) comme le séquentiel indexé, le hash-code, les B-arbres... Maintenant si l'utilisation de la base ne s'intéresse aux entités de iR qu'en les balayant toutes en séquentiel ou alors en les atteignant à partir des entités d'autres relations via des chemins d'accès, pourquoi conserver un tel accès coûteux pour R ? Dans ce cas le concepteur ne considère plus R comme relation d'entrée.

- l'inversion d'une relation d'un noeud ou index d'une relation:

c'est une situation très classique. Notre approche a beaucoup privilégié l'accès aux entités d'une relation R(*A*MNO) en connaissant les valeurs qu'elles prennent pour une clé. Mais il est bien sûr classique dans l'utilisation d'une base de données de vouloir accéder à des entités de R en connaissant les valeurs qu'elles prennent pour des constituants ne formant pas une clé de R.
Nous représentons ceci dans un GCA, par la création d'un nouveau noeud associé aux constituants sur lesquels porte l'inversion de la relation R; ce noeud est relié au point d'entrée; il est relié au noeud de R par un chemin d'accès monovalué partant du noeud de R, et d'un chemin d'accès multivalué réciproque.

Transformation de structures

Exemple d'inversion de R par MN:

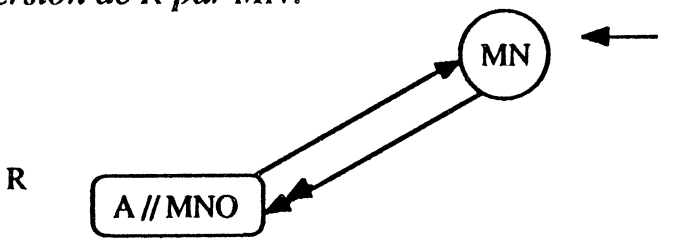

Figure 5.1. Inversion d'une relation.

MN forme alors un ***index*** de la relation R.

- *le rajout de redondance:*

nous étudions dans la troisième partie comment éviter la redondance dans les bases de données. Il peut paraître paradoxal d'étudier ici comment introduire de la redondance. La grande différence entre les deux parties provient de la position du concepteur. Dans la troisième partie nous cherchons à guider le concepteur vers un graphe de relation contenant le minimum de redondance; surtout nous cherchons à éviter des situations où le concepteur ne s'apercevrait pas de la redondance contenue dans le graphe de relation.
Ici le concepteur introduit de la redondance en connaissance de cause. Il sait qu'à toute redondance doivent être associés des contrôles de cohérence coûteux en temps d'exécution et de mise en place, mais qu'en contre partie une redondance judicieusement placée peut rendre beaucoup plus efficace l'exécution de traitements sur la base de données.
C'est à lui de prendre la décision en se rappelant que la redondance est pénalisante dans l'exécution de traitements seulement si elle porte sur des informations souvent modifiées.

Ce rajout de redondance se traduit en termes de GCA surtout de deux manières illustrées dans la figure suivante:
- rajout d'un (ou plusieurs) constituant C à une relation R (*A* B X) de clé A; pour que cette transformation maintienne la fidélité du GCA au graphe de relation initial, il faut qu'il existe la df A $\rightarrow$ C;
- rajout d'une paire de chemins d'accès réciproques dans le GCA, entre deux relations R et S dont les noeuds sont reliés par un chemin ne se résumant pas à un seul arc. Par exemple ce chemin est composé de deux arcs (=chemins d'accès) et passe par le noeud de la relation T: le premier arc est associé à la relation RT, le second à TS; alors la paire de chemins d'accès rajoutée correspond à la relation (RT * TS) [$KR^+ \cup KS^+$], KR^+ KS^+ étant les constituants clés de R et S.

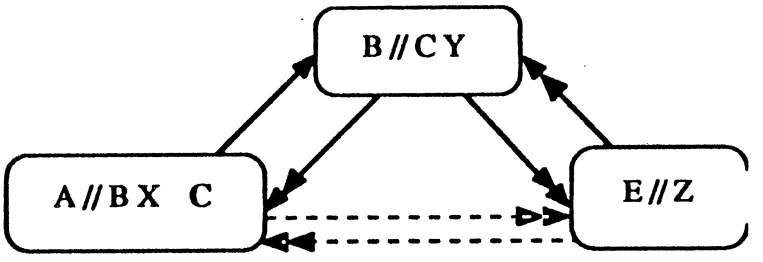

Figure 5.2. Redondance.

Dans cette figure nous avons mis en pointillé la redondance introduite en termes de chemins d'accès et en caractères gras celle introduite par un constituant.

- le dédoublement de noeuds:

soit une relation R(*A*MNOP) à laquelle correspond un noeud dans le GCA. Le concepteur peut être amené à décomposer cette relation R en deux (ou plusieurs) relations R_1 et R_2 ayant des clés équivalentes à celles de R, un constituant non clé de R ne se trouvant que dans une seule de ces relations par exemple R_1 (*A*MN) et R_2 (*A*OP). Le GCA devient alors:

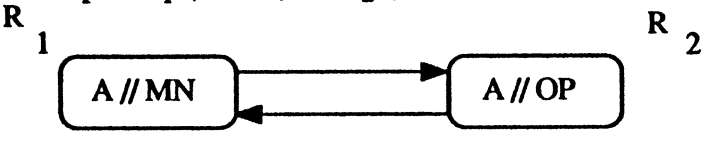

Figure 5.3. Dédoublement de noeuds.

Le noeud de R disparaît et est remplacé par deux noeuds associés aux relations R_1 et R_2 et reliés entre eux par deux chemins d'accès monovalués réciproques.
La question se pose concernant les chemins d'accès qui partaient du noeud de R ou qui y arrivaient. Une réponse possible est de choisir l'une des deux relations, R_1 par exemple, et d'accrocher au noeud de R_1 tous ces chemins d'accès. Cette méthode brutale peut être tempérée par une méthode plus souple qui conduit le concepteur à choisir entre les noeuds de R_1 et de R_2, pour chacun de ces chemins d'accès, le noeud auquel il va l'accrocher.

Pour quelles raisons un concepteur peut-il envisager de dédoubler un noeud ? En voici deux.
La première concerne la taille informatique d'une entité de R. Plus elle est grande, moins d'entités de R seront amenées par une seule opération informatique de lecture en mémoire centrale. Or certains constituants de R peuvent avoir des domaines demandant beaucoup de places

informatiques (une adresse postale par exemple) alors que d'autres en demandent très peu (un chiffre d'affaires par exemple); si de plus les traitements qui travaillent sur les uns sont très souvent différents de ceux qui travaillent sur les autres, il est avantageux de séparer les constituants.

Exemple:
CLIENT(NOCLIENT CAMENSUEL NBCOMMANDES ADRESSE NOM).
ADRESSE et NOM de CLIENT servent pour des traitements qui expédient du courrier aux clients; ces traitements ne sont pas ceux qui travaillent sur le nombre de commandes du client (NBCOMMANDES) et leurs chiffres d'affaires mensuels (CAMENSUEL). Le concepteur a sans doute intérêt dans cet exemple à décomposer la relation CLIENT en (NOCLIENT CAMENSUEL NBCOMMANDES) et (NOCLIENT ADRESSE NOM).
fex.

La deuxième raison concerne la concurrence d'accès; en reprenant l'exemple précédent si deux traitements T_1 et T_2 ont pour but respectivement l'un de modifier le CAMENSUEL du client 150386 et l'autre de modifier l'adresse du même client, ils vont rentrer en concurrence si la relation CLIENT n'est pas décomposée: s'ils s'exécutent en même temps l'un doit laisser passer l'autre. En fait logiquement ils ne se font pas concurrence. La décomposition précédente de CLIENT leur permet de s'exécuter sans se faire de concurrence.

5.2. Méthodes informatiques d'implantation des chemins d'accès

Nous examinons trois types d'arêtes d'un graphe de relation:

- *D*-arête : arc orienté entre les noeuds des relations R et S, quand KR^+, ensemble des constituants clés de R, contient KS^+ : on dit alors que R est *dé*pendante de S.

- *F*-arête : arc orienté entre les noeuds des relations R et S, R n'étant pas dépendante de S. (Il existe la dépendance fonctionnelle $KR \rightarrow KS$).

- *B*-arête : arête entre les noeuds des relations R et S (arête binaire).

Nous allons présenter les opérations élémentaires de manipulation de données relatives autant aux chemins d'accès qu'aux instances des relations associées aux noeuds; les opérations élémentaires seront affinées suivant le type de l'arête concernée.
Ensuite pour chaque type d'arête, nous rappelons brièvement différentes méthodes d'implantation des chemins d'accès utilisées dans les SGBD; nous comparons leur efficacité dans le cas de chacune des opérations élémentaires, pour dégager un algorithme de choix entre ces méthodes.
En dernier lieu, nous déduisons de nos algorithmes de choix, des algorithmes de mesures: dans la phase de conception d'une base de données, il est en effet important de savoir quels sont les paramètres qu'il est utile d'estimer.

Pour nous les choix possibles du concepteur à ce niveau concernent notamment la possibilité de ne pas implanter deux chemins d'accès réciproques. Soit les trois relations R(*A*X), S(*B*Y), T(*AB*) et le graphe de chemins d'accès suivant:

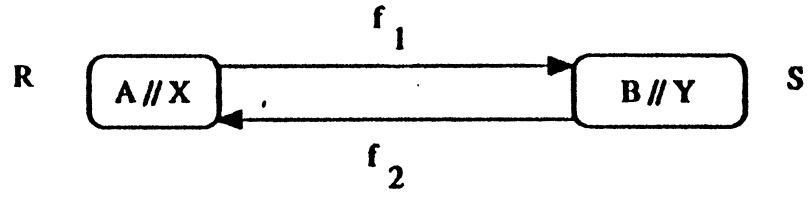

Figure 5.4. GCA élémentaire.

f_1 et f_2 sont les chemins d'accès correspondant à la relation T. L'étude de l'inutilité de l'un de ces deux chemins d'accès trouve son intérêt dans la redondance occasionnée par f_1 et f_2; pour que l'entité (ab) de T soit stockée dans la base, il faut que $a \in f_2(b)$ *et* $b \in f_1(a)$. Toute mise à jour d'une entité de T devra être répercutée sur ces deux chemins d'accès. Aussi pour des raisons autant d'économies de place que d'efficacité est-il avantageux de ne retenir que l'un ou l'autre de ces chemins d'accès.

5.2.1. Opérations atomiques de manipulation de données

Toutes les opérations que nous allons présenter, sont atomiques dans le sens où aucune d'elles ne peut être exécutée de manière informatique à l'aide d'autres opérations, avec des performances comparables.
Pour les présenter, nous utilisons les trois relations précédentes R(*A*X), S(*B*Y) et T(*AB*) et le graphe précédent de chemins d'accès.

5.2.1.1. Opérations atomiques relatives aux chemins d'accès

Lier: cette opération est notée :∈ dans (ABRIAL74). r:∈ $f_2(s)$ ou a:∈ $f_2(b)$ ou (a,b):∈ iT par exemple. Cette opération permet la création d'entités dans iT. Elle vérifie les règles d'intégrité qui la contiennent dans leur portée, vérifie que les entités r èt s existent dans iR et iS, et qu'elles ne sont déjà pas associées entre elles, exécute l'opération symétrique: s:∈ $f_1(r)$ (quand f_1 existe).

Délier: cette opération est notée :∉ dans (ABRIAL74): r :∉ $f_2(s)$ ou a :∉ $f_2(b)$ ou (a,b) :∉ iT. Elle permet la suppression d'entités dans iT. Elle contient la vérification des règles d'intégrité qui la contiennent dans leur portée, la vérification de l'existence des entités r et s dans iR et iS et de l'association entre elles, l'opération symétrique s :∉ $f_1(r)$ (quand f_1 existe).

Accéder: $f_2(s)$ permet l'accès à toutes les entités r de iR, reliées à l'entité s de iS par le chemin d'accès f_2. Elle vérifie l'existence de s dans iS. Nous notons également $f_2(b)$.

Modifier: r :∉ $f_2(s)$ et r:∈ f_2 (s_1) où s_1 est une entité de iS distincte de s. Nous notons également cette opération a :∉ $f_2(b)$ et a :∈ $f_2(b_1)$. Cette opération apparaît comme identique à la succession des deux opérations r :∉ $f_2(s)$ puis r :∈ $f_2(s_1)$; pourtant elle ne leur est pas équivalente car elle nécessite moins de contrôles de validation.
Une autre raison provient des règles d'intégrité qui la contiennent dans leur portée. L'état de la base après l'opération de délier peut être incohérent sans que l'état de la base après les deux opérations ne le soit.
Cette opération vérifie les règles d'intégrité qui la contiennent dans leur portée, vérifie que les entités r, s, s_1 et l'association entre les entités r et s existent et qu'aucune association entre r et s_1 n'existe déjà, et exécute l'opération symétrique: s :∉ $f_1(r)$ et s_1:∈ $f_1(r)$ quand f_1 existe.

5.2.1.2. Opérations atomiques relatives aux relations de noeuds

Créer: cette opération permet la création d'une nouvelle entité dans une instance d'une relation associée à un noeud (par exemple R). Nous la notons r :∈ iR.
Cette opération valide les règles d'intégrité qui la contiennent dans leur portée, et vérifie que r n'existe pas déjà dans iR; elle valide éventuellement les règles de dépendance si R est une relation dépendante d'une autre.

Exemple: soit les deux relations R(*AX*) et V(*ACZ*), V dépendante de R. (acz):∈ iV ne peut s'exécuter que s'il existe une entité r dans iR telle que r.A=a.

Supprimer: cette opération permet la suppression d'une entité d'une instance d'une relation associée à un noeud (par exemple R).
Nous la notons **r :∉ iR.** Cette opération:
- vérifie que r existe dans iR,
- valide les règles d'intégrité qui la contiennent dans leur portée,
- valide éventuellement les règles de dépendance s'il y a des relations dépendantes de R,
- et exécute les opérations de delier pour tous les chemins d'accès partant de r ou arrivant à r.

Exemple: soit le graphe de chemins d'accès suivant où le point d'entrée n'est pas indiqué ne jouant aucun rôle:

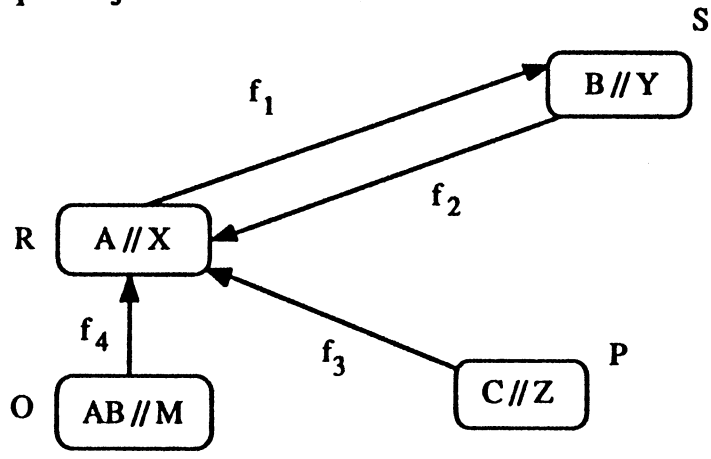

Figure 5.5. Suppression d'une entité.

O est une relation dépendante de R.
r=(a,x) :∉ iR contient les opérations suivantes après la validation de l'existence de r dans iR:
pour tout s ∈ f_1(r) *faire* r :∉ f_2(s) *refaire*
pour tout p de iP *faire*
    si r ∈ f_3(p) *alors* r :∉ f_3(p) *finsi*
refaire
pour tout o de iO *tel que* o.A = a *faire* o :∉ iO *refaire.*

Accéder: cette opération permet d'accéder à toutes les entités de iR par un parcours séquentiel; on note **prendre iR, r:** la nouvelle entité lue est stockée dans r, variable qui a été déclarée de type R.
Si dans le GCA, le noeud de relation R est relié au point d'entrée par une

arête, alors il est possible d'accéder à une entité de iR en connaissant la valeur qu'elle prend pour une clé KR; on note **r := R[KR=a].**

Modifier: il s'agit de remplacer une entité r de iR par une nouvelle r' telle que pour toute clé obligatoire KR de R, r.KR=r'.KR. Cette opération vérifie que r appartient à iR et que r' n'y appartient pas, valide les règles d'intégrité qui la contiennent dans leur portée. On note **r :∉ iR et r' :∈ iR.** Ici également, comme pour l'opération modifier relative aux chemins d'accès et pour le mêmes raisons, cette opération n'est pas identique à la séquence d'opérations r :∉ iR puis r':∈ iR.

5.2.2. Implantation des chemins d'accès correspondant à une arête de type F

5.2.2.a. Introduction

Soit les relations R(*A* X) S(*B* Y) et T(*A* B) et leur graphe de relation:

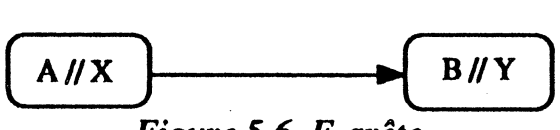

Figure 5.6. F-arête.

(où B n'est pas inscrit dans le noeud de R comme le permet l'étape E5 de l'algorithme du chapitre précédent), et leur GCA:

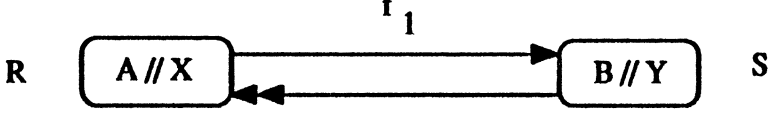

Figure 5.7. F-arête dans GCA.

Nous notons:

nR	le nombre d'entités de iR,
nS	le nombre d'entités de iS,
q	la probabilité qu'il existe au moins une entité de iR associée à une entité de iS,
p	la probabilité qu'il existe au moins une entité de iS associée à une entité de iR,
nBA	le nombre moyen d'entités de iR associées à une entité de iS quand il en existe au moins une,
nRmax	le nombre maximal d'entités de iR,
nSmax	le nombre maximal d'entités de iS,
nBAmax	le nombre maximum d'entités de iR associées à une entité de iS.

5.2.2.b. Méthodes informatiques pour implanter f_1 et f_2

Parmi les méthodes informatiques souvent utilisées dans les SGBD (notamment obéissant aux normes CODASYL) nous considérons:
pour f_1:
- *le pointeur* (1): on crée un nouveau constituant de R: Ref, dont le domaine est l'ensemble des adresses des entités de S: $\forall$ r $\in$ R, r.Ref désigne l'adresse de l'entité s de S telle que (r.A, s.B) $\in$ T.
- *le dédoublement* de B: on rajoute B à l'espace des constituants de R. Ainsi chaque entité de R contient la clé de l'entité correspondante de S.

pour f_2:
- *la liste (1)*: R(*A* X Suiv), S(*B* Y Tlist).
$\forall$ s $\in$ S s.Tlist indique la première entité de R correspondant à s,
$\forall$ r $\in$ R r.Suiv indique l'entité de R qui suit r dans la liste de s.

- *La liste hétérogène circulaire (2)*: R(*A* X Suiv Parent), S(*B* Y Tlist).
C'est une liste dont, en plus, la dernière entité pointe sur l'entité origine de la liste(Parent).

- *La liste doublée (3)*: R(*A* X Suiv Pre), S(*B* Y Tlist Finlist).
C'est une liste dont chaque entité a non seulement un pointeur indiquant la suivante (Suiv), mais également un autre indiquant la précédente (Pre). De plus $\forall$ s $\in$ S s.Finlist indique la dernière entité de la liste de s.

- *La liste hétérogène doublée (4)*:
R(*A* X Suiv Pre Parent), S(*B* Y Tlist Finlist).
C'est à la fois une liste doublée et une liste hétérogène circulaire (cf. un exemple en annexe § 5.4.1.).

- *Le tableau (5)*: R(*A* X), S(*B* Y PT).
PT est en fait un tableau de pointeurs; s.PT(i) indique l'adresse d'une entité r de R telle que (r.A, s.B) $\in$ T.
Il existe deux sortes d'implantation de tableaux: les tableaux de taille fixe et ceux avec zone de débordement (cf. annexe § 5.4.2).

- *L'inversion* de la relation R par B.
On construit un index B de la relation R(*A* B X); cette méthode est surtout intéressante si cet index a déjà été créé dans la phase précédente (§ 5.1.). Pour les comparaisons avec les autres méthodes nous ne l'avons pas retenue car elle est difficilement comparable; les opérations logiques d'E/S qu' elle demande, concerne autant des accès supplémentaires aux index que des accès aux entités des relations.

Du point de vue informatique, on peut classer ces méthodes suivant la facilité de les mettre en oeuvre, les premières étant les plus faciles:
- liste et tableau de taille fixe,
- liste doublée et liste hétérogène circulaire,
- tableau avec zone de débordement.

Ainsi pour implanter T, peut-on:
- implanter f_1 sans implanter f_2 (10);
- implanter f_2 sans implanter f_1 par liste (01), liste hétérogène circulaire (02), liste doublée (03), liste doublée hétérogène circulaire (04), tableau (05);
- implanter f_1 et f_2:
solution 11: f_2 est implantée à l'aide d'une liste;
solution 13: f_2 est implantée à l'aide d'une liste doublée;
solution 15: f_2 est implantée à l'aide d'un tableau.
Les solutions 12 (f_2 implantée à l'aide d'une liste hétérogène circulaire) et 14 (f_2 implantée à l'aide d'une liste hétérogène circulaire doublée) sont identiques respectivement à 11 et 13.

5.2.2.c. Comparaison de ces méthodes suivant la place occupée

Numéro de la solution	*Place en unité de pointeurs*
10	nR
01	nR + nS
02	2nR + nS (évent. nR + nS)
03	2nR + nS
04	3nR + nS (évent. 2nR + nS)
05	nBAmax.nS
11	2nR + nS
13	3nR + nS
15	nR + nBAmax nS

5.2.2.d. Calcul du nombre d'opérations d'entrée-sortie

Les opérations (a,x) :∈ iR ou de (b,y) :∈ iS ne jouent aucun rôle dans notre étude. Par contre nous affinons les opérations de:

- **Délier:** non seulement nous considérons s :∉ f_1(r) (ou (a,b) :∉ iT) mais également ∅ :∈ f_1(r) (ou (a,-) :∉ iT); dans le premier cas, l'état de la base contient une association entre r et s qui doit être détruite; dans le second cas, l'état de la base contient une association entre r et une entité non précisée de iS, qui doit être détruite.

Par exemple R(*A*X) désigne la relation **VOITURE**(NOVOITURE KM)
et S(*B*Y) la relation **PERSONNE**(NOM ADRESSE)
et T(*A*B) la relation **PROPRIÉTAIRE**(NOVOITURE NOM);
faire que la voiture de numéro 4806HN46 n'ait plus de propriétaire, peut demander de connaître son propriétaire: (4806HN46, Marilyne) :∉ iPROPRIÉTAIRE, ou non: (4806HN46,-) :∉ iPROPRIÉTAIRE.

- **Modifier** : la df A → B rompt la symétrie entre les relations R et S. On peut observer en programmant les algorithmes que d'une part (a,b) :∉ iT *et* (a,b_1) :∈ iT et d'autre part (a,b) :∉ iT *et* (a_1,b) :∈ iT sont deux opérations aux performances différentes. Comme précédemment il existe une variante à la première opération qui est: (a,-) :∉ iT *et* (a,b_1) :∈ iT.

Rapidement voici trois exemples avec les mêmes relations:
(4806 HN46, Marilyne) :∉ iPROPRIÉTAIRE *et* (4806HN46, Marion) :∈ iPROPRIÉTAIRE; cette opération change le propriétaire de la voiture de numéro 4806HN46: il était Marilyne, il devient Marion.
(4806HN46,-) :∉ iPROPRIÉTAIRE et (4806HN46, Marion):∈ iPROPRIÉTAIRE; la voiture 4806HN46 avait un propriétaire, elle en change et son nouveau est Marion.
(4806HN46, Marilyne) :∉ iPROPRIÉTAIRE et (GE101437, Marilyne) :∈ iPROPRIÉTAIRE. Marilyne n'est plus propriétaire de la voiture 4806HN46 mais de la voiture GE101437.

Pour le calcul de l'efficacité des différentes opérations dans le cadre des 9 méthodes d'implantation retenues, nous avons dénombré le nombre d'opérations logiques de lecture et d'écriture faites pour les instances iR et iS en sachant qu'une opération de modification d'une entité compte pour une seule opération.
Les tableaux considérés sont de taille fixe.
Le nombre d'opérations d'E/S pour chacune des neuf solutions et pour chacune des opérations se trouve en fin de ce chapitre (§ 5.4.4.).

Une solution s_1 est dite supérieure à une solution s_2 si pour toutes les opérations élémentaires, le nombre d'E/S nécessaires pour exécuter s_1 est inférieur à celui de s_2. Cette relation est une relation d'ordre partiel dans l'ensemble des solutions.

Nos résultats montrent que:

$15 > 13 > 04 > 03$,
$15 > 11 > 03 > 02$,
15, 05 et 10 ne pouvant être comparées.

5.2.2.e. Algorithme de choix que nous proposons

Le résultat précédent ne veut pas dire que le choix se limite aux seules solutions 15, 05, 10. En effet l'implantation de f_2 par un tableau de dimension fixe n'étant pas toujours possible, il faut alors choisir une autre solution.
Le choix entre ces solutions est commandé par les fréquences des opérations élémentaires déclenchées par les traitements prévus. Dans certains cas, certaines solutions sont équivalentes: on choisit alors la moins complexe à implanter.
Nous notons la fréquence d'une opération ainsi: F <nom de l'opération> <argument>.
Algorithme simplifié de choix:
si Fsupprimer (b,y) ≅ 0 et Ff_2(b) ≅ 0 *alors* choisir 10
sinon *si* la solution du tableau est possible alors choisir 15
sinon *si* nBA > 7 *alors* choisir 13
sinon choisir 11
finsi
finsi
finsi.

5.2.2.f. Algorithme de prise de mesures

Il s'agit de déduire de l'algorithme de choix précédent celui de la prise de mesures: en effet, il n'est pas toujours nécessaire de mesurer ou d'estimer toutes les fréquences et tous les paramètres (nAB...).
Algorithme:
si Fsupprimer (b,y) ≅ 0 et si Ff_2(b) ≅ 0 *alors*
mesurer nBA, q, nBAmax, nRmax, nR
finsi.

5.2.3. Implantation des chemins d'accès correspondant à une arête de type D

5.2.3.a. Introduction

Soit les relations R(*A* X) S(*A B* Y) et le graphe de relation:

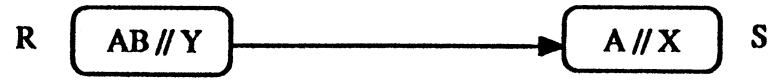

Figure 5.8. D-arête.

et le graphe de chemins d'accès:

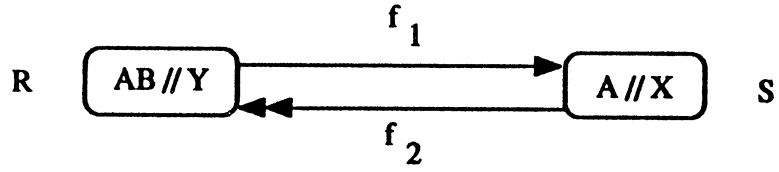

Figure 5.9. D-arête dans un GCA.

Voici les paramètres utilisés:

nR	nombre d'entités de iR,
nS	nombre d'entités de iS,
q	probabilité pour qu'il existe au moins une entité de iR associée à une entité de iS,
nSR	nombre moyen d'entités de iR associées à une entité de iS quand il en existe au moins une,
nSRmax	nombre maximum d'entités de iR associées à une entité de iS.

5.2.3.b. Méthodes pour implanter informatiquement f_1 et f_2

Parmi elles nous considérons:

pour f_1:

l'accès par une valeur du domaine de A ou par un pointeur, dont le domaine est l'ensemble des adresses des entités de S et qui remplace A dans R.

pour f_2:

- les mêmes solutions que dans les cas précédents s'appliquent, puisqu'il existe la dépendance fonctionnelle, AB → A;

- la contiguïté séquentielle (06): il s'agit de considérer un seul fichier qui contient deux sortes d'enregistrements correspondant aux entités de iR et aux entités de iS. Les entités de iR relatives au même objet *a* de A sont rangées en séquence après l'entité de iS relative à *a*. La codification des valeurs des clés A et A,B doit permettre de faire cette différence.
Si l'on connaît le nombre maximum nSRmax d'entités de R correspondant à une entité de S, cette méthode est la moins complexe à implanter de toutes les autres solutions car elle ne nécessite aucun pointeur; sinon elle nécessite la gestion d'une zone de débordement et rejoint en complexité la solution de tableau avec zone de débordement.

5.2.3.c. Comparaison de ces méthodes suivant la place

Les seules solutions à comparer sont 10, 11, 13, 15 et 16 (où f_1 est implantée et f_2 également suivant 06) car
- f_1 est toujours implantée,
- donc 12 et 14 sont respectivement équivalentes à 11 et 13 du point de vue des opérations élémentaires, mais avec une complexité plus grande pour l'implantation.

Numéro de la solution	*Place en unités de pointeurs*
10	---
11	nR + nS
13	2nR + nS
15	nSRmax.nS*
16	(nSRmax-qnR) nS LGR**

* dans le cas de tableau de dimension fixe.
** c'est la place perdue par cette solution; LGR désigne la longueur d'une entité de R en unité de pointeurs.

5.2.3.d. Calcul du nombre d'opérations

Nous avons calculé le nombre d'opération E/S nécessaires pour réaliser les opérations atomiques avec les mêmes conventions qu'au paragraphe précédent. Les opérations atomiques, lier et délier, sont enclenchées par les opérations effectuées sur les entités des instances iR et iS puisque R est dépendante de S.
La solution 16 ne peut être comparée valablement aux autres par notre méthode car les E/S se font surtout par des parcours séquentiels.

Le nombre d'opérations d'E/S exécutées par les opérations atomiques pour les différentes solutions se trouve en fin de chapitre (§ 5.4.4.).

5.2.3.e. Algorithme de choix que nous proposons

Nous utilisons les mêmes conventions que précédemment.

Si $Ff_2(a) \cong 0$ et Fsupprimer(a,x) $\cong 0$, *alors* choisir 10
sinon si la solution du tableau est possible
    alors choisir 15
    sinon choisir 13
    finsi
finsi.

5.2.3.f. Algorithme de prise de mesures

Si $Ff_2(a) \cong 0$ et *si* Fsupprimer(a,x) $\cong 0$ *alors*
mesurer nSR,q,nR,nSRmax
finsi.

5.2.4. Implantation des chemins d'accès correspondant à une arête de type B

5.2.4.a. Introduction

Soit les relations R(*A* X) S(*B* Y) et T(*A B*) et leur graphe de relation:

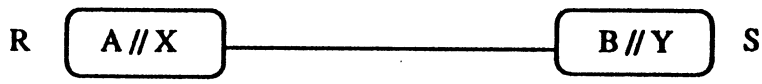

Figure 5.10. B-arête.

et leur graphe de chemins d'accès:

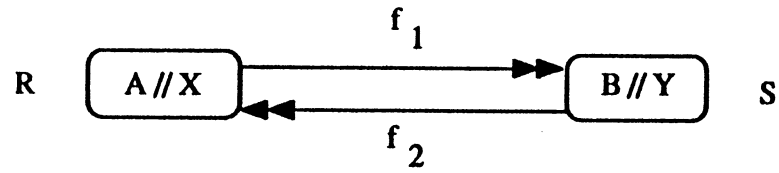

Figure 5.11. B-arête dans un GCA.

Nous notons:

nR le nombre d'entités de iR,
nS le nombre d'entités de iS,
p la probabilité qu'il existe au moins une entité t de iT telle que t.A=a,
q la probabilité qu'il existe au moins une entité t de iT telle que t.B=b,
nAB le nombre moyen d'entités de iT telles que t.A=a sachant qu'il en existe au moins une,
nBA le nombre moyen d'entités de iT telles que t.B=b sachant qu'il en existe au moins une.

5.2.4.b. Méthodes informatiques pour implanter f_1 et f_2

f_1 et f_2 jouent des rôles symétriques: toute méthode applicable pour l'une est valable pour l'autre. Parmi les méthodes possibles, nous considérons:

- *La table de correspondance*:
Il s'agit de créer une autre relation TABLE-R-S dont les constituants sont seulement des pointeurs et dont les entités font la correspondance entre les entités de R et celles de S. TABLE-R-S est dépendante de R et de S. Cette méthode est celle utilisée notamment dans les SGBD obéissant aux normes CODASYL. Les deux chemins d'accès f_1 et f_2 sont alors implantés:

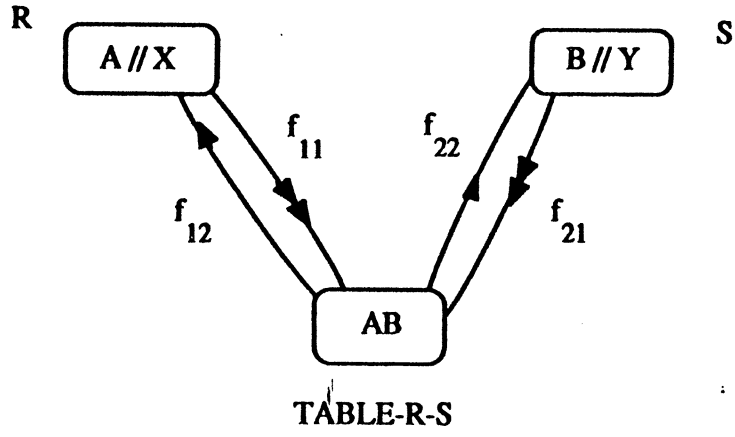

Figure 5.12. Table de correspondance.

Il s'agit d'implanter quatre chemins d'accès relatifs aux deux arêtes de dépendance. Parmi les solutions du paragraphe précédent, c'est la solution 13 qui est retenue:
- elle est plus efficace que 11 dans la plupart des cas,
- 16 ne peut être appliquée pour les deux arêtes de dépendance,
- 15 donne lieu à une implantation plus directe de f_1 et de f_2 par tableau et qui est étudiée ensuite.

Nous considérons donc les relations:
R(*A* X Atête),
S(*B* Y Btête),
TABLE-R-S(*NO* Asuiv Apre Aparent Bsuiv Bpre Bparent) où:
NO est un numéro discriminant des entités de TABLE-R-S,
Atête,Asuiv,Apre servent à implanter f_{11} (liste doublée),
Btête,Bsuiv,Bpre servent à implanter f_{21} (liste doublée),
Aparent sert à implanter f_{12} (pointeur),
Bparent sert à implanter f_{22} (pointeur).

- *le tableau:*
si l'on implante f_1 par tableau, les relations deviennent:
R(*A* X PTA) et S(*B* Y).où PTA est en fait un tableau de pointeurs.
r.PTA(i) indique l'adresse d'une entité de S, telle que (r.A, s.B) ∈ iT.

Il existe deux sortes de tableaux: ceux de taille fixe et ceux avec zone de débordement (cf. annexe § 5.4.2.).

- *la multiliste:*
si on implante f_1 par multiliste, les relations deviennent:
R(*A* X Atête Atpos) et S(*B* Y Asuiv Apos) où r.Atête donne l'adresse de la première entité s de S de la liste de r.A;
mais comme une entité s peut appartenir à plusieurs listes, il faut également indiquer dans r.Atpos, la position i de la liste de r.A dans les tableaux Asuiv et Apos: s.Asuiv(i) indique l'adresse de l'entité s' suivante dans la liste; s.Apos(i) indique la position de cette liste dans les tableaux Asuiv, Apos de s'.
Si le nombre d'objets *a* de A correspondant à un objet *b* de B n'a pas de maximum, alors il faut prévoir une zone de débordement comme dans le cas d'un tableau. Comme le tableau est une solution plus efficace et plus simple, la solution multiliste ne peut être retenue dans ce cas. Aussi considérons nous seulement les multilistes de taille fixe, c'est-à-dire dont les tableaux Asuiv et Apos sont de taille fixe (cf. annexe §5.4.3.).

- *le tableau (f_1) et la multiliste (f_2) (ou inversement):*
les relations deviennent: R(*A* X PTA Bsuiv) et S(*B* Y Btête).
On peut convenir alors, que:
∀ r ∈ R si r.PTA(j) = b alors r.Bsuiv(j) est relatif à la liste de b.

- Finalement pour implanter T on peut choisir entre:
la table de correspondance (33),
implanter f_1 et f_2 avec des tableaux (11),
implanter f_1 avec un tableau, f_2 avec une multiliste (12),
et inversement (21),
implanter f_1 avec un tableau sans implanter f_2 (10),
et inversement (01),
implanter f_1 avec une multiliste sans implanter f_2 (20),
et inversement (02).

La solution consistant à implanter f_1 et f_2 à l'aide de multilistes ne peut pas être retenue: elle suppose en effet que nABmax et nBAmax soient connues; mais alors 11 est plus simple à mettre en oeuvre et conduit à un nombre d'opérations d'entrée/sortie moins important.

Du point de vue informatique, on peut classer ces méthodes suivant la facilité de leur mise en oeuvre:

10 et 01	un tableau de taille fixe,
11	avec tableaux de tailles fixes,
20 et 02	une multiliste,
12 et 21	un tableau de taille fixe et une multiliste de taille fixe,
33	table de correspondance,
10, 01, 11	tableau de taille variable.

5.2.4.c. Comparaison de ces méthodes suivant la place

Les tableaux et les multilistes sont considérés de taille fixe.

Numéro de la solution	*Place en unité de pointeurs*
33	nR+nS+6pnR.nAB
10	nABmax.nR
11	nABmax.nR+nBAmax.nS
12	3nABmax.nR+nS
20	nR+nBAmax.nS

5.2.4.d. Calcul du nombre d'opérations

Nous avons calculé le nombre d'opérations E/S nécessaires pour réaliser les opérations atomiques dans le cas des 8 solutions proposées, en prenant les mêmes conventions que dans (§ 5.2.2.d.). Le tableau qui donne le nombre d'opérations E/S nécessaires pour chaque solution et pour chaque opération élémentaire, se trouve en fin de chapitre (§ 5.4.4.).

En prenant la même définition que dans le (§ 5.2.2.d.) de la supériorité d'une solution sur l'autre, on constate que:
11 > 33, 11 > 12, 11 > 20.

5.2.4.e. Algorithme de choix que nous proposons

Si la solution du tableau fixe pour f_1 est possible
et *si* la solution du tableau fixe pour f_2 est possible *alors*
choisir11
sinonsi la solution du tableau fixe pour f est possible
et $Ff_2(b) \cong 0$ et Fsupprimer(b,y) $\cong 0$
alors choisir 10
sinonsi la solution du tableau fixe pour f_1 est possible
et $Ff_1(a) \cong 0$ et Fsupprimer(a,x) $\cong 0$
alors choisir 01

sinon
    si la solution du tableau fixe pour f_1 est possible *alors*
        si $Ff_1(a) \cong 0$ et $Fsupprimer(a,x) \cong 0$ *alors* choisir 02
        sinon choisir 12
        finsi
    sinonsi la solution du tableau fixe pour f_1 est possible *alors*
        si $Ff_2(b) \cong 0$ et $Fsupprimer(b,y) \cong 0$ *alors* choisir 20
        sinon choisir 21
        finsi
    sinon
        choisir 33
    finsi
finsi.

5.2.4.f. Algorithme de mesures

Mesurer nR, nS, nAB, nBA, p,q, nABmax, nBAmax,nRmax, nSmax.

Si la solution du tableau fixe pour f_1 est impossible
ou la solution du tableau fixe pour f_2 est impossible *alors*
    si la solution du tableau fixe pour f_1 est possible
    ou la solution du tableau fixe pour f_2 est possible *alors*
        tester si $Fsupprimer(b,y) \cong 0$ et $Ff_2(b) \cong 0$
        tester si $Fsupprimer(a,x) \cong 0$ et $Ff_1(a) \cong 0$
    finsi
finsi

5.2.5. Relativité de ces résultats

Toute cette étude a dégagé des critères de choix de méthodes d'implantation des chemins d'accès. Ceux-ci reposent sur le nombre d'opérations d'entrée-sortie qui résultent de l'utilisation de ces chemins d'accès et de la méthode de leur implantation informatique.
Ces résultats ont leurs limites. Notamment ils ne tiennent pas compte des règles d'intégrité autres que les dépendances fonctionnelles intrinsèques aux relations. Aussi pour les appliquer faut-il s'assurer que la validation d'autres règles d'intégrité ne les modifie pas.

Ces résultats dégagent l'importance des fréquences de traitements pour faire des choix rigoureux. Si l'on peut facilement admettre que tous les chemins d'accès monovalués (ils associent une entité à au plus une autre entité) seront implantés car leur implantation est aisée, le problème

essentiel concerne les chemins d'accès multivalués qui associent une entité à plusieurs entités (f_2 par exemple): ceux-ci peuvent donner lieu à des implantations lourdes. La nécessité de les conserver dans la base de données dépend de deux fréquences $Ff_2(s)$ et Fsupprimer(s).
Il s'agit donc que l'analyse des traitements estime ces fréquences en cumulant les utilisations prévues des deux opérations atomiques $f_2(s)$ et supprimer(s). A notre avis il s'agit de pondérer les fréquences d'utilisation, d'une part par les conditions d'exploitation du traitement qui les déclenchent (exploitation en conversationnel, en temps différé, en traitements par lots...), et d'autre part par l'importance de ces traitements; certains traitements peuvent n'être exécutés que relativement rarement, mais par contre exiger des résultats très rapides quand ils sont réclamés.
C'est cette étude qui permet notamment au concepteur de savoir si un chemin d'accès est utile ou non (§ 5.1.).

5.3. Conclusions

Nous venons de dresser une liste des choix possibles que l'on rencontre dans la transformation d'un graphe de chemin d'accès brut dans une structure informatique de données.

Ces choix concernent d'abord le graphe de chemins d'accès:

- quels sont les chemins d'accès qu'il faut conserver ?
- comment introduire de la redondance?
- quels sont les index qu'il faut prévoir ?
- quelles sont les relations qu'il est préférable de décomposer ?

Ce sont les résultats de l'analyse des traitements prévus, qui permettent au concepteur de répondre à de telles questions. Nous avons précisé les paramètres qui doivent être estimés par une telle analyse: notamment les fréquences de parcours d'un chemin d'accès multivalué et de suppression d'entités des relations origines des chemins d'accès multivalués.
Le concepteur précise les noms de chemins d'accès retenus et leurs paramètres.

Ensuite ces choix concernent la méthode informatique utilisée pour implanter chacun des chemins d'accès retenu.

La transformation que nous proposons, d'un GCA brut dans une structure informatique de données, n'est pas un algorithme: elle n'est qu'un guide qui limite les modifications possibles pour que la transformation reste fidèle au sens du § 3.2.:
- elle préserve le contenu puisque les relations de noeuds et d'arêtes du GCA brut se retrouve dans le GCA final même si une relation de noeud a été décomposée;
- elle préserve l'accès puisque dans le GCA final, il y a toujours au moins un écoulement correspondant à une décomposition d'une relation espace: cette précaution est prise au moment de l'élimination de chemins d'accès;
- elle préserve l'efficacité: les df intrinsèques des relations du GCA_b sont identiques à celles du GCA final.

5.4. Annexe

5.4.1. Exemple de liste hétérogène doublée

Soit les relations R(*A*X), S(*B*Y), T(*A*B) et les instances suivantes:

iR	iS	iT
a_1x_1	b_1y_1	a_1--
a_2x_2	b_2y_2	a_2b_2
a_3x_3	b_3y_3	a_3b_1
a_4x_4		a_4b_2
a_5x_5		a_5b_2

Seul le chemin d'accès f_2 est implanté à l'aide d'une liste hétérogène doublée: les instances iR et iS sont alors:

			iR				iS			
Rang	*A*	*X*	*Suiv*	*Pre*	*Parent*	*Rang*	*B*	*Y*	*Tlist*	*Finlist*
1	a_1	x_1	Nul	Nul	Nul	1	b_1	y_1	3	3
2	a_2	x_2	4	Nil	Nil	2	b_2	y_2	2	5
3	a_3	x_3	Nil	Nil	1	3	b_3	y_3	Nul	Nul
4	a_4	x_4	5	2	Nil					
5	a_5	x_5	Nil	4	2					

Nul signifie "n'appartient à aucune liste"; Nil signifie "est la dernière entité d'une liste" pour Suiv, "est la première entité de la liste" pour Pre, "n'est pas la dernière entité de la liste" pour Parent.

5.4.2. Implantation d'un chemin d'accès par tableau

Soit les relations R(*A* X), S(*B* Y), T(*A* B) de graphe de chemins d'accès:

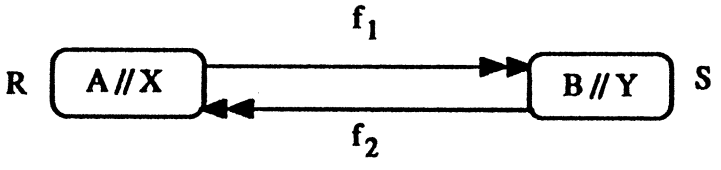

Figure 5.12 Tableau.

Nous considérons f_2 comme un chemin d'accès multivalué. Il peut associer une entité de iS à plusieurs entités de iR. Soit nBAmax la cardinalité maximale de $f_2(s)$ pour toute entité s de S. Suivant la sémantique des relations, ce paramètre peut être connu ou inconnu. Par exemple un réglement de bibliothèque peut limiter le nombre de prêts simultanés (relation T) de livres (relation R) à un adhérent (relation S) à 5: nBAmax=5. Par contre le nombre maximum de voyages (relationR) qu'un explorateur (relation S) a pu entreprendre (relation T) dans sa vie, n'a pas beaucoup de sens.
Si nBAmax est connu, alors il est possible d'implanter f_2 par un tableau de dimension fixe égale à nBAmax: chaque entité s de iS prend des valeurs non seulement pour les constituants B et Y mais également pour un tableau F2 de pointeurs indiquant les adresses des entités de iR qui lui sont associées à s.

Cette solution est très facile à implanter; mais elle peut ne pas être économique en place. En effet toutes les entités de iS ne sont pas obligatoirement associées à nBAmax entités de iR; l'ensemble des places occupées par les pointeurs n'est remplie que dans la proportion de qnBA/nBAmax. Il faut que ce rapport ne soit pas trop éloigné de 1 si nBAmax est assez grand, pour que la place perdue ne soit pas trop importante. Si tel est le cas nous disons que "la solution du tableau fixe est possible".

Sinon il est possible d'avoir recours à des tableaux avec zones de débordement. Alors on associe à chaque entité s de iS un tableau de pointeurs de dimension card80; pour les entités associées à plus de card80 entités de iR leurs tableaux sont prolongés par des zones de débordement. Par exemple s'il est difficile de savoir le nombre maximum de voitures (relation R) que peut posséder un ménage (relation S), il est par contre

facile de savoir que 80% des ménages n'ont pas plus de 2 voitures; l'entité relative à un ménage va contenir un tableau de dimension 2; un pointeur vers une zone de débordement sera utile pour les ménages plus motorisés.

S'il est possible d'estimer card80 et si les tableaux de pointeurs de cette dimension sont suffisamment remplis, nous disons que *"la solution du tableau est possible"*. La solution du tableau fixe en est un cas particulier.

Remarque:
certains SGBD implantent systématiquement tous les chemins d'accès multivalués à l'aide de tableaux.

5.4.3. Exemple de multiliste

Soit les relations R(*A*X), S(*B*Y), T(*AB*) et les instances suivantes:

iR	iS	iT
a_1x_1	b_1y_1	a_2b_1
a_2x_2	b_2y_2	a_2b_3
a_3x_3	b_3y_3	a_2b_4
a_4x_4	b_4y_4	a_3b_4
a_5x_5		a_4b_4
		a_4b_3
		a_3b_2

Seul le chemin d'accès f_1 (de iR vers iS) est implanté.
nBAmax est égal à 3 et f_1 est implanté à l'aide d'une multiliste. Les instances iR et iS sont alors:

iR:

Rang	*A*	*X*	*Atête*	*Atpos*
1	a_1	x_1	Nul	Nul
2	a_2	x_2	1	1
3	a_3	x_3	4	2
4	a_4	x_4	4	3
5	a_5	x_5	Nul	Nul

iS:

Rang	*B*	*Y*	*Asuiv*			*Apos*		
1	b_1	y_1	3	Nul	Nul	2	Nul	Nul
2	b_2	y_2	Nil	Nul	Nul	Nil	Nul	Nul
3	b_3	y_3	Nil	4	Nul	Nil	1	Nul
4	b_4	y_4	Nil	2	3	Nil	1	1

Ainsi à a_4 correspond b_4 car a_4.Atête = 4; il lui correspond également b_3 car a_4.Atpos = 3 et b_4.Asuiv(3) = 3. Il ne lui correspond plus aucune autre entité car b_4.Apos(3) = 1 et b_3.Asuiv(1) = Nil.

5.4.4. Tableaux des nombres d'opérations

Les tableaux admettent en lignes les différentes méthodes d'implantation et en colonnes les différentes opérations atomiques. Le premier tableau concerne l'implantation des D-arêtes, le second celui des F-arêtes, et le troisième celui des B-arêtes.

Dans le tableau relatif aux F-arêtes toutes les opérations atomiques contenant (a,b) : $\notin$ T ont fait l'objet de deux types d'algorithmes: le premier ("avec vérif") vérifie que la base contient bien l'entité (a,b) dans l'instance de T avant l'exécution de l'opération, le second ("sans vérif") ne le vérifie pas.
Toutes les méthodes d'implantation considérées par tableau considèrent des tableaux de dimension fixe.

Voici le tableau relatif aux D-arêtes:

		f2(a)	*(a,b,y):$\in$ R*	*(a,b,y):$\notin$ R*	*(a,x):$\notin$S*
10	R	nR	1	1	nR
	S	1	1	0	1
11	R	qnSR	1	(nSR+3)/2	qnSR
	S	1	2	1	1
13	R	qnSR	1+q	4-(2/nSR)	qnSR
	S	1	2	1/nSR	1
15	R	qnSR	1	1	qnSR
	S	1	1	1	1

Voici celui des F-arêtes suivi par celui des B-arêtes.

A →F B		$(a,b):\notin T$ et $(a,b_1):\in T$		$(a,b):\notin T$ et $(a_1,b):\in T$		$(a,-):\notin T$ et $(a,b_1):\in T$	$(a,b):\notin T$		$f_1(a)$	$f_2(b)$	$(a,b):\in T$	$(a,x):\notin R$	$(b,y):\notin S$	$(a,-):\notin T$
		avec vérif.	sans vérif.	avec vérif.	sans vérif.		avec vérif.	sans vérif.						
10	R	1	1	2	2	1	1	1	1	nR	1	1	nR	1
	S	0	0	0	0	0	0	0	p	0	0	0	1	0
01	R	$\frac{nBA+3}{2}$	$\frac{nBA+3}{2}$	$\frac{nBA+5}{2}$	$\frac{nBA+5}{2}$	$2+q\frac{nS-1}{2}nBA + \frac{nBA-1}{2}$	$\frac{nBA+3}{2}$	$\frac{nBA+3}{2}$	$1+p(nBA\times\frac{nS-1}{2}q + \frac{nBA-1}{2})$	qnBA	1	$1+p(qnBA\times\frac{nS-1}{2} + \frac{nBA-1}{2})$	qnBA	$2+qnBA\ \frac{nS-1}{2} + \frac{nBA-1}{2}$
	S	3	3	$1+\frac{1}{nBA}$	$1+\frac{1}{nBA}$	$2+\frac{nS+1}{2}$	1	1	$p\frac{nS+1}{2}$	1	2	$p\frac{nS+1}{2}$	1	$\frac{nS+1}{2}$
02	R	$\frac{nBA+3}{2}$	$\frac{nBA+3}{2}$	$\frac{nBA+5}{2}$	$\frac{nBA+5}{2}$	nBA+1	$\frac{nBA+3}{2}$	$\frac{nBA+3}{2}$	$1+p(\frac{nBA-1}{2})$	qnBA	1	1+p(nBA-1)	qnBA	nBA+1
	S	3	3	$1+\frac{1}{nBA}$	$1+\frac{1}{nBA}$	3	1	1	p	1	2	p	1	1
03	R	$3+q+\frac{nBA-1}{2}+\frac{1}{nBA}$	$4+q-\frac{1}{nBA}$	$5+\frac{nBA-1}{2}+\frac{2}{nBA}$	$4+\frac{2}{nBA}$	$4-\frac{2}{nBA}+q$	$\frac{nBA+5}{2}+\frac{1}{nBA}$	$4-\frac{2}{nBA}$	$1+p(\frac{nBA-1}{2})$	qnBA	1+q	$1+p(2-\frac{2}{nBA})$	qnBA	$4-\frac{2}{nBA}$
	S	3	$2+\frac{1}{nBA}$	$1+\frac{1}{nBA}$	$\frac{1}{nBA}$	$2+\frac{1}{nBA}\times\frac{nS+1}{2}$	$1+\frac{1}{nBA}$	$\frac{1}{nBA}$	$p\frac{nS+1}{2}$	1	2	$p\frac{(nS+1)}{2nBA}$	1	$\frac{nS+1}{2nBA}$
04	R	$3+q+\frac{nBA-1}{2}+\frac{1}{nBA}$	$4+q-\frac{1}{nBA}$	$5+\frac{nBA-1}{2}+\frac{2}{nBA}$	$4-\frac{2}{nBA}$	$5+q-\frac{3}{nBA}$	$\frac{nBA+5}{2}+\frac{1}{nBA}$	$4-\frac{2}{nBA}$	$1+p\frac{nBA-1}{2}$	qnBA	1+q	$2+p(3-\frac{3}{nBA})$	qnBA	$5-\frac{3}{nBA}$
	S	3	$2+\frac{1}{nBA}$	$1+\frac{1}{nBA}$	$\frac{1}{nBA}$	$2+\frac{1}{nBA}$	$1+\frac{1}{nBA}$	$\frac{1}{nBA}$	p	1	2	$\frac{p}{nBA}$	1	$\frac{1}{nBA}$
05*	R	0	0	0	0	0	0	0	0	qnBA	1	1	0	0
	S	3	3	2+nS	2+nS	$2+\frac{nS+1}{2}$	1	1	$\frac{nS+(1-p)nS+p}{2}$	1	nS+1	$\frac{nS+(1-p)nS+p}{2}$	1	$1+\frac{nS+(1-p)nS}{2}$
11	R	$\frac{nBA+3}{2}$	$\frac{nBA+3}{2}$	$\frac{nBA+5}{2}$	$\frac{nBA+5}{2}$	$\frac{nBA+3}{2}$	$\frac{nBA+3}{2}$	$\frac{nBA+3}{2}$	1	qnBA	1	$1+p\frac{nBA-1}{2}$	qnBA	$\frac{nBA+3}{2}$
	S	3	3	$1+\frac{1}{nBA}$	$1+\frac{1}{nBA}$	3	1	1	p	1	2	p	1	1
13		$4-\frac{2}{nBA}+q$ $2+\frac{1}{nBA}$	$4-\frac{2}{nBA}+q$ $2+\frac{1}{nBA}$	$5-\frac{2}{nBA}$ $\frac{1}{nBA}$	$5-\frac{2}{nBA}$ $\frac{1}{nBA}$	$4-\frac{2}{nBA}+q$ $2+\frac{1}{nBA}$	$4-\frac{2}{nBA}$ $\frac{1}{nBA}$	$4-\frac{2}{nBA}$ $\frac{1}{nBA}$	1 p	qnBA 1	1+q 2	$1+p(2-\frac{2}{nBA})$ $\frac{p}{nBA}$	qnBA 1	$4-\frac{2}{nBA}$ $\frac{1}{nBA}$
15*		2 3	2 3	4 1	4 1	2 3	2 1	2 1	1 p	qnBA 1	1 2	1 p	qnBA 1	2 1

Tous les résultats comportant $\frac{1}{nBA}$ ont été obtenus en supposant nBA constant.

* Les tableaux sont supposés de taille fixe.

A—B		(a,b) : ∉ T et (a,b₁) : ∈ T	(a,b) : ∉ T et (a₁,b) : ∈ T	(a,b).	$\tau_1(a)$	$f_2(b)$	(a,b) : ∈ T	(a,x) : ∉ R	(b,y) : ∉ S
33	R	0	$3+\frac{1}{nAB}$ (2)*	$1+\frac{1}{nAB}$ *	1	qnBA	2 (1)	2 *	$\frac{qnBA}{nAB}$ *
	S	$3+\frac{1}{nBA}$	0	$\frac{1}{nBA}$	pnAB	1	2	$\frac{pnAB}{nBA}$	2
	Table	$\frac{nAB+1}{2}+2\frac{nBA-1}{nBA}$ $+q+1$	$\frac{nAB}{2}+2\frac{nAB-1}{nAB}$ $+p+1$	$\frac{nAB+1}{2}+2\frac{nBA-1}{nBA}$ $+2\frac{nAB-1}{nAB}+1$	pnAB	qnBA	(qpnAB+1+q+p)	$pnAB(1+2\frac{nBA-1}{nBA})$	$qnBA(1+2\frac{nAB-1}{nAB})$
	Total	$6+q+\frac{nAB+1}{2}-\frac{2}{nBA}$ ou $6+q+\frac{nBA+1}{2}-\frac{2}{nBA}$	$6+p+\frac{nAB+1}{2}-\frac{1}{nAB}$ ou $6+p+\frac{nBA+1}{2}-\frac{1}{nAB}$	$6+\frac{nAB+1}{2}-\frac{2}{nAB}-\frac{2}{nBA}$ ou $6+\frac{nBA+1}{2}-\frac{2}{nAB}-\frac{2}{nBA}$	1+2pnAB	1+2qnBA	5+p+q+pqnAB ou 5+q+p+pqnBA	$1+3pnAB-p\frac{nAB}{nBA}$	$1+3qnBA-q\frac{nBA}{nAB}$
11	R	1	2	1	1	qnBA	1	1	qnBA
	S	2	1	1	pnAB	1	1	pnAB	1
	Total	3	3	2	1+pnAB	1+qnBA	2	1+pnAB	1+qnBA
12	R	$2+\frac{nBA}{2}$	$3+\frac{nBA}{2}$	$1+\frac{nAB}{2}$	1	qnBA	2	$1+p\frac{nAB \times nBA}{2}$	qnBA
	S	2	1	1	pnAB	1	1	pnAB	1
	Total	$4+\frac{nBA}{2}$	$4+\frac{nBA}{2}$	$2+\frac{nAB}{2}$	1+pnAB	1+qnBA	3	$1+pnAB+\frac{nAB \times nBA}{2}p$	1+qnBA
10	R	1	2	1	1	nR	1	1	nR
	S	0	0	0	pnAB	0	0	0	1
	Total	1	2	1	1+pnAB	nR	1	1	1+nR
20	R	$1+\frac{1}{nAB}$	$3+\frac{1}{nAB}$	$1+\frac{1}{nAB}$	1	qnR	1	1	qnR
	S	$(nAB)+4+(1-\frac{1}{nAB})$	$\frac{nAB-1}{2}+nAB+2$	$\frac{nAB+1}{2}+\frac{nAB-1}{nAB}$	pnAB	S $1+q(nR-nBA)pnAB+q\frac{nABnBA}{2}$ T $1+2qnR+q(1-\frac{p}{2})nABnBA$	pq(nAB-1)+2	pnAB	$1+q+q(nR-nBA)pnAB+q\frac{nABnBA}{2}$
	Total	nAB+6	$\frac{3nAB+9}{2}+\frac{1}{nAB}$	$\frac{nAB+3}{2}$	1+pnAB		3+pq(nAB-1)	1+pnAB	$1+q+2qnR+q(1-\frac{p}{2})nABnBA$

(1) Le résultat correspond au cas où pnAB ≪ qnBA

(2) * * * * nAB ≤ nBA

* Pour ce calcul nous avons supposé nAB ou/et nBA constants

TROISIÈME PARTIE

DÉCOMPOSITION D'UNE RELATION

6

CHOISIR UNE DÉCOMPOSITION D'UNE RELATION

L'étude de la *décomposition* d'une relation s'intéresse au difficile problème des *représentations équivalentes* d'informations. Nous allons considérer notamment l'équivalence entre une relation R et un ensemble de relations $\{R_1...R_i...R_n\}$; cette équivalence signifie que toute information prise en compte dans R sous forme d'une entité, doit pouvoir être prise en compte dans l'ensemble des relations $\{R_1...R_i...R_n\}$ sous forme d'entités de ces relations, et réciproquement. Formellement cette équivalence est construite à l'aide d'une part de l'opération de *projection* de relations qui permet de transformer une entité de R en entités des relations R_i, et d'autre part l'opération de *composition* de relations qui permet de transformer des entités compatibles des différentes relations R_i en une entité de R. Les relations $\{R_1...R_i...R_n\}$ forment alors une *décomposition* de la relation R.
D'abord nous allons illustrer l'intérêt de stocker les entités de R dans la base de données à l'aide de leurs projections sur R_i^+ plutôt que de les stocker elles-mêmes, à l'aide de l'exemple suivant.

Exemple: Voici la relation structure commerciale SC d'une entreprise de produits grand public. Cette relation est construite sur 3 constituants nom d'un représentant (R), nom d'un point de vente (PTV), nom d'un produit (P) avec le prédicat suivant: (r ptc p) est une entité de SC si le représentant r visite le point de vente ptv et y présente le produit p de son catalogue. SC (R PTV P) est décomposable en SC[R PTV], SC[R P] car la structure commerciale est ainsi organisée: tout représentant présente tous les produits de son catalogue à tous les points de vente qu'il visite.

Décomposition d'une relation

Nous pouvons stocker les informations suivant deux structures,
la première suivant SC (R PTV P) munie de la règle d'intégrité précédente exprimée sous forme de dépendance de composition,
la seconde suivant SC1 = SC[R PTV] et SC2 = SC[R P].
A l'aide d'un exemple d'une instance de R, nous allons montrer l'intérêt de la seconde structure par rapport à la première.

SC	(R	PTV	P)
	Duhêtre	Aupré	Abao
	Duhêtre	Croisement	Abao
	Duhêtre	Aupré	Evesele
	Duhêtre	Croisement	Evesele
	Morvié	Desert	Épine

Si maintenant M. Duhêtre visite le point de vente Gentil-Caviar, il faut dans la première structure retrouver tous les produits du catalogue de M. Duhêtre et *créer* ensuite les entités (Duhêtre Gentil-Caviar Abao) et (Duhêtre Gentil-Caviar Evesele); dans la seconde structure, il suffit de créer la nouvelle entité dans SC1: (Duhêtre Gentil-Caviar).
Si maintenant M. Duhêtre n'a plus à son catalogue le produit Abao, il faut aller *supprimer* toutes les entités de SC relatives à M. Duhêtre et à Abao dans la première structure; dans la seconde il suffit de supprimer l'entité (Duhêtre Abao) de SC2.
Si on applique la même méthode au cas où M. Morvié n'a plus à son catalogue le produit Épine, on va perdre dans la première structure l'information que M. Morvié visite Désert alors que cette information est conservée dans la seconde structure.
Si maintenant le produit Ampleur remplace Abao dans le catalogue de M. Duhêtre, il faut dans la première structure trouver toutes les entités de SC relatives à (Duhêtre Abao) et les *mettre-à-jour* en remplaçant Abao par Ampleur; par contre dans la seconde structure, il suffit de remplacer l'entité (Duhêtre Abao) par (Duhêtre Ampleur) dans SC2.
En conclusion, les opérations de modification (création, suppression, mise à jour) sont nettement facilitées dans la deuxième structure. Alors que dans la première structure, il faut prévoir des programmes de validation de la dépendance de composition coûteux en temps d'exécution, dans la seconde structure aucun programme de validation n'est à écrire et les modifications s'opèrent simplement.

En fait, la décomposition a restreint la *redondance d'informations*: dans la première structure, on écrit plusieurs fois que le catalogue de M. Duhêtre contient Evesele; dans la seconde on ne l'écrit qu'une fois.
fex.

Comme le fait pressentir l'exemple précédent, un intérêt d'une décomposition d'une relation R est d'éviter de la redondance d'informations et donc le maintien de la cohérence de cette redondance par des programmes de validation. Pour étudier cet intérêt dans le cas général, nous allons:
- d'abord montrer différentes manières de décomposer une relation R;
- ensuite dégager les liens existant entre une décomposition de R et la validation des règles d'intégrité définies sur R: formellement il s'agit de définir la *projection ou l'extension d'une règle d'intégrité*;
- finalement comparer différentes décompositions de R par rapport à l'efficacité de la validation de ces règles d'intégrité: notamment nous définissons des *qualités* d'une décomposition.

6.1. Décomposer une relation

6.1.1. Définition et propriétés d'une décomposition

C'est dans (DELOBEL73) que l'on trouve une première formalisation de la notion de décomposition.
Nous allons faire quelques rappels du § 2.5.1.

DÉFINITION

Une *décomposition* D d'une relation R est un ensemble de relations R_1, R_2,...R_i,...R_n vérifiant:

a) $R_i = R[R_i^+]$,

b) $R = \overset{n}{\underset{i=1}{*}} R_i$.

D^+ désigne l'ensemble des constituants des relations R_i: $D^+=R^+$.

Une décomposition D de R est *triviale* si elle contient une seule relation, R elle-même.

Une décomposition D de R est *nettoyée* si pour toute relation R_i de D admettant pour clé R_i^+, il n'existe aucune relation R_j de D telle que $R_i^+ \subseteq R_j^+$.

Décomposition d'une relation

Exemple:
R(ABCD) et F = {AB → C}.
R_1 = R[ABC], R_2 = R[ABD], R_3 = R[AB].
La décomposition D' = {R_0 R_1 R_2} n'est pas nettoyée à cause de R_3, la décomposition D = {R_1 R_2} l'est par contre.

DÉFINITION

D' est une *décomposition partielle* de la décomposition D
si D' est incluse dans D, et si D' est une décomposition de R' = R[D'$^+$].

Remarque:

Une relation R ne peut pas être décomposée suivant n'importe quel découpage de R^+ comme le montre l'exemple suivant: soit R(ABC) qui admet cet ensemble d'entités comme fermeture:

a_1	b_1	c_1
a_1	b_2	c_1
a_1	b_1	c_2

Elle ne se décompose pas suivant R[AB] et R[AC] car la composition des projections de la fermeture fournit (a_1 b_2 c_2) qui n'appartient pas à la fermeture.

Propriété (d) (rappel):

Si un ensemble de relations {R_1...R_i...R_n} vérifie la propriété a) et la propriété c) $\bigcup_{i=1}^{n} R_i^+ = R^+$, alors il vérifie: $R \subseteq \underset{i=1}{\overset{n}{*}} R_i$.

Les propriétés des décompositions et les propriétés des dépendances de composition sont étroitement liées.(§ 2.5.2.).

6.1.2. Instances

Propriété de complétude (rappel):

Si les relations R_1...R_i...R_n forment une décomposition D de la relation R, pour toute instance iR_i de la relation R_i il existe un ensemble d'instances compatibles entre elles iR_1 iR_2...iR_n formé d'une instance de chaque relation de la décomposition.

Propriété (di) (rappel):

Si $iR = *_{i=1}^{n} (iR_i)$, alors $iR[R_i^+] \subseteq i\,R_i$ pour (i=1,...,n).

Remarque:
Cette dernière propriété s'applique en particulier aux instances des relations R_i stockées dans la base au même instant.

Exemple:
Soit R(ABC) qui se décompose en $R_1 = R[AB]$ et $R_2 = R[AC]$.
iR_1 contient les entités (ab) et (a'b').
iR_2 contient les entités (ac) et (a"c").
iR contient une seule entité (abc).
Alors iR[AB] contient seulement (ab).
En effet (a'b') n'est compatible avec aucune entité de iR_2 et ne donne donc lieu à aucune entité de iR; il en est de même de (a"c").
fex.

Nous sommes alors confrontés au problème suivant qui concerne justement les entités (a'b') et (a"c"). Peut-il oui ou non exister une collection de données iR de R, c'est-à-dire une instance de R qui valide les ri de R, telle que $iR_1 \subseteq iR[R_1^+]$ et $iR_2 \subseteq iR[R_2^+]$?
Dans le premier cas, iR_1 et iR_2 sont des collections de données et fournissent les réponses correctes à la recherche des entités contenues dans la base relatives à R_1 et à R_2. Dans le second cas, les réponses correctes se limitent d'une part à (ab) et d'autre part à (ac) et sont obtenues par la composition des instances de iR_1 et de iR_2.

6.1.3. Différentes méthodes de décomposition d'une relation

Nous allons rappeler que les dépendances fonctionnelles permettent de décomposer une relation.

THÉORÈME (decdf1)

Une relation R(XYZ) munie de la df $X \rightarrow Y$ est alors munie de la dc {R[XY], R[XZ]} et peut se décomposer ainsi:
$R = R[XY] * R[XZ]$.

La démonstration est fournie dans le § 2.5.4.

Décomposition d'une relation

THÉORÈME (decdf2)

Une relation R munie d'un ensemble de df $F = \{f_1...f_i...f_n\}$ peut se décomposer de la manière suivante:
K désignant une clé de R obtenue à partir de F par application de l'algorithme d'obtention de clés du § 2.11.,
f_i^+ désignant les constituants de f_i,

$$R = \left(\overset{n}{\underset{i=1}{*}} R[f_i^+] \right) * R[K].$$

Elle est également munie de la dc associée à cette décomposition.

Si la relation R[K] empêche la décomposition d'être nettoyée, on peut l'éliminer et obtenir une décomposition nettoyée de R.

La preuve est fournie en annexe.

Exemples:
- Soit R(ABCD) et F = {A → B; B → C; B → D}. clé de R: A.
La décomposition D obtenue est la suivante:
R = R[AB] * R[BC] * R[BD] * R[A].
En la nettoyant on obtient R = R[AB] * R[BC] * R[BD].
- Une autre décomposition de la même relation R peut être obtenue en considérant seulement un sous-ensemble F' de F: F'={A → B; B → D}.
La clé de R par rapport à F' est alors AC; voici la décomposition obtenue:
R = R[AC] * R[AB] * R[BD]. Elle est nettoyée.
- Soit S(ABC) et F = {A → C; B → C}. clé de S: AB.
On obtient comme décomposition: R = R[AC] * R[BC] * R[AB].
Cette décomposition est nettoyée.

Définition:

Une décomposition D de R est dite *fonctionnelle de relation source R_1* si et seulement si il existe au moins un ordre de rangement des relations de D $\{R_1...R_i....R_n\}$ tel que:

$$\forall i \in (1,n-1)\ K_{i+1} \subseteq \bigcup_{j=1}^{i} R_j^+ \quad \text{où } K_{i+1} \text{ est une clé de } R_{i+1}.$$

Propriété:
Toute décomposition de R obtenue par l'application successive du théorème (decdf1) est une décomposition fonctionnelle.
La preuve est immédiate.

Propriété:

Toute décomposition de R obtenue à partir d'une couverture de df F par l'application du théorème (decdf2) est une décomposition fonctionnelle. La preuve se déduit du paragraphe c) de la preuve du théorème (decdf2).

PROPRIÉTÉ (dco)

Si R admet la dépendance de composition (dc) $\{R_1...R_n\}$ alors $\{R_1...R_n\}$ forme une décomposition de R (rappel du § 2.5.1.).

6.1.4. Conclusion

Un algorithme possible de décomposition d'une relation R considère l'ensemble des dépendances fonctionnelles et des dépendances de composition définies sur R; une première décomposition est obtenue à partir d'une couverture des df (théorème decdf2) ou à partir d'une df (théorème decdf1) ou à partir d'une dc totale définie sur R. L'algorithme se poursuit en tentant de décomposer les relations formant cette première décomposition à l'aide des df et des dc partielles. Et ainsi de suite.

Si la relation R est munie de plusieurs df et dc, cet algorithme va fournir *plusieurs décompositions* possibles de la relation R. C'est le difficile problème du choix entre ces décompositions que nous allons étudier maintenant.

6.2. Projection et extension d'une règle d'intégrité

La décomposition D d'une relation R: $D = \{R_1...R_i...R_n\}$ fournit une représentation équivalente de R et donc une autre possibilité de stockage des entités de R; celles-ci vont être stockées à l'aide de leurs projections dans les instances des différentes relations de D. La base de données est ainsi formée par les instances des relations R_i de D et l'instance iR de R devient virtuelle dans le sens qu'elle n'est pas stockée mais qu'elle peut être recalculée par la composition des instances. Les opérations élémentaires de modification de la base de données ne s'expriment plus dans les termes: créer, supprimer, mettre à jour une entité de R mais dans ceux-ci: créer, supprimer, mettre à jour une entité de R_i.

Décomposition d'une relation

Le problème de la cohérence de la base de données exprimé en termes de règles d'intégrité définies sur R, doit s'exprimer en termes de règles d'intégrité définies sur les R_i: les contrôles de cohérence sont effectués lors d'une modification d'une instance de R_i.

DÉFINITIONS

La *projection d'une règle d'intégrité* ri définie sur la relation R sur l'ensemble de constituants S^+ est la règle d'intégrité notée ri[S^+] définie sur la relation S = R[S^+] et dont le prédicat est validé par toutes les instances iS de S telles qu'il existe au moins une instance iR validant ri et se projetant sur S^+ en iS = iR[S^+].

L'*extension d'une règle d'intégrité* ri définie sur la relation R à l'ensemble de constituants S^+ contenant R^+ est la règle d'intégrité notée ext (ri/S^+) définie sur la relation S = (R/S^+) et dont le prédicat est validé par toutes les instances iS telle que iS[R^+] est une instance de R validant ri.

Théorème:

ri étant définie sur R^+, ri' sur S^+,
ri' = ext(ri/S^+) est équivalent à ri = ri'[R^+].
La preuve est immédiate.

Propriété (ridfl) et définition:

La projection sur S^+ de la dépendance fonctionnelle df: X $\rightarrow$ Y définie sur la relation R est égale à:
a) la règle d'intégrité triviale 1 si X n'est pas inclus dans S^+, et le domaine d'au moins un constituant de X - S^+ est illimité;
b) la règle d'intégrité triviale 1 si Y $\wedge$ S^+ = $\varnothing$;
c) la df X $\rightarrow$ W où W = Y $\wedge$ S^+ si X $\subseteq$ S^+.

Preuve:

a) Soit XS = X $\wedge$ S^+. Soit XR = X - XS.
Il suffit de considérer l'extension iR de iS telle que chaque entité de iS a une valeur différente des autres pour chaque constituant de XR; ceci est possible car au moins un de leurs domaines est illimité; iR valide ainsi X $\rightarrow$ Y.

b) Le même raisonnement s'applique en donnant aux entités de iR la même valeur pour Y.

c) Nous démontrons que $X \rightarrow W \leq df[S^+]$, et puis que $df[S^+] \leq X \rightarrow W$.
c1) $\forall$ iS qui valide $X \rightarrow W$, $\forall$ iR une extension de iS à R^+. Si iR ne valide pas $X \rightarrow W$, alors $\exists$ r, r' $\in$ iR telle que $r.X = r'.X$ et $r.W \neq r'.W$.
Comme $iS = iR[S^+]$, $\exists$ s, s' $\in$ iS telle que $s = r.S^+$ et $s' = r'.S^+$.
Ainsi $s.X = s'.X = r.X$ et $s.W \neq s'.W$.
Donc iS ne validerait pas $X \rightarrow W$, ce qui est impossible.
iR valide donc $X \rightarrow W$.

On peut facilement transformer iR en iR' en faisant en sorte que toutes les entités de iR' prennent les mêmes valeurs pour les constituants de Y-W.
iR' valide $X \rightarrow Y$ et sa projection sur S^+ est iS. Ainsi $X \rightarrow W \leq df[S^+]$.

c2) Réciproquement:
$\forall$ iS qui valide $df[S^+]$, par définition il existe une instance iR de R qui valide df et dont la projection sur S^+ est iS.
Si iS ne valide pas $X \rightarrow W$, alors
$\exists$ s, s' $\in$ iS telle que $s.X = s'.X$ et $s.W \neq s'.W$.
Comme $iS = iR[S^+]$, $\exists$ r, r' $\in$ iR telle que $r.S^+ = s$ et $r'.S^+ = s'$.
Alors $r.X = r'.X$ et $r.W \neq r'.W$.
Ainsi iR ne validerait pas $X \rightarrow W$ et donc ne validerait pas $X \rightarrow Y$.
Ce qui est impossible. iS valide $X \rightarrow W$.
Donc $df[S^+] \leq X \rightarrow W$.

c3) Finalement $df[S^+] = X \rightarrow W$.

Propriété (ridf2) et définition:

L'extension à S^+ de la df: $X \rightarrow Y$ définie sur la relation R (R^+ inclus dans S^+), est égale à la dépendance fonctionnelle $X \rightarrow Y$ définie dans la relation $S = ext(R/S^+)$.

Preuve:

X et Y sont inclus dans S^+ par construction.

a) Si iS valide $X \rightarrow Y$, alors $iR = iS[R^+]$ valide $X \rightarrow Y$ d'après la propriété précédente.

b) Si iS est telle que $iR = iS[R^+]$ valide $X \rightarrow Y$ alors
$\forall$ s, s' $\in$ iS telle que $s.X = s'.X$.
Soit $r = s.R^+$ et $r' = s'.R^+$. Comme iR valide $X \rightarrow Y$, $r.Y = r'.Y$.
Donc $s.Y = s'.Y$ et iS valide $X \rightarrow Y$.
Ainsi $X \rightarrow Y = ext(df/S^+)$.

Décomposition d'une relation

DÉFINITION

Un *ensemble* de règles d'intégrité $I=\{i_1...i_n\}$ définies sur R est une nouvelle règle d'intégrité i définie sur R telle que toutes les instances de R validant i valide chacune des *ri* de I:

$$i = i_1 \wedge i_2 \wedge ...i_n.$$

i est la *règle d'intégrité caractéristique* de I.

DÉFINITIONS

L'*extension* de I à S^+ (R^+ inclus dans S^+) est l'extension de la règle d'intégrité $i = i_1 \wedge i_2 \wedge ...i_n$.

L'*extension dispersée* de I à S^+ (R^+ inclus dans S^+) est la règle d'intégrité $ext(I/S^+) = \{ ext(i_1/S^+), ext(i_2/S^+)... ext(i_n/S^+)\}$.

La *projection* de I sur un ensemble de constituants X, est la projection de la règle d'intégrité: $i = i_1 \wedge i_2 \wedge ...i_n$.

La *projection dispersée* de I sur un ensemble de constituants X est la règle $I[X] = \{i_1[X], i_2[X], ... i_n[X]\}$.

Un ensemble de règles d'intégrité I définie sur R^+ est *complet pour la projection* si $\forall X \subseteq R^+$ la projection de I sur X est équivalente à sa projection dispersée sur X.

Un ensemble de règles d'intégrité définie sur R^+ est *complet pour l'extension* si $\forall$ X contenant R^+ l'extension de I à X est équivalente à son extension dispersée à X.

Un ensemble de règles d'intégrité définie sur R^+ est *continu* s'il est complet pour la projection et l'extension.

Propriété (ridf3):

Tout ensemble de df F est *complet* pour l'extension.

Preuve:
Soit $F = \{f_1...f_n\}$ défini sur R.
Soit f la ri caractéristique de F. L'extension de f à S^+ ($R^+ \subseteq S^+$) est la règle d'intégrité $g = ext(f/S^+)$; l'extension dispersée de F à S^+ est l'ensemble G.

D'après la propriété ($ridf_2$) G est un ensemble de df $\{g_1...g_n\}$ définies sur S^+ et s'exprimant de la même façon que les df $f_1...f_n$. Soit g' la règle d'intégrité caractéristique de G.
Il faut démontrer l'égalité de g et de g'.
a) $\forall$ iS validant g, soit iR = iS[R^+]. Comme iS valide g, iR valide f et donc valide toutes les df f_i; ainsi iS valide les df $g_i = ext(f_i/S^+)$ et donc g'.
b) $\forall$ iS validant g', elle valide toutes les df $g_i = ext(f_i/S^+)$; iR = iS[R^+] valide donc les df f_i (propriété $ridf_1$); iR valide donc f; iS valide g = $ext(f/S^+)$ par définition de l'extension.

Remarque:
Un ensemble de df F peut ne pas être complet pour la projection.

Exemple:
Soit R(ABC) et F = {A $\rightarrow$ B; B $\rightarrow$ C}.
F[AC] = $\varnothing$.
Si f est la ri caractéristique de F alors f[AC] = {A $\rightarrow$ C}.

Propriété (ridf 4):
La *base complète* d'un ensemble de df F est complète pour la projection quand les domaines des constituants sont illimités.

Preuve:
On suppose que F est une base complète de df définies sur R^+. Soit f la ri caractéristique de F et g = f[S^+] où $S^+ \subseteq R^+$. Soit g' la ri caractéristique de F[S^+]. Soit S = R[S^+].

a) $g \subseteq g'$?
$\forall$ iS validant g, il existe iR dont iS est la projection sur S^+, et qui valide f; donc iR valide tous les f_i. Par la définition de la projection, iR[S^+] valide tous les f_i[S^+] et donc iS valide g'.

b) $g' \subseteq g$?
$\forall$ iS validant g', iS valide toutes les df f_i[S^+] qui sont des df éventuellement triviales (propriété $rifd_1$).
Nous allons construire une extension iR de iS à R^+ qui valide toutes les df f_i: ainsi iR validera f et iS g par définition de la projection d'une règle d'intégrité.

Soit S* = {A, $S^+ \rightarrow$ A est une df de F**}.

Nous allons compléter les entités de iS de la manière suivante pour obtenir des entités de iR:

- à chaque entité s de iS correspond une entité r de iR telle que:$r.S^+ = s$;
- $\forall A \in S^* - S$:deux entités r_a et r_b de iR vérifiant $r_a.A = r_b.A$ si et seulement si il existe X inclus dans S^+ tel que $X \rightarrow A$ appartient à F et si $r_a.X = r_b.X$;
- $\forall A \in R^+ - S^*$, toutes les entités de iR prennent des valeurs distinctes les unes des autres.

Nous allons prouver que *iR ainsi formée valide F.*
Soit $Y \rightarrow A$ une df de F et r_a et r_b deux entités de iR telles que $r_a.Y = r_b.Y$.
Si l'intersection de Y et de $R^+ - S^*$ n'était pas vide, $r_a.Y$ serait distinct de $r_b.Y$ par construction. Donc Y est inclus dans S^*.
Ainsi il existe X' inclus dans S^+ tel que $X' \rightarrow Y$ est une df de F^{**}.
Alors il existe X inclus dans X' tel que $X \rightarrow A$ est une df de F^* (=F).
Si A est un constituant de S alors $X \rightarrow A$ appartient à $F[S^+]$ et iS la valide: $s_a.A = s_b.A$ et donc $r_a.A = r_b.A$.
Sinon par construction $r_a.A = r_b.A$ puisque $s_a.X = s_b.X$.

c) Ainsi g' = g et la base complète d'un ensemble de df F est complète pour la projection.

Propriété:
La *base complète* d'un ensemble de df de F est continue quand les domaines de constituants sont illimités.

Ainsi pour trouver les df définies sur $R_i = R[R_i^+]$, il suffit de former l'ensemble des df F_i tel que $\forall f_i \in F_i \; \exists f \in F^* \; f_i = f[R_i^+]$.

Propriété (ridf5):
Si iR_i est une instance de R_i validant F_i pour tout $i = \{1...n\}$ alors

$$iR = \overset{n}{\underset{i=1}{*}} (iR_i) \text{ est une instance de R validant } \bigcup_{i=1}^{n} F_i.$$

C'est une généralisation de la propriété (ridf2).

Conclusion:
Le schéma de décomposition ne fait plus seulement correspondre une relation R avec un ensemble de relations $R_1...R_i...R_n$ mais une *structure de relation* (R,I) formée de la relation R et de l'ensemble des règles d'intégrité I définies sur R avec un ensemble de structures de relations $\{(R_1,I_1)...(R_i,I_i)...(R_n,I_n)\}$ où R_i est la projection de R sur R_i^+ et I_i est la projection de I sur R_i^+. Dans le paragraphe suivant, nous allons nous intéresser aux différentes qualités que peut posséder une décomposition d'une relation R par rapport aux règles d'intégrité définies sur R.

6.3. Qualités d'une décomposition

Comme nous l'avons remarqué en conclusion du paragraphe 1, une relation R peut admettre plusieurs décompositions; nous allons au cours de ce chapitre tenter de les comparer et dans ce but nous allons dégager des *qualités* qu'une décomposition peut posséder. Toute notre étude va considérer comme *seules règles d'intégrité de la relation R*, des *dépendances fonctionnelles* et va dégager des résultats formels. En conclusion de ce paragraphe nous allons étendre la problématique au cas d'une relation R munie de règles d'intégrité qui ne sont pas toutes des dépendances fonctionnelles.
D'après les résultats du paragraphe précédent, les règles d'intégrité non triviales définies sur une des relations R_i de la décomposition D, sont des dépendances fonctionnelles F_i obtenues par la projection sur R_i^+ des df de la base complète F^* des df définies sur R ou appartenant à la fermeture $(F_i)^{**}$.

6.3.1. Cohérences locale et globale d'une base de données: décomposition franche

Nous allons supposer que chaque relation R_i de la décomposition dispose d'un mécanisme de validation des df de F_i. Ainsi toute modification d'une instance iR_i validant F_i la transforme dans une autre instance validant F_i (*cohérence locale*).
Il s'agit maintenant de savoir si la composition des différentes instances iR_i est une instance de R validant F (*cohérence globale*).

Soit $iR = \overset{n}{\underset{j=1}{*}} iR_j$; iR valide $\overset{n}{\underset{i=1}{\cup}} F_i$ (propriété ridf4).

Par construction $\overset{n}{\underset{i=1}{\cup}} F_i \subseteq F^*$.

DÉFINITION

iR valide F seulement si $(\overset{n}{\underset{i=1}{\cup}} F_i)^{**} = F^{**}$.

Dans ce cas la décomposition est qualifiée de *franche*: les cohérences locales suffisent pour garantir la cohérence globale.

Sinon il existe les df de: $F^{**} - (\bigcup_{i=1}^{n} F_i)^{**}$ qui ne sont pas validées par les mécanismes de validation de cohérence locale. Il faut mettre en place des mécanismes de validation de cohérence globale, pour les valider.

Bien sûr, la solution la plus simple est de choisir une décomposition franche.

Propriété:
Une décomposition obtenue par l'application du théorème (decdf2), est franche si et seulement si elle est construite à partir d'une couverture de l'ensemble des df définies sur R.

La preuve est immédiate.

Exemple:

Soit R(VOL AVION PILOTE) de prédicat (vol avion pilote) est une entité de R si ce vol est assuré par cet avion qui est piloté par ce pilote.
Il existe deux df:
VOL → AVION : un vol est assuré par un seul avion;
AVION → PILOTE : un avion est piloté par un seul pilote;
et par transitivité on en déduit VOL → PILOTE.
Ces trois df forment la base complète.

D'après le théorème (decdf1) et en se servant de la df VOL → AVION, on obtient la décomposition suivante:
R = R[VOL AVION] * R[VOL PILOTE].

Ici R_1 = R[VOL AVION] et F_1: VOL → AVION,
et R_2 = R[VOL PILOTE] et F_2: VOL → PILOTE.
$F^* - (F_1 \cup F_2)^* = \{AVION \rightarrow PILOTE\}$

La cohérence locale ne suffit pas pour garantir la cohérence globale comme le montre l'exemple suivant:

iR_1	VOL	AVION	iR_2	VOL	PILOTE
	SR999	X366		SR999	Laverdure
	AF001	X366		AF001	Tanguy

Ces deux instances sont cohérentes localement mais iR n'est pas cohérente car elle ne valide pas AVION → PILOTE:

iR	VOL	AVION	PILOTE
	SR999	X366	Laverdure
	AF001	X366	Tanguy

Ainsi une modification de iR_1 (resp. iR_2) ne doit pas seulement valider F_1 (resp. F_2) mais également la df AVION → PILOTE en tenant compte des entités de iR_2 (resp. iR_1).

6.3.2. DF imprimées dans la structure: décomposition lisse

Nous allons supposer que pour chaque instance iR_i un dispositif informatique garantisse que deux entités ne peuvent prendre les mêmes valeurs pour chacune des clés K_i de R_i. Ce dispositif est classique dans les systèmes de gestion de fichiers lorsque R_i a une seule clé. Dans ce cas, toutes les df de type $K_i \rightarrow A_i$ (A_i constituant de R_i) sont validées par ce dispositif et il n'est plus utile de se soucier de leur validation. Nous dirons que ces df sont *imprimées* dans la structure.

Une relation R_i est en *troisiéme forme normale de Boyce Codd Kent* (FNBCK) si toutes les df élémentaires définies sur R_i sont de la forme $K_i \rightarrow A_i$ où K_i est une clé de R_i et A_i un constituant de R_i.
Dans les relations en FNBCK, les df élémentaires sont imprimées dans la structure. Aussi va-t-on chercher à décomposer une relation R dans un ensemble de relations en FNBCK. Il existe ainsi un type remarquable de décomposition: une décomposition *lisse* est une décomposition franche formée de relations en FNBCK.

Malheureusement une décomposition lisse n'existe pas pour toute relation comme le montre l'exemple suivant:
soit R(ABC) avec les df suivantes: AB → C et C → B.
Cet ensemble est la base complète des df définies sur R. Les clés de R sont AB et AC.
La décomposition R = R[AC] * R[CB] n'est pas franche: la df AB → C n'est définie dans aucune des relations de la décomposition. La relation R elle-même n'est pas en FNBCK à cause de la df C → B. Il n'existe pas de décomposition lisse de cette relation.

Remarque:

Pour toute décomposition *non lisse* il existe au moins une df de F* qui demande une validation spécifique de cohérence.

Décomposition d'une relation

Propriété:

Toute décomposition D formée de relations en FNBCK par application du théorème (decdf2) a toujours pu être formée à partir d'une *base* de F. Comme le montre l'exemple suivant, elle peut être construite à partir d'un ensemble de df qui ne soit pas une base.

Preuve:
L'ensemble des df f_i: $K_i \rightarrow A_i$, où K_i est une clé de R_i et A_i un constituant de R_i, forme une couverture de F puisque D est construite à partir de (decdf2) et est franche.
Comme R_i est en FNBCK, f_i est une df élémentaire et cette couverture est en fait une base.

Exemple:
Soit R(ABCD), F = {AB → C; A → D; D → B}.
On décompose R suivant F; la clé de R est A;
R = R[ABC] * R[AD] * R[DB].
Ces trois relations sont en FNBCK.
Pourtant F n'est pas une base.
Voici la base complète: F* = {A → B; A → C; A → D; D → B}.
La décomposition précédente peut être également obtenue à partir de cette base.

Remarque:
Une base ne conduit pas toujours à une décomposition en FNBCK.

Exemple:
Soit R(ABC) munie de F* = {AB → C; C → B}
Les clés de R sont AB et AC;
R se décompose en R_1 = R[ABC] et R_2 = R[CB].
R_1 n'est pas en FNBCK.

6.3.3. Dépendances fonctionnelles à risque: décomposition étanche

Le point de départ de notre réflexion porte sur *l'obtention de toutes les entités de la relation R[X]* (X inclus dans R^+) à partir des instances des relations R_i de la décomposition de R. Le *seul algorithme actuellement autorisé* demande d'abord de composer toutes les instances des relations R_i et ainsi d'obtenir l'instance de R, et ensuite de la projeter sur X.

Cet algorithme que nous qualifions de rudimentaire a 3 inconvénients majeurs:
- il est extrêmement lent car déjà la composition de seulement deux instances est une opération généralement coûteuse en temps d'exécution;
- il est extrêmement dangereux: en effet, si un utilisateur veut toutes les entités de R_i, il doit d'abord former l'instance iR par composition puis la projeter sur R_i^+. Jamais un utilisateur n'aura l'idée extravagante d'appliquer un tel algorithme alors qu'il dispose d'un ordre très simple pour extraire l'instance iR_i s'il utilise un SGBD.
- il perd de l'information: en effet si $iR = \overset{n}{\underset{i=1}{*}} (iR_i)$

alors $(iR)[R_i^+] \subseteq iR_i$ (propriété (di)).

Exemple:

INVENTAIRE(MATÉRIEL PIÈCE IMMEUBLE DÉPARTEMENT).
(m p i d) est une entité d'INVENTAIRE si le matériel m se trouve dans la pièce p de l'immeuble i et s'il appartient au département d dont les bureaux se trouvent dans l'immeuble i.
Voici l'ensemble F des df définies dans INVENTAIRE:
f_1: MATÉRIEL → PIÈCE,
f_2: MATÉRIEL → DÉPARTEMENT,
f_3: PIÈCE → IMMEUBLE,
f_4: DÉPARTEMENT → IMMEUBLE.
INVENTAIRE admet une seule clé: MATÉRIEL.

Pour application du théorème (decdf2), INVENTAIRE peut se décomposer suivant les relations:
R_1 = R[MATÉRIEL PIÈCE],
R_2 = R[MATÉRIEL DÉPARTEMENT],
R_3 = R[PIÈCE IMMEUBLE],
R_4 = R[DÉPARTEMENT IMMEUBLE].

La base complète de F: F* est formée de f_1 f_2 f_3 f_4
et f_5: MATÉRIEL → IMMEUBLE.
Voici donc les ensembles de df définies sur les R_i:
$F_1=\{f_1\}$, $F_2=\{f_2\}$, $F_3=\{f_3\}$, $F_4=\{f_4\}$.

Cette décomposition est franche puisqu'elle a été obtenue à partir d'une couverture de F.

Elle est lisse car toutes les relations R_i sont en FNBCK.

Décomposition d'une relation

La composition des instances des relations R_i fournit une instance de R validant F sans validation spécifique de la cohérence globale.

En voici un exemple:

MATÉRIEL	PIÈCE
vx78001	l311
mac015	c304
mbi001	u100
cry001	a500

MATÉRIEL	DÉPARTEMENT
vx78001	cui
mac015	hec
mbi001	université
cry001	phy

PIÈCE	IMMEUBLE
l311	ruedulac
c304	candolle
u100	uni2
a500	expo

DÉPARTEMENT	IMMEUBLE
cui	ruedulac
hec	candolle
phy	arve

a) Si l'on veut obtenir la pièce où se trouve le matériel vx78001, le seul algorithme dont nous disposons pour l'instant, est rudimentaire: il fait la composition des instances des relations R_1 R_2 R_3 et R_4, ensuite sélectionne l'entité relative au matériel vx78001 et finalement retient que la valeur de cette entité pour pièce: L311. C'est un algorithme bien long !

b) Et pourtant si l'on accepte que l'utilisateur cherche la pièce d'un matériel directement dans l'instance de R_1, on obtient un résultat certes rapide mais peu fiable. Il est juste pour le matériel vx78001 mais il est entaché de *contradiction* pour le matériel cry001 et devient par là même *dangereux*: en effet ce matériel est associé d'une part à l'immeuble expo où se trouve la pièce a500 et d'autre part à l'immeuble arve où se trouve le département PHY, son propriétaire. Or un matériel doit être associé à un seul immeuble. Il existe donc une contradiction qui peut provenir aussi bien des entités (cry001 phy), (a500 expo) (phy arve) que de l'entité (cry001 a500). L'une au moins de celles-ci est fausse. Aussi à notre avis, la réponse "la pièce du cry001 est a500" obtenue à partir de R_1 est-elle dangereuse; elle n'est pas acceptable.

Par contre, l'algorithme rudimentaire construit l'instance iR suivante:

vx78001	l311	cui	ruedulac
mac015	c304	hec	candolle.

Il ne trouve aucune pièce pour le matériel cry001.

c) Si l'algorithme rudimentaire élimine les contradictions en construisant iR, par contre il peut faire perdre également de l'information. Ainsi dans l'exemple précédent il fait perdre la pièce du matériel mbi001 alors que cette information n'est entachée d'aucune contradiction.
fex.

Pour éviter l'utilisation de cet algorithme rudimentaire, il faut que la base de données vérifie au contraire le principe simple suivant: "*les entités de* $R[R_i^+]$ *sont les entités de* iR_i".
Dans ce but il faut qu'aucune des entités r_i de iR_i ne puisse être *en contradiction* avec une entité obtenue par composition d'entités d'autres instances.

Il ne doit donc exister aucune df f: X → A qui est définie à la fois sur R_i et sur une décomposition partielle ne contenant pas R_i. Le concept essentiel est alors celui de *dépendance fonctionnelle à risque.*

DÉFINITION (LUONG86)

Une dépendance fonctionnelle f est une *dépendance fonctionnelle à risque* dans D si et seulement si il existe deux décompositions fonctionnelles partielles D_1 et D_2 de D telles que:
$f^+ \subseteq D_1^+$ et $f^+ \subseteq D_2^+$,
et il peut exister une instance de la base telle que:
$iD_1[f^+] \cup iD_2[f^+]$ ne valide pas f_1.

Exemple:

Dans la décomposition proposée d'INVENTAIRE,

f_3: PIÈCE → IMMEUBLE est une df à risque:
$D_1 = \{R_3\}$ et $D_2 = \{ R_1 R_2 R_4\}$.

f_4: DÉPARTEMENT → IMMEUBLE en est une autre:
$D_1 = \{R_4\}$ et $D_2 = \{ R_1 R_2 R_3\}$.
fex.

Ainsi même si les df sont validées au niveau de chacune des relations de D, il n'empêche que $(iD_1[f^+] \cup iD_2[f^+])$ peut ne pas vérifier f_1:
iD_1 et iD_2 contiennent alors des *contradictions* dans le sens que la valeur *x* de X peut être associée à la valeur *a* de A dans iD_1 alors qu'elle est associée à la valeur *a'* dans iD_2. Ainsi les *df à risque* sont *sources de contradictions.*

Décomposition d'une relation

Propriété:
Les décompositions fonctionnelles obtenues à partir d'une couverture de F par l'application du théorème (decdf2) admettent toutes les df redondantes comme df à risque.

Preuve:
Soit C une couverture de F.
Soit f_1 une df de C qui est redondante: ainsi C' = C-f_1 est aussi une couverture de F.
Soit D, resp. D', la décomposition de R obtenue à partir de C (resp. C') par application du théorème (decdf2). D contient une relation R_1 construite à partir de f_1. D' ne contient pas R_1 et D' est une décomposition partielle de D. Comme f_1 peut être générée à partir de C', f_1^+ est inclus dans D'^+.
Donc f_1 est une dépendance fonctionnelle à risque où les deux décompositions concernées sont $D_1 = R_1$ et $D_2 = D'$.

Décomposition étanche:

Si un utilisateur souhaite extraire de la base les entités de $R[R_i^+]$ il doit pouvoir très simplement extraire l'instance iR_i de R_i et obtenir ainsi un résultat fiable et complet.

Malheureusement, la décomposition de R construite à partir d'une couverture irredondante ne fournit pas toujours une réponse complète à ce problème comme le montre l'exemple de la décomposition de la relation INVENTAIRE; les df f_1,f_2,f_3,f_4 forment la base irredondante de F. Pourtant nous avons montré que la base de données construite suivant cette décomposition peut contenir des contradictions.
Aussi allons nous étudier les décompositions qui permettent de résoudre le problème fondamental précédent: les entités de $R[R_i^+]$ sont celles de iR_i et iR_i les contient toutes. Nous appelons de telles décompositions, des *décompositions étanches.*

6.3.4. Synthèse et généralisation

6.3.4.a. Synthèse des résultats concernant une relation R munie seulement de df comme règles d'intégrité.

La propriété d'une décomposition d'être *franche* permet d'assurer la cohérence globale de la base de données à partir des seules cohérences

locales. Elle nécessite que les df définies dans chacune des relations forment une *couverture de F*. Nous allons donc considérer les décompositions de R obtenues par l'application du théorème (decdf2).
Les relations en FNBCK facilitent les contrôles de validation de leurs df. Nous allons donc tenter d'obtenir des décompositions formées de relations en FNBCK: nous considérons donc les décompositions de R obtenues à partir d'une *base de F* par l'application du théorème (decdf2).

Les df à risque sont sources de contradictions et interdisent un algorithme bien simple d'extraction de données de la base. Les df redondantes d'une base sont des df à risque pour la décomposition construite à partir de cette base. Nous considérons les décompositions de R obtenues à partir d'une *base irredondante de F* par l'application du théorème (decdf2). Une telle décomposition est appelée *décomposition directe* de R.

Une *décomposition directe* constitue la solution habituellement proposée (DELOBEL-CASEY73) (BERNSTEIN76) (ADIBA-DELOBEL-LÉONARD76) (FLORY82). Mais comme le montre l'exemple INVENTAIRE, une décomposition directe n'est pas pour autant étanche. C'est la raison pour laquelle nous allons proposer une autre solution dans le chapitre suivant.

6.3.4.b. Généralisation

La qualité d'une décomposition d'être *franche* signifie que la validation des ri définies sur les relations R_i est suffisante pour garantir la cohérence de: $iR = \overset{n}{\underset{i=1}{*}} (iR_i)$ par rapport aux règles d'intégrité de R.

Dans le cas général, la difficulté d'obtenir des résultats formels provient à notre avis de la nécessité d'exprimer la projection et l'extension d'une règle d'intégrité quelconque; dans le cas des df, la projection et l'extension s'expriment simplement à l'aide de df, ce qui simplifie la validation des instances iR_i. Les résultats que nous avons dans le cas des df sont obtenus grâce à notre connaissance d'un système valide et complet de dérivation de df.

La notion de *règle d'intégrité imprimée dans une structure* est toute aussi importante; une ri est imprimée dans une structure si sa validation ne repose sur aucun programme spécifique. Une dépendance de composition peut être imprimée dans une structure comme le montre l'exemple de la structure commerciale (introduction).

Décomposition d'une relation

La notion de *décomposition étanche* est toute aussi importante. Il s'agit de savoir si iR_i contient seulement des entités de $R[R_i^+]$ et si iR_i les contient toutes. Il faut que ces deux conditions soient remplies pour que l'utilisateur obtienne un résultat fiable et complet quand il va chercher les entités de $R[R_i^+]$ dans iR_i sans faire la composition de toutes les instances. Dans de nombreux cas, cette propriété est essentielle.

Si la décomposition n'est pas étanche, il faut la rendre étanche par le déclenchement de programmes spéciaux lors de modifications de la base. La décomposition est alors *artificiellement étanche*. Nous abordons ce problème dans le chapitre suivant.

6.4. ANNEXE

THEOREME (decdf2)

Une relation R munie d'un ensemble de df $F = \{f_1 ... f_i ... f_n\}$ admet la dépendance de composition (dc) suivante:
K désignant une clé de R,
f_i^+ désignant les constituants de f_i,

$$R = \left(\overset{n}{\underset{i=1}{*}} R[f_i^+] \right) * R[K].$$

On nettoye la décomposition en éliminant R[K] si nécessaire.

Preuve:

Dans cette preuve, nous ne travaillons qu'avec des instances de relation comme il n'y a aucun risque d'ambiguïté, nous les notons R_1, R_i et non iR_1, iR_i par simplification.

a) Comme $\forall A \in R^+$, $K \rightarrow A$ appartient à F^{**}; donc $K \rightarrow A$ peut être générée par les df de F.

Ainsi $$R^+ \subseteq \bigcup_{i=1}^{n} f_i^+ \cup K.$$

Comme par construction, la réciproque est vérifiée,

on a donc $$R^+ = \bigcup_{i=1}^{n} f_i^+ \cup K.$$

b) Soit $R_i = R[f_i^+]$ et $RK = R[K]$. D'après la propriété initiale (d) on sait qu'au niveau des relations et ainsi également au niveau des instances on a:

$$R \subseteq (\ *_{i=1}^{n} R_i\ ) * R_k.$$

Il suffit de démontrer la réciproque.

c) Pour poursuivre la démonstration au point d), nous allons construire des ensembles de df F_o F_1...F_n vérifiant les propriétés c1, c2 et c3.
Soit $K_o = K^+$ (ensemble des constituants clés de R).
$F_o = \{ f \in F / f: X \rightarrow A \text{ et } X \subseteq K_o \}$;
$T_o = \{ A \in R^+ - K_o / \exists f: X \rightarrow A \text{ et } f \in F_o \}$;
$K_1 = K_o \cup T_o$.

Et par récurrence: $F_j = \{ f \in F / f: X \rightarrow A \text{ et } X \subseteq K_j \}$;
$T_j = \{ A \in R^+ - K_j / \exists f: X \rightarrow A \text{ et } f \in F_j \}$;
$K_{j+1} = K_j \cup T_j$.

(c1) Il existe un rang p tel que $K_p = K_{p+1}$.
En effet, la suite des K_j est croissante et bornée (par R^+): elle se stabilise donc.

(c2) T_j est vide si et seulement si $R^+ = K_j$.
En effet, si $R^+ = K_j$ alors T_j est vide par construction.
Réciproquement: si K_j est strictement inclus dans R^+, T_j n'est pas vide.
En effet soit $A \in R^+ - K_j$, $K_j \rightarrow A$ appartient à F^{**} car K_j contient une clé de R^+.

Soit $K' \rightarrow A$ de F^* et $K' \subseteq K_j$.
Il existe une génération dans F de $K' \rightarrow A$, d'après le théorème de génération, suivant un arbre de génération (f_1 f_2...f_p) où f_p est de type $X_p \rightarrow A_p$ avec $A_p = A$ et où $f_1: X_1 \rightarrow A_1$ vérifie que X_1 est inclus dans K' donc dans K_j.
Si A_1 n'est pas un constituant de K_j, la preuve que T_j n'est pas vide est apportée.

Sinon $K_1' = A_1 \cup K'$.

Par récurrence on suppose que tous les constituants A_1... A_{e-1} appartiennent à K_j et que $f_e: X_e \rightarrow A_e$ appartiennent à T_j ($e < p$).

X_e est inclus dans K_{e-1}' et donc dans K_j. Si A_e n'est pas un constituant de K_j alors T_j n'est pas vide; sinon on forme $K_e' = A_e \cup K_{e-1}'$ qui est inclus

dans K_j. De proche en proche on trouve un constituant A_m (éventuellement m = p) et alors $A = A_p$ n'appartenant pas à K_j. Ainsi T_j n'est pas vide.

(c3) T_p est vide car $R^+ = K_p$.

d) $\forall rr \in (\overset{n}{\underset{i=1}{*}} R[f_i^+] * R[K]), \exists rr_0 \in R[K]: rr.K = rr_0,$
$\forall i \in (1,n) \ \exists rr_i \in R[f_i^+]: rr.f_i^+ = rr_i.$

Comme les instances $R[f_i^+]$ et R[K] sont des projections,
$\exists r'_0 \in R: r'_0.K^+ = rr_0, \forall i \in (1,n) \exists r'_i \in R: r'_i.f_i^+ = rr_i.$

Nous allons montrer que $r'_0 = rr$ et ainsi que rr est une entité de R.

Le raisonnement s'appuie sur le fait que les entités $r'_0 ... r'_i ... r'_n$ ne peuvent prendre des valeurs, indépendamment les unes des autres car les df $f_1 ... f_n$ sont validées dans R.
- $\forall f_i: X_i \rightarrow A_i \in F_o$, $X_i \subseteq K^+$ par construction.
Alors $r'_0.X_i = rr_0.X_i = rr.X_i = rr_i.X_i = r'_i.X_i$.
Comme f_i est validée dans R, $r'_0.A_i = r'_i.A_i = rr.A_i$.

- Il suffit d'appliquer ce même raisonnement pour les différents ensembles $F_1 ... F_{p-1}$ pour conclure que:
$\forall A \in R^+ \ r'_0.A = rr.A$ et ainsi rr est une entité de R.

7

CRITÈRES DE CHOIX D'UNE DÉCOMPOSITION FONCTIONNELLE

7.1. Introduction

Nous faisons deux critiques aux décompositions directes d'une relation qui sont obtenues à partir d'une base irredondante de df (§ 6.3.4.a): celle de privilégier des clés implicitement et celle de ne pas fournir comme résultat une décomposition étanche. Notre but est d'améliorer ces résultats.

7.1.1. Cas de relations ayant plusieurs clés

Dans le cas d'une relation ayant plusieurs clés, il y a généralement plusieurs bases irredondantes de df; l'une d'elles sert à la construction d'une décomposition directe et privilégie l'une des clés de la relation. Le concepteur ne choisit pas directement la clé privilégiée: il choisit en fait une base irredondante! Nous souhaitons proposer une méthode de conception qui mette les concepteurs devant des choix clairs: ils auront à privilégier des clés, s'ils le souhaitent, lorsque le processus de conception sera suffisamment avancé.

Nous allons illustrer notre point de vue avec les exemples suivants.

Décomposition d'une relation

Exemples:

a) *Exemple AFFECTATION*

Soit la relation AFFECTATION(Machine Heure Produit Noemp Place) de prédicat: à l'heure h (Heure) d'aujourd'hui l'employé de numéro no (Noemp) conduit la machine m (Machine) pour la production du produit p (Produit) à la place pl (Place).
a = (m h p no pl) est alors une entité de AFFECTATION c'est-à-dire une affectation.
Voici les dépendances fonctionnelles définies sur AFFECTATION:

Machine	Heure	→	Produit	Noemp,
Noemp	Heure	→	Produit	Machine,
Produit	Heure	→	Noemp	Machine.

La relation AFFECTATION admet 3 clés: Machine Heure; Noemp Heure; Produit Heure.

Pour nous (m h), (no h), (p h) sont trois manières distinctes mais équivalentes de désigner une affectation. Si aucune affectation ne peut exister sans que soient connus l'employé et l'heure concernés, la clé Noemp Heure est obligatoire.

b) *Exemple SURVEILLANCE* (prolongement de l'exemple précédent)

Soit R(Machine Heure Produit Noemp Place Nocontr Nomcontr) la relation qui associe à un numéro de contrôleur (Nocontr) son nom (Nomcontr) et qui associe à ce numéro et à une heure donnée une affectation formée d'une machine, d'un produit, d'un numéro d'employé et d'une place.
R est munie des df suivantes:

Nocontr		→	Nomcontr,		
Nocontr	Heure	→	Machine	Produit	Noemp,
Machine	Heure	→	Produit	Noemp	Place,
Produit	Heure	→	Machine	Noemp	Place,
Noemp	Heure	→	Machine	Produit	Place.

Voici une décomposition possible de R:
R = AFFECTATION * CONTROLEUR * SURVEILLANCE
où AFFECTATION = R [Machine Heure Produit Noemp Place],
CONTROLEUR = R [Nocontr Nomcontr],
SURVEILLANCE = R [Nocontr Heure Machine Produit Noemp].
Cette décomposition préserve le rôle équivalent des trois clés de AFFECTATION.

c) L'obtention d'une décomposition directe conduit dans l'exemple AFFECTATION à demander au concepteur de choisir une base irredondante parmi notamment les trois suivantes:
F1 = FC ∪ (Noemp Heure → Place),
F2 = FC ∪ (Machine Heure → Place),
F3 = FC ∪ (Produit Heure → Place),

avec FC = {Noemp Heure → Machine;
Machine Heure → Produit;
Produit Heure → Noemp}.

Si le concepteur choisit la base irredondante F1, il doit comprendre qu'il sera possible de stocker la place d'une affectation seulement si le numéro de l'employé de cette affectation est connu. Ainsi la clé Noemp Heure est privilégiée.
Comme la correspondance entre le choix d'une base irredondante et celui d'une clé privilégiée n'est pas évidente, nous préférons respecter l'équivalence des clés à ce stade du processus de conception.

7.1.2. Une décomposition directe peut ne pas être étanche

Dans l'exemple de la relation INVENTAIRE, nous avons considéré la décomposition directe d'INVENTAIRE. Voici le graphe de relation correspondant:

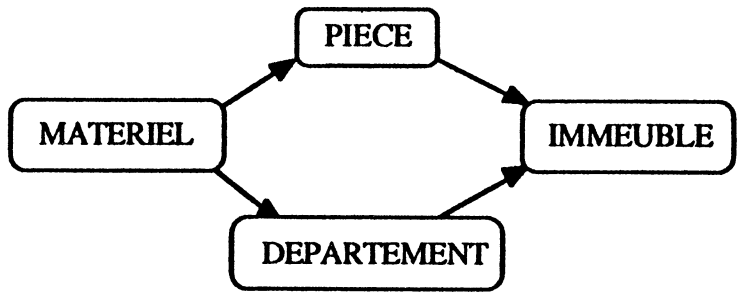

Figure 7.1. Décomposition directe non étanche

Nous avons montré (§ 6.3.3.) que pour interroger cette base de données, il est nécessaire de composer les instances des relations R_1 R_2 R_3 R_4 si l'on veut obtenir des résultats non entachés de contradictions.
La décomposition directe n'est pas étanche à cause des df PIÈCE → IMMEUBLE et DÉPARTEMENT → IMMEUBLE qui sont des df à risque sans pour autant être redondantes.

Décomposition d'une relation

Nous allons proposer des mécanismes qui rendent les décompositions artificiellement étanches. Ils vont permettre de répondre "naturellement" aux questions des utilisateurs: ainsi par exemple pour toute question portant sur R_4 = INVENTAIRE [DÉPARTEMENT IMMEUBLE] il suffira d'interroger l'instance iR_4 sans être obligé de faire la composition des instances iR_1, iR_2, iR_3 et iR_4 au préalable.

7.1.3. Notre objectif

Notre objectif est de:
- mettre en évidence des décompositions de relation qui ne privilégient aucune clé parmi plusieurs clés d'une relation. Cette partie fait l'objet du chapitre suivant;
- spécifier des mécanismes informatiques qui assurent *artificiellement* l'étanchéité des décompositions.

Dans les paragraphes suivants, nous allons donner une première spécification de ces mécanismes. Dans un chapitre suivant (§ 10.3.), nous élargissons les fonctions de ces mécanismes.

7.1.4. Hypothèses de notre étude

(h0) Toute décomposition considérée n'a jamais deux relations ayant une même clé. Les seules règles d'intégrité prises en compte sont les df intrinsèques.

(h1) Toute question à la base de données est relative à une relation R' obtenue par composition de relations: $R_1 \ldots R_m$ formant une décomposition fonctionnelle partielle (§ 6.1.3.) de R.

(h2) Chaque relation R_i de D est munie d'un mécanisme appelé *mécanisme de clés* qui valide pour toute instance iR_i les df intrinsèques de R_i.

(h3) La méthode "naturelle" d'utilisation d'une base de données est d'une part de consulter une seule relation R_i de D plutôt que de composer les relations de D, dans le cas où l'interrogation ne porte que sur R_i **(h31)**.
D'autre part, si l'interrogation porte sur des constituants formant une ou plusieurs clés de R_i, la méthode "naturelle" d'interrogation de la base considère seulement R_i sans consulter les autres relations **(h32)**.

Exemple INVENTAIRE:

INVENTAIRE(MATÉRIEL PIÈCE DÉPARTEMENT IMMEUBLE) est une relation qui se décompose suivant les relations:
R_{12}(*MATÉRIEL* PIÈCE DÉPARTEMENT),
R_3(*PIÈCE* IMMEUBLE),
R_4(*DÉPARTEMENT* IMMEUBLE).

Les hypothèses précédentes s'expriment ainsi:
(h1) Nous ne nous intéressons à aucune question concernant la relation $R_{34} = R_3 * R_4$ car la décomposition (R_3, R_4) n'est pas fonctionnelle.
(h2) Le mécanisme de clés garantit par exemple que dans iR_3 il n'existe pas deux entités relatives à la même pièce.
(h31) Si l'on veut trouver l'immeuble dans lequel sont les bureaux d'un département, on va chercher la réponse dans R_4 sans avoir besoin des relations R_{12} R_3 qui forment une décomposition fonctionnelle.
(h32) Si l'on veut trouver toutes les pièces considérées dans la base de données, on consulte R_3 et non pas R_{12}.

NOTATION

D est une décomposition de R formée des relations R_1, R_2,...,R_n. Dans tout le chapitre nous adoptons la convention suivante: T est une relation de D, K est une clé de T, f: K $\rightarrow$ A est une df définie sur T non triviale.

7.2. Problèmes rencontrés

DÉFINITION

Une décomposition D_1 est *concernée* par Y (Y inclus dans R^+) si D_1 est une décomposition fonctionnelle partielle qui admet Y dans D_1^+.

7.2.1. (pb1) Complétude de clés

Une décomposition D_1, concernée par K, *ne déborde par pour K* si pour toute instance de la base, $iD_1[K] \subseteq iT[K]$.

Lemme (1):
Une décomposition D_1, concernée par K, ne déborde pas pour K si et seulement si elle contient T.

Décomposition d'une relation

Preuve:
la condition est suffisante: si D_1 contient T, alors $iD_1[T^+] \subseteq iT$ (propriété (di) § 1.2.) et ainsi $iD_1[K] \subseteq iT[K]$.
Réciproquement:
si T n'est pas une relation de D_1, il existe une instance de la base telle que:
- les instances des relations n'appartenant pas à D_1 sont vides;
- les instances des relations appartenant à D_1 contiennent toutes une seule entité prenant la valeur 1 pour chaque constituant.
Pour chaque instance, les df intrinsèques sont bien vérifiées puisqu'il y a au plus une entité par instance. Ici $iD_1[K]$ n'est pas inclus dans iT[K].

Remarque:
s'il existe une décomposition D_1 débordante pour K alors l'obtention de iR[K] demande de faire $iD_1[K] \cup iT[K]$: cette méthode s'oppose à la méthode "naturelle" d'utilisation de la base (h32).

7.2.2. (pb2) Complétude d'entités

Une décomposition D_1 concernée par AK ne déborde pas pour AK (ou pour f: K → A) si pour toute instance de la base,
$iD_1[AK] \subseteq iT[AK]$.

Lemme (2):

Une décomposition D_1 concernée par AK ne déborde pas pour AK si et seulement si elle contient T.
La démonstration est identique à celle du lemme précédent.

Remarque:
s'il existe une décomposition D_1 débordante pour AK alors l'obtention de iR[AK] demande de faire $iD_1[AK] \cup iT[AK]$. Cette méthode s'oppose à la méthode "naturelle" d'utilisation de la base (h31, h32).

7.2.3. (pb3) Validité d'une décomposition fonctionnelle partielle D'

D'^+ contient KA.
D' valide f: K → A si et seulement si pour toute instance de la base, iD' valide f.
Si D' contient T, D' valide f car iD'[KA] est contenue (propriété (di))

dans iT[KA] qui valide f par application du mécanisme des clés (h2).
Par contre, si D' ne contient pas T, D' ne valide pas f dans le cas général comme l'exemple suivant le montre.

Exemples:
- R = (*S*KB) * (*K*A) * (*B*A) (décomposition D de R).
D' = {(*S*KB), (*B*A)}.

D' ne valide pas K → A car il existe les instances suivantes des relations de D:

s k b	k a	b a
s' k b'	k a	b' a'

- Par contre soit: R = (BCADZ) * (*B*E) * (*EC*A).
La première relation T admet deux clés BC et AD et la df f: BC → A.

D' = {(*B*E), (*EC*A)} ne contient pas T.
Pourtant D' valide f: BC → A puisqu'elle valide B → E et EC → A.

7.2.4. (pb4) Compatibilité de décompositions fonctionnelles partielles pour une df

Deux décompositions fonctionnelles partielles D_1 et D_2 concernées par f sont *compatibles pour f* si et seulement si pour toute instance de la base, $iD_1[f^+] \cup iD_2[f^+]$ valide f. Elles sont *incompatibles pour f* sinon.

Lemme (3):
Toute décomposition fonctionnelle partielle D_1 concernée par f est compatible avec T pour f si et seulement si D_1 contient T.

Preuve:
si D_1 contient T, alors $iD_1[KA]$ est inclus dans iT[KA] (propriété (di)) et ainsi $iD_1[KA] \cup iT[KA]$ est égal à iT[KA] et valide f.
Réciproquement:
si T n'est pas une relation de D_1, il est possible de considérer l'instance suivante de la base:
- toutes les instances des relations de D n'appartenant pas à D_1 sont vides, sauf celle de T;
- T contient une seule entité qui prend la valeur 1 pour tous ses constituants sauf pour A où elle prend la valeur 0;
- l'instance de chaque relation de D_1 contient une seule entité prenant la valeur 1 pour chaque constituant.

Décomposition d'une relation

Les df intrinsèques sont vérifiées dans chaque instance car chacune contient au plus une entité.
- $iD_1[f^+]$ contient seulement une entité prenant la valeur 1 pour les constituants de K et la valeur 1 pour A;
- $iT[f^+]$ contient seulement une entité prenant la valeur 1 pour les constituants de K et la valeur 0 pour A;
- $iD_1[f^+] \cup iT[f^+]$ ne valide pas f. Ce qui est impossible par hypothèse. T appartient donc à D_1.

7.3. Eléments structurels

DÉFINITION

Une décomposition D_1 est *minimale pour* Y (respectivement pour f) si et seulement si c'est une décomposition fonctionnelle partielle de D concernée par Y (respectivement par f) et la suppression d'une relation de D_1 conduit à un ensemble de relations qui soit n'est plus une décomposition fonctionnelle partielle de D, soit reste une décomposition fonctionnelle partielle mais n'est plus concernée par Y (respectivement par f).

Propriété (1):

Si D_1 déborde pour K et si D_{11} est incluse dans D_1 et est concernée par K, alors D_{11} déborde pour K.

Preuve:

Comme, si D_{11} est incluse dans D_1, $iD_1[K] \subseteq iD_{11}[K]$ et ainsi si $iD_1[K]$ n'est pas contenu dans iT[K], il en est de même pour $iD_{11}[K]$

DÉFINITION

K est une *clé à risque linéaire* si toutes les décompositions minimales pour K qui débordent pour K sont réduites à une seule relation.
Elle est une *clé à risque cyclique* sinon.

Propriété (2):
Si D_1 déborde pour AK et si D_{11} est incluse dans D_1 et est concernée par AK, alors D_{11} déborde pour AK.

La preuve s'établit de manière identique à celle de la propriété précédente.

Propriété (3):

Si D_1 ne valide pas f alors toute décomposition D_{11} incluse dans D_1 et concernée par f ne valide pas f.

Preuve:

$D_1[KA] \subseteq D_{11}[KA]$ d"après la propriété (di).
Si $D_1[KA]$ ne valide pas f, il existe une instance de la base telle que $iD_1[KA]$ ne valide pas f et alors l'instance de D_{11} ne valide pas f. D_{11} ne valide pas f.

Propriété (4):

Si deux décompositions D_1 et D_2 sont incompatibles pour f, alors toutes décompositions D_{11} incluse dans D_1 , et D_{22} incluse dans D_2 , concernées par f, sont incompatibles pour f.

Preuve:

Comme D_1 et D_2 sont incompatibles pour f, il existe deux instances iD_1 et iD_2 telles que $iD_1[f^+] \cup iD_2[f^+]$ ne valide pas f.
Soit iD_{11} et iD_{22} les instances de D_{11} et D_{22} qui ont servi à construire respectivement iD_1 et iD_2.
D'après la propriété (di), $iD_1[f^+] \subseteq iD_{11}[f^+]$ et $iD_2[f^+] \subseteq iD_{22}[f^+]$.
Ainsi, $iD_{11}[f^+] \cup iD_{22}[f^+]$ contient $iD_1[f^+] \cup iD_2[f^+]$ et ne valide pas f.

Propriété (5):

f est une df à *risque* (§ 6.3.3.) si et seulement si il existe deux décompositions minimales pour f incompatibles pour f.

Preuve:
la condition est suffisante: si deux telles décompositions existent, f est une dépendance à risque par définition.
Réciproquement:
si f est une df à risque, il existe deux décompositions partielles D_1 et D_2 incompatibles pour f. Il existe une décomposition minimale pour f incluse dans chacune d'elles: D_{11} et D_{22}; d'après la propriété précédente D_{11} et D_{22} sont incompatibles pour f et ne peuvent donc être identiques.

Décomposition d'une relation

Théorème (1):

f est une dépendance à risque si et seulement si il existe une décomposition D_1 minimale pour f qui soit incompatible avec T pour f.

Preuve:
la condition est suffisante à cause de la définition d'une df à risque.
Elle est nécessaire:
d'après la propriété précédente, f est à risque si et seulement si il existe deux décompositions minimales D_1 et D_2 pour f_1 incompatibles pour f. T est une décomposition minimale pour f_1. Si D_1 et T sont compatibles pour f_1 alors d'après le lemme 3, T est un élément de D_1; comme D_1 est minimale, D_1 se réduit à T. T et D_2 sont alors incompatibles pour f par hypothèse.
Sinon D_1 et T sont incompatibles pour f.

DÉFINITIONS

Une df f est à *risque cyclique* si et seulement si il existe deux décompositions minimales pour f et incompatibles pour f dont l'une au moins n'est pas réduite à une seule relation de D.

Une df f est à *risque linéaire* si et seulement si toute paire de deux décompositions minimales et incompatibles pour f est en fait formée de deux relations de D.

7.4. Une solution: mécanismes de complétude de clés et d'entités

Pour permettre l'utilisation "naturelle" de la base de données, il faut garantir (h3) que:

- toute décomposition D_1 concernée par toute clé K de T et débordante pour K vérifie que pour toute instance de la base $iD_1[K] \subseteq iT[K]$,

- et toute décomposition D_2 concernée par KA débordante pour KA vérifie que pour toute instance de la base $iD_2[KA] \subseteq iT[KA]$.

D'après les propriétés structurelles précédentes relatives d'une part à la complétude de clés et d'autre part à la complétude des entités, il suffit en fait de garantir que:
- toute décomposition D_1 minimale pour K vérifie que pour toute instance de la base $iD_1[K] \subseteq iT[K]$
- et toute décomposition D_2 minimale pour KA vérifie que pour toute instance de la base $iD_2[KA] \subseteq iT[KA]$.

Mais dans le cas général, ces propriétés ne peuvent être vérifiées qu'"artificiellement", c'est-à-dire que par déclenchement de deux mécanismes, le *mécanisme de complétude de clés* et le *mécanisme de complétude d'entités*, lors des modifications de la base puisque nous privilégions les interrogations par rapport aux modifications (h3). Nous disons que ces deux mécanismes rendent *artificiellement* non débordantes les décompositions D_1 et D_2 précédentes.

Le mécanisme de complétude d'entités va dans le cas précédent contrôler que chaque nouvelle valeur k de $iD_1[K]$ se trouve dans iT[K]; si elle n'y est pas, il crée une nouvelle entité t telle que t.K = k et il complète ainsi T.
Le mécanisme de complétude d'entités va contrôler que chaque nouvelle entité (xa) de $iD_2[KA]$ se trouve dans iT[KA]; si elle n'y est pas, il crée une nouvelle entité t de T telle que t.KA = ka. La *limite* d'application de ce mécanisme est qu'il ne concerne que les df intrinsèques.
Nous disons que ces deux mécanismes rendent *artificiellement* non débordantes les décompositions D_1 et D_2 précédentes respectivement pour K et KA.

Propriété:

Le mécanisme de complétude d'entités rend toute décomposition D_1 concernée par f, *artificiellement valide* pour f et toute décomposition D_2 incompatible avec T pour f, *artificiellement compatible* avec T et *artificiellement non débordante pour T.*
Pour toute instance de la base on a: $iD_2[KA] \subseteq iT[KA]$.

Preuve:

Si D_1 est non valide pour f, elle ne contient pas T. L'application du mécanisme de complétude d'entités force le fait que pour toute instance de la base $iD_1[KA] \subseteq iT[KA]$. Comme iT[KA] valide f à cause du mécanisme de clés (h2), $iD_1[KA]$ valide également f.
Ainsi D_1 est forcée de valider f.
Si D_2 n'est pas compatible avec T pour f, elle ne contient pas T.

Décomposition d'une relation

L'application du mécanisme de complétude d'entités force le fait que pour toute instance de la base $iD_2[KA] \subseteq iT[KA]$.
Ainsi $iD_2[KA] \cup iT[KA]$ est égal à $iT[KA]$ et valide f. D_2 est forcée d'être compatible avec T pour f_1.

Structure d'une décomposition:

Les propriétés précédentes (2,3,4) et le théorème (1) montrent que pour une df $K \rightarrow A$ intrinsèque d'une relation T, il suffit de garantir artificiellement que les décompositions minimales par rapport à f ne débordent pas pour KA, valident f et soient compatibles avec T pour f, pour que toutes les autres décompositions fonctionnelles contenant KA aient les mêmes propriétés.

DÉFINITION

Les décompositions minimales contenant KA non réduites à une seule relation sont qualifiées de *décompositions minimales cycliques pour f (ou pour KA).*

La propriété précédente (1) montre que pour une clé K d'une relation T, il suffit de garantir artificiellement que les décompositions minimales par rapport à K ne débordent pas pour K, pour que toutes les autres décompositions contenant K aient la même propriété.

DÉFINITION

Les décompositions minimales contenant K non réduites à une seule relation sont qualifiées de *décompositions minimales cycliques pour K.*

(LUONG86) a montré que si le graphe de relation obtenu à l'aide de l'algorithme du chapitre 4 à partir de la décomposition initiale, ne contient aucun cycle fonctionnel alors les décompositions minimales ne sont pas cycliques.

Nous allons dresser la liste des décompositions minimales cycliques pour chaque relation T de la décomposition compacte (§ 4.0.) et pour chaque clé K de T. Pour chacune d'elles nous notons:
la relation source,
les relations qui la composent,
la relation pour laquelle elle est cyclique (c'est-à-dire la relation T),
les clés de la relation T concernées (parmi elles il y a K),
les df par rapport auxquelles elle est cyclique (il y a au moins f: $K \rightarrow A$).

Exemple:

Nous considérons une relation U qui se décompose à l'aide des relations suivantes formant une décomposition compacte de U. L'ensemble des df dérivées des clés de chacune de ces relations forme ici une *base* des df définies sur U, qui est *sans circuit.* Chaque relation a une clé dont les constituants sont en italique:

RS(*S*IJG)	RP(A)
RV(*G*ADH)	T(*BC*A)
R_1(*I*B)	R_2(*J*C)
R_3(*H*BC)	R_4(*AD*E)
R_5(B)	R_6(C).

Le graphe de relation obtenu à partir de cette décomposition à l'aide de l'algorithme GR est le suivant:

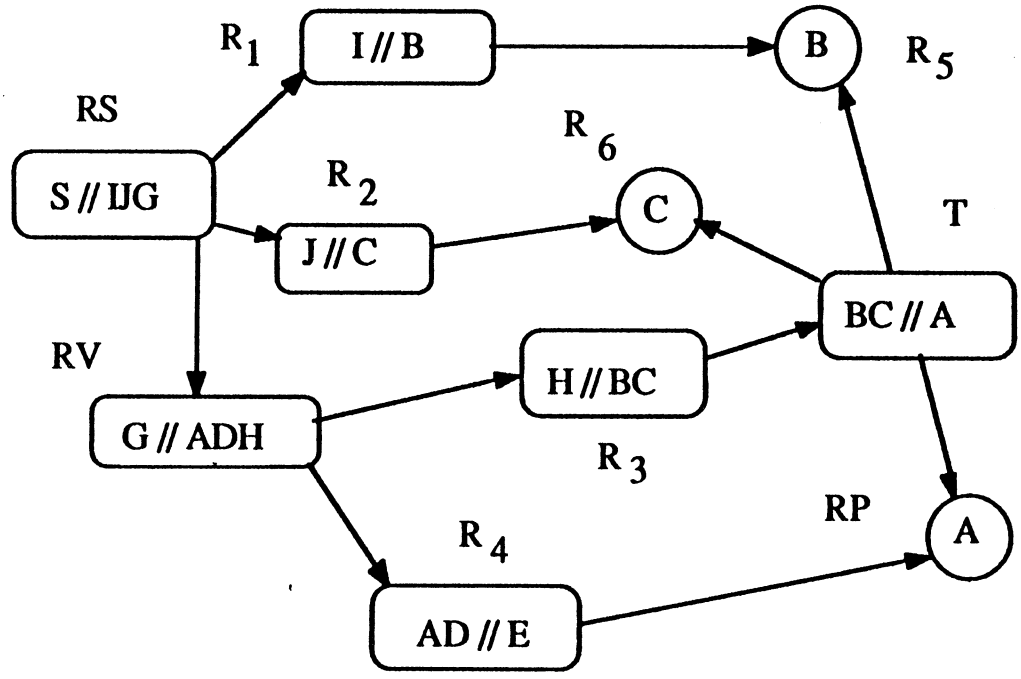

Figure 7.2. Décompositions cycliques et mécanismes

a) La décomposition minimale D_1 formée seulement de RS(SIJG) *déborde* pour I, J et G mais elle n'est pas cyclique. En effet il est possible de créer une entité (sijg) dans iRS sans que pour autant il existe une entité de clé i dans iR_1, une de clé j dans iR_2, une de clé g dans iRV.
Le mécanisme de complétude de clés va rendre obligatoire la création de ces entités ou alors interdire la création de l'entité (sijg).

b) La décomposition minimale D_2 {(SIJG), (IB), (JC)} *déborde* pour BC qui est clé de la relation T.
En effet il est possible de créer les entités (sijg) et (ib) et (jc); par composition on obtient l'entité (sijgbc) et par projection sur BC: (bc).
Sans l'intervention du mécanisme de complétude de clés, il se pourrait qu'il n'existât aucune entité dans iRT ayant pour clé bc.
Cette décomposition minimale est *cyclique* de source la relation RS, par rapport à la relation T et à sa clé BC.

c) La décomposition minimale D_3 {(SIJG), (IB), (JC), (GADH)}

- déborde bien sûr pour BC: elle ne contient pas T et contient D_2, mais D_3 n'est pas minimale pour ce fait;

- déborde pour BCA: en effet une instance possible de la base possible, sans l'intervention du mécanisme de complétude d'entités, est la suivante: (sijg), (ib), (jc), (gadh), (bc-) (- désignant une valeur obscure);

- ne valide pas f: BC → A: en effet, sans l'intervention du mécanisme de complétude d'entités, une instance possible de la base est la suivante:

(sijg) et (s'ijg') dans iRS,
(gadh) et (g'a'd'h') dans iRV,
(ib) dans iR_1 et (jc) dans iR_2.

La composition de ces entités donne deux entités dans iD_3:
(sijgbcadh)
et (s'ijg'bca'd'h');
iD_3 ne valide pas BC → A.

- n'est pas compatible avec T: sans l'intervention du mécanisme de complétude d'entités, un état possible de la base est le suivant:
(sijg), (ib), (jc), (gadh), (bca');
$iD_3[BCA] \cup iT[BCA]$ contient les entités (bca) et (bca') qui ne valident pas BC → A.

Cette décomposition minimale D_3 est *cyclique*;
sa source est la relation RS, sa cause la relation T, concernant la clé BC, et la df BC → A.

Dans cet exemple il existe d'autres décompositions minimales cycliques comme par exemple D_4 {RS, RV, R_3} de source RS, de cause R_1, concernant la clé I de R_1 et la df I → B.

7.5. Conclusions

La solution que nous proposons pour résoudre le problème des df à risque, est de rajouter dans les fonctionnalités d'un SGBD, deux mécanismes celui de la complétude de clés et celui de la complétude d'entités. Ces deux mécanismes permettent alors d'utiliser une base de données selon une méthode "naturelle" (h3) et rendent en partie la décomposition initiale *artificiellement étanche*. Ce résultat n'est que partiel car il s'applique aux seules df intrinsèques.

Dans un prochain chapitre (§ 10.3.), nous allons étendre l'application de ces deux mécanismes aux df qui ne sont pas intrinsèques, en les intégrant dans d'autres mécanismes fort utiles pour la réalisation d'applications de bases de données.

La mise en place de tels mécanismes, nécessaires dans le cas des décompositions directes même simples (comme dans l'exemple INVENTAIRE), permet de travailler avec des décompositions obtenues à partir de couvertures C de df qui ne sont pas forcément irredondantes. En effet, les df redondantes de telles couvertures sont des df à risque de la décomposition (propriété du § 6.3.3.) et leurs validations sont garanties par les mécanismes assurant artificiellement l'étanchéïté des décompositions.

L'objectif du chapitre suivant est de profiter de cette liberté pour mettre en évidence une couverture particulière de df qui n'est pas obligatoirement une base irredondante, qui permet d'obtenir automatiquement une première décomposition où toutes les clés de chaque relation y ont un rôle équivalent. Cette décomposition sera le point de départ du travail du concepteur ensuite.

8

DÉCOMPOSITION FONDAMENTALE

8.1. Introduction

L'objectif de ce chapitre est d'étudier une forme de décomposition de relation qui d'une part respecte l'équivalence des clés des relations et d'autre part facilite l'implantation des mécanismes de complétude de clés et d'entités.
Nous avons présenté les propriétés fondamentales des df dans un chapitre précédent (§ 2.4.). Après avoir démontré des propriétés mathématiques des df, nous proposons notre solution: *la* décomposition fondamentale d'une relation (elle est unique pour chaque relation).
Dans les chapitres suivants, nous intégrons la détermination de la décomposition fondamentale dans le processus de conception de base de données et nous décrivons à partir d'elle les spécifications de mécanismes qu'il faudrait implanter dans les SGBD.

NOTATION

Soit f: $X \rightarrow A$ une df.
g(f) et d(f) désignant respectivement la partie gauche (X) et la partie droite (A) de la df.

Dans tout ce chapitre, U désigne une relation sur laquelle est défini un ensemble de df F; R désigne la projection de la relation U sur l'ensemble des constituants R^+ (qui est inclus dans celui de U: U^+).

Décomposition d'une relation

Propriété 1 et définitions (rappels):

A chaque ensemble de df F, correspond une fermeture F** qui contient toutes les df qui peuvent être générées à partir de F par application des propriétés de dérivation des df et une base complète F*, ensemble des df élémentaires de F.

G est une *couverture* de F si G** = F**; cette couverture est une base si elle est incluse dans F*.

G est une *couverture irredondante* si tout ensemble G' obtenu par suppression d'une df de G, n'est pas une couverture de F.

8.2. Clés d'une relation

Nous allons étendre la définition au cas des dépendances ayant en partie droite plusieurs constituants.

DÉFINITION

$X \rightarrow Y$ est une df *élémentaire* si et seulement si:
$\forall Y_i \in Y \quad X \rightarrow Y_i \in F^*$.
$X \rightarrow Y$ est une df *quasi élémentaire* si et seulement si:
$\exists Y_i \in Y \; X \rightarrow Y_i \in F^*$ et $X \rightarrow Y \in F^{**}$.
$X \rightarrow Y$ est une df *structurellement élémentaire* si et seulement si
$\forall X' \subseteq X \quad X' \rightarrow Y \notin F^{**}$ et $X \rightarrow Y \in F^{**}$.

Remarque:

L'ensemble de ces propriétés peut être ordonné: élémentaire, quasi élémentaire, structurellement élémentaire; alors toute df admettant l'une de ces propriétés, admet les suivantes.

Exemple 3 U(ABCMN):

si $F^* = \{AB \rightarrow M; AC \rightarrow N\}$
alors $ABC \rightarrow MN$ est structurellement élémentaire;
si $F^* = \{ABC \rightarrow M; A \rightarrow N\}$ alors $ABC \rightarrow MN$ est quasi élémentaire;
si $F^* = \{ABC \rightarrow M; ABC \rightarrow N\}$ alors $ABC \rightarrow MN$ est élémentaire.

Propriété 2:

Soit K une clé de la relation R;
soit K' un sous-ensemble de R^+.
Alors K' est une autre clé de R si et seulement si:
$K \rightarrow K' \in F^{**}$, et $K' \rightarrow K$ est une df structurellement élémentaire.

Preuve:

Si K et K' sont des clés de R, alors $K \rightarrow K'$ et $K' \rightarrow K \in F^{**}$.
Soit $K'' \subseteq K'$ tel que $K'' \rightarrow K \in F^{**}$ alors $\forall X \subseteq R^+ \; K'' \rightarrow X$;
K" est une clé de R;
comme K' en est une également, K', K" sont identiques et $K' \rightarrow K$ est structurellement élémentaire.

Réciproquement:

Si $K \rightarrow K' \in F^{**}$ et $K' \rightarrow K$ est structurellement élémentaire,
alors $\forall X \in R^+ \; K' \rightarrow X \in F^{**}$ et $\forall K'' \subseteq K$
$\exists B \in K$ tel que $K'' \rightarrow B \notin F^{**}$
puisque $K' \rightarrow K$ est structurellement élémentaire.
Par conséquent, K' est une clé de R.

Propriété 3:

Soit K une clé de R ($K \subseteq R^+$).
S'il existe $K' \subseteq U^+$ tel que $K \rightarrow K'$ et $K' \rightarrow K$ appartiennent à F^{**}, alors la relation $T = U[R^+ \cup K']$ admet K et K" comme clés où $K'' \subseteq K'$.

Preuve:

S'il existe $K_1 \subseteq K$ tel que K_1 est une clé de T,
alors $K_1 \rightarrow K \in F^{**}$ et K_1 est une clé de R;
comme K est également une clé de R, $K = K_1$ et donc K est une clé de T.
Comme $K' \rightarrow K \in F^{**}$, $\exists K'' \subseteq K'$ tel que $K'' \rightarrow K$ est structurellement élémentaire.
D'après la propriété précédente, K" est également une clé de T.

DÉFINITION

Si K est une clé de R contenue dans R^+
et si $U[R^+ \cup K']$ admet comme clés K et K',
alors K' est une *clé potentielle* de R.

Décomposition d'une relation

Propriété 4:

Soit deux relations R et S projections de U respectivement sur R^+ et S^+ et telles que KR $\rightarrow$ KS et KS $\rightarrow$ KR appartiennent à F^{**} où KR et KS désignent respectivement une clé de R et une de S.
Alors KR $\rightarrow$ KS et KS $\rightarrow$ KR sont structurellement élémentaires et KR et KS sont deux clés de la relation T projection de U sur l'union de R^+ et de S^+.

Preuve:

Si KR' $\subseteq$ KR tel que KR' $\rightarrow$ KS $\in$ F^{**} alors KR' $\rightarrow$ KR $\in$ F^{**} puisque KS $\rightarrow$ KR $\in$ F^{**}.
Comme KR est une clé de R, KR = KR' et KR $\rightarrow$ KS est structurellement élémentaire.
KR $\subseteq$ R^+ comme clé de R et KR $\rightarrow$ KS $\rightarrow$ S^+; si KR' $\subseteq$ KR est clé de T, KR' $\rightarrow$ KR et KR' est une clé de R: donc KR'= KR et KR est clé de T.
KS l'est également.

Propriété 5:

Soit K_1 et K_2 deux clés de R. Soit K $\subseteq$ U^+. Si K $\rightarrow$ K_1 appartient à F^{**} et est structurellement élémentaire, alors K $\rightarrow$ K_2 appartient à F^{**} et est également structurellement élémentaire.

Preuve:
K $\rightarrow$ K_2 $\in$ F^{**} puisque K $\rightarrow$ K_1 $\in$ F^{**} et K_1 $\rightarrow$ K_2 $\in$ F^{**}.
Soit K' $\subseteq$ K tel que K' $\rightarrow$ K_2 $\in$ F^{**}, alors K' $\rightarrow$ K_1 $\in$ F^{**} et comme K $\rightarrow$ K_1 est structurellement élémentaire, K' = K:
ainsi K $\rightarrow$ K_2 est structurellement élémentaire.

DÉFINITION

R *contient toutes ses clés* si et seulement si,K étant une clé de R appartenant à R^+, $\forall$ K' $\in$ U^+ tel que K $\rightarrow$ K' $\in$ F^{**} et que K' $\rightarrow$ K est structurellement élémentaire de F^{**}, alors K' $\subseteq$ R^+.

Exemple 4:
Soit U(ABC) une relation munie de l'ensemble des df:
F = {A $\rightarrow$ C; A $\rightarrow$ B; B $\rightarrow$ A}.
R = U[AC] est une relation qui ne contient pas toutes ses clés; la clé de R est A mais on peut ajouter B à R^+;
et alors R1 = U[ABC] admet comme clé d'une part A et d'autre part B.

8.3. Classes d'équivalence minimales de df

Dans tout ce chapitre les df considérées n'ont qu'un constituant en partie droite. Les propriétés d'additivité et de projection des df font que cette hypothèse n'enlève rien à la généralité des résultats.

8.3.1. Consensus et génération de df

Les résultats de ce paragraphe sont de (GAROCHE-LÉONARD78).

DÉFINITION

Le *consensus* de deux df f: X → A et g: Y → B est, quand il existe, une nouvelle df h: Z → C obtenue par pseudo-transitivité de f et de g et notée : h = f +> g.
Aussi pour que le consensus existe, faut-il que A ∈ Y;
alors: Z = X ∪ (Y-A) et C = B.
Cette notion de consensus est celle de (KUNTZMANN72) qui l'a définie dans un contexte plus général des monômes d'une fonction booléenne.

Exemple 5:
MN → A +> AP → B = MNP → B.

Théorème de la pseudo-associativité:
soit trois df: f,g,h.
Si (f +> g) +> h existe, alors f +> (g +> h) existe également et de plus,
((f +> g) +> h) < (f +> (g +> h)).
La démonstration est fourni dans l'annexe (§ 8.11.1.).

Exemple 6:
f: M → A; g: AN → B; h: ABO → C;
i = (f +> g) +> h: MNAO → C; k = f +> (g +> h): MNO → C;
Ainsi i < k.

Lemmes relatifs au consensus:
Soit f,g,f',g' des df telles que f < f' et g < g'.
Si f +> g existe alors f' +> g existe et(f' +> g) > (f +> g).
Si f +> g' existe, alors (f +> g') > (f +> g).
Les preuves de ces lemmes sont immédiates.

Décomposition d'une relation

DÉFINITION

Une *génération* de f dans l'ensemble F de df, et à partir de la df f_0 de F, est la suite récurrente d'opérations de consensus suivantes:

$$m_0 = f_0,$$
$$m_1 = f_1 +> m_0,$$
$$m_2 = f_2 +> m_1,$$
$$\vdots$$
$$m_i = f_i +> m_{i-1},$$
$$\vdots$$
$$m_n = f_n +> m_{n-1},$$
$$f < m_n,$$

où $\forall\, i \in (1,n)\quad f_i \in F$ et $d(f_i) = d(f_0)$ et où n désigne un nombre fini (éventuellement 0). On dit alors que f_0 *génère* f dans F ou que f peut être générée à partir de f_0 ou que f est générée par $(f_0,f_1...f_n)$ dans F.

THÉORÈME DE GÉNÉRATION

A toute df de F** correspond une génération dans F.

Preuve:
Il suffit de démontrer le théorème pour les df élémentaires de F* puisque les autres se déduisent de celles-ci par augmentation qui est une génération particulière.
Par définition même de la base complète F*, toute dépendance f de F* est construite à partir des dépendances de F par une succession d'opérations de consensus: démontrer que cette succession d'opérations de consensus peut être réalisée à l'aide d'une génération repose sur l'application récursive de la pseudo associativité de l'opération de consensus. La démonstration se trouve dans (§8.11.1.).

Propriété 6 (démonstration dans § 8.11 3.):
Si f est générée par $(f_0,f_1...f_n)$ dans F alors $\forall\, i \in (1,n)$
$\forall\, A_i \in g(f_i)$: $g(f) \rightarrow A_i \in F^{**}$.

Remarque:
La notion de génération est une notion aussi générale mais plus précise que celle de dérivation introduite dans (BEERI-BERNSTEIN79) car elle indique l'ordre le mieux approprié pour effectuer les opérations de consensus. Le lemme précédent est l'équivalent du lemme 2 de leur papier dans le contexte de la génération.

8.3.2. Relation d'ordre sur l'ensemble des classes d'équivalence de df de F**

DÉFINITION

La df f_1 de F** est *générée* par la df f_0 de F** s'il existe une génération de f dans F** à partir de f_0. On dit aussi que f_0 *génère* f dans F**.

Théorème (démonstration dans § 8.11 4.):
La relation "f_0 génère f dans F**" définit un *préordre* sur l'ensemble de df F**.

Remarque:
s'il y a une génération de f à partir de f_0 dans F et s'il en existe une de f' à partir de f dans F, alors il en existe une de f' à partir de f_0 dans F.

Cette relation n'est pas antisymétrique comme le montre l'exemple suivant:
Soit F = {AB → C, A → B, A → C}.
A → C génère AB → C (par augmentation);
mais AB → C génère également A → C car
(A → C) = (A → B) +> (AB → C).

DÉFINITION

La relation f_1 *génère* f_2 *dans* F** et f_2 *génère* f_1 *dans* F** est une relation d'équivalence définie sur F**. Nous notons f* la classe d'équivalence de la dépendance f de F**.

THÉORÈME

La relation $f^* < h^*$ (où f et h appartiennent à F**) définie sur l'ensemble des classes d'équivalence de la manière suivante:
$f^* < h^* \ \exists\ f_1 \in f^*, \exists\ h_1 \in h^*$ telles que f_1 génère h_1 dans F**
est une *relation d'ordre.*

Remarque:
si $f^* < h^*$ alors $\forall\ f_1 \in f^*, \forall\ h_1 \in h^*$ f_1 génère h_1 dans F**.
Ces résultats sont des résultats classiques dérivés d'une relation de préordre.
On note $f^* << h^*$ si $f^* < h^*$ et s'il n'existe aucune autre classe g^* ($g^* \neq f^*$ et $g^* \neq h^*$) telle que $f^* < g^* < h^*$.

8.3.3. Classes d'équivalence minimales

8.3.3.1. Aspects structurels

DÉFINITION

Une classe d'équivalence f* est *minimale* s'il n'existe aucune autre classe d'équivalence h* telle que h* < f*.

Propriété 7 et définition:

Une classe d'équivalence minimale contient au moins une df élémentaire. Les df élémentaires d'une classe minimale sont dites *minimales* et FMIN désignera l'ensemble des df élémentaires des différentes classes minimales.

Preuve:

Soit X → A une df non élémentaire d'une classe minimale; alors il existe Y → A élémentaire telle que Y ⊆ X. Comme X → A peut être générée par Y → A par augmentation et que la classe d'équivalence de X → A est minimale, Y → A appartient à cette classe.

Propriété 8:

Si f: X → A et h: Y → A sont deux df élémentaires d'une même classe d'équivalence minimale, alors les deux df (dont les parties droites sont des ensembles de constituants) X → Y et Y → X appartiennent à F** et sont de plus structurellement élémentaires.

Preuve:
Comme f et h appartiennent à la même classe d'équivalence, f génère h et h génère f. D'après le lemme précédent X → Y et Y → X sont des df de F**. Si X' tel que X' ⊆ X et X' → Y, alors par transitivité X' → A et, comme X → A est élémentaire, X' = X. Ainsi X → Y et Y → X sont structurellement élémentaires.

Remarque:
X et Y sont alors des *clés* de la relation formée sur les constituants des dépendances de f* et munie des df de f*.
On dit que X et Y sont des *clés de f**.

8.3.3.2. Formation algorithmique des classes minimales

Propriété 9:

Soit B un constituant.
Soit f* et h* des classes d'équivalence de df.
influence (f*) = {B, $\exists$ K clé de f* et K $\rightarrow$ B $\in$ F*};
inflirr (f*) = {B $\in$ influence (f*); $\forall$ h* < f* B $\notin$ influence (h*)}.
Alors FMIN = {X $\rightarrow$ B $\in$ F*, $\exists$ h* et X est clé de h* et B $\in$ inflirr (h*)}.

Preuve:
$\forall$ f: X $\rightarrow$ B $\in$ FMIN f* est une classe minimale et X est une clé de f*.
Comme f* est minimale, B $\in$ inflirr (f*).

Réciproquement:
$\forall$ f: X $\rightarrow$ B $\in$ h* telle que X est clé de h*, B $\in$ inflirr (h*) et X $\rightarrow$ B $\in$ F*.
Soit Y $\rightarrow$ B $\in$ F* telle que Y $\rightarrow$ B peut générer X $\rightarrow$ B; soit e* la classe de Y $\rightarrow$ B; alors e* < h* et B $\in$ influence (e*).
Comme B $\in$ inflirr (h*), e* = h* et h* est une classe minimale.

8.3.3.3. Classes d'équivalence minimales et bases de F

Propriété 10:

Toute base de F contient au moins une df élémentaire de chaque classe d'équivalence minimale.

Preuve:
Soit G une base de F.
$\forall$ f $\in$ F*, il y a une génération de f à partir de f_0 dépendance de G d'après le théorème de génération. En particulier si f appartient à une classe minimale, alors comme f_0 génère f, f_0 appartient à la même classe minimale; ainsi G contient au moins une df élémentaire de chaque classe minimale.

Propriété 11:
Toute df f de F* peut être générée par une df de FMIN.

Preuve:
Si f $\in$ FMIN, la démonstration est terminée.
Sinon, $\exists$ h $\in$ F** telle que h* < f* et h* est minimale: alors il existe $h_1 \in h^*$ telle que $h_1 \in F^*$ et h_1 génère f.

Remarque:
FMIN n'est pas toujours une base comme le montre l'exemple suivant:

Exemple 7:
F = {S → A; S → B; A → B; B → A}. F est une base complète.
La classe d'équivalence de S → A ne peut contenir B → A car si B → A génère S → A, l'inverse est faux: B → S n'appartient pas à F**.
La classe d'équivalence de S → A n'est pas minimale, celle de S → B non plus. Les seules classes d'équivalence minimales sont celles contenant l'une A → B et l'autre B → A. Donc FMIN = {A → B, B → A}.
FMIN n'est pas une base de F.

8.4. Blocs de constituants et regroupements de df

Le but de ce paragraphe est de mettre en évidence la notion de regroupement de df et deux relations d'ordre partiel définies sur l'ensemble des regroupements.

DÉFINITION
Deux ensembles de constituants X et Y sont équivalents si X → Y et Y → X sont des df de F**.

Propriété 12:
Cette relation définit une relation d'équivalence.
La preuve est immédiate.

DÉFINITION
On appelle *bloc* de l'ensemble de constituants X la classe d'équivalence formée autour de X par la relation d'équivalence précédente.
On le note b(X).

DÉFINITION
b(X) > b(X') si et seulement si X → X' appartient à F**.

Propriété 13:
Cette relation est une relation d'ordre partiel définie sur l'ensemble des blocs déterminés à partir de F**.

La preuve est immédiate.

La notation b(X) >> b(X') signifie qu'il n'existe aucun autre bloc b(Y) vérifiant b(X) > b(Y) > b(X'); le bloc b(X) est directement supérieur au bloc b(X').

DÉFINITION

$b^+(X)$ désigne l'ensemble des constituants qui appartiennent à au moins un ensemble de constituants Y équivalent à X et donc élément de b(X). Une *clé* d'un bloc b(X) est une clé de relation Rb = U[$b^+(X)$].

Propriété 14:

La relation Rb contient toutes ses clés.

Preuve:
Soit Y une clé potentielle de Rb; alors Y → X appartient à F**. D'autre part comme Rb est construit à partir du bloc de X, X est une clé de Rb ou X contient une clé de Rb. Dans tous les cas, X → Y appartient à F**; ainsi Y appartient à b(X) et Rb contient toutes ses clés.

DÉFINITION

Le bloc b(X2) est *dépendant* du bloc b(X1) si et seulement si l'ensemble des constituants-clés de b(X1) est contenu dans celui de b(X2).

Propriété 15:
Si le bloc b(X2) est dépendant du bloc b(X1), alors toute clé K1 de b(X1) et toute clé K2 de b(X2) vérifient: K2 → K1 appartient à F**.

La preuve est immédiate.

Propriété 16:
La relation de dépendance définit une relation d'ordre partiel sur l'ensemble des blocs.

Preuve:
Elle est réflexive par construction.
Elle est antisymétrique: b(X1) étant dépendant de b(X2), si la réciproque est vraie, alors K1 → K2 et K2 → K1 appartiennent à F**;
K1 et K2 appartiennent au même bloc et ainsi b(X1) = b(X2).
Elle est transitive: si b(X3) est dépendant de b(X2) et b(X2) est dépendant de b(X1), alors toute clé de b(X1) est contenue dans l'ensemble des constituants clés de b(X2) et donc dans celui de b(X3).

Décomposition d'une relation

Notation:
G désigne un sous-ensemble de F**.

DÉFINITION

Un *regroupement* r de G est un sous-ensemble de df de G dont les parties gauches appartiennent à un même bloc que l'on note b(r) et que l'on appelle le bloc du regroupement r. r^+ désigne l'ensemble des constituants sur lesquels sont formées les df de r.

Propriété et définition:

Les clés potentielles de la relation Rr = U[r^+] sont les clés de b(X).
On les appelle les *clés* du regroupement r.

Preuve:

Le regroupement r contient au moins une df X → A et le bloc de r est par définition le bloc de X. Toute clé Y de b(X) vérifie que Y → X et X → Y appartiennent à F** et ainsi Y contient une clé potentielle de Rr: Y'.
Comme Y' est une clé potentielle de Rr, Y' → X est une df de F**;
Y = Y' car sinon Y ne serait pas clé de b(X).

Y est une clé potentielle de Rr.
Toute clé Z de Rr vérifie que Z → X et X → Z appartiennent à F**.
Z appartient donc à b(X). D'après les deux df précédentes, il contient une clé Z' de b(X): Z' doit être égal à Z pour que Z soit une clé de Rr.
Z est donc une clé de b(X).

Propriété 17:

Les parties gauches des df de F* appartenant à un regroupement sont des clés de ce regroupement.

Preuve:

Si X → A appartient à F* et à un regroupement r, alors par définition du regroupement, X contient une clé de r: soit X' cette clé. Alors X' → X → A. Comme X → A appartient à F*, X' est égal à X et X est une clé de r.

DÉFINITION

r > r' si et seulement si b(r) > b(r').

Propriété 18:
Cette relation est une relation d'ordre partiel définie sur l'ensemble des regroupements de G. Les regroupements qui n'ont pas d'inférieurs sont appelés *minimaux.*
La preuve est immédiate.

On note r >> r' le fait qu'il n'existe aucun autre regroupement r" vérifiant r > r" > r'.

DÉFINITION

Le regroupement r est *dépendant* du regroupement r' si et seulement si b(r) est dépendant de b(r').

Propriété 19:

Cette relation est une relation d'ordre partiel définie sur l'ensemble des regroupements de G.

La preuve est immédiate.

DÉFINITION

Le regroupement r est *directement dépendant* du regroupement r' si et seulement si r est dépendant de r' et si aucun autre regroupement r" ne vérifie: r est dépendant de r" et r" de r'.

DÉFINITION

MIN(r) (respectivement MIN(b(r))) désigne l'ensemble des regroupements r' de G (respectivement des blocs b(r')) vérifiant:
r >> r' ou r est directement dépendant de r'.

Remarques:
Une classe d'équivalence de df de G (§ 8 3.2.), contient des df dont les parties droites sont égales et les parties gauches appartiennent à un même bloc. Elles appartiennent donc à un même regroupement de G.

Exemple 8:
G = {S → A; S → B; A → B; B → A}.
G admet quatre classes d'équivalences qui admettent chacune l'une des df de G. G admet deux regroupements:
r = {S → A; S → B}; r' = {A → B; B → A} et r > r'.

8.5. Invariant des bases irredondantes

DÉFINITION

Un regroupement r_0 de F* est *obligatoire* si et seulement si chacune de ses df f_0 vérifie que toute génération de f_0 dans F* contient une df de r_0.

Le bloc d'un regroupement obligatoire est un *bloc obligatoire*; chaque clé d'un bloc obligatoire est une *racine* de constituants.

Remarque:

Un regroupement r minimal de F* ne contient que des df de FMIN: une df ne peut être générée que par une df de r car r est minimal. Ainsi un regroupement minimal est obligatoire.

Propriété 20:

L'ensemble des df des regroupements obligatoires B_0 forme une base de F*.

Preuve:

Soit B_0, l'ensemble des df des regroupements obligatoires.
Grâce à la relation d'ordre définie précédemment sur les regroupements, il est possible d'attribuer un rang à chaque regroupement (tri topologique (KNUTH73)):
.les regrougements minimaux pour rang 1;
.le regroupement non minimal r a pour rang i (i > 1) si tous les regroupements r' tels que r > r' ont un rang inférieur à i et s'il en existe un qui a le rang i-1.

Nous formulons l'hypothèse de récurrence suivante: les df des regroupements de rang n peuvent être générées dans B_0.
Cette propriété est vraie pour le rang 1 car les regroupements minimaux sont obligatoires.
Nous supposons cette propriété vraie pour les rangs inférieurs au rang n (n > 1).

Soit r un regroupement non obligatoire de rang n. Alors pour toute

dépendance f de r il existe une génération de f dans F*: GN = {f_1,f_2,...f_p, f'} où aucune des df f_i et f' n'appartient à r.
Comme de plus d'après la propriété 6 il existe g(f) → g(f_i) (i=1..p), chaque df de GN appartient à un regroupement de rang strictement inférieur à celui de r.
La propriété de récurrence étant supposée vraie jusqu'au rang n-1, chaque dépendance de GN peut être générée dans B_0; ainsi en est-il de f.

La propriété est vérifiée pour le rang n. B_0 est une base de F*.

Propriété 21 (démonstration dans § 8.11.5.):

Toute base contient au moins une df de chaque regroupement obligatoire.

Propriété 22 (démonstration dans § 8.11.5.):

Tout df d'une base irredondante appartient à un regroupement obligatoire.

THÉORÈME

Les racines de F*, c'est-à-dire les clés des regroupements obligatoires de F*, sont les clés des regroupements de toute base irredondante de F*.
Cette propriété constitue un invariant des bases irredondantes de F*.
La preuve découle des deux propriétés précédentes.

8.6. Couvertures homogènes et la couverture fondamentale

DÉFINITION

G est une *couverture homogène* de F si et seulement si:
a) G est une couverture de F: G** = F**.
b) G ne contient aucune df triviale.
c) Si X → A appartient à G, et si Y est une autre clé du bloc b(X) dans F**, alors Y → A appartient à G à moins qu'elle ne soit triviale;
de plus pour tout constituant B de Y, X → B appartient à G à moins qu'elle ne soit triviale.
d) Pour tout bloc b' de MIN(b(X))) dans G et pour tout constituant clé B de b', X → B appartient à G à moins qu'elle ne soit triviale.

Décomposition d'une relation

Exemple 9:

a) soit F = {S → A, S → B, A → B, B → A}.
CV = {S → A, A → B, B → A}. CV est une couverture de F, c'est même une base irredondante.
CV n'est pas une couverture homogène car A et B sont deux clés équivalentes et il manque S → B dans CV;
CV' = {S → A, S → B, A → B, B → A} est une couverture homogène.

b) soit F = {AB → G, DF → G, AB → D, G → E, AB → E, DE → AB, DG → AB}.
Voici la base complète:
F* = {AB → DEG, DF → ABEG, G → E, DE → ABG, DG → AB}.

Voici une couverture de F:
CV = (AB → GD, DE → ABG, G → E, DF → G); elle n'est pas homogène car AB, DE et DG sont des clés équivalentes, et DF et ABF également.
Il existe en fait trois blocs:
b_1 qui a pour clés DF et ABF,
b_2 qui a pour clés AB, DE et DG,
b_3 qui a pour clé G.
b_2 est dépendant de b_3. De plus $b_1 > b_2 > b_3$.
Min(b_1) contient seulement b_2.
Voici une couverture homogène construite à partir de CV:
CV' = {DF → ABEG, ABF → DEG, AB → GDE, DE → ABG,
DG → ABE, G → E}.

Notation:
B_0 désigne l'ensemble des df élémentaires appartenant aux regroupements obligatoires de F*.

DÉFINITION

La couverture fondamentale CF de F** est formée d'un ensemble de df f: X → A de F** vérifiant chacune que:
a) elle n'est pas triviale;
b) X est une racine, clé d'un regroupement obligatoire r_0;
c) elle appartient à FMIN,
ou il existe une autre clé Y du bloc de X, dont A est un des constituants,
ou il existe une clé Y d'un regroupement obligatoire r_1 appartenant à MIN(r_0) et A est un des constituants de Y.

Propriété 23:

La couverture fondamentale CF de F** est une couverture homogène de F* et elle est unique.

Preuve:

Nous allons montrer que CF permet de générer B_0.

Soit X une racine et f: X → A une df élémentaire.
Soit r_0 le regroupement obligatoire de f.
Si f appartient à FMIN, alors f appartient à CF.
Sinon il existe une df g: Y → A de FMIN qui permet de générer f. g appartient à CF.

Soit r_n le regroupement obligatoire de g. Si r_n n'appartient pas à MIN(r_0), il existe un regroupement obligatoire r_1 de MIN(r_0) tel que r_1 est dépendant de r_n ou que r_1 est supérieur à r_n.
Il existe ainsi une suite de regroupements obligatoires r_0, r_1 ... r_n telles que r_{i+1} appartient à MIN(r_i) (éventuellement n = 0 ou 1). Soit K_0...K_i K_n une clé de chaque regroupement, les df $K_i \rightarrow K_{i+1}$ appartiennent à CF par construction. K_0 = X et K_n = Y en particulier.

Ainsi $X \rightarrow K_1$, $K_1 \rightarrow K_2$,...$K_{n-1} \rightarrow Y$ et $Y \rightarrow A$ fournissent une génération de X → A dans CF sauf si A est un constituant d'une clé K_i.
Dans ce cas le même raisonnement s'applique en considérant
$X \rightarrow K_1$,... $K_{i-1} \rightarrow A$.

CF est donc une couverture de F puisqu'elle permet de générer une base de F. CF est homogène et unique par construction.

Algorithme de formation de la couverture fondamentale de F:

-calculer F* la base complète;
-calculer FMIN (propriété 9);
-calculer une base irredondante F^+ (algorithmes de BERNSTEIN76, PICHAT-DELOBEL79, LUCAS81);
- pour chaque racine X et son regroupement obligatoire r, générer les clés équivalentes (LUONG 86) et calculer MIN(r);
- pour chaque racine X, en déduire les df de CF (propriété c de la définition) qui viennent s'ajouter aux df de FMIN.

Décomposition d'une relation

Exemple 10:
Nous prolongeons l'exemple 9.b et nous allons calculer la couverture fondamentale.

a) *Calcul de FMIN:*

Nous formons les classes d'équivalences des df de F*:
1 = {G → E}; 2 = {AB → D}; 3 = {AB → G, DE → G};
4 = {DG → A, DE → A};
5 = {DG → B, DE → B}; 6 = {AB → E};
7 = {DF → A}; 8 = {DF → B}; 9 = {DF → E};
10 = {DF → G}.

Elles sont ordonnées de la manière suivante:
1 < 6 < 9; 3 < 10; 4 < 7; 5 < 8.
Les classes d'équivalence minimales sont donc 1,2,3,4,5.
FMIN = {G → E, AB → D, AB → G, DE → G, DG → A, DE → A,
DG → B, DE → B}.

b) *Calcul d'une base irredondante:*

BIRR = { AB → G, DF → G, AB → D, G → E, DE → A, DE → B }.
Les regroupements de BIRR sont:
r_1 = { G → E} clé: G;
r_2 = { AB → G, AB → D, DE → A, DE → B } clés: AB, DE, DG;
r_3 = { DF → G } clés: DF, ABF.
Nous avons l'ordonnancement suivant: $r_1 << r_2 << r_3$.

c) *Calcul de la couverture fondamentale:*

Pour la racine G: G → E qui appartient à FMIN.
Pour les racines AB, DE, DG:
- AB → G, AB → D, DE → A, DE → B qui appartiennent à FMIN;
- AB → DE, DE → AB, DG → AB, AB → DG, DE → G, DG → E
à cause des différentes clés;
- AB → G, DE → G car $r_2 >> r_1$;
Pour les racines DF, ABF:
- DF → AB et ABF → D à cause des différentes clés;
- DF → ABEG et ABF → DEG à cause de $r_3 >> r_2$.

En résumé voici la couverture fondamentale:
CF = {G → E, AB → DEG, DE → ABG, DG → ABE, DF → ABEG,
ABF → DEG }.

8.7. Décompositions homogènes et la décomposition fondamentale d'une relation

DÉFINITION

Une décomposition D d'une relation U est une *décomposition homogène de U* si elle est obtenue à partir d'une couverture homogène des df définies sur U par application du théorème (decdf2) (§ 6.1.3.) et s'il existe une relation particulière de D contenant toutes les clés de U.

Toute relation qui contient une clé d'une autre relation, doit contenir toutes les clés de celle-ci.

Propriétés 24:

Une décomposition homogène vérifie les propriétés suivantes:
a) Chaque relation R_i de D contient toutes ses clés.
b) La relation: $R_i > R_j$ définie sur l'ensemble des relations de D et signifiant qu'il existe la df $K_i \rightarrow K_j$ dans U, est une relation d'ordre partiel.
c) La relation: *R_i est dépendante de R_j* définie sur l'ensemble des relations de D et signifiant que l'ensemble des constituants clés de R_i contient l'ensemble des constituants-clés de R_j , est une relation d'ordre partiel.

Ce sont des propriétés que l'on peut facilement démontrer.

Exemple 11:

La décomposition homogène obtenue à partir de la couverture homogène CV' de l'exemple 9.b est la suivante:

U = U[ABDFEG] * U [ABDEG] * U [GE].

Les clés de U sont ABF et DF et sont les clés de la première relation.

DÉFINITION

La *décomposition fondamentale de U* est la décomposition homogène de U obtenue à partir de la couverture fondamentale des df de U par application du théorème (decdf2).

Exemple 12:
La décomposition fondamentale obtenue à partir de la CF de l'exemple 10 est la même que celle obtenue dans l'exemple 11.

ALGORITHME

Pour obtenir la décomposition fondamentale d'une relation U, il suffit de:
- construire la couverture fondamentale et former les relations correspondantes R_i;
- déterminer les clés de U;
- si les clés de U ne sont pas des racines, former la relation UK obtenue par projection de U sur les clés de U et sur les constituants-clés des relations dont UK est directement supérieure ou dépendante.

8.8. Décomposition complètement homogène

Notre but est d'obtenir une décomposition D qui ne privilégie aucune clé d'une relation de D quand celle-ci en admet plusieurs. Ce résultat est presque atteint si D est une décomposition homogène. Mais il manque encore un détail que nous illustrons à l'aide de l'exemple suivant:

Exemple 13:
Soit la relation U(ABCDSUV).
F = {S → ABCDU; AB → CUV; UB → ACV; CD → UVA; UD → ACV;
AC → XUV; UV → XAC }.
F est la couverture fondamentale.
La décomposition fondamentale est formée des relations:

U_1 = U[SABCDU]	clé : S;
U_2 = U[ABCUV]	clés: AB; BU;
U_3 = U[CDUVA]	clés: CD; DU;
U_4 = U[ACUVX]	clés: AC; UV.

La relation U_1 contient une clé de U_4: AC sans contenir l'autre UV.
fex.

Pour résoudre cette difficulté, on applique de manière récursive un algorithme simple à chaque relation R_i d'une décomposition homogène:
si R_i contient dans ses constituants une clé d'une autre relation R_j, on lui rajoute tous les constituants-clés de R_j.
Cet algorithme est fini car nécessairement $R_i > R_j$.

DÉFINITION

Les *décompositions complètement homogènes* D garantissent l'équivalence des clés des relations les formant car si K_j clé d'une relation R_j de D se trouve dans R_i^+ alors toutes les autres clés de R_j se trouvent dans R_i^+. L'objectif de préserver l'équivalence des clés (§ 7.1.3.) est ainsi atteint.

8.9. Graphe de relation d'une décomposition complètement homogène

Soit GR le graphe de relation obtenu à partir d'une décomposition *complètement homogène* D_h rendue *compacte* (§ 4.), par l'application de l'algorithme GR.

$INF(R_i)$ est l'ensemble des relations R_j de D_h telles que
$R_i > R_j$ ou R_i est dépendante de R_j,
et les constituants-clés de R_j sont des constituants de R_i.

$MINFD(R_i)$, l'ensemble associé à une relation R_i de D_h par le processus de construction d'un graphe de relation (§ 4.1.1.E3), est l'ensemble des relations R_j de $INF(R_i)$ telles qu'il n'existe pas une relation R_k de $INF(R_i)$ dépendante de R_j.
Toutes les relations R_j de D_h dont R_i est directement supérieure ou directement dépendante appartiennent à $MINFD(R_i)$.

Les propriétés suivantes sont démontrés dans (§ 8.11.6.).

Propriété (GR1):
GR est sans circuit.

Propriété (GR2):
Si la relation R_1 est dépendante de la relation R_n alors il existe un chemin reliant le noeud de R_1 à celui de R_n.

Propriété (GR3):
Si la relation R_1 appartient à $INF(R_n)$ alors il existe un chemin dans GR reliant le noeud de R_1 à celui de R_2.

Décomposition d'une relation

Propriété (GR4):
Si A est un constituant d'une charnière de deux relations R_1 et R_2, alors il existe un noeud dont la relation R_j admet A comme constituant-clé et vérifie que $R_1 > R_i$ et $R_2 > R_i$.

Propriété (GR5):
GR est connexe.

Propriété (GR6):
Si X est une clé à risque linéaire (ou si f: X $\rightarrow$ A est une df à risque linéaire) dans une décomposition homogène D_h de U, alors ces risques sont représentés dans le graphe orienté de relation GR par des chemins reliant le noeud d'une relation contenant X (ou f^+) à la relation admettant X comme clé.

Si X est une clé à risque cyclique de R_i (ou si f: X $\rightarrow$ A est une df intrinsèque à risque cyclique de R_i) dans une décomposition homogène, alors ces risques sont représentés dans le graphe orienté de relation GR par des cycles fonctionnels contenant R_i et admettant comme constituant-clé du puits, des constituants de X (ou A dans le cas d'une df à risque cyclique).

8.10. Conclusions

Notre objectif est de construire une structure de données initiale qui d'une part préserve l'équivalence des clés et d'autre part facilite la mise en oeuvre des mécanismes de complétude de clés et d'entités.
Cette étude nous montre que pour une relation U munie d'un ensemble de df F, il existe une couverture unique, la couverture fondamentale qui permet ensuite d'obtenir une décomposition homogène de U unique préservant aux mieux l'équivalence des clés: c'est la décomposition fondamentale D_f de U.
Pour que cette équivalence soit complètement réalisée, il faut rendre cette décomposition *complètement homogène.*

De plus nous avons montré que les situations de df à rique cyclique et celles de clés cycliques correspondent justement à des cycles dans le graphe orienté de relation GR_f obtenu à partir de D_f, et que les situations de df à risque linéaire et de clés à risque linéaire correspondent dans GR_f à des chemins.

Dans la quatrième partie, nous montrons comment ces dernières propriétés permettent d'étendre les fonctionnalités des deux mécanismes de complétude.

Remarque:

Nous pouvons maintenant prouver que la structure de données construite à partir d'un GR_f, va préserver l'accès (§ 4.3.): GR_f étant connexe (propriété GR4), la composition de toutes les relations de D_f est implantée dans GR_f (§ 3.5.).

8.11. Annexe DF

8.11.1. Théorème de la pseudo-associativité

Soit trois df f,g,h.
Si (f +> g) +> h existe, alors f +> (g +> h) existe également et de plus,
(f +> g) +> h < f +> (g +> h).

Preuve:

Soit f: $X \rightarrow A$; g: $Y \rightarrow B$; h: $Z \rightarrow C$.
A,B,C sont des constituants, X,Y,Z des ensembles de constituants.

Comme i = (f +> g) +> h existe, $A \in Y$ et $B \in Z$:
i: $X \cup (Y-A) \cup (Z-B) \rightarrow C$.

j = g +> h existe puisque $B \in Z$; j: $Y \cup (Z-B) \rightarrow C$;
k = f +> j existe puisque $A \in Y$; k: $X \cup ((Y \cup (Z-B))-A) \rightarrow C$.

Si $A \notin Z$ alors k: $X \cup (Y-A) \cup (Z-B) \rightarrow C$ et k = j.

Par contre si $A \in Z$, alors $g(k) \cup A = g(i)$ et $k > i$.

8.11.2. Théorème de génération

A toute df de F** correspond une génération dans F.

Preuve:
a) Par définition même de la fermeture F**, toute df f de F** est construite par une succession d'opérations de consensus à partir de dépendances de F et de dépendances triviales: $h_1 ... h_n$.
On peut écrire cette succession d'opérations de consensus à l'aide d'un *arbre d'évaluation*:
- à chaque noeud terminal autre que la racine est associée une df;
- à chaque autre noeud de l'arbre est associée l'opération de consensus.

Exemple d'arbre d'évaluation permettant de construire ABEHI → L:

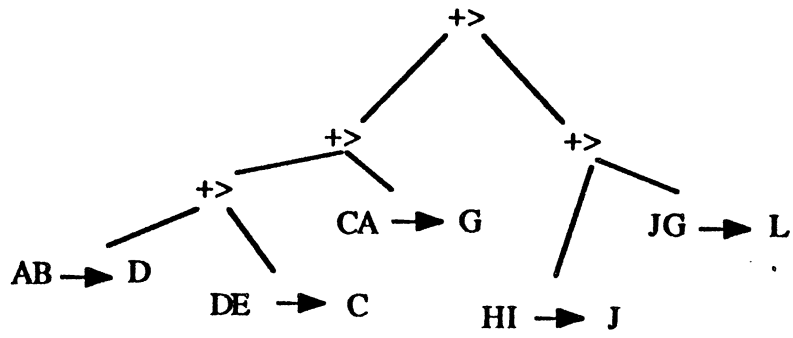

Figure 8.1. Arbre d'évaluation

fex.

Le noeud *prédécesseur* N du noeud N_1 dans l'arbre est le noeud qui se trouve sur le chemin de N_1 à la racine de l'arbre et qui est adjacent à N_1. N_1 est alors un *successeur* de N. Chaque noeud non terminal a 2 et seulement 2 successeurs N_1, successeur gauche, et N_2, successeur droite.

b) *L'évaluation* de l'arbre est un processus récursif qui demande de:
- choisir n'importe quel couple de noeuds terminaux N_1 et N_2 ayant même prédécesseur N;
- détruire N_1 et N_2 et associer à N le consensus f_{12} des df h_1, h_2 associées respectivement à N_1 et N_2: $h_{12} = h_1 +> h_2$;
- recommencer a) et b) tant que la racine n'est pas atteinte: si elle est atteinte, le résultat final est associé à la racine.

Un arbre d'évaluation est *bien formé* si, effectivement, son évaluation conduit à une df associée à sa racine.

Remarque:
Tout arbre d'évaluation extrait d'un arbre d'évaluation bien formé est lui-même bien formé.

Dans le cas d'une génération, l'arbre d'évaluation vérifie par construction que tous les successeurs gauches des noeuds non terminaux sont des noeuds terminaux puisque la partie gauche de toute opération de consensus dans une génération est une df de F. Un tel arbre d'évaluation est appelé *arbre de génération.*

Exemple d'arbre de génération de ABEHIHG → L:

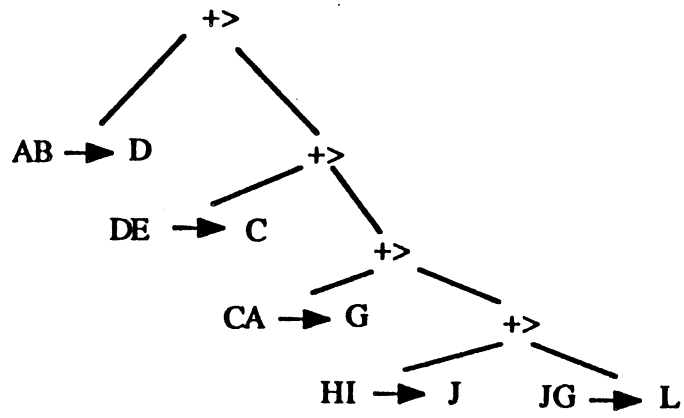

Figure 8.2. Arbre de génération

c) Le *basculement* d'un arbre d'évaluation A dans un autre arbre d'évaluation A' est défini seulement si le successeur gauche NG de la racine R de A n'est pas terminal, et obéit aux règles suivantes:
la racine R' de A' est NG;
le successeur gauche NG' de R' dans A' est le successeur gauche SG de NG dans A;
le successeur droit ND' de R' dans A' est la racine R de A;
le successeur gauche de ND' dans A' est le successeur droit de NG dans A;
les autres noeuds de A' sont les mêmes que ceux de A avec les mêmes prédécesseurs et successeurs.
Les noeuds de A' sont associés aux mêmes renseignements (opérateur de consensus ou df de F) que les noeuds de A correspondants.

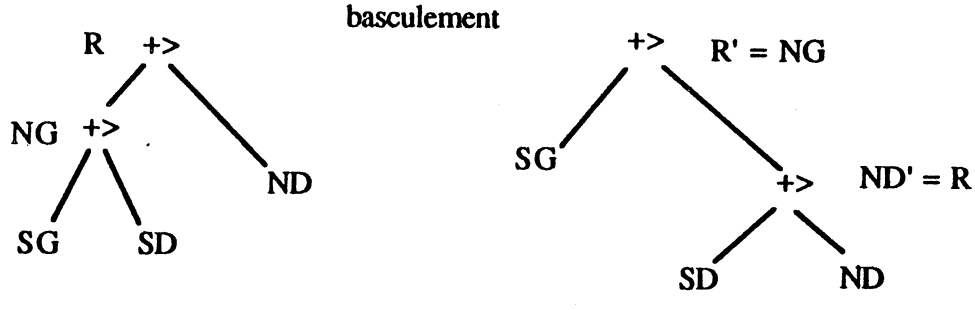

Figure 8.3. Basculement

Décomposition d'une relation

Propriété A1:

Le basculement d'un arbre d'évaluation A d'une df de F^+ donne un arbre d'évaluation A' bien formé, dont le résultat est supérieur ou égal à celui de A.

La démonstration est une application immédiate de la propriété de pseudo-associativité de l'opération de consensus.

d) Transformation d'un arbre d'évaluation A d'une df f de F^{**} dans un arbre de génération.

- Faire autant de basculements qu'il existe de noeuds séparant dans A la racine du noeud terminal le plus à gauche; on obtient ainsi un arbre d'évaluation A_1 dont le successeur gauche de la racine est terminal et qui conduit au même résultat que A puisque f appartient à F^{**}.
- Le successeur droit de la racine peut être considéré comme la racine d'un arbre d'évaluation A'_1, sous-arbre de A_1; A'_1 est un arbre d'évaluation d'une df de F^+. Son basculement donne un nouvel arbre d'évaluation A''_1 dont le résultat g' est égal ou supérieur à celui g de A'_1; s'il est supérieur, il faut rajouter une nouvelle opération de consensus entre une df triviale (la partie droite est incluse dans la partie gauche) et la dépendance g' pour obtenir g. On obtient ainsi un arbre d'évaluation A_2 dont le résultat est identique à A_1 et dont la racine R et son successeur droit admettent comme successeur gauche un noeud terminal.
- En considérant un nouveau sous-arbre de A_2 et en refaisant le même parcours, on transforme de proche en proche l'arbre d'évaluation initial A en un arbre de génération A' qui admet le même résultat que A.
Mais A' contient des df triviales.

e) La partie suivante de la démonstration consiste à montrer que toutes les df triviales de A' qui ont été introduites peuvent être éliminées.
L'évaluation déduite de A' s'écrit:

$$m_0 = f_0,$$
$$m_1 = f_1 +> m_0,$$
$$\vdots$$
$$m_i = f_i +> m_{i-1},$$
$$\vdots$$
$$m_p = f_p +> m_{p-1},$$
$$f < m_p.$$

On peut déduire de cette évaluation une autre évaluation construite sur les règles suivantes:
i) $e_0 = f_0$
ii) $\forall$ i $\in$ (1,p): si f_i n'est pas triviale et si le consensus de f_i et de e_{i-1} existe alors $e_i = f_i +> e_{i-1}$;
sinon $e_i = e_{i-1}$.
iii) $f' = e_p$.

Nous allons montrer que $\forall$ i $\in$ (0,p) $m_i < e_i$.
Cette propriété est vraie pour i = 0.
Nous la supposons vraie pour q (q < p-1) et nous allons montrer qu'elle est vraie pour q+1: $m_q < e_q$.

- Si f_{q+1} est triviale, alors $g(m_{q+1}) = g(m_q) \cup g(f_{q+1})$ et ainsi $m_{q+1} < e_q = e_{q+1}$;
- si le consensus n'existe pas entre f_{q+1} et e_q, alors $d(f_{q+1}) \notin g(e_q)$:
donc $g(m_{q+1}) = (g(m_q) - d(f_{q+1})) \cup g(f_{q+1}) \subseteq g(e_q)$
et ainsi $m_{q+1} < e_q = e_{q+1}$;
- si f_{q+1} n'est pas triviale et si le consensus existe entre f_{q+1} et e_q, alors $(g(m_q)-d(f_{q+1})) \cup g(f_{q+1})$ contient $(g(e_q)-d(f_{q+1})) \cup g(f_{q+1})$
puisque $g(m_q)$ contient $g(e_q)$:ainsi $m_{q+1} < e_{q+1}$.
- f < f'.
On a ainsi obtenu une évaluation de f construite seulement à partir de df de F et dont l'arbre d'évaluation est un arbre de génération.

f) Nous allons montrer qu'il est possible de construire un arbre de génération tel que toutes les df f_i d'un arbre de génération ($i \neq 0$) vérifient $d(f_i) \neq d(f_0)$.

Soit k $\in$ (1,p) tel que $d(f_k) = d(f_0)$ et $\forall$ i > k $d(f_i) \neq d(f_0)$ et $e_k = f_k +> e_{k-1}$.
Si l'arbre d'évaluation est bien formé, alors $d(f_0) \in g(e_{i-1})$.
Comme $d(e_{i-1}) = d(e_0) = d(f_0)$, e_{i-1} est en fait une df triviale et $f_k > e_k$.

On peut alors déduire une nouvelle évaluation:
i) $h_0 = f_k$;
ii) $\forall$ i $\in$ (k+1,p): si le consensus de f_i et de h_{i-1-k} existe,
alors $h_{i-k} = f_i +> h_{i-1-k}$; sinon $h_{i-k} = h_{i-1-k}$;
iii) $f'' = h_{p-e}$.

Nous allons démontrer que $\forall$ i $\in$ (k,p) $h_{i-k} > e_i$.
Cette propriété est vraie pour i=k puisque $h_0 = f_k > e_k$.
Nous la supposons vraie pour q (q < p-1) et nous allons montrer qu'elle est vraie pour q+1: ainsi $h_{q-k} > e_q$:

- si le consensus de f_{q+1} et de h_{q-k} n'existe pas alors $h_{q+1-k} = h_{q-k}$;
  si $e_q = e_{q+1}$ alors $h_{q+1-k} > e_{q+1}$;
  sinon $g(e_{q+1}) = (g(e_q) \cup g(f_{q+1})) - d(f_{q+1})$;
  comme $d(f_{q+1}) \notin g(h_{q-k})$ et que $g(h_{q-k}) \subseteq g(e_q)$,
  alors $g(h_{q-k}) \subseteq g(e_{q+1})$ et $e_{q+1} < h_{q+1-k}$;
- si le consensus de f_{q+1} et h_{q-k} existe,
  alors $g(h_{q+1-k}) = (g(f_{q+1}) \cup g(h_{q-k})) - d(f_{q+1})$
  et $g(h_{q+1-k}) \subseteq (g(f_{q+1}) \cup g(e_q)) - d(f_{q+1})$;
  ainsi $h_{q+1-k} > e_{q+1}$
- ainsi $f'' > f' > f$.

8.11.3. Propriété 6

Si f est générée par $(f_0,f_1...f_n)$ dans F alors
$\forall i \in (1,n) \forall A_i \in g(f_i)$: $g(f) \rightarrow A_i \in F^{**}$.

Preuve:

$m_i = f_i +> m_{i-1}$. Ainsi $g(m_i) = g(f_i) \cup (g(m_{i-1}) - d(f_i))$.
Donc $g(m_i) \rightarrow A_i \in F^{**} \ \forall A_i \in g(f_i)$.
$g(m_i) \rightarrow d(f_i) \in F^{**}$. $g(m_i) \rightarrow B_{i-1} \in F^{**} \ \forall B_{i-1} \in g(m_{i-1})$.

Ainsi, par applications successives de la pseudo- transitivité
$\forall i \in (1,n), \forall j \in (1,i-1), \forall B_j \in g(m_j)$, $g(m_i) \rightarrow B_j$.

Comme $f < m_n$, $\forall A_n \in g(m_n)$ $g(f) \rightarrow A_n \in F^{**}$ et
$\forall i \in (1,n) \forall B_i \in g(m_i)$ $g(f) \rightarrow B_i \in F^{**}$;
donc $\forall A_i \in g(f_i)$ $g(f) \rightarrow A_i \in F^{**}$

8.11.4. Théorème du préordre défini sur F**

La relation "f_0 génère f dans F**" définit un *préordre* sur l'ensemble de df F**.

Preuve:

La relation est réflexive par construction même d'une génération. Elle est transitive: si f_0 génère f et si f génère f' dans F** alors il existe deux générations S et S':

S: $m_0 = f_0$; $\forall i \in (1,n)$ $m_i = f_i +> m_{i-1}$; $f < m_n$,
et S': $m_0 = f$; $\forall j \in (1,p)$ $m'_j = f'_j +> m_{j-1}$; $f' < m'_p$,
avec $\forall i \in (1,n)$ $f_i \in F^{**}$ et $\forall j \in (1,p)$ $f'_j \in F^{**}$.

A partir de S', on peut construire une nouvelle génération S" vérifiant:
$m_0" = m_n$,
$\forall j \in (1,p)$, si $d(f'_j) \in g(m"_{j-1})$, $m"_j = f'_j +> m"_{j-1}$, sinon $m"_j = m"_{j-1}$.

Montrons par récurrence que $\forall j \in (0,p)$ $m"_j > m'_j$;
cette propriété est vraie pour $j = 0$ puisque $m"_0 = m_n > f = m'_0$.
Supposons la vraie pour $\forall j \in (0,q)$ où $q < p$.

Alors si $d(f'_{q+1}) \in g(m"_q)$, le consensus existe:
$g(m"_{q+1}) = (g(m"_q) - d(f'_{q+1})) \cup g(f'_{q+1})$;
comme $g(m"_q) \subseteq g(m'_q)$, $g(m"_{q+1}) \subseteq g(m'_{q+1})$ et $m"_{q+1} > m'_{q+1}$.

Sinon $d(f'_{q+1}) \notin g(m"_q)$: alors $g(m"_q) \subseteq g(m'_q) - d(f'_{q+1})$
et $g(m'_{q+1}) = (g(m'_q) - d(f'_{q+1})) \cup g(f'_{q+1})$ contient $g(m"_q)$;
comme $g(m"_q) = g(m"_{q+1})$, $m'_{q+1} < m"_{q+1}$.

Ainsi S" est une nouvelle génération de f' ($f' < m'_p < m"_p$) à partir de m_n.
La réunion des deux générations S et S" fournit ainsi une génération de f à partir de f_0.

La relation est donc transitive.

8.11.5. Regroupements obligatoires

Propriété 21:

Toute base contient au moins une dépendance de chaque regroupement obligatoire.

Preuve:

Le regroupement obligatoire r_0.contient la dépendance f qui n'appartient pas à la base B. Comme B est une base, il existe f' dans B qui génère f dans B: ainsi $g(f) \rightarrow g(f')$ et le rang du regroupement de f' est inférieur ou égal à celui de r_0. S'il est égal, alors f et f' appartiennent au même regroupement r_0: B contient une dépendance de r_0. S'il n'est pas égal, alors comme la génération dans B qui construit f à partir de f' est

également une génération dans F*, elle contient obligatoirement une dépendance de r_0.par définition d'un regroupement obligatoire.

Propriété 22:

Tout df d'une base irredondante appartient à un regroupement obligatoire.

Preuve:

Chaque dépendance f d'un regroupement r non obligatoire peut être générée dans F* à l'aide d'une génération G telle que G = (f_1, f_2...f_n, f') et f_1,f_2...f_n, f' n'appartiennent pas à r. Ainsi les rangs des regroupements contenant les dépendances de G sont strictement inférieurs à celui de r.

Soit BIRR une base irredondante et BIRR(n) les dépendances de BIRR appartenant à des regroupements dont les rangs sont inférieurs strictement à n (n est le rang de r).
Comme BIRR est une base, chaque dépendance de G peut être générée dans BIRR à l'aide de dépendances appartenant nécessairement à BIRR(n): en effet si la génération de f_1 dans BIRR contient la dépendance firr, alors $g(f_1) \rightarrow g(firr)$ (propriété 6) et ainsi le regroupement de f_1 a un rang supérieur ou égal au rang du regroupement de firr.

Ainsi f appartient à la fermeture formée à partir de BIRR(n). f ne peut donc appartenir à BIRR.

8.11.6. Graphe de relation d'une décomposition complètement homogène

8.11.6.1. Propriété (GR1)

GR est sans circuit.

Preuve:

Si les noeuds des relations ($R_1R_2...R_n$) forment un circuit, alors $R_1 > R_2...R_{n-1} > R_n$ et $R_n > R_1$. Ceci est impossible, car la relation définie sur les relations est une relation d'ordre.

8.11.6.2. Propriété (GR2)

Si la relation R_1 est dépendante de la relation R_n alors il existe un chemin reliant le noeud de R_1 à celui de R_n.

Preuve:

R_n appartient à $INF(R_1)$ car KR_n^+ est inclus dans KR_1^+. Si R_n appartient à $MINFD(R_1)$, la propriété est démontrée.
Sinon il existe R_2 de $MINFD(R_1)$ par construction du graphe de relation telle que KR_n^+ est inclus dans KR_2^+: R_2 est dépendante de R_n. Les noeuds de R_1 et de R_2 sont reliés par un arc partant du noeud de R_1.

On peut appliquer le même raisonnement aux relations R_2 et R_n, en remarquant toutefois que la relation de dépendance est une relation d'ordre partiel: aussi existe-t-il une suite de relations $R_2...R_n$ telles que:
- R_i est dépendante de R_j si $i < j$, (i et j compris entre 2 et n).
- R_{j+1} appartient à $MINFD(R_i)$ pour tout $i \in (2, n-1)$. Il existe donc un chemin reliant le noeud de R_1 à celui de R_n.

8.11.6.3. Propriété (GR3)

Si la relation R_1 appartient à $INF(R_n)$ alors il existe un chemin dans GR reliant le noeud de R_1 à celui de R_2.

Preuve:

Si R_1 appartient à $INF(R_n)$, il existe une relation R_2 qui appartient à $MINFD(R_n)$ et qui est dépendante de R_1 d'après la 3ème étape de l'algorithme: éventuellement R_1 et R_2 sont identiques. Il existe un arc reliant le noeud de R_n à celui de R_2, et il existe un chemin reliant le noeud de R_2 à celui de R_1 d'après la propriété précédente.

8.11.6.4. Propriété (GR4)

Si A est un constituant d'une charnière de deux relations R_1 et R_2, alors il existe un noeud dont la relation R_i admet A comme constituant-clé et vérifie que $R_1 > R_i$ et $R_2 > R_i$.

Preuve:

Si les relations R_1 et R_2 admettent une charnière Ch_{12}, alors D_h qui est compacte contient une relation R_{12} dont les clés sont identiques à celles de

Ch_{12}. Si A n'est pas un constituant clé de R_{12}, D_h contient une relation R_{121} dont les clés sont identiques à celles de $Ch_{121} = Ch_{12} - KR_{12}^+$. Ainsi de proche en proche, il existe une relation R_i dont A est un constituant-clé. Par construction $R_1 > R_i$ et $R_2 > R_i$.

Par contre, la quatrième étape de la construction du graphe de relation (§ 4.4.4.) pourrait éliminer R_i. Mais, il faut remarquer que par construction R_i appartient à $INF(R_1)$ et à $INF(R_2)$. Aussi existe-t-il une relation R_3 de $MINFD(R_1)$ qui est dépendante de R_i (éventuellement égale à R_i). R_3 admet donc A comme constituant-clé .

De même il existe une relation R_4 de $MINFD(R_2)$ dépendante de R_i.
Si R_3 et R_4 sont identiques, leur noeud est relié à deux noeuds distincts, celui de R_1 et celui de R_2 et ne peut être supprimé au cours de la quatrième étape; sinon il existe une relation R_5 (éventuellement égale à R_i).dépendante de R_i et dont R_3 et R_4 sont dépendantes.
R_5 ne peut être supprimée au cours de la quatrième étape.

A la fin de l'algorithme, il existe au moins une relation admettant A comme constituant-clé.

8.11.6.5. Propriété (GR5)

GR est connexe.

Preuve:

D_h étant une décomposition homogène d'une relation U, il existe, par définition, une relation R source de D qui admet pour clés, les clés de U, et qui est supérieure à toutes les autres relations R_i de D.

Dans le graphe orienté de relations obtenu à la fin de la troisième étape de l'algorithme de transformation d'une décomposition compacte dans un graphe de relation (§ 4.1.1.), R est associée à un noeud.

D'après la propriété précédente (GR3), ce noeud est relié à tous les autres noeuds par un chemin puisque $R > R_i$.
Le graphe orienté de relation est connexe.

Les étapes suivantes de l'algorithme ne modifient pas cette caractéristique même si la relation R n'est plus associée à un noeud mais à une arête dans le graphe de relation final.

8.11.6.6. Propriété (GR6)

Si X est une clé à risque linéaire (ou si f: X → A est une df à risque linéaire) dans une décomposition homogène D_h de U, alors ces risques sont représentés dans le graphe orienté de relation GR par des chemins reliant le noeud d'une relation contenant X (ou f^+) à la relation admettant X comme clé.

Si X est une clé à risque cyclique de R_i (ou si f: X → A est une df intrinsèque à risque cyclique de R_i) dans une décomposition homogène, alors ces risques sont représentés dans le graphe orienté de relation GR par des cycles fonctionnels contenant R_i et admettant comme constituant-clé du puits, des constituants de X (ou A dans le cas des df à risque cyclique).

Preuve:

1. X est une clé de la relation R_i de D_h. Il existe une décomposition fonctionnelle partielle D' de D_h qui déborde pour X; D' est minimale et soit R la source de D'. R est supérieure à R_i car K → X est une df de U. Il existe donc un chemin reliant le noeud de R à celui de R_i (propriété GR3). Dans D', il existe une relation R_k telle que R_k^+ et X admettent au moins un constituant commun B. Comme R_k est une relation de D', $R > R_k$.

D'après la propriété (GR4), il existe un noeud dans GR dont la relation R_{ik} admet B comme constituant-clé et vérifie: $R_k > R_{ik}$ et $R_i > R_{ik}$.

1.a) Si $R_{ik} = R_k$, alors $R > R_i > R_k$.
Comme R_i n'appartient pas à D', il existe deux chemins qui relient R à R_k et il existe donc un cycle fonctionnel de source R, de puits R_k, contenant R_i dont un constituant-clé du puits est B, constituant-clé de R_i.

1.b) Si $R_{ik} = R_i$, alors $R_k > R_i$.
Les clés des constituants communs de R_k et de R_i sont celles de R_i. Comme la décomposition est complètement homogène, R_k^+ contient KR_i^+ et donc X.
Alors $R = R_k$ et X est une clé à risque linéaire: il existe un chemin reliant les noeuds de R_k et de R_i.

1.c) Sinon il existe trois relations distinctes R_i, R_k, R_{ik}.
Un chemin de R à R_k ne passe pas par R_i car R_i n'est pas une relation de D'. Il existe alors un cycle de source R et de puits R_{ik}, contenant R_i et admettant comme constituant-clé du puits, au moins un constituant de X.

Décomposition d'une relation

2. f: X → A est une df intrinsèque d'une relation R_i de D_h. Il existe une décomposition fonctionnelle partielle D' de D_h qui déborde pour f; D' est minimale. Elle contient une relation R_k qui admet A comme constituant. Il suffit d'appliquer le même raisonnement que précédemment.

9

ANNEXE: AMORTISSEMENT DE CYCLES FONCTIONNELS

Nous avons mis en évidence dans un chapitre précédent (chap.7) le rôle des df à risque cyclique dans le maintien de la cohérence d'une base de données. Les df à risque cyclique sont associées aux cycles fonctionnels (§ 3.4.3.) que l'on peut détecter dans le graphe de relation d'une structure fondamentale GR_f. Même si tout cycle fonctionnel d'un GR_f ne contient pas obligatoirement une df à risque cyclique, un grand nombre d'entre eux en contient une et demande pour le maintien de la cohérence de la base, le déclenchement des *mécanismes* de complétude des entités et des clés. Comme ces mécanismes sont relativement coûteux dans le contrôle de cycle, nous proposons maintenant de minimiser leur utilisation.
Dans ce but, nous allons *"amortir"* certains cycles fonctionnels, c'est-à-dire les transformer. Nous obtenons ainsi un nouveau graphe de relation GR_{am}.

Cette étude provient de (LUONG(86).
Après avoir indiqué le principe d'une telle opération, nous nous attacherons à déceler les conditions qui doivent être remplies pour qu'une telle opération soit *valide*. Cette validité est étroitement liée à la *fidélité* (§ 3.2.) de GR_{am} par rapport à la décomposition initiale.
Ainsi les conditions de validité vont assurer que l'amortissement envisagé d'un cycle fonctionnel préserve:

(f1) le contenu de la base,
(f2) l'accès aux entités,
(f3) l'efficacité de la validation des df.

Décomposition d'une relation

Nous commençons ce chapitre par un exemple.

Exemple 1:

Voici le graphe de relation GR_f obtenu par notre algorithme:

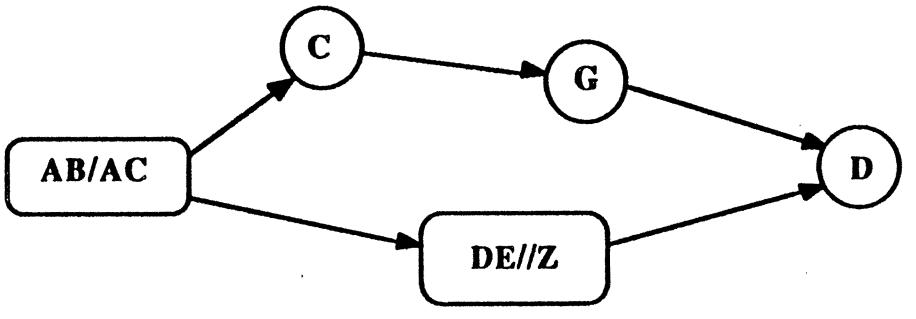

Figure 9.1. Cycle fonctionnel à amortir

Les relations de la décomposition initiale étaient: R_s(ABCDE), R_1(CG), R_2(GD) et R_{oa}(DEZ).

L'amortissement de ce cycle fonctionnel va conduire au graphe de relation suivant GR_{am}:

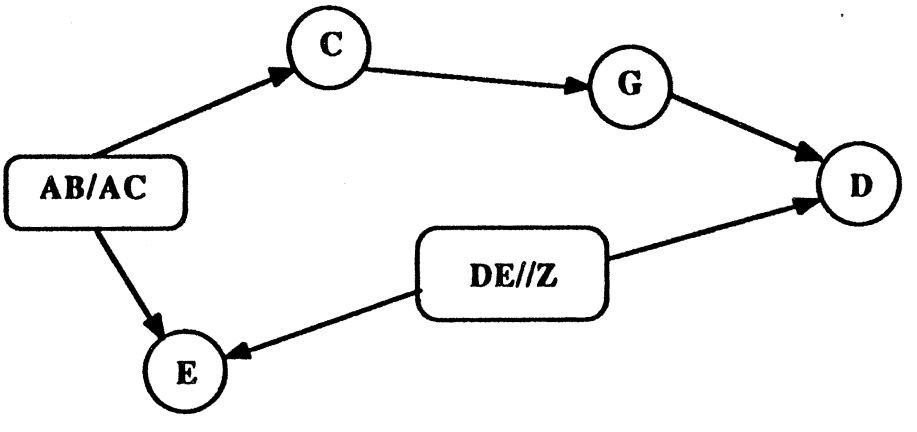

Figure 9.2. Cycle fonctionnel amorti

GR_{am} ne contient plus de cycle fonctionnel.

La df G → D qui est à risque cyclique dans GR_f: $((R_s * R_1)[GD])$, ne l'est plus dans GR_{am}.

Les relations déduites de GR_{am} sont $R_s'(ABCE)$, $R_1(CD)$ et $R_{oa}(DEZ)$.

Comme on peut facilement retrouver R_s en composant R_s' R_1 et R_2, le contenu de la base est préservé. L'ensemble des df intrinsèques dans GR_{am} permet de déduire $AB \rightarrow D$ et $AC \rightarrow D$.

L'accès à l'entité de R_{oa} relative à une entité de R_s' se fait au travers de R_1 et de R_2. Nous allons considérer cet amortissement comme valide et retenir GR_{am} comme structure de la base de données.

Par contre il reste une clé cyclique DE et il faut assurer que: $(iR_s' * iR_1 * iR_2)$ [DE] $\subseteq iR_{oa}$ [DE].

9.1. Amortissement d'un cycle fonctionnel

DÉFINITION

Soit un graphe de relation GR contenant un ensemble de cycles fonctionnels (CF) de même source R_s.
L'*amortissement* de cet ensemble de CF consiste à transformer R_s en une nouvelle relation R_s' telle que:

(am1) $R_s'^+ \subseteq R_s^+$ et $\forall$ A $\in$ $R_s^+ - R_s'^+$ il existe une relation T_i de MINFD(R_s) (§ 4.1. E3) telle que A est un constituant clé de T_i;

(am2) $KR_s^+ = KR_s'^+$;

Toute relation T_i de MINFD(R_s) telle que KT_i^+ n'est pas inclus dans $R_s'^+$ est appelée *origine de l'amortissement*; nous la notons dorénavant R_{oai}.

Soit GR_{am} le graphe de relation obtenu après cette transformation; comme il ne contient plus d'arc reliant les noeuds de R_s' et R_{oai}, il contient moins de cycles fonctionnels que GR.

Exemple 2:

Voici un graphe de relation GR_f:

Décomposition d'une relation

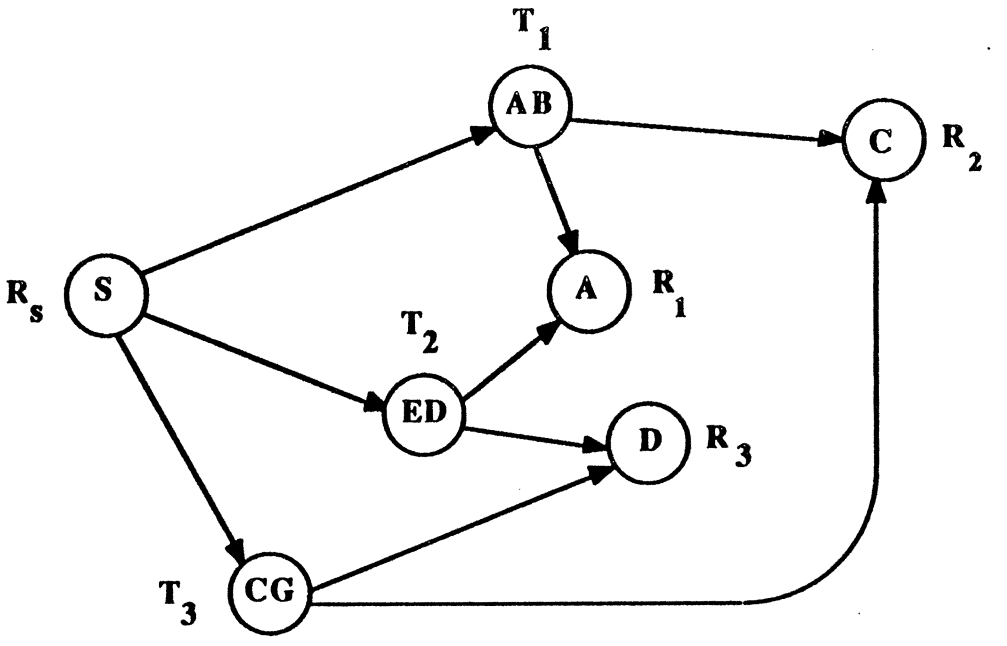

Figure 9.3. Cycle fonctionnel (CF)

R_s est formée sur les constituants SABCDEG.

Il existe trois CF, de même source R_s et de puits respectifs R_1, R_2 et R_3.

Voici une première possibilité d'amortir ces cycles (GR_{am1}):

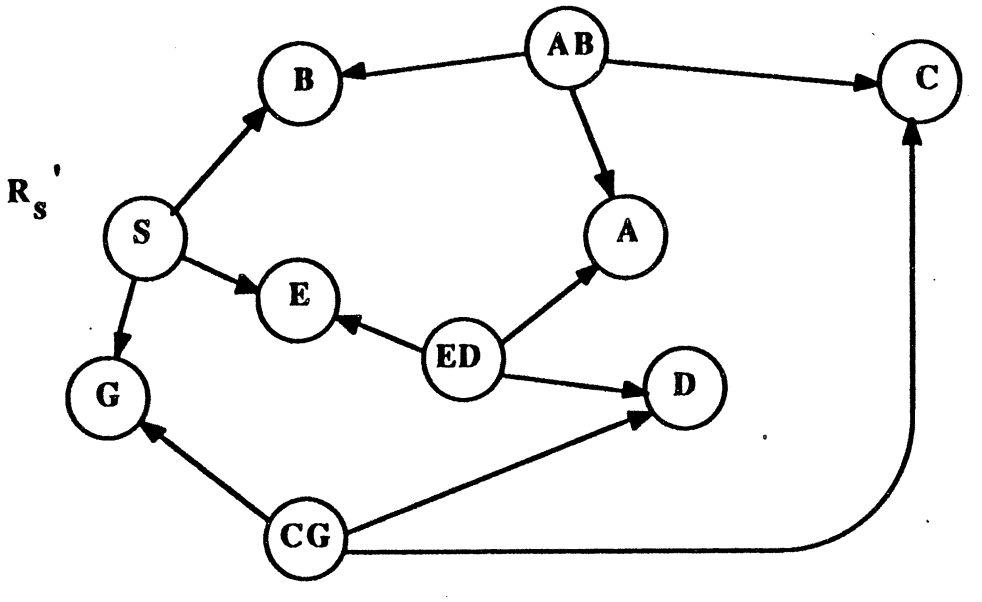

Figure 9.4. Amortissement 1 de CF

R_s' est formée sur les constituants SBEG.

Voici une seconde possibilité d'amortir ces cycles (GR_{am2}):

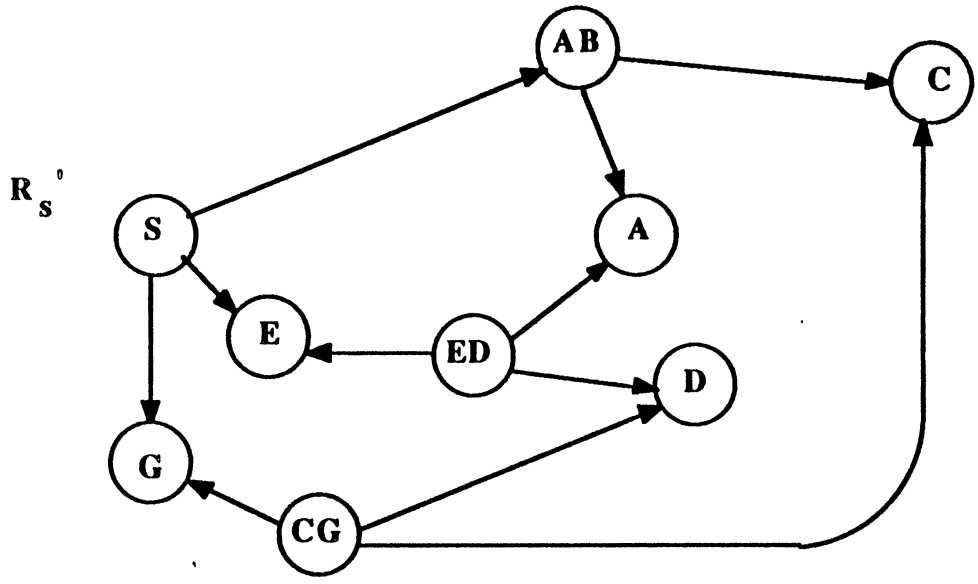

Figure 9.5. Amortissement 2 de CF

GR_{am2} ne contient aucun cycle fonctionnel.

Les paragraphes suivants vont montrer que GR_{am1} et GR_{am2} ne sont pas valides et nous ne les retiendrons pas.

9.2. Validation d'un amortissement

Dans l'introduction de ce chapitre, nous avons situé la validation d'un amortissement dans le contexte du respect des conditions de fidélité à la décomposition initiale.
Nous allons maintenant préciser ces conditions dans le cadre de cette étude.

Soit GR le graphe de relation initial, GR_{am} le graphe de relation obtenu après amortissement, R_s la relation de GR transformée en R_s' par l'amortissement, R_{oai} les relations origines de l'amortissement.

9.2.1. Préserver le contenu de la base et l'efficacité de la validation des df

Il faut que certaines relations de GR_{am} forment une décomposition fonctionnelle qui permettent de retrouver R_s. Il faut donc que l'ensemble des df intrinsèques des relations de GR_{am}, F_{am} ait la même fermeture que l'ensemble de celles de GR: F.

(am3) $(F_{am})^{**} = F^{**}$.

Exemple:

Dans l'exemple 2, GR_{am_1} ne vérifie pas cette condition:

$S \rightarrow ADC$ n'appartient pas à $(F_{am})^{**}$;
$F_{am} = (S \rightarrow BEG; AB \rightarrow C; ED \rightarrow A; CG \rightarrow D)$.
Ainsi GR_{am_1} n'est pas valide.
Par contre GR_{am_2} vérifie cette condition.

9.2.2. Ne pas augmenter le nombre de df à risque cyclique

(am4) L'ensemble des df à risque cyclique de GR_{am} doit être inclus ou égal à celui défini dans GR.

Si cette condition n'est pas remplie, l'amortissement ne sert à rien.
Il peut même conduire à des situations pires comme le montre l'exemple suivant !

Exemple:

Dans l'exemple 2, GR_{am_2} ne vérifie pas cette condition.

Dans GR_f, il n'y a aucune df à risque cyclique (malgré les cycles): par exemple $ED \rightarrow A$ est intrinsèque à T_2 et se trouve définie dans R_s : elle est seulement à risque linéaire.

Par contre cette même df $ED \rightarrow A$ est à risque cyclique dans GR_{am_2} : elle est intrinsèque à T_2 et se trouve définie dans $R_s' * T_1 * T_3$ qui est une décomposition minimale pour cette df.

9.2.3. Préserver l'accès

Soit $R_s'^*$ la relation qui peut être obtenue par composition de toutes les relations de GR_{am} dont on peut atteindre les noeuds par un écoulement du noeud de R_s'.

Toutes les clés d'une relation R_{oa} origine d'un amortissement qui appartiennent à l'ensemble des constituants de $R_s'^*$ sont dites *privilégiées* par rapport aux autres clés de R_{oa}.

(am5) GR_{am} préserve l'accès si pour chaque relation origine de l'amortissement R_{oa}, le concepteur accepte que l'association d'une entité rs' de R_s' et d'une entité r_{oa} de R_{oa} passe nécessairement par la connaissance de la valeur associée à une clé privilégiée de R_{oa} et ne peut plus être faite par la connaissance de la valeur associée à une autre clé de R_{oa}.

Exemple:

Soit le GR_f suivant:

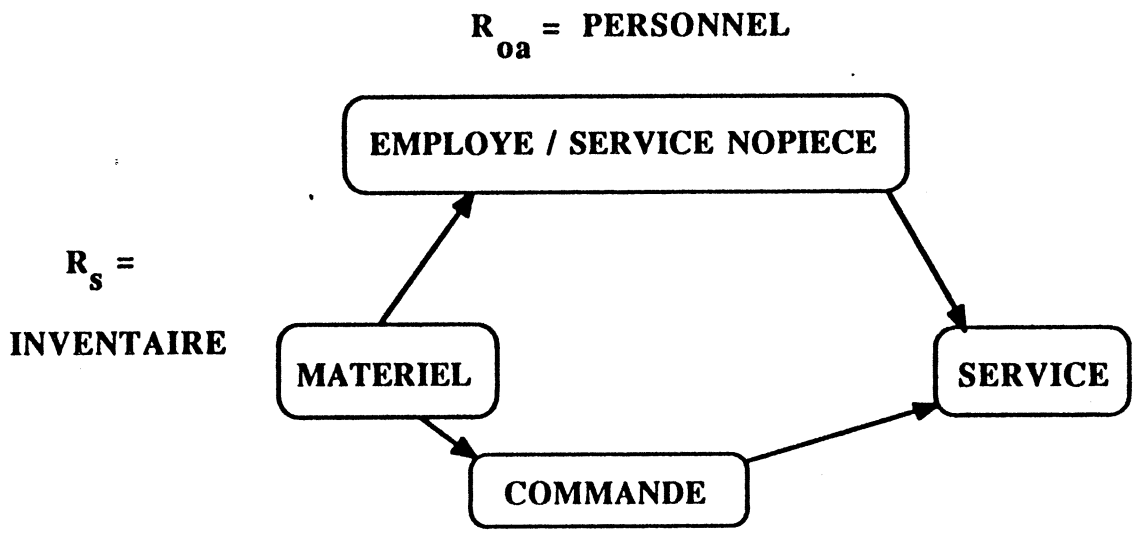

Figure 9.6.Inventaire

Relation **INVENTAIRE:**
Un matériel est acheté lors d'une commande, il est utilisé par un employé et se trouve dans une pièce d'un service (NOPIÈCE SERVICE).

Relation **COMMANDE:**
Une commande est passée par un seul service.

Décomposition d'une relation

Relation **PERSONNEL:**
Un employé travaille dans une seule pièce de son service. Dans une pièce travaille un seul employé.

Voici le GR_{am} obtenu:

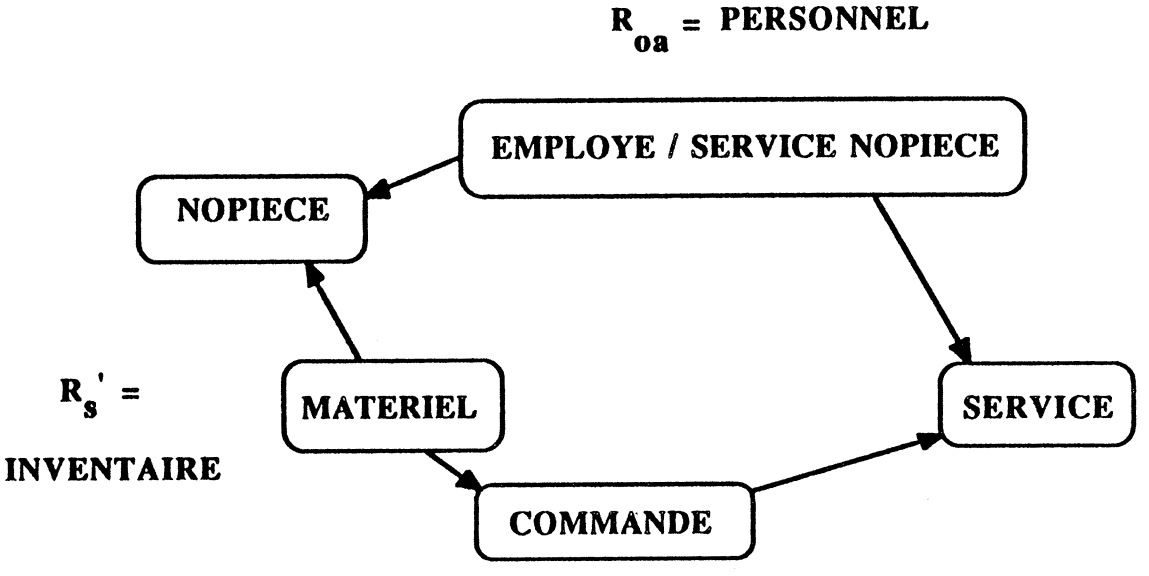

Figure 9.7. Amortissement de Inventaire

GR_{am} est valide seulement si pour le concepteur, les associations entre une entité de INVENTAIRE (R_s) et une de PERSONNEL (R_{oa}) se font par la connaissance de la pièce où se trouve le matériel et non par celle de l'employé qui utilise le matériel.

Par contre si le concepteur pense que certaines de ces associations se font par la connaissance de l'employé qui utilise le matériel, alors GR_{am} n'est pas valide et il faut conserver GR_f.

QUATRIÈME PARTIE

PERSPECTIVES

10

PERSPECTIVES

Nous avons tenté tout au long de ces chapitres de travailler une nouvelle matière celle de l'information structurée qui va être stockée dans une base de données. Cette matière est commune au domaine de l'informatique et au domaine de l'organisation. C'est en parlant de cette matière qu'informaticiens et gestionnaires travaillant à la mise en place d'une base de données peuvent se comprendre. Autant leur fournir un langage adapté à leurs discours (relation, graphe de relation, ...). Notre objectif est de participer à la formation de concepteurs de bases de données et pour nous, de telles personnes doivent devenir des spécialistes de cette nouvelle matière.

Dans le premier paragraphe, nous rappelons les différentes parties du processus de conception d'une structure d'une base de données en resituant le cadre des interventions du concepteur.

A l'aide de ces résultats, nous introduisons dans le second paragraphe des extensions au modèle relationnel de données, qui proviennent de différents auteurs. Nous n'avons pas voulu le faire auparavant car ces extensions auraient alourdi notre présentation. Ces extensions fournissent un langage plus riche aux gestionnaires et aux informaticiens pour se comprendre et aux concepteurs pour spécifier. Mais ces spécifications doivent être facilement implantées dans des outils informatiques. Les SGBD doivent évoluer et proposer des modèles de données plus étendus que le simple modèle relationnel de données: avec d'autres chercheurs, nous (JUNET-FALQUET-LÉONARD86), avons développé un exemple d'un tel nouveau SGBD que nous appelons ÉCRINS. Finalement dans le troisième paragraphe, nous présentons de nouveaux mécanismes que devraient contenir de futurs SGBD afin de faciliter considérablement (à

notre point de vue) la réalisation d'applications de bases de données. Nous avons découvert ces mécanismes par notre étude du domaine de la conception et la place que nous donnons à ce paragraphe (le dernier !) témoigne de l'importance que nous lui accordons.

10.1. Processus de conception de bases de données

Notre approche de la conception d'une structure de bases de données se divise en trois parties, comme de nombreuses autres approches:

- détermination d'une première modélisation de données;

- détermination d'une structure logique de données;

- détermination d'une structure informatique de données.

10.1.1. Détermination d'une première modélisation de données

C'est une partie d'analyse du champ d'application. Le modèle relationnel de données permet de stocker de manière précise les résultats des analyses. Nous avons montré comment les règles d'intégrité peuvent être d'une aide efficace dans cette partie. Non seulement les analystes doivent les mettre en évidence pour garantir la cohérence des données dans la future base de données, mais ils peuvent en les cherchant dans des situations remarquables, approfondir leur modélisation: notamment les dépendances d'inclusion sont d'une aide indispensable pour la compréhension des cycles dans une modélisation.

Nous avons montré aussi combien il est nécessaire pour des modélisations de données un peu complexes de déterminer les espaces et les contextes: les futurs utilisateurs intrépides pourront poser des questions ne respectant pas les contours de ces espaces ou de ces contextes; ils le feront à leurs risques et périls mais la responsabilité des concepteurs sera dégagée dans le cas où ces futurs utilisateurs intrépides ou inconscients

poseraient des questions sans discernement en planant au-dessus des contours des espaces ou des contextes.

Finalement le dossier de modélisation va comprendre:
les domaines,
les constituants, chacun avec ses domaines,
les relations, chacune avec ses constituants,
ses clés (dont une obligatoire),
les valeurs inconnues permises,
les règles d'intégrité, chacune avec son contexte, sa portée,
les espaces et les contextes.

Pour chaque espace U nous allons travailler avec une de ses *décompositions complètement homogènes*, afin de ne privilégier aucune clé à ce stade du processus (c'est une différence avec d'autres approches de conception (BERNSTEIN76)). Dans ce but, nous avons deux possibilités qui correspondent à deux approches déjà connues:
- approche par les df (appelée aussi approche par synthèse). En fait, on considère l'ensemble des df définies sur U et on construit une décomposition de U seulement à partir des df. Dans ce cas, nous préconisons l'emploi de *la* décomposition fondamentale de U.
- approche par décomposition. On conserve la décomposition de U formée à l'aide des relations obtenues précédemment lors de la modélisation. Dans ce cas, nous préconisons de rajouter à chaque relation toutes ses clés potentielles pour obtenir une décomposition complète et ensuite de la rendre complètement homogène et compacte.
Avec l'une ou l'autre de ces approches nous obtenons une décomposition complètement homogène D_h de U qui ne privilègie aucune clé.

10.1.2. Détermination d'une structure logique de données

L'algorithme GR transforme la décomposition homogène D_h dans un graphe orienté de relation GR_h de manière automatique. Les compétences du concepteur vont lui permettre d'éliminer certaines charnières et même de renommer certains constituants et d'amortir certains cycles: pour chaque cycle le concepteur doit, s'il souhaite l'amortir, accepter qu'une clé d'une relation soit privilégiée par rapport aux autres. Bien sûr c'est un algorithme qui indique les cycles qui peuvent être amortis et, pour chacun d'eux, la relation et la clé qu'il convient de privilégier.
Une fois tout ce travail réalisé, un algorithme introduit les éventuelles arêtes binaires (§ 4.1.1. E6).

10.1.3. Détermination d'une structure informatique de données

Cette partie dépend du système de gestion de bases de données utilisé. Nous en avons présenté les aspects importants dans (§ 5.).

Le concepteur peut dédoubler des noeuds,
supprimer des chemins d'accès,
inverser des relations.

Ces transformations sont contrôlées par un algorithme qui garantit par exemple qu'il existe toujours un algorithme simple d'obtention des entités de la relation espace.

Pour bien remplir sa tâche, le concepteur doit posséder des résultats de l'analyse des futurs traitements, essentiellement des fréquences de suppressions d'entités et d'accès aux entités.

10.1.4. Commentaire

La transformation de la décomposition homogène initiale dans une structure informatique de données réclame les compétences du concepteur. Son travail est guidé par des algorithmes qui garantissent la fidélité de la structure informatique finale à la décomposition homogène initiale.

Notre approche place le concepteur d'une structure de données face à des choix dont il peut clairement comprendre les enjeux. Des questions aussi importantes que les formes normales de relation sont partiellement réduites à un seul point de vue d'optimisation et peuvent être traitées de manière purement algorithmique.

Nous allons comparer nos résultats avec ceux d'autres approches à l'aide d'exemples.

10.1.4.1. Exemple

Soit la relation R (SABCEGHX) munie de l'ensemble de df suivant:
$F = \{S \rightarrow ABCEGH, E \rightarrow C, H \rightarrow A, AB \rightarrow CG, CG \rightarrow ABX\}$.

L'approche purement booléenne donnerait le choix suivant de bases irredondantes au concepteur de la base:

$FIRR_{11}$ = {S → EHB, E → C, H → A, AB → CGX, CG → AB};
$FIRR_{12}$ = {S → EHB, E → C, H → A, AB → CG, CG → ABX};
$FIRR_{21}$ = {S → EHG, E → C, H → A, AB → CGX, CG → AB};
$FIRR_{22}$ = {S → EHG, E → C, H → A, AB → CG, CG → ABX}.

AB et CD étant des clés de R[ABCGX], on peut considérer que le concepteur doit choisir entre d'une part $FIRR_{11}$ (ou $FIRR_{12}$) et d'autre part $FIRR_{21}$ (ou $FIRR_{22}$), ce qui conduit à deux décompositions possibles de R:
D_1 : R = R[SEHB]*R[EC]*R[HA]*R[ABCGX];
D_2 : R = R[SEHG]*R[EC]*R[HA]*R[ABCGX].
Le GR associé à D_1 est le suivant:

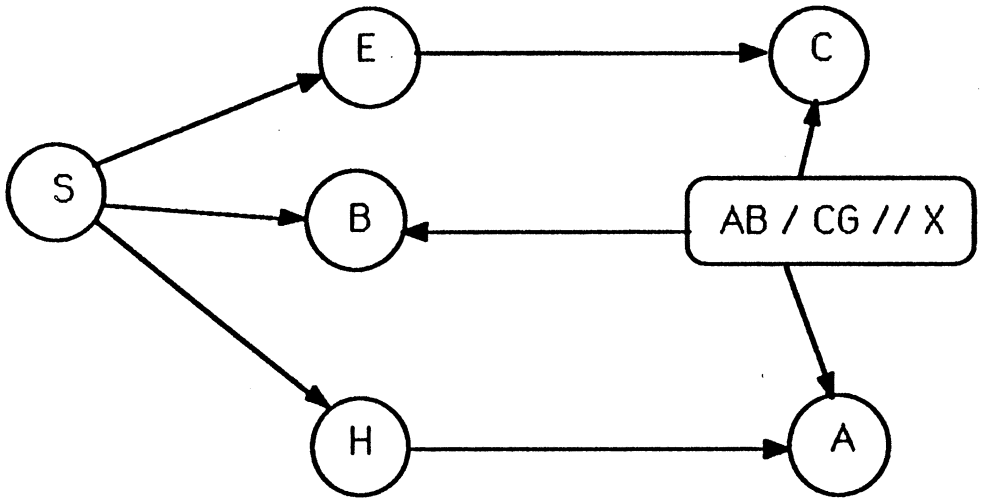

Figure 10.1. GR construit à partir d'une base irredondante

Dans D_1, la df E → C est à risque cyclique.
Les créations des entités des relations de D_1 ne peuvent donc être exécutées indépendamment les unes des autres si l'on veut garantir la cohérence globale;
en effet l'existence de : (s b) dans R[SB], (s h) dans R[SH], (h a) dans [RHA], (a b c g x) dans R[ABCGX], (s e) dans R[SE], (e c') dans R[EC], nécessite : c = c'.

D_2 conduit à la même difficulté.

Notre approche conduit à ne pas amortir le cycle fonctionnel et obtenir le graphe de relation suivant:

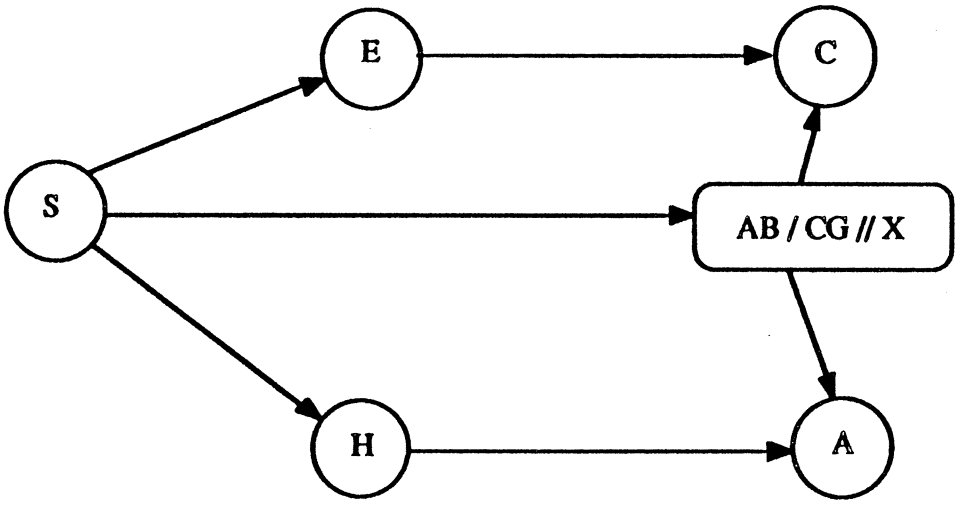

Figure 10.2. Notre solution

Ce GR correspond à la décomposition suivante:
R = R[SABCEGH] * R[EC] * R[HA] * R[ABCGX].

Les df E → C et H → A sont seulement à risque linéaire et un amortissement rend l'une ou l'autre à risque cyclique (propriété am4 du § 9.).

10.1.4.2. Exemple (PICHAT-DELOBEL79)

Soit la relation R(ABCDEFG) munie d'un ensemble de df dont voici une des bases irredondantes :
{G → F, AF → E, A → BC, C → A, D → A, H → I, I → J}.
DHG est la clé de R.

Voici la décomposition directe que l'on peut en déduire :
R = R[DHG] * R[GF] * R[AFE] * R[ABC] * R[DA] * R[HI] * R[IJ].

Elle privilégie le rôle de A par rapport à celui de C alors qu'elles sont deux clés équivalentes de R[ABC].

Par notre approche, nous obtenons la décomposition fondamentale suivante:
R = R[DHGACF] * R[GF] * R[ACFE] * R[DAC] * R[ACB] * R[HIJ] * R[IJ].

La relation R[ACFE] est source d'un amortissement valide (§ 9.) et nous obtenons le graphe de relation suivant:

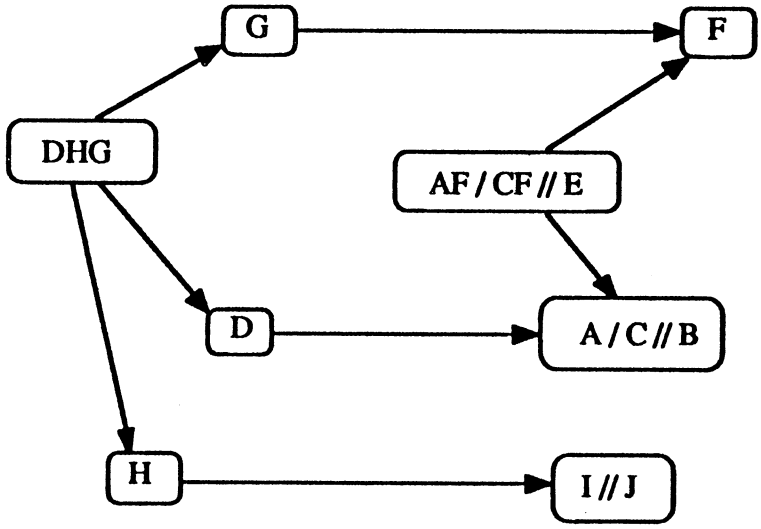

Figure 10.3. GR obtenu pour l'ex.Pichat-Delobel

10.1.4.3. Exemple (BEERI-BERNSTEIN79)

Soit la relation R (ABCDE) munie de l'ensemble des df:
$F = \{A \rightarrow BC, BC \rightarrow A, AD \rightarrow E, E \rightarrow C\}$.

Nous proposons la décomposition fondamentale D suivante de R
R = R[ABC] * R[ABCDE] * R[EC] * R[C] dont le GR est le suivant:

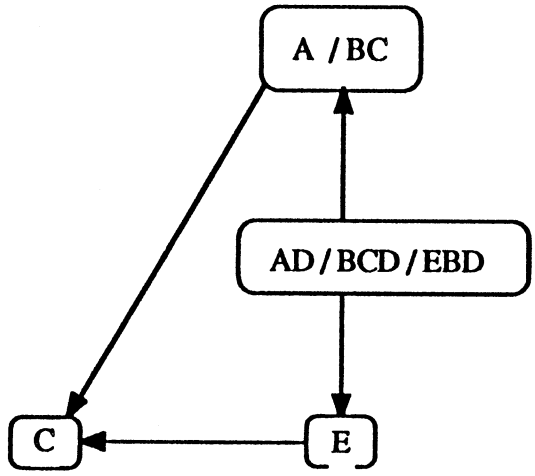

Figure 10.4. GR obtenu pour l'ex. Beeri- Bernstein

Les auteurs proposent deux décompositions:
$D_1 = \{R_1 = R[ABC], R_2 = R[ADE], R_3 = R[EC]\}$;
$D_2 = \{S_1 = R[ABC], S_2 = R[BCDE], S_3 = R[EC]\}$.

Perspectives

La seule différence entre D_1 et D_2 d'une part et D d'autre part est que R_2 et S_2 ne contiennent pas toutes leurs clés.
Les auteurs notent que la structure déduite de D_1 est en forme normale de Boyce Codd Kent (BCKNF) alors que celle déduite de D_2 ne l'est pas. S'il est vrai que les mises à jour d'une relation en BCKNF peuvent être exécutées de manière indépendante, par contre les mises à jour de relations isolées, toutes en BCKNF et formant une décomposition, doivent parfois être contrôlées de manière globale pour faciliter l'interrogation.
Ainsi si l'on accepte la décomposition D_1, si (a b c) $\in$ iR_1, (a d e) $\in$ iR_2, (e c') $\in$ iR_3, le contrôle global exige c = c'.

Les différences entre D_1, D_2 n'apparaissent plus comme évidentes, et nous proposons la décomposition D qui a l'avantage de donner un rôle équivalent aux clés AD et BCD.

10.1.4.4. Exemple (BERNSTEIN76)

Soit la relation R (ABC X1 X2) munie de l'ensemble des df:
F = {X1X2 → A, C → X1X2, AX1 → B, BX2 → C}.

(BERNSTEIN76) fournit comme décomposition finale:
D'={R_1'=R[X1 X2 C A], R_2'=R[A X1 B], R_3'=R[B X2 C]}
en donnant comme clés de R_1': X1 X2 et C,
R_2': A X1,
R_3': B X2.

En fait B X2 est également une clé de R_1' car X1 X2 → A X1 → B d'une part et B X2 → C → X1X2 d'autre part.
Nous n'avons aucune information sur le bien fondé de la décomposition de la relation R_1 = R[X1 X2 C A B] dans les deux relations R_1' R_3'. De plus cette décomposition D' demande que toute mise-à-jour soit vérifiée à l'aide des instances des 3 relations, si l'on veut éviter des contradictions.
Notre proposition fait apparaître un cycle fonctionnel qui, traité par les mécanismes que nous proposons (§ 10.3.), évite les contradictions.

Voici la décomposition que nous proposons:
D = {R_1 = R[X1 X2 C A B], R_2 = R[A X1 B], R_3 = R[B]};
clés de R_1: X1X2, C, B X2.

Cette décomposition ne privilégie aucune clé et fournit toutes les clés des relations.

10.2. Modéle relationnel de données étendu

Depuis l'introduction du modèle relationnel de données par (CODD70), de nombreux autres modèles de données sont apparus. Ils essaient tous d'avoir une syntaxe plus riche afin de prendre en compte plus facilement des modélisations mêmes complexes. Ils facilitent ainsi la détermination d'une première modélisation de données. Nous présentons ici seulement 4 concepts.

10.2.1. Sous-séquence de constituants d'une relation

Une relation est formée sur un ensemble de constituants rangés en séquence. Une *sous-séquence de constituants* dans cette relation est commandée par un constituant de la relation appelé *constituant générique* de la sous-séquence dont le domaine doit être de type mot.
Elle est soumise à une proposition à une variable qui prend ses valeurs dans le domaine du constituant générique.

Une entité d'une relation admet des valeurs pour les constituants de cette sous-séquence seulement si la proposition est vérifiée par la valeur prise par l'entité pour le constituant générique.

Exemple:

Soit la relation PERSONNE formée sur les constituants NOM, -CIVIL .et la sous-séquence de constituants DATE-DE-MARIAGE, LIEU-DE-MARIAGE.
DATE-DE-MARIAGE et LIEU-DE-MARIAGE constituent une sous-séquence de constituants de la relation PERSONNE:
une entité de PERSONNE ne prend des valeurs pour ces constituants que si la valeur prise pour le constituant générique ÉTAT-CIVIL n'est pas "célibataire".

Remarque:

Les sous-séquences peuvent être imbriquées les unes dans les autres.

10.2.2. Sous-relation

Une *sous-relation* SR est une relation définie par rapport à une *relation de référence* R et commandée par un constituant de R, appelé *constituant générique* de SR (SMITH77).
Le constituant générique doit avoir un domaine de type mot et n'appartenir à aucune sous-séquence de R.
La sous-relation SR est liée à une seule valeur du domaine du constituant générique.

Une entité de R est une entité de SR seulement si elle prend cette valeur pour le constituant générique. Réciproquement, une entité de SR peut toujours être considérée comme une entité de R.

Deux sous-relations de R ne peuvent être associées à la même valeur d'un même constituant générique.

De manière récursive, il est possible de définir une nouvelle sous-relation SSR admettant SR comme relation de référence: alors on dit que SSR est également une *sous-relation de R*. Cette expression se justifie par le fait qu'une entité de SSR est une entité de SR et donc de R.

Une sous-relation SR de R peut avoir des constituants propres.

Exemple:

Soit la relation PERSONNE admettant pour constituants:
NOMPERS fournissant le nom et le prénom de la personne et servant de clé à la relation PERSONNE;
EMPLOI indiquant si une personne est "au travail","chômeur", "hors travail" ou "retraité(e)".

CHOMEUR, AUTRAVAIL sont des sous-relations admettant PERSONNE comme relation de référence; elles admettent EMPLOI comme constituant générique.

Une entité de PERSONNE est une entité de CHOMEUR si elle prend la valeur "chômeur" pour EMPLOI; elle l'est de AUTRAVAIL si elle prend la valeur "au travail" pour EMPLOI.

CHOMEUR admet un constituant propre NBANCH indiquant le nombre d'années de chômage de la personne.

10.2.3. Relation d'association ou relation associative

Une relation d'association (CHEN76) est une relation définie par rapport à deux *relations de référence* R_1 et R_2. Les entités d'une relation d'association permettent d'associer une entité d'une des deux relations de référence avec une entité de l'autre. Chaque relation de référence remplit un *rôle* dans l'association et à chaque rôle on fait correspondre un paramètre *cardmax* indiquant le nombre maximum d'entités RA que l'on peut construire à partir d'une entité de la relation de référence remplissant ce rôle. Il peut exister plusieurs relations d'association distinctes ayant les mêmes relations de référence.

Exemple:
Soit les relations PERSONNE et VILLE qui ne sont pas des relations d'association et soit les deux relations d'association NAISSANCE et HABITATION admettant toutes les deux PERSONNE et VILLE comme relations de référence. PERSONNE remplit le rôle "personne née à" dans NAISSANCE, et à une personne on fait correspondre au maximum 1 entité de NAISSANCE: une personne a une seule ville de naissance ! PERSONNE remplit le rôle "personne habitant à" dans HABITATION. VILLE remplit le rôle "ville de naissance de" dans NAISSANCE, et à une ville on fait correspondre un nombre maximum inconnu d'entités de NAISSANCE: le nombre maximum de personnes nées dans une ville n'a pas de sens comme règle d'intégrité. Elle joue un rôle "ville d'habitation de" dans HABITATION avec également un cardmax inconnu.
fex.

Dans le cas où les deux relations de référence R_1 et R_2 sont, soit une même relation, soit des sous-relations d'une même relation R, R pouvant être l'une des deux relations R_1 ou R_2, la relation d'association s'appelle alors une *boucle*.
Si la boucle porte sur une même relation, elle peut être *symétrique* si les deux rôles sont identiques, ou *non symétrique* dans le cas contraire; sinon elle est forcément non symétrique.

Exemple:
Soit la même relation PERSONNE et soit les boucles AMI(E) et FILIATION admettant PERSONNE comme relation de référence. AMI(E) est une boucle symétrique si "la personne X est amie de la personne Y" est équivalent à "la personne Y est amie de la personne X". Dans ce cas, les deux rôles d'AMI(E) sont identiques avec une cardinalité

maximale inconnue. FILIATION est non symétrique et admet deux rôles distincts: PARENTS DE et ENFANT DE. La cardinalité maximale associée au rôle ENFANT DE est 2. Ainsi, à une entité de PERSONNE, on peut à travers le rôle ENFANT DE créer au plus 2 entités de FILIATION.
fex.

La relation de référence R_1 de RA remplit un rôle *déterminant* dans RA si sa clé est inclue dans la clé de RA. Si les clés sont égales, à une entité de la relation R_1 correspond au plus une entité de RA à travers ce rôle (cardinalité maximale égale à 1).
Dans l'exemple précédent, PERSONNE remplit un rôle déterminant dans NAISSANCE mais ne l'est pas de HABITATION.

10.2.4. Relation complémentaire

Une relation complémentaire RC est une relation ayant une relation de référence R; la clé de RC est formée de celle K de R à laquelle on a rajouté un ou plusieurs constituants propres à RC. Une entité de RC ne peut exister que si l'entité de R ayant les mêmes valeurs pour K existe.

Exemple:
La relation QUALIFICATION est une relation complémentaire de la relation PERSONNE: elle admet pour clé le constituant NOMPERS, formant la clé de PERSONNE, et NOQUALIF (numéro de qualification) qui admet un domaine de type rang. Les autres constituants de QUALIFICATION fournissent le libellé (LIBELLÉ) de la qualification, le nombre d'années d'expérience de cette personne dans cette qualification (NBANNÉES), le lieu principal où elle a fait un travail exigeant cette qualification (LIEUTRAVAIL), le diplôme principal que la personne a obtenu dans cette qualification (DIPLOME) et la date d'obtention de ce diplôme (DATEDIPLOME).
Ainsi à une personne, c'est-à-dire à une entité de la relation PERSONNE, on peut faire correspondre les différentes qualifications de cette personne à travers les entités de la relation complémentaire QUALIFICATION avec les informations propres à chaque qualification.
Pour créer une entité de QUALIFICATION d'une personne X, il faut qu'une entité de PERSONNE existe dans la base pour cette personne X. La suppression d'une entité de PERSONNE provoque la suppression des entités de QUALIFICATION qui lui sont rattachées.
fex.

Le paramètre *cardmax* d'une relation complémentaire RC désigne le nombre maximum d'entités qui sont associées à une entité de la relation de référence R.
Dans notre exemple, le cardmax de QUALIFICATION est de 5 si à une personne correspond au plus 5 qualifications pour la base de données considérée.

10.2.5. Relation de base

Une *relation de base* est une relation qui n'est ni une sous- relation, ni une relation complémentaire, ni une relation d'association.

10.2.6. Représentation graphique

La représentation graphique d'un ensemble de relations formant une structure de données est construite à partir d'un graphe dont les arêtes représentent les relations d'association, les noeuds les autres relations (BOUILLÉ78).

Un noeud est associé à une et une seule relation R qui n'est pas d'association. Le nom de R est inscrit à l'intérieur de la courbe fermée représentant ce noeud.

Il peut contenir lui-même d'autres noeuds qui sont associés à des relations R_1...R_n: celles-ci doivent alors être des sous-relations de R. Les noeuds de sous-relations relatives à un même constituant générique de R sont reliés au nom de R par un même arbre ayant pour racine le symbole ou exclusif.

Une arête peut être associée à une seule relation d'association RA. Elle porte le nom de cette relation comme renseignement. Elle relie les noeuds N_1 et N_2 des relations de référence R_1 et R_2 de RA.
Elle peut être orientée ou non: une flèche orientée dans le sens de N_1 vers N_2 signifie qu'à une entité de R_1 peut correspondre au plus une entité de R_2 à travers RA; une arête non orientée de N_1 vers N_2 signifie alors qu'à une entité de R_1 peuvent correspondre plusieurs entités de R_2.

Si une arête n'est pas associée à une relation d'association, alors elle relie le noeud d'une relation complémentaire à celui de sa relation de référence. Elle est alors représentée en un trait gras.

Exemple:
Soit les relations des exemples précédents PERSONNE, VILLE, HABITATION, NAISSANCE, QUALIFICATION, CHOMEUR, AUTRAVAIL, AMI, FILIATION. Nous rajoutons la relation d'association LIEUTRAVAIL ayant comme relations de référence AUTRAVAIL et VILLE. Nous supposons que le cardmax du rôle de AUTRAVAIL dans LIEUTRAVAIL est de 1: à une entité de AUTRAVAIL peut correspondre au plus une entité de VILLE à travers LIEUTRAVAIL.
La représentation graphique de l'ensemble de ces relations est la suivante:

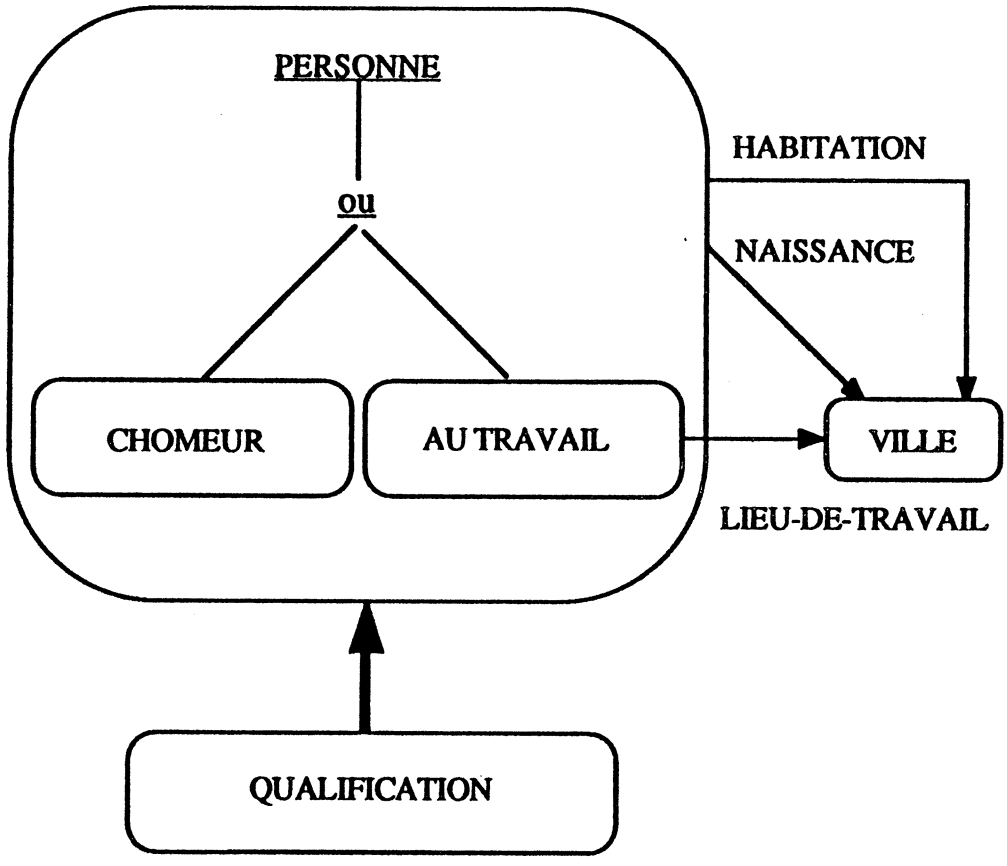

Figure 10.5. Modélisation avec le modèle relationnel étendu

10.2.7. Conclusions

Les concepts de relation d'association et de relation complémentaire peuvent très facilement être traduits en termes du simple modèle relationnel de données. Ils apparaissent communément comme plus simples à utiliser dans l'établissement d'une modélisation et c'est la raison de leur utilité.

Par contre les concepts de sous-séquence de constituants et de sous-relations ne sont traduisibles en termes du simple modèle relationnel de données qu'en termes laborieux avec l'écriture de règles d'intégrité fastidieuses. Pourtant si l'on utilise des systèmes de gestion de bases de données relationnelles, dont le modèle de données est le simple modèle relationnel de données, il faut en passer par là. Nous pensons que la solution de l'avenir est celle de la réalisation de SGBD dont le modèle de données intègre ces concepts (VELEZ-LOPEZ87, projet TIGRE par exemple).
En fait, au fur et à mesure que les bases de données sont utilisées dans des domaines les plus divers, il faut se rendre compte qu'un modèle de données général de bases de données n'est plus assez pertinent. Il y aura à notre avis de plus en plus de modèles de données spécifiques et il faut espérer qu'il y aura des SGBD acceptant ces modèles de données spécifiques: nous donnons un exemple dans le domaine de la CAO (RIEU84), un autre dans le domaine de l'économétrie (SNELLA84, projet PIRÉE), un autre dans celui de l'analyse de données (LÉONARD86, projet FARANDOLE). Dans ces trois exemples, les chercheurs ont développé et le modèle de données spécifique et le SGBD l'acceptant.

10.3. Mécanismes de contrôle

Pour garantir artificiellement l'étanchéité de décompositions nous avons élaboré deux mécanismes: celui de complétude de clés et celui de complétude d'entités. Nous allons les spécifier dans le cas de décompositions complètement homogènes et nous allons constater qu'ils s'insèrent dans un cadre plus vaste de mécanismes qui nous apparaissent comme fondamentaux.

Dans un premier paragraphe, nous allons présenter tous ces mécanismes. Dans un second paragraphe, nous montrons qu'ils permettent de rendre artificiellement étanche **une décomposition complètement homogène et compacte** et qu'ils valident les df intrinsèques ou non.

Notation:
GR_h désigne le graphe orienté de relation déduit d'une décomposition complètement homogène et compacte D_h d'une relation R par l'algorithme GR;
"-" désigne une valeur obscure.

Perspectives

Définitions:

Une *relation étendue* R^* d'une relation R est obtenue par composition des relations formant un écoulement de source R dans GR_h. A une entité r de R correspond une *entité étendue* r^*, entité de R^* telle que $r^*.R^+ = r$.

Un *chemin d'entités* $(r_1.r_2...r_n)$ est un ensemble d'entités compatibles dont les relations $R_1, R_2,..., R_n$ forment un chemin dans GR_h.

Les instances des relations de D_h dont nous allons parler, forment une *instance de la base de données.*

10.3.1. Mécanisme de clés

Pour toute relation R_i de D_h et pour toute clé K_i de R_i, le mécanisme de clés garantit qu'il ne peut exister dans une instance iR_i deux entités prenant la même valeur pour K_i.

10.3.2. Mécanisme linéaire

Soit une relation R_i dont l'ensemble de constituants contient une clé K_j d'une autre relation R_j.
Si une entité r_i de iR_i prend une valeur claire k_j pour K_j, *le mécanisme linéaire garantit* qu'il existe une entité r_j de iR_j prenant la valeur k_j pour K_j.

De plus si R_i^+ contient une autre clé K_{j1} de R_j, *ce mécanisme garantit* que pour chaque constituant A_{j1} de K_{j1}:
si $r_j.A_{j1}$ est claire alors $r_j.A_{j1}=r_i.A_{j1}$ sinon $r_i.A_{j1}$ est obscure.

Ce mécanisme solutionne les problèmes relatifs aux clés à risque linéaire et aux df à risque linéaire: en effet dans ces deux cas, les décompositions minimales à considérer se réduisant à une seule relation, le mécanisme précédent garantit que:

$iR_i [K_j] \subseteq iR_j [K_j]$ et même $iR_i [K_j K_{j1}] \subseteq iR_j [K_j K_{j1}]$
aux valeurs inconnues près (cf. mécanisme de clés partielles).

10.3.3. Mécanisme complémentaire d'association (mécanisme du hors piste)

La situation la plus simple de son utilisation est illustrée par l'exemple de la relation U(ABCXYZ) qui se décompose suivant $R_1(ABX)$ $R_2(BCY)$ et $R_3(CZ)$. A cette décomposition fondamentale correspond le graphe de relation GR_{h1}:

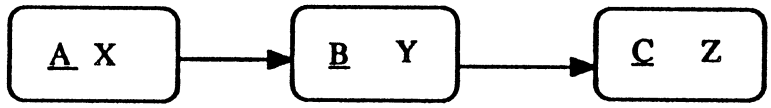

Figure 10.6. Mécanisme de hors piste

Dans l'utilisation de cette base de données, il peut se faire que l'on connaisse l'entité (ac) de U[AC] avant de connaître les entités (ab) de U[AB] et (bc) de U[BC]. Mais avec la décomposition fournie, il n'est pas possible de stocker dans la base l'entité (ac).
Le but du mécanisme complémentaire d'association est de le permettre. Nous introduisons la notion d'*arc complémentaire* orientée du noeud de (AX) à celui de (CZ) et d'association complémentaire entre les relations (AX) et (CZ). Elle signifie que si à une entité (ax) de R_1 ne correspond aucune entité (by) de R_2, il est possible de lui associer une entité (cz) de R_3 par une association complémentaire . Ce mécanisme va veiller à ne pas stocker de la redondance dans la base de données: ainsi dès que l'on va connaitre l'entité (by) correspondant à l'entité (ax), l'association complémentaire entre (ax) et (cz) est détruite; mais comme les faits (ac) et (ab) existent et qu'il y a la df $B \rightarrow C$, le mécanisme va en déduire la création de l'association entre (by) et (cz). Bien sûr si l'entité (by) est déjà associée à une autre entité (c'z'), le mécanisme va refuser l'association de l'entité (ax) avec celle (by) si c et c' sont différents.

Un *chemin d'entités est complémentaire* s'il contient au moins une association entre entités qui est complémentaire. L'utilité des associations complémentaires ne concerne pas seulement le stockage de données dans la base; il concerne également la suppression de données dans la base.

Ainsi dans le premier exemple, on suppose que la base contienne les entités (abx), (bcy) et (cz). Que l'on supprime l'entité (bcy) ou que l'on supprime l'association (ab) ou (bc) en mettant à jour les entités (a-x) ou (b-y), la manière habituelle de stockage des données dans une base provoque la suppression de l'association (ac). Ce comportement peut

conduire à des pertes d'information. L'utilisation des associations complémentaires fournit un remède facile. Si une association complémentaire existe entre les relations (*AX*) et (*CZ*), les suppressions envisagées précédemment ne pourront être exécutées que si l'utilisateur a signifié au système s'il doit supprimer également l'entité (ac) ou la conserver.

Pour simplifier la présentation de ce mécanisme, nous allons supposer qu'il existe une association complémentaire systématiquement entre deux relations R_i et R_j, R_i supérieure à R_j sans lui être directement supérieure. Bien sûr il serait possible d'affiner ce canevas: le concepteur de la base pourrait choisir les associations complémentaires nécessaires. A un degré plus fin, on pourrait distinguer les associations complémentaires servant lors de la création d'associations, celles servant lors de la suppression d'entités ou d'associations ou celles servant aux deux !

Avant d'en arriver à ce stade de détails, il faut convaincre les responsables de systèmes de gestion de bases de données de la relative facilité à implanter un tel mécanisme; il pourrait selon nous, faciliter considérablement le travail de développement des applications d'une base de données et celui de la définition d'une structure de base de données.

Soit $R_i > R_j$ et R_i n'est pas directement supérieure à R_j.
Le mécanisme complémentaire d'association garantit que si une association complémentaire existe entre deux entités r_i et r_j alors il n'existe aucun chemin entre r_i et r_j dans la base; de plus pour aucune entité r_m de iR_m avec $R_i > R_m > R_j$, il n'existe aucune association complémentaire entre r_i et r_m.

10.3.4. Mécanisme des clés partielles; entité obscure

Voici la situation la plus simple qui illustre le fonctionnement de ce mécanisme. Soit la relation U(SABXY) qui se décompose suivant R_1(*S*AB), R_2(*AB*X), R_3(*A*Y). Le graphe de relation correspondant est alors le suivant:

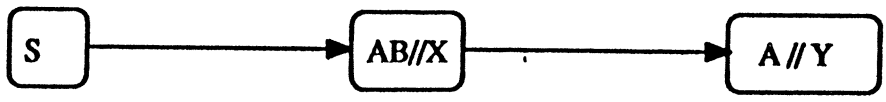

Figure 10.7. Mécanisme de clés partielles

Dans l'utilisation de cette base de données, il se peut que l'on connaisse l'entité (sa-). Implicitement cette entité fait une association entre une entité r_1 de iR_1 de clé s et une entité r_3 de iR_3 de clé a. Comme R_1 n'est pas directement supérieure à R_3 il suffit de créer une association complémentaire entre r_1 et r_3. Dans ce cas particulier (R_2 est dépendant de R_3), le mécanisme de clés partielles va l'implanter simplement en créant une entité (a--) dans iR_2. Une telle entité est appelée *obscure* car pour aucune clé elle ne prend une valeur claire.

Une entité obscure peut être associée avec une autre entité obscure ou non (voir mécanisme de cycle). Elle prend à sa création la valeur obscure pour tout constituant non clé. Deux entités obscures sont incomparables sauf dans un cas:
soit deux relations R_i et R_j, une clé de l'une K_i contenant une clé de l'autre K_j. On peut ainsi écrire $K_i=K_j$ X. Il existe une entité obscure r_j et deux entités r_{j1}, r_{i2} de iR_i également obscures telles que les valeurs de r_{j1}.X et r_{i2}.X sont claires et identiques. Si r_{j1} et r_{i2} sont associées toutes les deux à la même entité r_j, alors elles sont identiques.
Voici un exemple simple qui illustre cette situation.

Exemple:

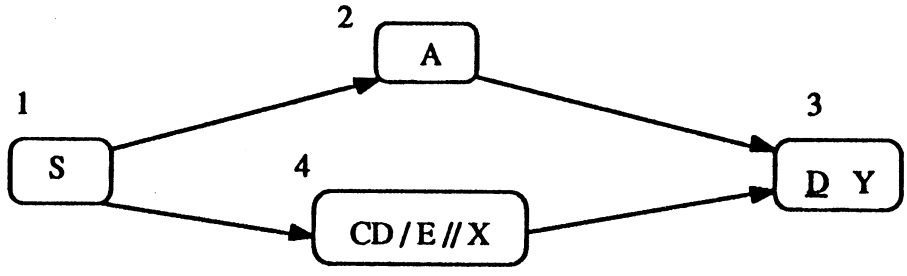

Figure 10.8. Situation où des entités obscures peuvent être égales

Les relations sont R_1(*S*ACDE), R_2(*A*D), R_3(*D*Y) et R_4 (CDEX). La base contient r_1=(sac--) et r_1'=(s'ac--) dans iR_1, r_2=(a-) dans iR_2, r_3 dans iR_3 et r_4=(c---) et r_4'=(c---), obscures dans iR_4. (r_1,r_2,r_3) (r_1',r_2,r_3), (r_1',r_4,r_3) forment des chemins d'entités.
Ici r_4 et r_4' sont identiques car elles sont associées à la même entité r_3 et prennent la même valeur pour C. Ainsi leurs valeurs pour la clé CD ne peuvent qu'être identiques.

Remarque:

Le mécanisme de clés partielles est une extension du mécanisme linéaire.

Soit deux relations R_i et R_j et une clé K_j de R_j contenue dans R_i^+.
Le mécanisme de clés partielles garantit que pour toute entité r_i de iR_i telle que $r_i.K_j$ est obscure, il existe une entité r_j dans iR_j éventuellement obscure telle que $r_i.K_j=r_j.K_j$ et qu'il existe un chemin d'entités (éventuellement complémentaire) reliant r_i à r_j.
Il garantit de plus que si $r_j.K_j$ devient claire, alors $r_i.K_j$ le devient également.

10.3.5. Mécanisme des clés cycliques

Voici un exemple que illustre son fonctionnement.
Soit la décomposition fondamentale suivante d'une relation:
$U=R_1(SACDEX) * R_2(AB) * R_3(CDY) * R_4(CEZB) * R_5(BCV)$
et le graphe de relation correspondant:

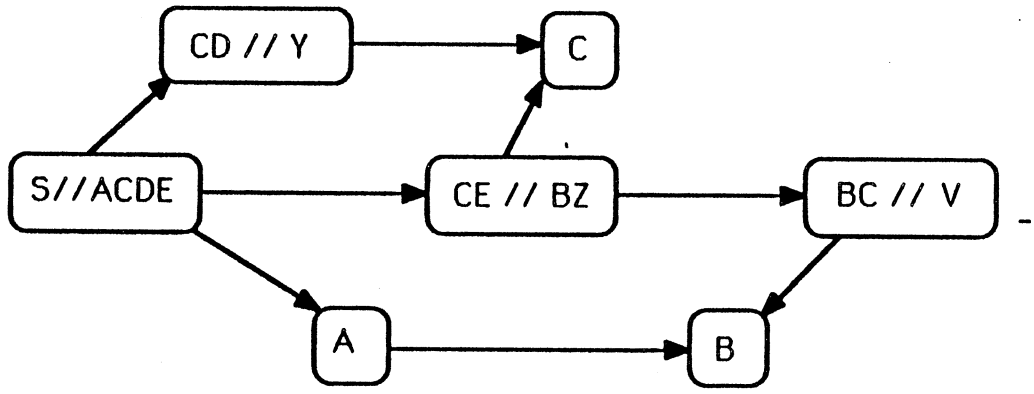

Figure 10.9. Mécanisme de clés cycliques

Voici les entités stockées dans la base: (cdy) et (ab).

On crée l'entité r_1=(sacd-x). Que fait le mécanisme de clés cycliques ?

Le mécanisme des clés cycliques va créer r_4=(c--) dans iR_4 et associer r_1 et r_4. Le mécanisme des clés cycliques va remarquer que BC est une clé à risque cyclique à cause de la décomposition fonctionnelle partielle $R_1*R_2*R_3$; il constate la création d'une nouvelle valeur (bc) de BC. Comme iR_5 est pour l'instant vide, il crée l'entité r_5=(bc-) dans iR_5. Ensuite il va déclencher le mécanisme complémentaire d'association pour s'assurer qu'un chemin complémentaire ou non dans la base existe entre r_1 et r_5 : il va créer l'association entre r_4 et r_5.

Le *mécanisme des clés cycliques garantit que* s'il existe une entité s dont une entité étendue s* prend une valeur claire ou obscure pour une clé KR d'une relation R alors
- il existe une entité r éventuellement obscure dans iR telle que pour chaque constituant A de KR tel que s*.A est claire, r.KR = s*.KR.
- il existe un chemin entre s et r qui est éventuellement complémentaire.
Ce mécanisme s'intéresse aux clés à risque cyclique. Pour toute df partielle D' de D_h qui ne contient pas R, et dont KR est un sous ensemble de constituants, ce mécanisme garantit que iD'[KR] $\subseteq$ iR[KR]. Il intervient dès qu'une entité d'une instance d'une relation de D' est créée ou dès que l'on tente de supprimer une entité de iR.

Remarque:
Ce mécanisme de clés cycliques peut être considéré comme une extension du mécanisme linéaire. Il est une extension de ceux proposés par (SMITH77; ZANIOLO79; KUCK-SAGIV83).

10.3.6. Mécanisme de cycle

La situation la plus simple qui correspond à ce mécanisme, est illustrée par la relation U(SABCDWXYZ) qui se décompose suivant R_1(*S*AC), R_2(*A*BX), R_3(*B*DY), R_4(*C*DZ), R_5(*D*W). En voici le graphe de relation:

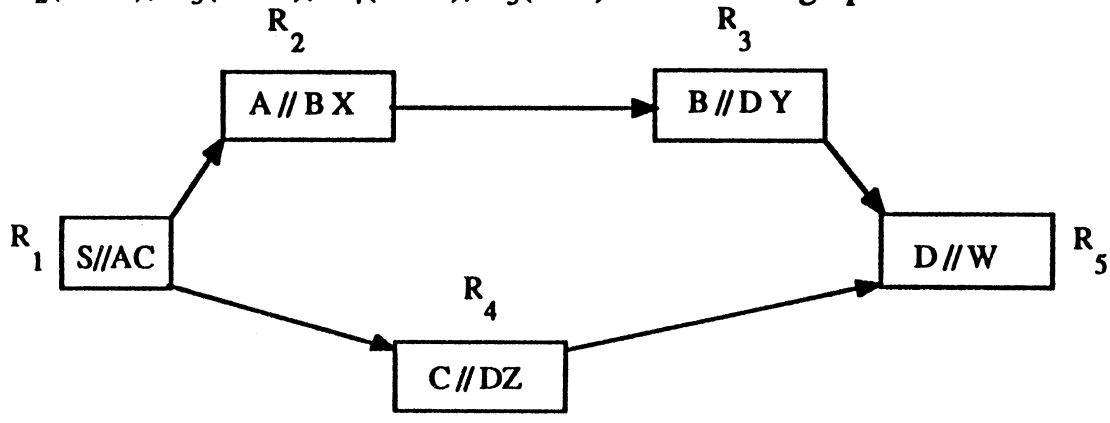

Figure 10.10. Mécanisme de cycle

Voici un état de la base qui comprend les entités (sac) (a-x) (cdz) (dw) et (d'w'). On crée ensuite l'entité (bd'y) et l'association (ab). Cet état de la base est incohérent car il existe deux chemins d'entités :
(sac) (abx) (bd'y) (d'w) et (sac) (cdz) (dw).
Le premier associe s à d', le second s à d alors qu'il existe la df S $\rightarrow$ D déduite des autres par transitivité.

Perspectives

Le mécanisme de cycle va justement vérifier que les deux chemins d'entités convergent vers la même entité du puits du cycle.
Pour cela il va toujours chercher à anticiper plutôt que d'être dans la position brutale de constater l'"erreur". Ainsi à partir du premier état de la base, il constate qu'il existe un chemin qui partant d'une entité de la source du cycle r_1=(sac) arrive à associer à r_1 une entité de la relation puits (sac) (cdz) (dw). L'autre chemin est de longueur 1: (sac) (a-x). (Il serait de longueur 0 si l'on avait seulement (s-c)).
Le mécanisme va déclencher le mécanisme complémentaire d'association pour créer une association complémentaire entre (a-x) et (dw): en effet à partir des données stockées dans la base, il est possible de déduire l'entité (ad) de U[AD].
Cette association complémentaire est implantée à l'aide d'une entité obscure de R_3.

Le *mécanisme de cycle* s'applique au cas d'un cycle fonctionnel de source S et de puits P; pour toute entité s de iS, il *garantit* l'existence d'un chemin d'entités pour chacune des branches du cycle qui relie s à la même entité p de iP. Chaque chemin d'entités est formé d'une entité éventuellement obscure pour chaque relation de la branche.

Ce mécanisme intervient dès que l'on crée ou supprime une association entre deux entités de relations appartenant à un même cycle. On peut trouver dans (LUONG86) la description d'une partie de ce mécanisme.

Ce mécanisme est plus fin que celui proposé par (SAGIV81) et (BROSDA-VOSSEN85) comme l'a montré (LUONG86).
Il est très proche du processus de chasse ("chase process" de MAIER-MENDELZON-SAGIV79) appliqué aux seules df.

Dans l'exemple précédent, nous supposons maintenant que la base de données contient une seule entité r_2=(a-x) dans iR_2.

La création de l'entité r_1(sa-) dans iR_1 déclenche le mécanisme du cycle; celui-ci provoque l'association entre r_1 et r_2 et surtout crée les entités obscures r_3 r_4 r_5 de telle manière que (r_1 r_2 r_3 r_5) et (r_1 r_4 r_5) forment deux chemins d'entités.

Ensuite la création de l'entité r_1'=(s'a-) dans iR_1 déclenche également le mécanisme du cycle, celui-ci constate le chemin d'entités (r_1' r_2 r_3 r_5), et provoque la création d'une entité obscure r_4' pour former le chemin (r_1' r_4' r_5).

10.3.7. Utilité de ces mécanismes

THÉORÈME

Le mécanisme de clés, le mécanisme complémentaire d'associations, le mécanisme de clés cycliques, le mécanisme de cycle et le mécanisme de clés partielles garantissent la validité des df intrinsèques et des df pouvant être générées à partir des df intrinsèques. Deux décompositions fonctionnelles partielles D_1 et D_2 concernées par l'une de ces df sont compatibles. Ces mécanismes garantissent artificiellement l'étanchéité de la décomposition homogène initiale.

Nous démontrons ce théorème dans l'annexe.

Un prototype (ALCOREZA88) de tous ces mécanismes a été réalisé pour un seul cycle fonctionnel.

10.3.8. Exemples

a) Voici les relations formant la décomposition:
{S(*S*ABCEGH); R_1 (EGHX); R_3 (*AB*); R_4 (*BC*E)}.
Les clés de R_1 sont EG et H; les df se déduisent des clés.
Voici le graphe de relation:

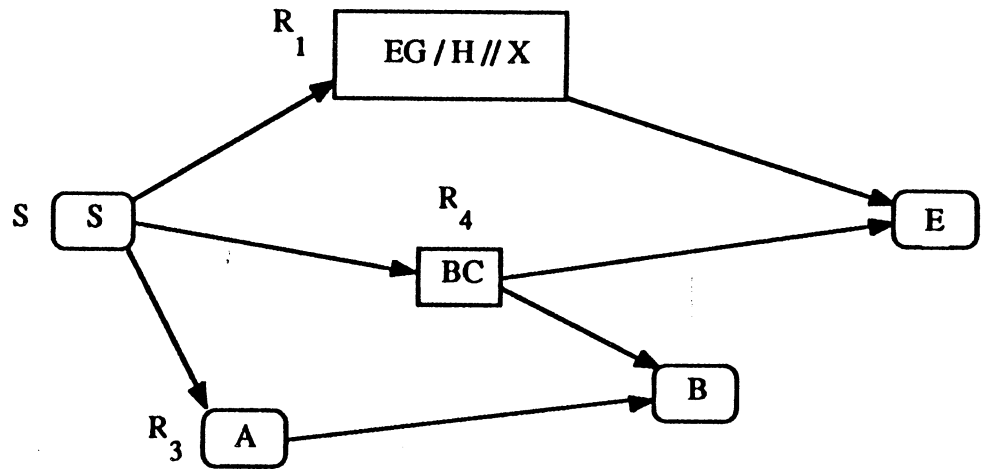

Figure 10.11. Utilisation des mécanismes (1)

La df f: AC → E se déduit de A → B et BC → E; elle est définie dans la relation S. Bien sûr comme AC n'est pas une clé de S, le mécanisme de

clés appliqué à iS n'est pas chargé de la valider.
Soit la décomposition fonctionnelle partielle D' formée de S [SABC], R_3 et R_4. Deux entités s et s' vérifiant s.AC = s'.AC en clair, existent dans iS s'il existe une entité r_3 vérifiant r_3.A = s.A dans iR_3.
De plus le mécanisme de clés partielles crée si besoin une entité r_4 de iR_4: r_4.C = s.C, et une autre r_4': r_4'.C = s'.C.
Le mécanisme de cycle appliqué au cycle fonctionnel de source S et de puits B, garantit que r_4.B = r_3.B = r_4'.B.
Il en déduit que r_4 et r_4' sont identiques.
s et s' sont ainsi reliées à la même entité de iR_4; leurs entités étendues à D' prennent la même valeur pour E. f est valide dans iD'.

Elle l'est également dans S à cause du mécanisme du cycle appliqué au cycle fonctionnel (S R_1 R_4); il garantit que les entités r_1 associée à s et r_1' associée à s' vérifient: r_1.E = s.E = r_4.E = s'.E = r_1'.E.

b) Voici le graphe de relation d'un autre exemple:

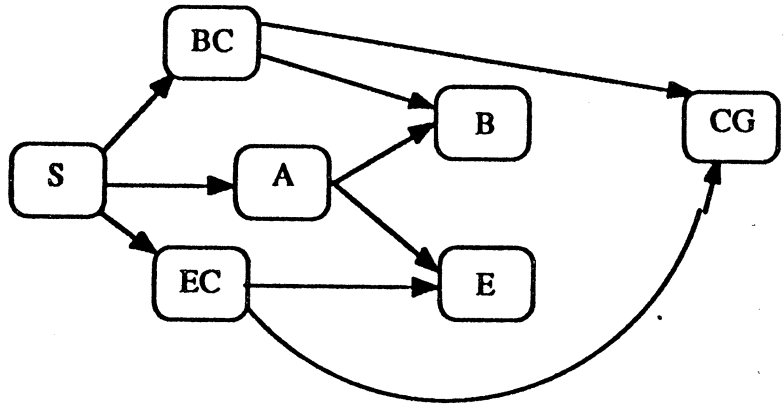

Figure 10.12. Utilisation des mécanismes (2)

La df f: AC → G n'est intrinsèque d'aucune relation; elle se déduit aussi bien de A → B et BC → G que de A → E et EC → G.
Il existe deux décompositions fonctionnelles partielles concernées par f: d'une part {S[SABC], R_2} et d'autre part {S[SACE], R_3}.

S'il existe dans la base une entité s de iS telle que s.AC est claire, alors le mécanisme des clés partielles construit si besoin, une entité r_2 dans iR_2 telle que s.C=r_2.C, et une entité r_3 dans iR_3 telle que r_3.C = s.C .
Le mécanisme de cycle appliqué au cycle fonctionnel de source S et de puits R_5 et celui de clés partielles associent à ces deux entités une entité r_5 de iR_5 telle que r_5.C=r_3.C=r_2.C=s.C.
Ainsi s est associée à une seule entité de R_5 et à une seule valeur de G que

ce soit par le chemin passant par R_2 ou par le chemin passant par R_3.

Si maintenant il existe une autre entité s' de iS vérifiant s'.AC = s.AC en clair, il est facile de montrer avec le même raisonnement que précédemment, que s et s' sont associées aux mêmes entités r_2 et r_3 à cause des mécanismes de clés partielles et de cycle appliqués respectivement au cycle fonctionnel de source S et de puits R_6, et à celui de souce S et de puits R_7. Ainsi s et s' sont associées à la même entité r_5.

Non seulement les deux décompositions fonctionnelles valident f, mais elles sont compatibles entre elles pour f.

c) Voici le graphe de relation d'un dernier exemple:

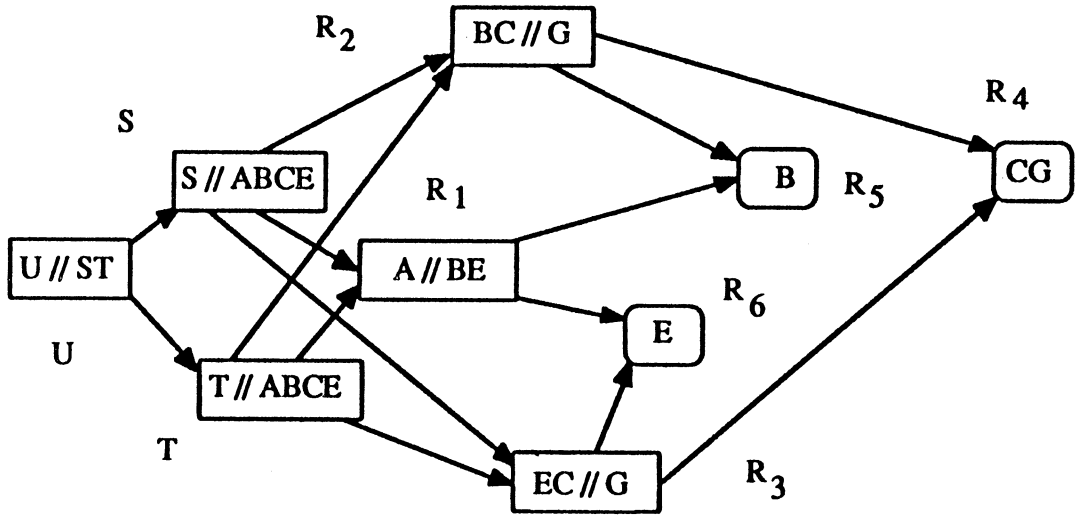

Figure 10.13. Utilisation des mécanismes (3)

AC → G se déduit de A → B et BC → G et aussi de A → E et EC → G.
S'il existe une entité s=(sac--), il existe l'entité r_1=(a--) et de plus le mécanisme de clés partielles crée si besoin r_2=(-c-) et r_3=(-c-).

Le mécanisme de cycle appliqué au cycle (S R_2 R_4 R_3) impose que r_2.G=r_3.G.Il impose également que r_2.B=r_1.B à cause du cycle (S R_2 R_5 R_1) et également r_3.E = r_1.E à cause du cycle (S R_3 R_6 R_1)

Si maintenant on crée une entité t=(tac--), elle est reliée à r_1 et donc à r_5 et r_6. Le mécanisme de clés partielles la relie à une nouvelle entité r_3'=(-c-).
Le mécanisme de cycle appliqué à (T R_1 R_6 R_3) impose que r_3'.E=r_1.E et donc que r_3'.E = r_3.E. r_3 et r_3' sont identiques.
Ainsi s et t sont associées à la même entité de R_4 et AC → G est validée !

10.4. Finalement

Comme nous l'avons expliqué dans l'introduction, nous ne fournissons aucune méthodologie de conception d'une structure d'une base de données. Bien que nous divisions en trois parties le processus de conception, nous ne fournissons aucune méthode de conception. Nous ne disons pas qu'un concepteur doit commencer par la première partie, puis la deuxième partie puis terminer par la troisième partie. Non ! Ces trois parties servent seulement à distinguer les problèmes de conception entre eux.

Pour nous cet ouvrage n'est ni une méthodologie, ni une méthode de conception de bases de données mais seulement une réflexion sur la base des écrins que sont les bases de données.

10.5. Annexe: utilité de ces mécanismes

Nous allons montrer que tous ces mécanismes garantissent la validation de toute df intrinsèque et l'étanchéité artificielle de la décomposition pour les df intrinsèques et pour toute clé.
De plus ils valident également les df qui ne sont pas intrinsèques.
Soit la relation T de clé X et la df $X \rightarrow A$ intrinsèque à T.

10.5.1. Propriété (M1)

La df $X \rightarrow A$ est validée dans iT grâce au mécanisme de clés.

10.5.2. Propriété (M2): étanchéité artificielle par rapport à X

Le mécanisme linéaire garantit que pour toute relation R qui est directement supérieure à T, si iR contient une entité r telle que r.X est claire, alors il existe une entité t dans iT telle que r.X=t.X. Le mécanisme de clés cycliques étend ce résultat à toute relation R qui n'est pas

directement supérieure à T, mais dont toute relation étendue ne contenant pas T contient une ou plusieurs clés de T comme constituants.

Ces deux mécanismes garantissent donc artificiellement l'étanchéité de toute décomposition fonctionnele partielle par rapport à X.
En particulier le mécanisme linéaire garantit la règle d'intégrité existentielle de dépendance entre relations.

10.5.3. Propriété (M3): étanchéité linéaire artificielle par rapport à XA

Lemme:

Soit R > T et deux entités r et t dans la base reliées entre elles par un chemin d'entités complémentaire ou non. A est un constituant commun à R et T. r.A et t.A sont des valeurs claires. Avec l'application des mécanismes précédents r.A et t.A sont identiques.

Preuve:
Il existe une charnière RT entre R et T qui admet A comme constituant clé car la décomposition initiale est compacte.

a) Si les clés de RT ne sont pas celles de T, alors RT est distincte de T. D'après le mécanisme de clés partielles, il existe deux entités de iRT rt et rt' telles que rt.A = r.A et rt'.A = t.A et deux chemins d'entités d'une part entre t et rt' qui prolonge le chemin entre r et t, et d'autre part entre r et rt.
S'il y a un cycle de source R et de puits RT, une branche du cycle passant par T, l'autre non et si les deux chemins d'entités précédents correspondent à ces deux branches distinctes, alors l'application du mécanisme de cycle conduit à garantir que rt = rt' et ainsi r.A = t.A = rt.A.
Sinon il existe un seul chemin d'entités reliant r à rt. L'application du mécanisme complémentaire d'associations garantit que ce chemin passe par t; comme il existe une seule entité de iRT qui peut être reliée à t, rt et rt' sont identiques: rt.A = t.A = r.A.

b) Si les clés de RT sont celles de T, alors RT est confondue avec T et A est un constituant clé de T. D'après le mécanisme des clés partielles, r ne peut être reliée qu'à une entité t de iT telle que r.A = t.A.
*

Perspectives

Si une relation R admet comme constituants XA, alors pour toute entité r de iR telle que r.X est claire, il existe une entité t de iT telle que r.X = t.X d'après le mécanisme de clés cycliques; il garantit même l'existence d'un chemin d'entités de r à t.

Si maintenant r.A est claire, on peut appliquer le lemme précédent et en déduire r.A = t.A. Ainsi iR[XA] $\subseteq$ iT [XA].
Ainsi est garantie artificiellement l'étanchéité linéaire de toute décomposition fonctionnele partielle concernée par XA.

10.5.4. Propriété (M4): étanchéité cyclique

Lemme:
Soit R > T et deux entités r et t dans la base reliées entre elles par un chemin d'entités complémentaire ou non.
Soit R* une relation étendue de R ne contenant pas T.
A est un constituant commun à R* et T.
Avec l'application des mécanismes précédents, r*.A et t.A sont identiques quand l'une des deux au moins est claire.

Preuve:

Ce lemme se réduit au lemme précédent quand R* = R.
Soit V une relation qui a servi à construire R* et dont A est un constituant. V est distincte de R. R est supérieure à V par construction.
De plus iV est telle qu'elle contient une entité v reliée à r par un chemin d'entités et r*.A = v.A.
Il existe TV la charnière entre T et V qui admet A comme constituant clé car la décomposition est compacte.

a) Si les clés de TV ne sont pas celles de T, alors TV est distincte de T. D'après le mécanisme des clés partielles, il existe deux entités de iTV tv et tv' telles que tv.A = v.A et tv'.A = t.A et deux chemins d'entités reliant r à une entité de iTV: (r, v, tv) et (r, t, tv'): si ces chemins sont dstincts, il existe un cycle fonctionnel de source R et de puits TV et le mécanisme de cycle garantit que tv et tv' désignent une même entité; sinon c'est le mécanisme complémentaire d'association qui le garantit. Ainsi tv.A = v.A = t.A = r*.A.

b) Si les clés de TV sont les mêmes que celles de T, alors TV et T sont confondues et A est un constituant-clé de T. D'après le mécanisme de clés

partielles, v doit être reliée à une entité t' de iT vérifiant v.A = t'.A. Comme r est reliée à v par un chemin d'entités, elle est également reliée à t'; d'après le mécanisme de cycle ou le mécanisme complémentaire d'association garantit comme précédemment que t=t'; ainsi t.A = r*.A.

Si une relation R admet une relation étendue R* ne contenant pas T mais dont l'ensemble des constituants contient XA, alors pour toute entité r de iR telle que r*.X est claire, il existe une entité t de iT telle que r*.X=t.X d'après le mécanisme de clés cycliques; ce mécanisme garantit également l'existence d'un chemin d'entités de r à t.

Si maintenant r*.A est claire, on peut appliquer le lemme précédent et en déduire r*.A = t.A. Ainsi iR*[XA] $\subseteq$ iT[XA].

Ceci garantit artificiellement l'étanchéité cyclique et également la validité de X $\rightarrow$ A dans iR*.

10.5.5. Théorème (M5)

Les quatre mécanismes, le mécanisme de clés, le mécanisme complémentaire d'association, le mécanisme de clés cycliques et le mécanisme de cycle garantissent l'étanchéité des décompositions fonctionnelles partielles par rapport aux clés des relations formant la décomposition et par rapport aux df intrinsèques. Ils garantissent la validité des df intrinsèques dans la base de données.
Nous allons maintenant étendre ce résultat aux df qui ne sont pas intrinsèques.

10.5.6. Cas des df qui ne sont pas intrinsèques

Soit une df f X $\rightarrow$ A élémentaire qui peut être générée à partir des df intrinsèques.

10.5.6.1. Propriété (M6)

Nous allons démontrer que si les mécanismes précédents sont actifs, f est validée pour toute instance d'une décomposition fonctionnelle D admettant XA comme constituants. Soit S la source d'une telle décomposition.

Perspectives

Preuve:

a) Il existe une décomposition fonctionnelle particulière DG de source S et qui admet XA comme constituants.

En effet, f: X → A peut être générée à partir des df ($f_1 f_2 ... f_n$) qui sont intrinsèques et qui forment un arbre de génération bien formée (§ 8.11.). Soit $R_1, R_2 ... R_n$ les relations correspondantes. Nous avons démontré que $X \rightarrow g(f_i)$ pour i = (1...n) ($g(f_i)$ désignant la partie gauche du f_i). KS étant une clé de S, comme S est la source de D, KS → X existe et donc KS → $g(f_i)$. Nous avons demontré que $g(f_i)$ contient une clé de R_i. Ainsi S est supérieure à R_i et il existe un chemin dans le graphe de relation reliant S à chacune des relations R_i (propriété GR_3 § 8.11.6.3.). Toutes ces chemins forment une décomposition fonctionnelle particulière DG de source S et admettant XA comme constituants.

b) Nous allons montrer que iDG valide f: X → A grâce aux mécanismes. Nous supposons qu'il existe deux entités de iS étendues à DG: s* et s'* telles que s*.X et s'*.X sont claires et identiques. Dans le théorème de génération des df, nous avons appris que X est l'ensemble des constituants sources de l'arbre, et A le constituant puits. Donc il existe au moins une df f_1 dont la partie gauche est formée seulement de constituants sources X_1 inclus dans X. Une clé de R_1 est X_1 et le mécanisme complémentaire d'association garantit l'existence d'un chemin d'entités reliant s et s' à la même entité r_1 de iR_1 telle que $s^*.X_1 = r_1.X_1$. Il en est de même avec toutes les df. $f_2 ... f_p$ ($p \leq n$) telles que g(f) est inclus dans X. Le mécanisme complémentaire d'association garantit l'existence d'un chemin d'entités reliant s et s' à la même entité r_j ($j \leq p$) de iR_j telle que $s^*.X_j = r_j.X_j$. Et ainsi $s^*.A_j = s'^*.A_j$ où A_j désigne la partie droite de f_j.

Les parties droites A_j des df f_j ($1 \leq j \leq p$) ne peuvent pas être des constituants sources de l'arbre de génération; elles appartiennent à des parties gauches d'autres df; il existe au moins une df f_k telle que $g(f_k)$ contient seulement des constituants X_k de X et des A_j, par construction de l'arbre. Soit R_k la relation correspondante. Il existe un chemin reliant S à R_k dans le graphe de relation. D'après les mécanismes des clés cycliques et de clés partielles, il existe un chemin d'entités reliant s à une entité r_k de iR_k et un autre reliant s' à une entité r_k' de iR_k. Comme $s^*.X_k = s'^*.X_k$, $r_k.X_k = r_k'.X_k$. Si $s^*.A_j$ est claire, $r_k.A_j$ l'est également et $r_k.A_j = r_k'.A_j = s^*.A_j$.
Comme s et s' sont reliées à la même entité r_j, $r_k.A_j$ et $r_k'.A_j$ obscures ou non sont identiques (mécanisme de cycle ou complémentaire d'association). Ainsi s et s' sont reliées à la même entité r_k ($=r_k'$) de iR_k.

De proche en proche, on démontre que s et s' sont reliées à la même entité pour chaque iR_i ($1 \leq i \leq p$) formant DG. Ainsi s*.A =s'*.A que la valeur soit claire ou obscure.
Donc iDG valide f:X → A.

c) Nous allons montrer que iD valide X → A et que iD est compatible avec iDG pour f_1.
Soit S* la relation étendue de S à DG. Soit S_1* la relation étendue de S à D.

XA forme un sous-ensemble de S^{*+} et de S_1^{*+}. Pour tout constituant A_i de XA, il existe une relation R de DG et une relation R_1 de D (éventuellement égale à R) telles que A_i est un constituant de R et de R_1. Il existe donc une charnière CH entre R et R_1 qui admet A_i comme constituant clé car la décomposition est compacte: cette charnière peut être R ou R_1. Ainsi dans le graphe de relation soit il existe un chemin reliant S R et R_1 soit il existe un cycle de source S et de puits CH.

Si l'on considère une entité s de iS son entité étendue s* à DG et son entité étendue s_1* à D alors grâce soit au mécanisme du cycle dans le cas du cycle soit au mécanisme complémentaire d'association dans le cas d'un chemin reliant S, R et R_1, $s^*.A_i=s_1^*.A_i$. Et ainsi $s^*.X=s_1^*.X$. De même $s^*.A=s_1^*.A$.

Si l'on considère une autre entité s' de iS, son entité étendue s'* à DG et son entité étendue s_1'* à D telle que $s_1^*.X=s_1'^*X$ en clair, alors on peut en déduire d'après ce qui précède que $s^*.X=s'^*.X$ en clair. D'après le paragraphe précédent, comme iDG valide X → A, $s^*.A=s'^*.A$. Finalement d'après ce qui précède, $s_1^*.A=s_1'^*.A$.

iD valide également f: X → A et de plus elle est compatible avec iDG pour f. On sait même que iD[XA] et iDG[XA] contiennent les mêmes entités contenant des valeurs claires pour X.

10.5.6.2. Propriété (M7)

Si deux décompositions fonctionnelles D et D' sont concernées par f: X → A, elles sont compatibles pour f quand les mécanismes sont actifs.

Preuve:
Ce résultat est déjà démontré précédemment si D et D' ont la même source. Il s'agit maintenant de considérer des décompositions de sources

différentes S pour D, S' pour D'.

Nous considérons la décomposition DG précédente de source S et nous reprenons les mêmes notations. Comme KS' → X est vérifiée, et que X → $KR_i (1 \leq i \leq n)$, on a aussi, KS' → KR_i, ainsi S' > R_i $(1 \leq i \leq n)$.
Il existe donc une décomposition DG' de source S' construite à partir des relations R_i.

Soit s une entité de iS, s* l'entité s étendue à DG; soit s' une entité de iS', s'* l'entité s' étendue à DG'; nous supposons que s*.X = s'*.X en clair. Par le même raisonnement que précédemment (§ 10.5.6.1.b.), on va obtenir de proche en proche que s*.A=s'*.A.
Ainsi iDG et iDG' sont compatibles pour f.
Comme d'après ce qui précède, d'une part iDG[XA] et iD[XA] et d'autre part iDG'[XA] et iD'[XA] contiennent les mêmes entités prenant des valeurs claires pour X, iD et iD' sont compatibles pour f.

10.5.6.3 Théorème (M8)

Ainsi les mécanismes de clés, le mécanisme complémentaire d'associations, le mécanisme de clés cycliques, le mécanisme de cycle, le mécanisme de clés partielles garantissent la validité des df pouvant être générées à partir de l'ensemble des df intrinsèques. Ils garantissent l'étanchéité des décompositions fonctionnelles par rapport à ces df.

La preuve de ce théorème découle du théorème précédent pour les cas des df intrinsèques et des propriétés précédentes pour les autres df.

BIBLIOGRAPHIE

ABITEBOUL85 ABITEBOUL S., *Cocktail de dépendances,* Thèse, Paris 11, 1985.

ABITEBOUL-VIANU85 ABITEBOUL S.; VIANU V., *Transaction and Constraints,* Conf. PODS, 1985.

ABRIAL74 ABRIAL J.R., *Data semantics,* Conf. IFIP in Data Management Systems, North Holland, 1974.

ADIBA-DELOBEL-LÉONARD76 ADIBA M.; DELOBEL C.; LÉONARD M., *An Unified Approach for Modelling Data in Logical Data Base Design.,* IFIP Workshop TC2, Freudenstadt, Jan 1976.

AHO-BEERI-ULLMAN79 AHO A.V.; BEERI C.; ULLMANN J.D., *The Theory of Joins in Relational Databases,* ACM TODS, 4, 3, Sept 1979.

ALCOREZA88 ALCOREZA J., *Mécanismes de cohérences dans les cycles fonctionnels* Mémoire de licence d'Informatique de Gestion, U. de Genève, Fév 1988.

ARMSTRONG74 ARMSTRONG W.W., *Dependency Structures in Relational Databases.,* Conf. IFIP, 1974.

BEERI-BERNSTEIN79 BEERI C.; BERNSTEIN P., *Computational Problems Related to the Design of Normal Form Relational Schemas,* ACM TODS, 4, 1, Mars 1979.

Bibliographie

BEERI-FAGIN-HOWARD77 BEERI C.; FAGIN R.; HOWARD J.H., *A Complete Axiomatization for Functional and Multivalued Dependencies in Database Relations,* ACM SIGMOD, 1977.

BEERI-RISSANEN80 BEERI C.; RISSANENS J., *Faithful Representations of Relational Database Schemes,* IBM RJ 2722 (34837), 1980.

BENCI-ROLLAND79 BENCI G.; ROLLAND C., *Bases de données: conception canonique pour une réalisation extensible,* Editions SCM, Paris, 1979.

BERGE70 BERGE C.,*Graphes et Hypergraphes,* Dunod, 1970.

BERNSTEIN76 BERNSTEIN P.A., *Synthesizing Third Normal Form relations from Functional Dependencies,* ACM TODS, 1, 4, Déc 1976.

BLAUSTEIN81 BLAUSTEIN B.T., *Enforcing Database assertions: Techniques and Applications,* Thèse PHD, Harvard, 1981.

BODART-PIGNEUR83 BODART F.; PIGNEUR Y., *Conception assistée des applications informatiques: 1. étude d'opportunité et analyse conceptuelle,* Masson, 1983.

BOUILLE78 BOUILLE F., *A Survey of the Hypergraphed Based Data Structure Application to Cartography and Mapping,* International User's Conference on Computer Mapping Software and Data Bases,Cambridge, Mass., 1978.

BRIAND-COCHET77 BRIAND H.; COCHET C., *Analyse fonctionnelle en Informatique de gestion,* Dunod, Paris, 1977.

BROSDA-VOSSEN85 BROSDA V.; VOSSEN G., *Updating a Relational Database Through a Universal Relational Interface,* Report 101, Technical University of Aachen, Schriften zur Informatik und Angewandte Mathematik, Août 1984.

BRY-MANTHEY86 BRY F.; MANTHEY R. *Checking Consistency of Database Constraints: A Logical Basis,* Conf. VLDB, 1986.

BULA78 BULA E., *Méthode d'approche pour la conception d'une banque de données,* Thèse, U. de Neuchâtel, 1978.

CASANOVA82 CASANOVA M.A., *A Theory of Data Dependencies Over Relational Expressions,* Conf. PODS, 1982.

CASANOVA-FAGIN-PAPADIMITRIOU82 CASANOVA M.A.; FAGIN R.; PAPADIMITRIOU C.H., *Inclusion Dependencies and their Interaction with Functional Dependencies,* Conf. PODS, 1982.

CAVARÉRO-HÉRIN-AIME82 CAVARÉRO J.L.; HÉRIN-AIME D., *La conception des systèmes d'information. Un modèle, un dossier standard, des méthodes.,* Masson, 1982.

CHEN76 CHEN P.P.S., *The Entity Relationship Model: Towards a Unified View of Data,* ACM TODS, 1, 1, Mars 1976.

CODD70 CODD E.F., *A Relational Model for Large Shared Data Banks,* CACM, 13, 6, Juin 1970.

CODD79 CODD E.F., *Extending the Relational Database Model to Capture More Meaning,* ACM TODS, 4, 4, Décembre 1979.

COSMADAKIS-KANELLAKIS84
COSMADAKIS S.S.; KANELLAKIS P.C., *Functional and Inclusion Dependencies: A Graph Theoretic Approach,* Conf. PODS, 1984.

DATE75 DATE C.J., *Introduction to Database Systems,* Addison-Wesley Publ Co, 1975.

DELOBEL73 DELOBEL C, *Contributions théoriques à la conception et l'évaluation d'un système d'informations appliqué à la gestion,* Thèse, U. de Grenoble, Oct 1973.

DELOBEL78 DELOBEL C., *Normalization and Hierarchical Dependencies in the Relational Data Model,* ACM TODS, 3, 3, Sept 1978.

DELOBEL-ADIBA82 DELOBEL C.; ADIBA M., *Bases de données et systèmes relationnels,* Dunod Informatique, 1982.

DELOBEL-CASEY72 DELOBEL C.; CASEY R.G., *Decomposition of a Database and the Theory of Boolean Switching Functions,* IBM Journal of Research and Development, Sept 1972.

Bibliographie

DELOBEL-LÉONARD74 DELOBEL C.; LÉONARD M., *The Decomposition Process in a Relational Model,* IRIA-Workshop on Data Structures, Namur, Mai 1974.

DEMOLOMBE-YAZDANIAN-NICOLAS85 DEMOLOMBE R.; YAZDANIAN K.; NICOLAS J.M., *Modèle complet, modèle irredondant pour un schéma de base de données relationnelle,* Journées Base de Données, Grenoble, 1985.

FADOUS-FORSYTH75 FADOUS R.; FORSYTH J., *Finding Candidate Keys for Relational Data Bases,* ACM SIGMOD, 1975.

FAGIN77 FAGIN R., *Multivalued Dependencies and a New Normal Form of Relational Databases,* ACM TODS, 2, 3, Sept 1977.

FAGIN82 FAGIN R., *Horn Clauses and Database Dependencies,* JACM, 29, 4, Octobre 1982.

FINKELSTEIN-SCHKOLNICK-TIBERIO88 FINKELSTEIN S.; SCHKOLNICK M.; TIBERIO P., *Physical database design for relational databases,* ACM TODS, 13, 1, Mars 1988.

FLORY82 FLORY A., *Base de données: Conception et Réalisation,* Economica, Paris, 1982.

GALACSI86 GALACSI: BRIAND H.; CRAMPES I.B.; DUCATEAU C.; HÉBRAIL Y.; HÉRIN-AIME D.; KOULOUMDJAN J.; SABATIER R, *Les systèmes d'information: analyse et conception,* Dunod, 1986.

GALLAIRE81 GALLAIRE H., *Impacts of Logic on Databases,* Conf. VLDB, 1981.

GALLAIRE-MINKER-NICOLAS84 GALLAIRE H.; MINKER J.; NICOLAS J.M., *Logic and Databases: A Deductive Approach,* ACM Computing Surveys, 16, 2, Juin 1984.

GARDARIN84 GARDARIN G., *Bases de données: les systèmes et leurs langages,* Eyrolles, 1984.

GAROCHE-LÉONARD78 GAROCHE-REYNAUD F.; LÉONARD M. *Basepirr: algorithmes d'obtention de monômes premiers, des couvertures et bases irredondantes des fonctions booléennes*

particulières, Journées des modèles relationnels, Institut de Programmation de Paris, Mars 1978.

GAROCHE-LÉONARD84 GAROCHE-REYNAUD F.; LÉONARD M. *On a Class of Boolean Functions with Matroid Properties,* Communication in Discrete Mathematics 49, North Holland, 1984.

GUYOT86 GUYOT, J., *Un modèle de traitement pour les bases de données,* Thèse, U. de Genève, Ed. Le Concept Moderne, Genève, 1986.

HAINAUT86 HAINAUT J.L., *Conception assistée des applications informatiques:*
2. conception de la base des données, Masson, Paris, 1986.

HUONG87 HUONG Ph., *Dépendances d'inclusion et bases de données,* Rapport interne de l'U. de Genève, Juin 1987.

JUNET86 JUNET M., *Design and Implementation of an Extended Entity-Relationship Database Management System. (ÉCRINS/86),* Conf. Entity-Relationship Model, Dijon, 1986.

JUNET-FALQUET-LÉONARD86 JUNET M.; FALQUET G.; LÉONARD M., *ÉCRINS/86: An Extended Entity-Relationship Database Management System and its Semantic Query Language,* Conf. VLDB, 1986.

KNUTH73 KNUTH D., *The Art of Computer Programming,* 3, Addison Wesley Publ Co, 1973.

KUCK-SAGIV83 KUCK S.M.; SAGIV Y., *Designing globally consistent network schemas,* ACM-SIGMOD Conference, San Jose, Calif.,Mai 1983.

KUNDU75 KUNDU S, *An Improved Algorithm for Finding a Key of a Relation.,* Conf. PODS, 1975.

KUNTZMANN72 KUNTZMANN J., *Théorie des réseaux-graphes,* Dunod; 1972.

LAFAYE82 LAFAYE M.C., *Outils d'aides à la conception des bases de données relationnelles ou réseau (multigraphe de projection),* Thèse, U. de Rennes, Déc 1982.

Bibliographie

LÉONARD83 LÉONARD M., *Observation des dépendances fonctionnelles non réduites à leurs propriétés booléennes,* Séminaire INFORSID, Campo Dell'Oro, 1983.

LÉONARD-GALLAND-JUNET-TSCHOPP85 LÉONARD M.; GALLAND A.; JUNET M.; TSCHOPP R., *ÉCRINS: un modèle relationnel étendu et un système de gestion de petites bases de données,* Convention Informatique Latine, Barcelone, Avril 1985.

LÉONARD86 LÉONARD M.; SNELLA J.J.; ABDELJAOUED A., *FARANDOLE: A RDBMS for Statistical Data Analysis,* 7th Symposium on Computational Statistiques, Conf. COMPSTAT, Rome (Italie), 1986.

LUCAS81 LUCAS A., *Metodologias de concepçao de bases de dados,* Thèse, Lab. Nacional de Engenharia Civil, Lisboa, Nov. 1981.

LUCCHESI-ORSBORN76 LUCCHESI C.L.; ORSBORN S.L., *Candidate keys for relations,* Technical report, U. of Waterloo, Ontario, Canada, 1976.

LUONG86 LUONG DONG THI, B.T., *Une approche de conception d'une base de données cohérentes et complètes,* Thèse, U. de Genève, Ed. Le Concept Moderne, Genève, 1986.

MAIER83 MAIER D., *The Theory of Relational Databases,* Computer Science Press, 1983.

MAIER-MENDELZON-SAGIV79 MAIER D.; MENDELZON A.; SAGIV Y., *Testing Implications of Data Dependencies,* ACM TODS, 4, 4, Déc 1979.

MAIER-ULLMAN83 MAIER D.; ULLMAN J.D., *Maximal Objects and the Semantics of Universal Relation Databases,* ACM TODS, 8, 1, Janvier 1983.

MAIER-ULLMAN-VARDI84 MAIER D.; ULLMAN J.D.; VARDI M.Y., *On the Foundations of the Universal Relation Model,* ACM TODS, 9, 2, Juin 1984.

MELKANOFF-ZANIOLO80 MELKANOFF M.-A.; ZANIOLO C., *Décomposition of Relations and Synthesis of Entity Relationship*

Diagrams, In Entity-Relationship Approach to System Analysis and Design, North-Holland, 1980.

MENDELZON79 MENDELZON A.O., *On Axiomatizing Multivalued Dependencies in Relational Databases,* JACM, 26, 1, Janvier 1979.

MIRANDA-BUSTA84 MIRANDA S.M.; BUSTA J.M., *L'art des bases de données: 1. Introduction aux bases de données,* Eyrolles, 1984.

MITCHELL83 MITCHELL J.C., *Inference Rules for Functional and Inclusion Dependencies,* Conf. PODS, 1983.

NICOLAS78 NICOLAS J.M., *Mutual Dependencies and Some Results on Undecomposable Relations,* Conf. VLDB, 1978.

PARADAENS-JANSSENS81 PARADAENS J.; JANSSENS D., *Decomposition of Relations: A Comprehensive Approach,* in "Advances in Database Theory", 1, Plenum Press, New York, 1981.

PÉPIN85 PÉPIN, *Introduction aux systèmes de gestion de bases de données,* Eyrolles, 1985.

PICHAT-DELOBEL79 PICHAT E.; DELOBEL C., *Designing Third Normal Form for Relational Data Base Schema,* Rapport de recherche, IMAG 149, U. de Grenoble, 1979.

PORTAL75 PORTAL D., *Conception d'un système automatisé de gestion de la scolarité de l'enseignement supérieur,* Thèse, U. de Grenoble, Mars 1975.

REITER84 REITER R., *Towards a Logical Reconstruction of Relational Database Theory,* in "On Conceptual Modelling", Springer Verlag, 1984.

REYNAUD75 REYNAUD F.,*Théorie des écoulements dans les graphes orientés sans circuit,* Discrete Mathematics 12, North Holland Publ Co, 1975.

RIEU84 RIEU D., *Modèle et fonctionnalités d'un SGBD pour les applications CAO,* Thèse, U. de Grenoble, Juillet 1984.

RISSANEN77 RISSANEN J., *Independent Components of Relations.,* ACM TODS, 2, 4, Déc 1977.

Bibliographie

ROLLAND78 ROLLAND C., *Concepts for Information System Conceptual Schema and its Utilization in the REMORA project,* Conf. VLDB, 1978.

SAGIV83 SAGIV Y., *A Characterization of Globally Consistent Databases and their Correct Access Paths,* ACM TODS, 8, 2, Juin 1983.

SAGIV-DELOBEL-PARKER-FAGIN81 SAGIV Y.; DELOBEL C.; PARKER D.S.jr; FAGIN R., *An Equivalence Between Relational Database Dependencies and a Subclass of Propositional Logic,* JACM, 28, 3, Juillet 1981.

SCIORE82 SCIORE E., *Complete Axiomatization of Full Join Dependencies,* JACM, 29, 2, Avril 1982.

SCIORE83 SCIORE E., *Inclusion Dependencies and the Universal Instance,* Conf. PODS, 1983.

SMITH77 SMITH J.M.; SMITH D.C.P., *Database Abstractions: Aggregation and Generalization,* ACM TODS, 2, 2, Juin 1977.

SNELLA84 SNELLA J.J.; LÉONARD M., *Pirée: un système de gestion de bases de données économiques,* Conf. INFORSID, Bandol, 1984.

TARDIEU-ROCHFELD-COLLETTI83 TARDIEU H.; ROCHFIELD A.; COLLETTI R., *La méthode MERISE: Principe et outils,* Les Editions d'organisation, 1983.

ULLMAN80 ULLMAN J.D., *Principles of Database Systems,* Computer Science Press, 1980.

VARDI82 VARDI M.Y., *The Implication and Finite Implication Problems for Types Template Dependencies,* Conf. PODS, 1982.

VELEZ-LOPEZ87 VELEZ F.; LOPEZ M., *Projet TIGRE: bilan et enseignement,* Troisième Journées des Bases de Données Avancées, Port-Camargue, Mai 1987.

VETTER-MADDISON81 VETTER M.; MADDISON R., *Database Design Methodology,* Prentice Hall, 1981.

WASSERMAN83 WASSERMAN T., *The Unified Support Environnement: Tool Support for the User Software Engineering Methodology*, In CRIS 2.IFIP Conf., York (U.K.), Juillet 1983.

ZANIOLO76 ZANIOLO C., *Analysis and Design of Relational Schemata for Data Base Systems*, PhD UCLA, 1976.

ZANIOLO79 ZANIOLO C., *Design of Relational Views over Network Schemas*, ACM SIGMOD, Juin 1979.

ZANIOLO-MELKANOFF81 ZANIOLO C.; MELKANOFF M., *On the Design of Relational Database Schemata*, ACM TODS, 6, 1, Mars 1981.

Bibliographie

INDEX

AUTORISATION DE SOUTENANCE

DOCTORAT D'ETAT

Vu les dispositions de l'Article 5 de l'Arrêté du 16 avril 1974,

Vu les rapports de M.M. HAINAUT - BODARD

M. KOUCOUMDJIAN

M. JEOBEL

M. LEONARD Michel est autorisé à présenter une thèse en vue de l'obtention du grade de DOCTEUR D'ETAT ES SCIENCES.

Fait à Grenoble, le

Le Président de l'U.S.T.M.G.

J.J PAYAN