LE MODÈLE RELATIONNEL

1. INTRODUCTION : LES OBJECTIFS DU MODÈLE

Le modèle relationnel a été introduit par E. F. Codd [Codd70], qui travaillait au fameux centre de recherche d'IBM San-José et s'opposait au développement du modèle DIAM, un modèle utilisant des entités liées par de multiples pointeurs. La première volonté du modèle relationnel fut d'être un modèle ensembliste simple, supportant des ensembles d'enregistrements aussi bien au niveau de la description que de la manipulation. Les premières idées d'un modèle ensembliste avaient été proposées un peu avant, notamment dans [Childs68]. Le modèle relationnel est aujourd'hui la base de nombreux systèmes, et les architectures permettant d'accéder depuis une station de travail à des serveurs de données s'appuient en général sur lui. Le relationnel a donc atteint ses objectifs au-delà de toute espérance.

Les premiers objectifs du modèle ont été formulés par E. F. Codd [Codd70] comme suit :

1. Permettre un haut degré d'indépendance des programmes d'applications et des activités interactives à la représentation interne des données, en particulier aux choix des ordres d'implantation des données dans les fichiers, des index et plus généralement des chemins d'accès.

2. Fournir une base solide pour traiter les problèmes de cohérence et de redondance des données.

Ces deux objectifs, qui n'étaient pas atteints par les modèles réseau et hiérarchique, ont été pleinement satisfaits par le modèle relationnel, d'une part grâce à la simplicité des vues relationnelles qui permettent de percevoir les données sous forme de tables à deux dimensions, et d'autre part grâce aux règles d'intégrité supportées par le modèle et ses fondements logiques.

En addition aux objectifs fixés par E. F. Codd, le modèle relationnel a atteint un troisième objectif :

PDF-XChange Click to buy NOW! www.docu-track.com

3. Permettre le développement de langages de manipulation de données non procéduraux basés sur des théories solides.

Ce troisième objectif est atteint d'une part à l'aide de l'algèbre relationnelle qui permet de manipuler des données de manière très simple et formelle — comme les opérateurs arithmétiques permettent de manipuler des entiers —, et d'autre part à l'aide des langages assertionnels basés sur la logique qui permettent de spécifier les données que l'on souhaite obtenir sans dire comment les obtenir.

Finalement, le modèle relationnel a atteint deux autres objectifs non prévus initialement :

4. Etre un modèle extensible permettant de modéliser et de manipuler simplement des données tabulaires, mais pouvant être étendu pour modéliser et manipuler des données complexes.

5. Devenir un standard pour la description et la manipulation des bases de données.

L'objectif 4 est très important, car il a permis d'intégrer de nouveaux concepts au modèle relationnel, notamment la plupart des concepts de l'orienté objets que nous étudierons dans la troisième partie de cet ouvrage. Les premiers travaux de recherche dans ce sens ont été effectués par Codd lui-même [Codd79], puis poursuivis à Bell Laboratories [Zaniolo83], à Berkeley [Stonebraker87] et, en France, à l'INRIA [Gardarin89].

L'objectif 5 a été réalisé en particulier grâce à IBM : le modèle relationnel et son langage SQL ont été normalisés au niveau international en 1986 [ISO89]. Nous ferons un point sur le langage SQL et sa standardisation dans le chapitre suivant.

Dans ce chapitre, nous allons tout d'abord présenter les concepts structuraux de base du modèle relationnel qui permettent de modéliser les données sous forme de tables à deux dimensions. Nous exposerons ensuite les règles de cohérence des relations qui sont considérés comme partie intégrante du modèle. Puis nous introduirons l'algèbre relationnelle, qui est l'outil formel indispensable pour manipuler des relations. Les nombreuses extensions de cette algèbre seront également présentées. En conclusion, nous démontrerons les vastes possibilités d'enrichissement offertes par le modèle relationnel, possibilités qui seront étudiées plus en détail au niveau des modèles objets et logiques, sujets d'autres parties de ce livre.

2. LES STRUCTURES DE DONNÉES DE BASE

2.1 Domaine, attribut et relation

Le modèle relationnel est fondé sur la théorie mathématique bien connue des relations. Cette théorie se construit à partir de la théorie des ensembles. Trois notions sont importantes pour introduire les bases de données relationnelles. La première permet de définir les ensembles de départ. Ces ensembles sont les **domaines** de valeurs.

> **Notion VI.1 : Domaine (*Domain*)**
>
> Ensemble de valeurs caractérisé par un nom.

Les domaines sont donc les ensembles dans lesquels les données prennent valeur. Comme un ensemble, un domaine peut être défini en extension, en donnant la liste des valeurs

PDF-XChange Click to buy NOW! www.docu-track.com

composantes, ou en intention, en définissant une propriété caractéristique des valeurs du domaine. Au départ, les domaines `ENTIER, REEL, BOOLEEN, CARACTERES` (une chaîne de caractères de longueur fixe ou variable) sont définis en intention. A partir de ces domaines, il est possible de définir en intention des domaines plus spécifiques tels que `MONNAIE` (réel avec 2 chiffres derrière la virgule), `DATE` (entier de 6 chiffres jour, mois et an), `TEMPS` (heure en secondes), etc. Un domaine peut toujours être défini en extension, par exemple le domaine des couleurs de vins : `COULEUR-VINS = {Rosé, Blanc, Rouge}`.

Rappelons que le produit cartésien d'un ensemble de domaines `D1, D2,..., Dn` est l'ensemble des vecteurs <`v1, v2, ..., vn`> où, pour i variant de 1 à n, `vi` est une valeur de Di. Par exemple, le produit cartésien des domaines `COULEUR-VINS = {ROSE, BLANC, ROUGE}` et `CRUS = {VOLNAY, SANCERRE, CHABLIS}` est composé des neuf vecteurs représentés figure VI.1.

COULEUR-VINS = {ROSE, BLANC, ROUGE}

CRUS = {VOLNAY, SANCERRE, CHABLIS}

ROSE	VOLNAY
ROSE	SANCERRE
ROSE	CHABLIS
BLANC	VOLNAY
BLANC	SANCERRE
BLANC	CHABLIS
ROUGE	VOLNAY
ROUGE	SANCERRE
ROUGE	CHABLIS

Figure VI.1 — Exemple de produit cartésien

Nous pouvons maintenant introduire la notion de **relation**, à la base du modèle.

> **Notion VI.2 : Relation (*Relation*)**
>
> Sous-ensemble du produit cartésien d'une liste de domaines caractérisé par un nom.

Sous-ensemble d'un produit cartésien, une relation est composée de vecteurs. Une représentation commode d'une relation sous forme de table à deux dimensions a été retenue par Codd. Elle est généralement utilisée. Chaque ligne correspond à un vecteur alors que chaque colonne correspond à un domaine du produit cartésien considéré ; un même domaine peut bien sûr apparaître plusieurs fois. Par exemple, à partir des domaines `COULEURS_VINS` et `CRUS`, il est possible de composer la relation `COULEURS_CRUS` représentée sous forme tabulaire figure VI.2.

COULEURS_CRUS	Couleur	Cru
	ROSE	SANCERRE
	ROSE	CHABLIS
	BLANC	SANCERRE
	ROUGE	VOLNAY
	ROUGE	SANCERRE
	ROUGE	CHABLIS

Figure VI.2 — Un premier exemple de relation

Pour pouvoir distinguer les colonnes d'une relation sans utiliser un index, et ainsi rendre leur ordre indifférent tout en permettant plusieurs colonnes de même domaine, il est nécessaire d'associer un nom à chaque colonne. Une colonne se distingue d'un domaine en ce qu'elle prend valeur dans un domaine et peut à un instant donné comporter seulement certaines valeurs du domaine. Par exemple, si le domaine est l'ensemble des entiers, seules quelques valeurs seront prises à un instant donné, par exemple {10, 20, 30}. L'ensemble des valeurs d'une colonne de relation est en général fini. Afin de bien distinguer domaine et colonne, on introduit la notion **d'attribut** comme suit :

> **Notion VI.3 : Attribut (*attribute*)**
>
> Colonne d'une relation caractérisée par un nom.

Le nom associé à un attribut est souvent porteur de sens. Il est donc en général différent de celui du domaine qui supporte l'attribut. Ainsi, la première colonne de la relation `COULEUR-CRUS` pourra être appelée simplement `COULEUR` et la deuxième `CRU`. COULEUR et `CRU` seront donc deux attributs de la relation `COULEUR-CRUS`.

Les lignes d'une relation correspondent à des n-uplets de valeurs. Une ligne est aussi appelée **tuple** (du mot anglais *tuple*) : un tuple correspond en fait à un enregistrement dans une relation (encore appelée table).

> **Notion VI.4 : Tuple (*tuple*)**
>
> Ligne d'une relation correspondant à un enregistrement.

La notion de relation est bien connue en mathématiques : une relation n-aire est un ensemble de n-uplets. Un cas particulier souvent employé est la relation binaire, qui est un ensemble de paires ordonnées. Une relation n-aire est souvent représentée par un graphe entre les ensembles composants. Ainsi, la relation `LOCALISATION`, donnant la région de chaque cru et certains prix moyens des vins associés, est représentée par un graphe figure VI.3.

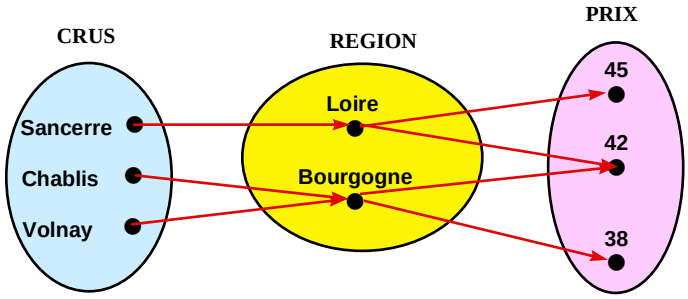


Figure VI.3 — Graphe de la relation `LOCALISATION`

Une relation peut aussi être représentée sur un diagramme à n dimensions dont les coordonnées correspondent aux domaines participants. Ainsi, la relation `STATISTIQUE`, d'attributs `AGE` et `SALAIRE`, donnant la moyenne des salaires par âge, est représentée figure VI.4.

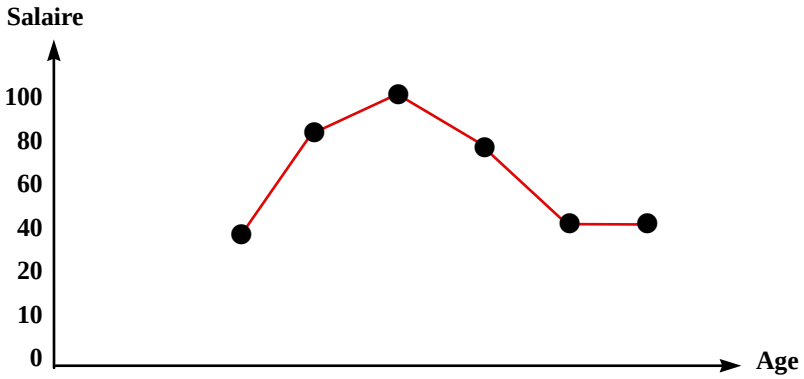


Figure VI.4 — Diagramme représentant une relation binaire

Une autre perception mathématique d'une relation consiste à voir chaque tuple t comme une fonction `t : Ai -> Di` pour `i = 1, 2 ... n` qui applique donc les attributs sur les domaines. Ainsi, une relation peut être vue comme un ensemble fini de fonctions. Bien sûr, cet ensemble varie en fonction du temps. Tout ceci montre la diversité des outils mathématiques qui peuvent être appliqués aux relations. En conséquence, le modèle relationnel peut apparaître comme un modèle mathématique simple des données. De plus, grâce à la représentation tabulaire, il est simple à appréhender pour les non mathématiciens.

2.2 Extensions et intentions

Comme tous les modèles de données, le modèle relationnel permet de décrire des données dont les valeurs varient en fonction du temps. Les relations varient en fonction du temps en ce sens que des tuples sont ajoutés, supprimés et modifiés dans une relation au cours de sa vie. Cependant, la structure d'une relation caractérisée par les trois concepts de domaine, relation et attribut est un invariant pour la relation (elle ne change pas en fonction du temps). Cette structure est capturée dans le **schéma de la relation**.

> **Notion VI.5 : Schéma de relation (*Relation Schema*)**
>
> Nom de la relation suivi de la liste des attributs et de la définition de leurs domaines.

Un schéma de relation est noté sous la forme :

```
R (A1 : D1, A2 : D2, ..., Ai : Di, ... An : Dn)
```

R est le nom de la relation, Ai les attributs et Di les domaines associés. A titre d'exemple, la figure VI.5 propose deux schémas de relations : celui de la relation LOCALISATION introduite ci-dessus et celui de la relation VINS qui représente des vins caractérisés par un numéro, un cru, un millésime et un degré. Les domaines utilisés sont ceux des entiers, des réels et des chaînes de caractères de longueur variable (CHARVAR). La plupart du temps, lorsqu'on définit le schéma d'une relation, les domaines sont implicites et découlent du nom de l'attribut ; aussi omet-on de les préciser.

```
LOCALISATION ( CRU : CHARVAR, REGION : CHARVAR, PRIX : FLOAT)
VINS (NV :INTEGER, CRU : CHARVAR, MILL :INTEGER, DEGRE : FLOAT)
```

Figure VI.5 — Schémas des relations LOCALISATION et VINS

Soulignons que le schéma d'une relation représente son **intention**, c'est-à-dire les propriétés (au moins certaines) communes et invariantes des tuples qu'elle va contenir au cours du temps. Au contraire, une table représente une **extension** d'une relation (voir figure VI.2 par exemple), c'est-à-dire une vue des tuples qu'elle contient à un instant donné. Une extension d'une relation R est aussi appelée **instance** de R. L'intention est le résultat de la description des données, alors qu'une extension (ou instance) fait suite à des manipulations et représente un état de la base.

3. LES RÈGLES D'INTÉGRITÉ STRUCTURELLE

Les règles d'intégrité sont les assertions qui doivent être vérifiées par les données contenues dans une base. Il est possible de distinguer les règles structurelles qui sont inhérentes au modèle de données, c'est-à-dire nécessaires à sa mise en œuvre, et les règles de comportement propres au schéma particulier d'une application. Le modèle relationnel impose a priori une règle minimale qui est l'unicité des clés, comme nous allons le voir ci-dessous. Il est commode et courant [Date81] d'ajouter trois types de règles d'intégrité supplémentaires afin d'obtenir les règles d'intégrité structurelle supportées par le modèle relationnel : les contraintes de références, les contraintes d'entité et les contraintes de domaine.

3.1 Unicité de clé

Par définition, une relation est un ensemble de tuples. Un ensemble n'ayant pas d'élément en double, il ne peut exister deux fois le même tuple dans une relation. Afin d'identifier les tuples d'une relation sans donner toutes les valeurs et d'assurer simplement l'unicité des tuples, la notion de **clé** est utilisée.

> **Notion VI.6 : Clé (*Key*)**
>
> Ensemble minimal d'attributs dont la connaissance des valeurs permet d'identifier un tuple unique de la relation considérée.

De manière plus formelle, une clé d'une relation R est un ensemble d'attributs K tel que, quels que soient les tuples t1 et t2 d'une instance de R, $t1(K) \neq t2(K)$, c'est-à-dire que t1 et t2 ont des valeurs de K différentes. Un ensemble d'attributs contenant une clé est appelée **super-clé** [Ullman88].

Toute relation possède au moins une clé car la connaissance de tous les attributs permet d'identifier un tuple unique. S'il existe plusieurs clés, on en choisit en général une arbitrairement qui est appelée **clé primaire.** Par exemple, NV peut constituer une clé primaire pour la relation VINS. Le couple <CRU,MILLESIME> est une clé alternative.

Soulignons que la notion de clé caractérise l'intention d'une relation : dans toute extension possible d'une relation, il ne peut exister deux tuples ayant même valeur pour les attributs clés. La détermination d'une clé pour une relation nécessite donc une réflexion sur la sémantique de la relation, c'est-à-dire sur toutes les extensions possibles et non pas sur une extension particulière.

3.2 Contraintes de références

Le modèle relationnel est souvent utilisé pour représenter des entités du monde réel qui sont les objets ayant une existence propre, et des associations entre ces objets [Chen76]. Une entité correspond alors à un tuple dans une relation qui comporte à la fois la clé de l'entité et ses caractéristiques sous forme d'attributs. Une entité est identifiée par la valeur de sa clé. Un type d'association est modélisé par une relation comportant les clés des entités participantes ainsi que les caractéristiques propres à l'association.

A titre d'exemple, nous considérons les entités BUVEURS et VINS du monde réel : la clé de l'entité BUVEURS est le numéro de buveur NB et celles de l'entité VINS le numéro de vin NV. Ces entités peuvent être associées par une consommation d'un vin par un buveur : une consommation sera par exemple modélisée par une association ABUS entre le buveur et le vin. Cette base typique et simple, qui servira souvent d'exemple, est appelée DEGUSTATION. Le diagramme entité-association de cette base est représenté figure VI.6. En relationnel, chaque entité est représentée par une table. Une association est aussi représentée par une table, dont les attributs seront les clés des entités participantes, c'est-à-dire NB et NV, ainsi que les attributs caractérisant l'association, par exemple la date et la quantité bue. On aboutit ainsi au schéma relationnel représenté figure VI.7.

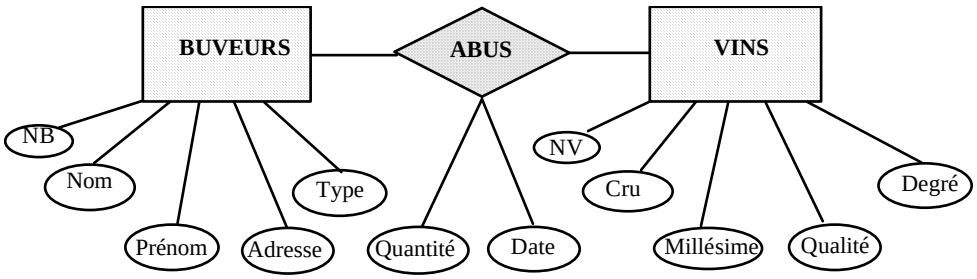


Figure VI.6 — Diagramme entité-association de la base DEGUSTATION

BUVEURS (NB, NOM, PRENOM, ADRESSE, TYPE)
VINS (NV, CRU, MILL QUALITE, DEGRE)
ABUS (NB, NV, DATE, QUANTITE)

Figure VI.7 — Schéma relationnel de la base DEGUSTATION

L'utilisation du modèle relationnel pour représenter des entités et des associations ne permet pas jusque-là de représenter le fait que l'association entre une consommation et un vin est

PDF-XChange Click to buy NOW! www.docu-track.com

obligatoire, c'est-à-dire que tout abus doit correspondre à un vin existant dans la base. De même, le fait que le lien entre une consommation et un buveur soit obligatoire est perdu. Le maintient de liens obligatoires a justifié l'introduction de la notion de **contrainte référentielle** [Date81].

> **Notion VI.7 : Contrainte référentielle (*Referential constraint*)**
>
> Contrainte d'intégrité portant sur une relation R1, consistant à imposer que la valeur d'un groupe d'attributs apparaisse comme valeur de clé dans une autre relation R2.

Une telle contrainte d'intégrité s'applique en général aux associations obligatoires : une telle association ne peut exister que si les entités participant à l'association existent déjà. Notez que dans la définition, `R1` et `R2` ne sont pas forcément distinctes : l'association peut en effet porter sur deux entités de même type, par exemple entre une personne et ses parents.

La représentation de contraintes de référence peut s'effectuer par la définition de **clés étrangères** dans une relation : une clé étrangère est un groupe d'attributs qui doit apparaître comme clé dans une (autre) relation. Par exemple, la relation `ABUS (NB, NV, DATE, QUANTITE)` a pour clé `<NB, NV, DATE>` et a deux clés étrangères, `NB` dans `BUVEURS` et `NV` dans `VINS`.

Les contraintes référentielles définissent des liens obligatoires entre relations. Ce sont des contraintes très fortes qui conditionnent le succès des opérations de mises à jour. Lors de l'insertion d'un tuple dans une relation référençante, il faut vérifier que les valeurs de clés étrangères existent dans les relations référencées. Par exemple, lors de l'insertion d'un abus, il faut que le vins et le buveurs associés existent, sinon l'insertion est refusée pour cause de violation d'intégrité. Lors de la suppression d'un tuple dans une relation référencée, il faut vérifier qu'aucun tuple n'existe dans chaque relation référençante ; s'il en existe, le système peut soit refuser la suppression, soit la répercuter en cascade (c'est-à-dire, supprimer les tuples référençants). Par exemple, la suppression d'un vin entraînera la vérification qu'aucun abus ne référence ce vin. Les contraintes d'intégrité référentielles sont donc des liens forts qui rendent les relations dépendantes ; en quelque sorte, elles introduisent des liens hiérarchiques depuis les relation référencées vers les relations référençantes.

3.3 Valeurs nulles et clés

Lors de l'insertion de tuples dans une relation, il arrive fréquemment qu'un attribut soit inconnu ou non applicable ; par exemple, la quantité de vin bu par un buveur à une certaine date peut être inconnue ; si un vin ne possède pas de millésime, l'attribut millésime est inapplicable à ce vin. On est alors amené à introduire dans la relation une valeur conventionnelle, appelée **valeur nulle**.

> **Notion VI.8 : Valeur nulle (*Null value*)**
>
> Valeur conventionnelle introduite dans une relation pour représenter une information inconnue ou inapplicable.

La signification précise d'une valeur nulle est souvent ambiguë ; le groupe ANSI/X3/SPARC a recensé quatorze significations possibles, parmi lesquelles valeur inconnue et valeur inapplicable sont les plus caractéristiques.

Tout attribut dans une relation ne peut prendre une valeur nulle ; en effet, l'existence d'une clé unique impose la connaissance de la clé afin de pouvoir vérifier que cette valeur de clé n'existe pas déjà. Ainsi, une contrainte structurelle du modèle relationnel est la **contrainte d'entité** définie ci-dessous [Date81].

> **Notion VI.9 : Contrainte d'entité (*Entity constraint*)**
>
> Contrainte d'intégrité imposant que toute relation possède une clé primaire et que tout attribut participant à cette clé primaire soit non nul.

A moins qu'il n'en soit spécifié autrement par une contrainte sémantique, le modèle relationnel n'impose pas que les clés étrangères qui n'appartiennent pas à une clé primaire soient non nulles. Cela peut permettre une certaine souplesse, par exemple d'enregistrer des employés qui ne sont attachés à aucun service.

3.4 Contraintes de domaines

En théorie, une relation est construite à partir d'un ensemble de domaines. En pratique, les domaines gérés par les systèmes sont souvent limités aux types de base entier, réel, chaîne de caractères, parfois monnaie et date. Afin de spécialiser un type de données pour composer un domaine plus fin (par exemple, le domaine des salaires mensuels qui sont des réels compris entre 6 000 et 1 000 000 de francs), la notion de **contrainte de domaine** est souvent ajoutée aux règles d'intégrité structurelle du relationnel. Cette notion peut être introduite comme suit :

> **Notion VI.10 : Contrainte de domaine (*Domain constraint*)**
>
> Contrainte d'intégrité imposant qu'une colonne d'une relation doit comporter des valeurs vérifiant une assertion logique.

L'assertion logique est l'appartenance à une plage de valeurs ou à une liste de valeurs. Par exemple, `SALAIRE ≥ 5 000 et ≤ 1 000 000, COULEUR ∈ {BLEU, BLANC, ROUGE}`, etc. Les contraintes permettent de contrôler la validité des valeurs introduites lors des insertions ou mises à jour. La non-nullité d'une colonne peut aussi être considérée comme une contrainte de domaine, par exemple `DEGRE IS NULL`.

4. L'ALGEBRE RELATIONNELLE : OPERATIONS DE BASE

L'algèbre relationnelle a été inventée par E. Codd comme une collection d'opérations formelles qui agissent sur des relations et produisent les relations en résultats [Codd70]. On peut considérer que l'algèbre relationnelle est aux relations ce qu'est l'arithmétique aux entiers. Cette algèbre, qui constitue un ensemble d'opérations élémentaires associées au modèle relationnel, est sans doute une des forces essentielles du modèle. Codd a initialement introduit huit opérations, dont certaines peuvent être composées à partir d'autres. Dans cette section, nous allons introduire six opérations qui permettent de déduire les autres et qui sont appelées ici opérations de base. Nous introduirons ensuite quelques opérations additionnelles qui sont parfois utilisées. Des auteurs ont proposé d'autres opérations qui peuvent toujours se déduire des opérations de base [Delobel83, Maier83].

Les opérations de base peuvent être classées en deux types : les opérations ensemblistes traditionnelles (une relation étant un ensemble de tuples, elle peut être traitée comme telle) et

les opérations spécifiques. Les opérations ensemblistes sont des opérations binaires, c'est-à-dire qu'à partir de deux relations elles en construisent une troisième. Ce sont l'union, la différence et le produit cartésien. Les opérations spécifiques sont les opérations unaires de projection et restriction qui, à partir d'une relation, en construisent une autre, et l'opération binaire de jointure. Nous allons définir toutes ces opérations plus précisément.

4.1 Les opérations ensemblistes

4.1.1 Union

L'**union** est l'opération classique de la théorie des ensembles adaptée aux relations de même schéma.

> **Notion VI.11 : Union (*Union*)**
>
> Opération portant sur deux relations de même schéma `RELATION1` et `RELATION2` consistant à construire une relation de même schéma `RELATION3` ayant pour tuples ceux appartenant à `RELATION1` ou `RELATION2` ou aux deux relations.

Plusieurs notations ont été introduites pour cette opération, selon les auteurs :

- `RELATION1 ∪ RELATION2`
- `UNION (RELATION1, RELATION2)`
- `APPEND (RELATION1, RELATION2)`

La notation graphique représentée figure VI.8 est aussi utilisée. A titre d'exemple, l'union des relations `VINS1` et `VINS2` est représentée figure VI.9.

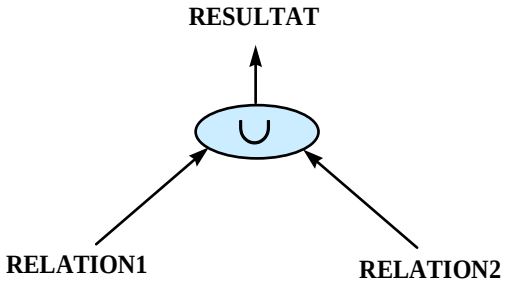


Figure VI.8 — Représentation graphique de l'union

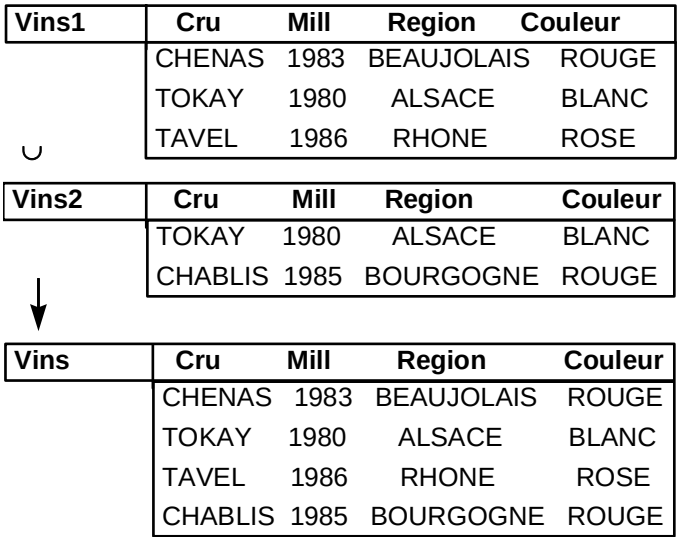


Figure VI.9 — Exemple d'union

4.1.2 Différence

La **différence** est également l'opération classique de la théorie des ensembles adaptée aux relations de même schéma.

> **Notion VI.12 : Différence (*Difference*)**
>
> Opération portant sur deux relations de même schéma `RELATION1` et `RELATION2`, consistant à construire une relation de même schéma `RELATION3` ayant pour tuples ceux appartenant à `RELATION1` et n'appartenant pas à `RELATION2`.

La différence est un opérateur non commutatif : l'ordre des relations opérandes est donc important. Plusieurs notations ont été introduites pour cette opération, selon les auteurs :

- `RELATION1 - RELATION2`
- `DIFFERENCE (RELATION1, RELATION2)`
- `REMOVE (RELATION1, RELATION2`
- `MINUS (RELATION1, RELATION2)`

La notation graphique représentée figure VI.10 est aussi utilisée. A titre d'exemple, la différence des relations `VINS1 - VINS2` est représentée figure VI.11.

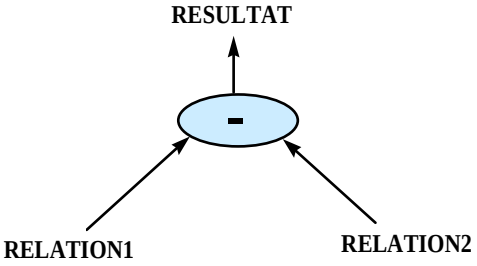


Figure VI.10 — Représentation graphique de la différence

Vins1	Cru	Mill	Region	Couleur
	CHENAS	1983	BEAUJOLAIS	ROUGE
	TOKAY	1980	ALSACE	BLANC
-	TAVEL	1986	RHONE	ROSE

Vins2	Cru	Mill	Region	Couleur
	TOKAY	1980	ALSACE	BLANC
	CHABLIS	1985	BOURGOGNE	ROUGE

↓

Vins	Cru	Mill	Region	Couleur
	CHENAS	1983	BEAUJOLAIS	ROUGE
	TAVEL	1986	RHONE	ROSE

Figure VI.11 — Exemple de différence

4.1.3 Produit cartésien

Le **produit cartésien** est l'opération ensembliste que nous avons rappelée ci-dessus pour définir le concept de relations. Elle est adaptée aux relations. Cette fois, les deux relations n'ont pas nécessité d'avoir même schéma.

> **Notion VI.13 : Produit cartésien (*Cartesian product*)**
>
> Opération portant sur deux relation `RELATION1` et `RELATION2`, consistant à construire une relation `RELATION3` ayant pour schéma la concaténation de ceux des relations opérandes et pour tuples toutes les combinaisons des tuples des relations opérandes.

Des notations possibles pour cette opération sont :

- `RELATION1 X RELATION2`
- `PRODUCT (RELATION1, RELATION2)`
- `TIMES (RELATION1, RELATION2)`

La notation graphique représentée figure VI.12 est aussi utilisée. A titre d'exemple, la relation VINS représentée figure VI.13 est le produit cartésien des deux relations `CRUS` et `ANNEES` de la même figure.

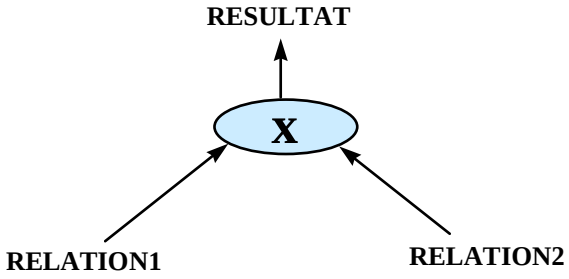


Figure VI.12 — Représentation graphique du produit cartésien

Crus	Cru	Region	Couleur
	CHENAS	BEAUJOLAIS	ROUGE
	TOKAY	ALSACE	BLANC
	TAVEL	RHONE	ROSE

X

Années	Mill
	1980
	1985

Vins	Cru	Region	Couleur	Mill
	CHENAS	BEAUJOLAIS	ROUGE	1980
	TOKAY	ALSACE	BLANC	1980
	TAVEL	RHONE	ROSE	1980
	CHENAS	BEAUJOLAIS	ROUGE	1985
	TOKAY	ALSACE	BLANC	1985
	TAVEL	RHONE	ROSE	1985

Figure VI.13 — Exemple de produit cartésien

4.2 Les opérations spécifiques

4.2.1 Projection

La **projection** est une opération spécifique aux relations qui permet de supprimer des attributs d'une relation. Son nom provient du fait qu'elle permet de passer d'une relation n-aire à une relation p-aire avec p<n, donc d'un espace à n dimensions à un espace à moins de dimensions.

> **Notion VI.14 : Projection (*Projection*)**
>
> Opération sur une relation `RELATION1` consistant à composer une relation `RELATION2` en enlevant à la relation initiale tous les attributs non mentionnés en opérandes (aussi bien au niveau du schéma que des tuples) et en éliminant les tuples en double qui sont conservés une seule fois.

Les notations suivantes sont utilisées pour cette opération, en désignant par `Attributi`, `Attributj`, `... Attributm` les attributs de projection :

- ∏`Attributi, Attributj, ... Attributm (RELATION1)`
- `RELATION1 [Attributi, Attributj, ... Attributm]`

- `PROJECT (RELATION1, Attributi, Attributj, ... Attributm)`

La notation graphique représentée figure VI.14 est aussi utilisée. Le trapèze horizontal signifie que l'on réduit le nombre de colonnes de la relation : partant du nombre représenté par la base, on passe au nombre représenté par l'anti-base. La figure VI.15 donne un exemple de projection d'une relation `VINS` comportant aussi l'attribut `QUALITE` sur les attributs `MILLESIME` et `QUALITE`.

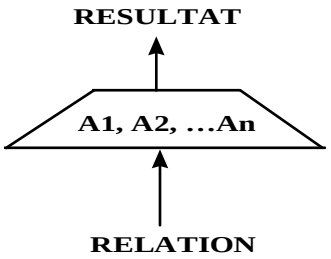


Figure VI.14 — Représentation graphique de la projection

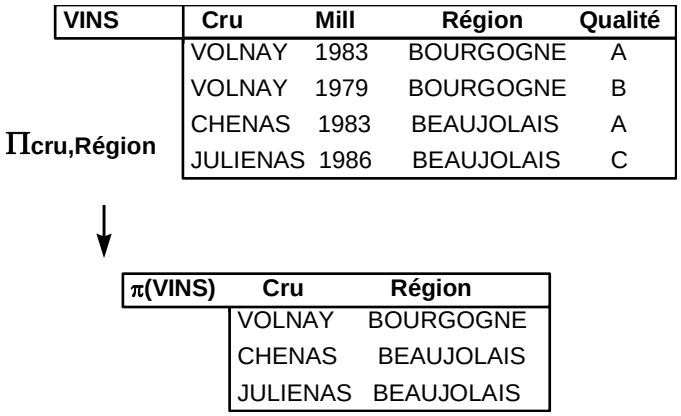


Figure VI.15 — Exemple de projection

4.2.2 Restriction

La **restriction** est aussi une opération spécifique unaire, qui produit une nouvelle relation en enlevant des tuples à la relation opérande selon un critère.

> **Notion VI.15 : Restriction (*Restriction*)**
>
> Opération sur une relation `RELATION1` produisant une relation `RELATION2` de même schéma, mais comportant les seuls tuples qui vérifient la condition précisée en argument.

Les conditions possibles sont du type :

```
<Attribut>  <Opérateur>  <Valeur>
```

où l'opérateur est un opérateur de comparaison choisi parmi {=,<, ≤ , ≥, >, ≠}. L'attribut doit appartenir à la relation sur laquelle s'applique le critère. Par exemple, pour la relation `VINS`, `CRU = "Chablis"` est une condition de restriction possible. `DEGRE > 12` est une autre condition possible. Il est aussi possible d'utiliser des compositions logiques de critères simples, c'est-à-dire des « et » et « ou » de conditions élémentaires. On pourra par exemple

utilisé le critère `CRU = "Chablis" et DEGRE > 12`, ou encore le critère `CRU = "Chablis" ou DEGRE = 12`. Toute composition de critères valides par conjonction et disjonction (des parenthèses peuvent être utilisées pour préciser les priorités) est valide. Notons que les compositions logiques peuvent aussi être obtenues par union et intersection de relations restreintes (voir ci-dessous).

Les notations suivantes sont utilisées pour la restriction :

- $\sigma_{condition}$ (RELATION1)
- RELATION1 [Condition]
- RESTRICT (RELATION1, Condition)

ainsi que la notation graphique représentée figure VI.16. Le trapèze vertical signifie que l'on réduit le nombre de tuples de la relation : partant du nombre représenté par le côté gauche, on passe au nombre représenté par le côté droit. La figure VI.17 représente la restriction d'une relation `VINS` enrichie d'un attribut `QUALITE` par la condition `QUALITE = "BONNE"`.

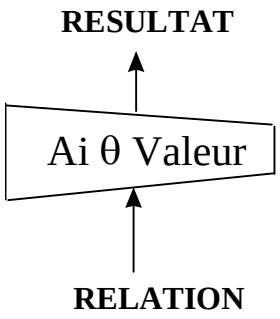


Figure VI.16 — Représentation graphique de la restriction

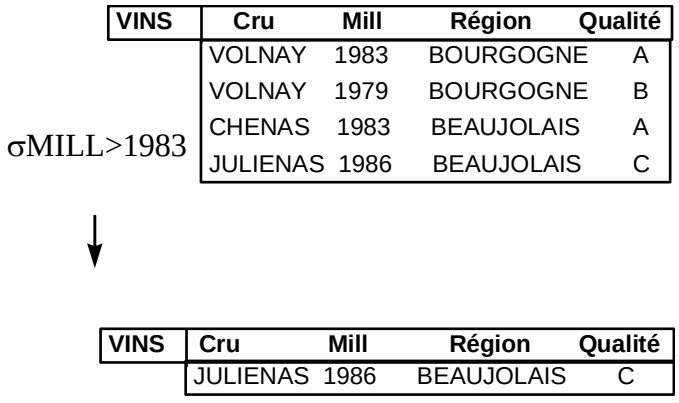


Figure VI.17 — Exemple de restriction

4.2.3 Jointure

La **jointure** est une des opérations essentielles de l'algèbre relationnelle, sans doute la plus difficile à réaliser dans les systèmes. La jointure permet de composer deux relations à l'aide d'un critère de jointure. Elle peut être vue comme une extension du produit cartésien avec une condition permettant de comparer des attributs. Nous la définirons comme suit :

Notion VI.16 : Jointure (*Join*)

Opération consistant à rapprocher selon une condition les tuples de deux relations `RELATION1` et `RELATION2` afin de former une troisième relation `RELATION3` qui contient l'ensemble de tous les tuples obtenus en concaténant un tuple de `RELATION1` et un tuple de `RELATION2` vérifiant la condition de rapprochement.

La jointure de deux relations produit donc une troisième relation qui contient toutes les combinaisons de tuples des deux relations initiales satisfaisant la condition spécifiée. La condition doit bien sûr permettre le rapprochement des deux relations, et donc être du type :

```
<Attribut1> <opérateur> <Attribut2>
```

où `Attribut1` appartient à `RELATION1` et `Attribut2` à `RELATION2`. Selon le type d'opérateur, on distingue :

- **l'équi-jointure** dans le cas où l'opérateur est =, qui est une véritable composition de relations au sens mathématique du terme ;
- **l'inéqui-jointure** dans les autres cas, c'est-à-dire avec un des opérateurs <, ≤, >, ≥, ≠.

Dans le cas d'équi-jointure, les deux attributs égaux apparaissent chacun dans le résultat : il y a donc duplication d'une même valeur dans chaque tuple. Afin d'éliminer cette redondance, on définit la **jointure naturelle** comme suit :

Notion VI.17 : Jointure naturelle (*Natural join*)

Opération consistant à rapprocher les tuples de deux relations `RELATION1` et `RELATION2` afin de former une troisième relation `RELATION3` dont les attributs sont l'union des attributs de `RELATION1` et `RELATION2`, et dont les tuples sont obtenus en composant un tuple de `RELATION1` et un tuple de `RELATION2` ayant mêmes valeurs pour les attributs de même nom.

L'opération de jointure est représentée par l'une des notations suivantes, la condition étant simplement omise dans le cas de jointure naturelle (c'est alors l'égalité des attributs de même nom) :

- ```
  RELATION1 |X| RELATION2
        Condition
  ```
- ```
  JOIN (RELATION1, RELATION2, Condition)
  ```

La figure VI.18 donne la représentation graphique de l'opération de jointure ; la figure VI.19 illustre cette opération en effectuant la jointure naturelle des deux relations `VINS` et `LOCALISATION`. L'inéqui-jointure de ces deux relations selon la condition `QUALITE > QUALITE MOYENNE` est représentée figure VI.20. On suppose que les qualités sont codées par ordre décroissant `A, B, C, D, E`.

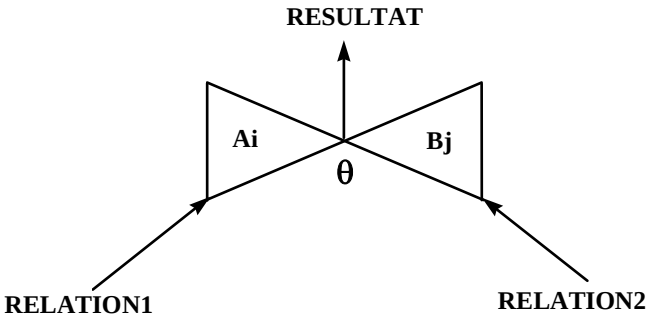


Figure VI.18 — Représentation graphique de la jointure

VINS	Cru	Mill	Qualité
	VOLNAY	1983	A
	VOLNAY	1979	B
	CHABLIS	1983	A
	JULIENAS	1986	C

⋈

LOCALISATION	Cru	Région	QualMoy
	VOLNAY	Bourgogne	A
	CHABLIS	Bourgogne	A
	CHABLIS	Californie	B

↓

VINSREG	Cru	Mill	Qualité	Région	QualMoy
	VOLNAY	1983	A	Bourgogne	A
	VOLNAY	1979	B	Bourgogne	A
	CHABLIS	1983	A	Bourgogne	A
	CHABLIS	1983	A	Californie	B

Figure VI.19 — Jointure naturelle des relations `VINS` et `LOCALISATION`

VINS	Cru	Mill	Qualité
	VOLNAY	1983	A
	VOLNAY	1979	B
	CHABLIS	1983	A
	JULIENAS	1986	C

⋈ Qualité ≠ QualMoy

LOCALISATION	Cru	Région	QualMoy
	VOLNAY	Bourgogne	A
	CHABLIS	Bourgogne	A
	CHABLIS	Californie	B

↓

VINSNLOC	Cru	Mill	Qualité	Cru	Région	QualMoy
	VOLNAY	1983	A	CHABLIS	Californie	B
	VOLNAY	1979	B	VOLNAY	Bourgogne	A
	VOLNAY	1979	B	CHABLIS	Bourgogne	A
	CHABLIS	1983	A	CHABLIS	Californie	B
	JULIENAS	1986	C	VOLNAY	Bourgogne	A
	JULIENAS	1986	C	CHABLIS	Bourgogne	A
	JULIENAS	1986	C	CHABLIS	Californie	B

Figure VI.20 — Inéqui-jointure des relations `VINS` et `LOCALISATION`

PDF-XChange Click to buy NOW! www.docu-track.com

La jointure n'est pas toujours considérée comme une opération de base de l'algèbre relationnelle. En effet, si l'on étend la définition de la restriction de manière à considérer des conditions multi-attributs du type `<Attribut1> <Opérateur> <Attribut2>`, alors la jointure peut être obtenue par un produit cartésien suivi d'une restriction du résultat comme suit :

```
JOIN (RELATION1, RELATION2, Condition) =

RESTRICT ((RELATION1 X RELATION2), Condition)
```

Compte tenu de son jointure, nous avons préféré ici définir la jointure comme une opération de base.

5. L'ALGEBRE RELATIONNELLE : OPERATIONS DERIVEES

Les opérations présentées ci-dessous sont parfois utilisées pour manipuler des relations. Elles peuvent en général être obtenues par combinaison des opérations précédentes. Dans certains cas (complément, jointure externe), leur expression à partir des opérations de base nécessite la manipulation de relations constantes à tuples prédéfinis, telles que la relation obtenue par le produit cartésien des domaines, ou encore celle composée d'un seul tuple à valeurs toutes nulles.

5.1 Intersection

L'**intersection** est l'opération classique de la théorie des ensembles adaptée aux relations de même schéma.

> **Notion VI.18 : Intersection (*Intersection*)**
>
> Opération portant sur deux relations de même schéma `RELATION1` et `RELATION2` consistant à construire une relation de même schéma RELATION3 ayant pour tuples ceux appartenant à la fois à `RELATION1` et `RELATION2`.

Plusieurs notations ont été introduites pour cette opération selon les auteurs :

- `RELATION1 ∩ RELATION2`
- `INTERSECT (RELATION1, RELATION2)`
- `AND (RELATION1, RELATION2)`

La notation graphique représentée figure VI.21 est aussi utilisée.

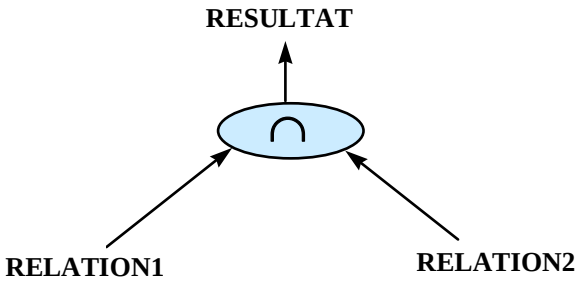


Figure VI.21 — Représentation graphique de l'intersection

L'intersection est une opération redondante avec les opérations de base, puisqu'il est possible de l'obtenir à partir de la différence à l'aide d'une des formules suivantes :

- `RELATION1 ∩ RELATION2 = RELATION1 - (RELATION1 - RELATION2)`
- `RELATION1 ∩ RELATION2 = RELATION2 - (RELATION2 - RELATION1)`

5.2 Division

La **division** permet de rechercher dans une relation les sous-tuples qui sont complétés par tous ceux d'une autre relation. Elle permet ainsi d'élaborer la réponse à des questions de la forme « quel que soit x, trouver y » de manière simple.

> **Notion VI.19 : Division (Division)**
>
> Opération consistant à construire le quotient de la relation `D (A1, A2... Ap, Ap+1, ... An)` par la relation `d(Ap+1, ... An)` comme la relation `Q(A1, A2... Ap)` dont les tuples sont ceux qui concaténés à tout tuple de `d` donnent un tuple de `D`.

De manière plus formelle, désignons par ai une valeur quelconque de l'attribut Ai. Un tuple est alors une suite de valeurs <a1, a2, a3 ...>. Utilisant ces notations, le quotient de D par D est défini par :

```
Q = {<a1, a2, ... ap> tel-que quel-que-soit <ap+1, ... an>
  de d, <a1, a2, ... ap, ap+1, ..., an> appartient à D}
```

Les notations possibles pour la division sont :

- `D ÷ d`
- `DIVISION (D, d)`

ainsi que la représentation graphique de la figure VI.22. Un exemple de division est représenté figure VI.23.

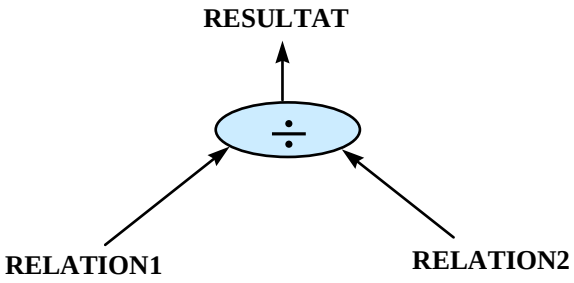


Figure VI.22 — Représentation graphique de la division

VINS	Cru	Mill	Qualité
	VOLNAY	1983	A
	VOLNAY	1979	B
	CHABLIS	1983	A
	CHABLIS	1979	A
	JULIENAS	1986	A

÷

QUALITE	Mill	Qualité
	1983	A
	1979	A

↓

CRUS	Cru
	CHABLIS

Figure VI.23 — Exemple de division

La division peut être obtenue à partir de la différence, du produit cartésien et de la projection comme suit :

- `D - d = R1 - R2 avec`
- `R1 =` $\prod$`A1, A2, ... Ap(D)`
- `R2 =` $\prod_{\texttt{A1, A2, ... Ap}}$`((R1 X d) - D)`

5.3 Complément

Le **complément** permet de trouver les tuples qui n'appartiennent pas à une relation. Il suppose a priori que les domaines sont finis (sinon on obtient des relations infinies).

> **Notion VI.20 : Complément (*Complement*)**
>
> Ensemble des tuples du produit cartésien des domaines des attributs d'une relation n'appartenant pas à cette relation.

Le complément d'une relation `RELATION1` est noté au choix :

- RELATION1
- NOT(RELATION1)
- COMP(RELATION1)

La figure VI.24 illustre cette opération dans un cas simple. C'est une opération peu utilisée du fait qu'elle permet de générer des tuples qui ne sont pas dans la base, en général très nombreux. Si l'on note par X Di le produit cartésien des domaines, le complément d'une relation RELATION1 est obtenu à partir de la différence comme suit :

- NOT (RELATION1) = X Di - RELATION1

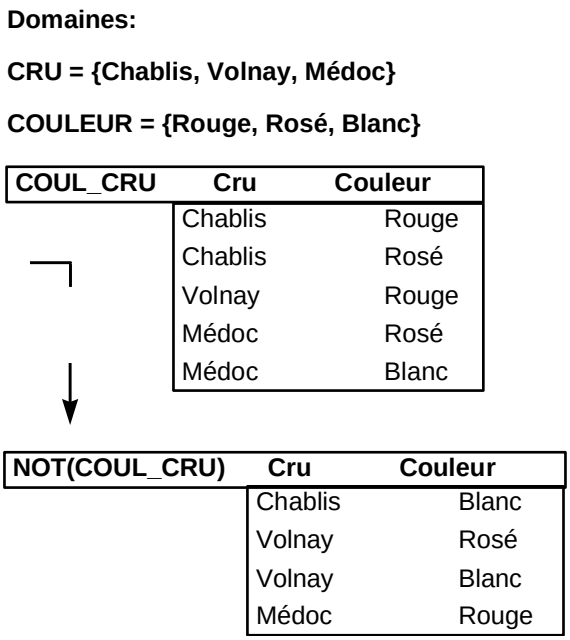


Figure VI.24 — Exemple de complément

5.4 Éclatement

L'**éclatement** [Fagin80] est une opération qui n'appartient pas vraiment à l'algèbre rationnelle puisqu'elle donne deux relations en résultats, à partir d'une. Elle est cependant utile pour partitionner une relation horizontalement en deux sous-relations. A ce titre, elle est considérée comme une extension de l'algèbre.

> **Notion VI.21 : Éclatement (*Split*)**
>
> Opération consistant à créer deux relations à partir d'une relation RELATION1 et d'une condition, la première contenant les tuples de RELATION1 et la deuxième ceux ne la vérifiant pas.

Cet opérateur appliqué à la relation RELATION1 génère donc deux relations R1 et R2 qui seraient obtenues par restriction comme suit :

- R1 = RESTRICT (RELATION1, Condition)

PDF-XChange
Click to buy NOW!
www.docu-track.com

- `R2 = RESTRICT (RELATION1, ¬ Condition)`

5.5 Jointure externe

Les jointures définies ci-dessus perdent des tuples d'au moins une relation quand les relations jointes n'ont pas de projections identiques sur l'attribut de jointure. Pour préserver toutes les informations dans tous les cas, il est nécessaire de définir des jointures qui conservent les tuples sans correspondant avec des valeurs nulles associées quand nécessaire. C'est dans ce but que Codd [Codd79] a introduit les **jointures externes**.

> **Notion VI.22 : Jointure externe (*External Join*)**
>
> Opération générant une relation `RELATION3` à partir de deux relations `RELATION1` et `RELATION2`, par jointure de ces deux relations et ajout des tuples de `RELATION1` et `RELATION2` ne participant pas la jointure, avec des valeurs nulles pour les attributs de l'autre relation.

Cette opération est très utile, en particulier pour composer des vues sans perte d'informations. Elle se note en général comme suit :

- `RELATION1 [X] RELATION2`
- `EXT-JOIN (RELATION1, RELATION2)`

La jointure externe permet par exemple de joindre des tables `CLIENTS` et `COMMANDES` sur un numéro de client commun, en gardant les clients sans commande et les commandes sans client associé. Elle est donc très utile en pratique. On peut aussi distinguer la jointure externe droite qui garde seulement les tuples sans correspondant de la relation de droite. On notera celle-ci `|X]` ou `REXT-JOIN`. De même, on peut distinguer la jointure externe gauche (`[X|` ou `LEXT-JOIN`). Un exemple de jointure externe complète apparaît figure VI.25.

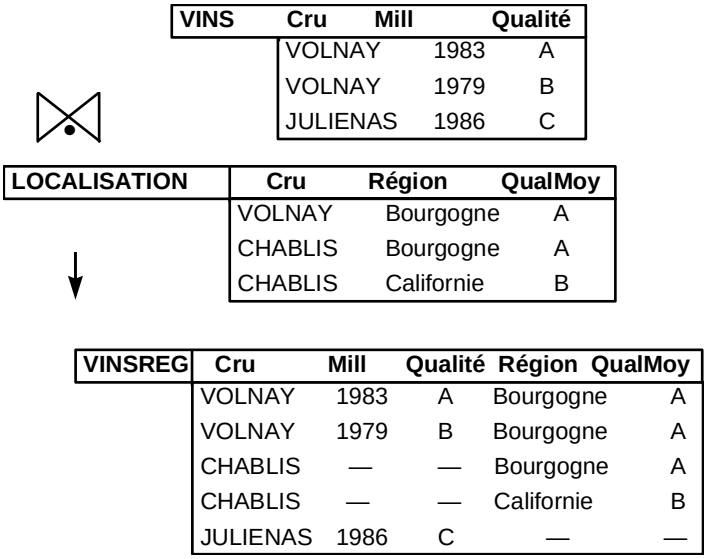


Figure VI.25 — Exemple de jointure externe

5.6 Semi-jointure

Dans certains cas, lors de l'exécution d'une jointure, il n'est pas nécessaire de conserver tous les attributs des deux relations en résultat : seuls les attributs d'une des deux relations sont conservés. Une opération spécifique de **semi-jointure**, très utile pour optimiser l'évaluation des questions, a été définie [Berstein81].

> **Notion VI.23 : Semi-jointure (*Semi-join*)**
>
> Opération portant sur deux relations `RELATION1` et `RELATION2`, donnant en résultat les tuples de `RELATION1` qui participent à la jointure des deux relations.

La semi-jointure de la relation `RELATION1` par la relation `RELATION2` est notée :

- `RELATION1 |⋉ RELATION2`
- `SEMI-JOIN (RELATION1, RELATION2)`

Elle est équivalente à la jointure des relations `RELATION1` et `RELATION2` suivie par une projection du résultat sur les attributs de la relation `RELATION1`. Notez que l'opération n'est pas symétrique puisque seuls des tuples de la première relation sont conservés. Elle peut être vue comme une restriction de la relation `RELATION1` par les valeurs des attributs de jointure figurant dans la relation `RELATION2`. La figure VI.26 illustre cette opération de semi-jointure.

VINS	Cru	Mill	Qualité
	VOLNAY	1983	A
	VOLNAY	1979	B
	CHABLIS	1983	A
	JULIENAS	1986	C

⋈

LOCALISATION	Cru	Région	QualMoy
	VOLNAY	Bourgogne	A
	CHABLIS	Bourgogne	A
	CHABLIS	Californie	B

↓

VINSLOC	Cru	Mill	Qualité
	VOLNAY	1983	A
	VOLNAY	1979	B
	CHABLIS	1983	A

Figure VI.26 — Exemple de semi-jointure

5.7 Fermeture transitive

La **fermeture transitive** est une opération très particulière qui permet d'ajouter des tuples à une relation. Elle n'appartient pas à l'algèbre rationnelle, mais peut être vue comme une de ses extensions. Il n'est pas possible de constituer cette opération avec un nombre fixe d'opérations de l'algèbre rationnelle : elle peut être effectuée par une série de jointure/projection/union, mais le nombre d'opérations à effectuer dépend du contenu de la relation. Certains auteurs considèrent que c'est une faiblesse de l'algèbre relationnelle que de ne pas pouvoir exprimer une fermeture transitive par une expression constante d'opérations élémentaires.

> **Notion VI.24 : Fermeture transitive (*Transitive closure*)**
>
> Opération sur une relation R à deux attributs (A1, A2) de même domaine consistant à ajouter à R tous les tuples qui se déduisent successivement par transitivité, c'est-à-dire que si l'on a des tuples <a, b> et <b, c>, on ajoute <a, c>.

Cette opération se note suivant les auteurs :

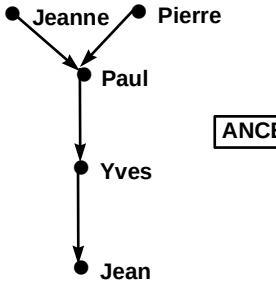
- $\tau(R)$
- R^+
- CLOSE(R)

Pour effectuer une fermeture transitive, il est nécessaire d'effectuer une boucle d'opérations jusqu'à obtention de la fermeture complète. On doit donc utiliser un langage de programmation avec une boucle *while*, comme suit :

$\tau(R)$ = *while* $\tau(R)$ *change do* $\tau(R) = \tau(R) \cup \Pi_{A1,A4}$ (R |X| $\tau(R)$).

Cette opération permet par exemple de calculer à partir d'une relation PARENTS (Parent, Enfant) la relation ANCETRES (Ascendant, Descendant), qui donne toute la

filiation connue d'une personne. La figure VI.27 illustre cette opération. La fermeture transitive de PARENTS est calculée par jointures/projections/unions successives de la relation PARENTS avec elle-même jusqu'à saturation, c'est-à-dire jusqu'à obtention d'une relation stable à laquelle une nouvelle jointure/projection/union n'apporte plus de tuples. On voit que la relation ANCETRES représente le graphe correspondant à la fermeture transitive du graphe de la relation PARENTS.



Graphe de la relation PARENTS

PARENTS	Parent	Enfant
	Pierre	Paul
	Jeanne	Paul
	Paul	Yves
	Yves	Jean

ANCETRES	Ascendant	Descendant
	Pierre	Paul
	Jeanne	Paul
	Paul	Yves
	Yves	Jean
	Pierre	Yves
	Jeanne	Yves
	Pierre	Jean
	Jeanne	Jean
	Paul	Jean

Figure VI.27 — Exemple de fermeture transitive

6. LES EXPRESSIONS DE L'ALGEBRE RELATIONNELLE

A partir des opérations de l'algèbre relationnelle, il est possible de composer un langage d'interrogation de bases de données. Une question peut alors être représentée par un arbre d'opérateurs relationnels. Le paraphrasage en anglais de telles expressions est à la base du langage SQL que nous étudierons dans le chapitre suivant.

6.1 Langage algébrique

En utilisant des expressions d'opérations de l'algèbre relationnelle, il est possible d'élaborer les réponses à la plupart des questions que l'on peut poser à une base de données relationnelle. Plus précisément, les opérations de base de l'algèbre relationnelle constituent un **langage complet**, c'est-à-dire ayant la puissance de la logique du premier ordre.

> **Notion VI.25 : Langage complet (*Complete language*)**
>
> Langage équivalent à la logique du premier ordre.

Nous avons étudié plus précisément la logique du premier ordre et les langages d'interrogation qui en découlent au chapitre 5. Disons simplement que l'algèbre relationnelle permet d'exprimer toute question exprimable avec la logique du premier ordre sans fonction, par exemple avec le calcul de tuple ou de domaine. L'algèbre relationnelle peut d'ailleurs être étendue avec des fonctions [Zaniolo85].

Voici quelques questions sur la base DEGUSTATION dont le schéma a été représenté figure VI.7. Elles peuvent être exprimées comme des expressions d'opérations, ou comme des opérations successives appliquées sur des relations intermédiaires ou de base, générant des

PDF-XChange Click to buy NOW! www.docu-track.com

relations intermédiaires. Pour des raisons de clarté, nous avons choisi cette deuxième représentation. L'expression peut être obtenue simplement en supprimant les relations intermédiaires notées Ri.

(Q1) Donner les degrés des vins de crus Morgon et de millésime1978 :

```
R1 = RESTRICT (VINS, CRUS = "MORGON")
R2 = RESTRICT (VINS, MILLESIME = 1978)
R3 = INTERSECT (R1, R2)
RESULTAT = PROJECT (R3, DEGRE)
```

(Q2) Donner les noms et prénoms des buveurs de Morgon ou Chenas :

```
R1 = RESTRICT (VINS, CRU = "MORGON")
R2 = RESTRICT(VINS, CRUS = "CHENAS")
R3 = UNION (R1, R2)
R4 = JOIN (R3, ABUS)
R5 = JOIN (R4, BUVEURS)
RESULTAT = PROJECT (R5, NOM, PRENOM)
```

(3) Donner les noms et adresses des buveurs ayant bu plus de 10 bouteilles de Chablis 1976 avec le degré de ce vin :

```
R1 = RESTRICT (ABUS, QUANTITE > 10)
R2 = RESTRICT (VINS, CRU = "CHABLIS")
R3 = RESTRICT (VINS, MILLESIME = 1976)
R4 = INTERSECT (R2, R3)
R5 = JOIN (R1, R4)
R6 = PROJECT (R5, NB, DEGRE)
R7 = JOIN (R6, BUVEURS)
RESULTAT = PROJECT (R7, NOM, ADRESSE, DEGRE)
```

(Q4) Donner les noms des buveurs n'ayant bu que du Morgon :

```
R1 = JOIN(BUVEURS,ABUS,VINS)
R2 = RESTRICT(R1, CRU = "MORGON")
R3 = PROJECT(R2, NOM)
R4 = RESTRICT(R1, CRU ≠ "MORGON")
R5 = PROJECT(R1, NOM)
RESULTAT = MINUS(R3 - R5)
```

Si l'on accepte les relations constantes, l'algèbre relationnelle permet aussi d'exécuter les mises à jour. Par exemple, l'intersection du vin `[100, TOKAY, 1978, 13]` s'effectuera comme suit :

```
R1 = CONSTANTE (VINS, [100, TOKAY, 1978, 13])
VINS = UNION (VINS, R1)
```

où `CONSTANTE` permet de définir une relation de même schéma que la relation `VINS` contenant le seul tuple indiqué en argument.

6.2 Arbre algébrique

Une question exprimée sous forme d'un programme d'opérations de l'algèbre relationnelle peut être représentée par un **arbre relationnel**. Les nœuds correspondent aux représentations graphiques des opérations indiquées ci-dessus et les arcs aux flots de données entre opérations.

> **Notion VI.26 : Arbre relationnel (*Relational tree*)**
>
> Arbre dont les nœuds correspondent à des opérations de l'algèbre relationnelle et les arcs à des relations de base ou temporaires représentant des flots de données entre opérations.

Ainsi, la question « Noms et Prénoms des buveurs habitant Paris ayant bu du Chablis depuis le 1er janvier 1992 » peut être exprimée à l'aide de l'arbre représenté figure VI.28. Plusieurs arbres équivalents peuvent être déduits d'un arbre donné à l'aide de règles de transformation simples, telles que la permutation des jointures et restrictions, la permutation des projections et des jointures, le regroupement des intersections sur une même relation, etc. Ces transformations sont à la base des techniques d'optimisation de questions qui dépassent le sujet de ce chapitre. La figure VI.29 propose un arbre équivalent à celui de la figure VI.28, obtenu par descente de projections et regroupement d'intersections.

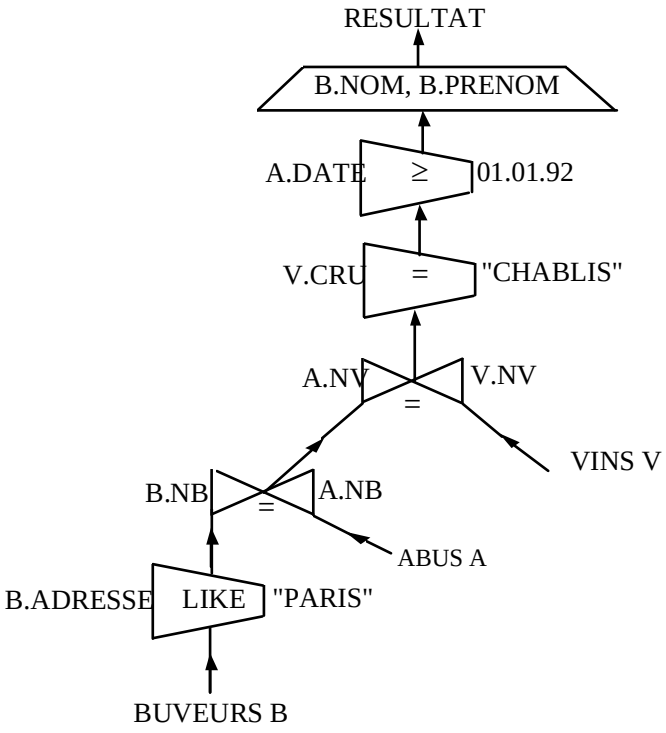


Figure VI.28 — Exemple d'arbre représentant une question

PDF-XChange
Click to buy NOW!
www.docu-track.com

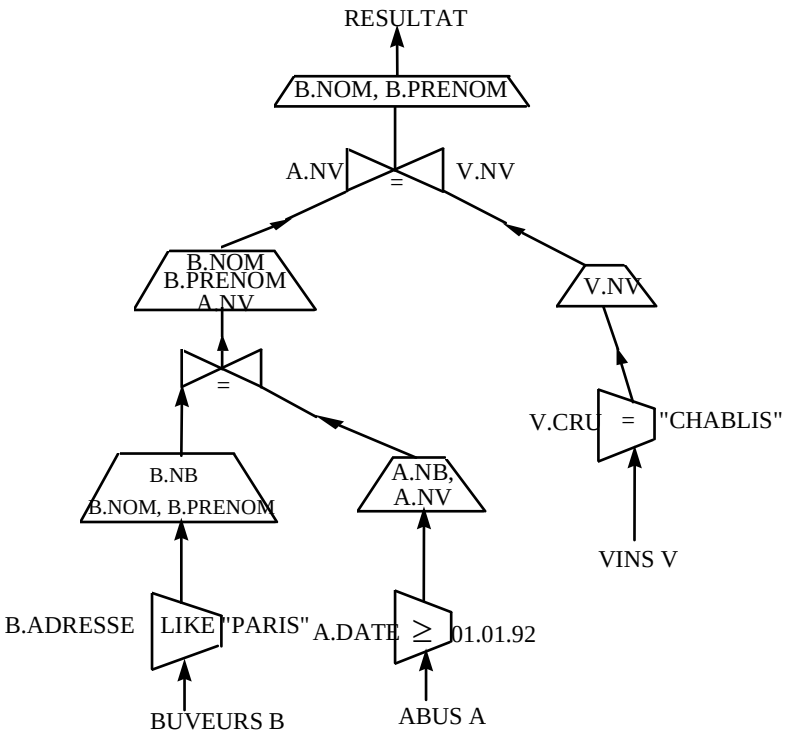


Figure VI.29 — Arbre équivalent à l'arbre précédent

La composition d'opérations de l'algèbre relationnelle ne nécessite pas toujours d'attendre le résultat de l'opération précédente pour exécuter l'opération suivante. Restriction, projection et jointure peuvent ainsi être exécutées par des algorithmes à flots de données. Des opérateurs se succédant sur un arbre peuvent donc être exécutés en parallèle par des algorithmes « pipe-line ». Des opérateurs figurant sur des branches distinctes d'un arbre peuvent aussi être exécutés en parallèle de manière indépendante. La représentation par arbre algébrique met ainsi en évidence les possibilités de parallélisme et les enchaînements nécessaires. Ces propriétés des arbres relationnels sont importantes pour optimiser les questions dans des contextes parallèles.

7. FONCTIONS ET AGREGATS

L'algèbre relationnelle est insuffisante pour traiter de véritables applications des bases de données, telles la suivie de production, la gestion de budget, etc. Il est en effet nécessaire d'effectuer des calculs sur la base pour supporter de telles applications. C'est l'objet de l'introduction des fonctions de calcul au sein de l'algèbre et du support des agrégats.

7.1 Fonctions de calcul

La possibilité d'effectuer des calculs sur les attributs est simplement introduite en généralisant projection, restriction et jointure par introduction de fonctions. Ainsi, tout attribut apparaissant en argument d'une opération est remplacé par une **expression d'attributs**.

Notion VI.27 : Expression d'attributs (*Attribut expression*)

Expression arithmétique construite à partir d'attributs d'une relation et de constantes, par application de fonctions arithmétiques successives.

Voici des exemples d'expressions d'attributs :

```
10 + 50 - 23 ;
QUANTITE * 50 ;
QUANTITE * DEGRE / 100 ;
QUANTITE - QUANTITE * DEGRE / 100.
```

De telles expressions peuvent donc être utilisées comme arguments de projections, de restrictions voire de jointures. Le programme d'algèbre relationnelle suivant illustre ces possibilités :

```
R1 = JOIN (VINS, ABUS, DEGRE*QUANTITE/100 > QUANTITE/10)
R2 = RESTRICT(R1, DEGRE*QUANTITE/100 > 10)
RESULTAT = PROJECT(R2, CRU, QUANTITE - QUANTITE*DEGRE/100)
```

La notion d'expression d'attributs peut être généralisée avec des fonctions autres que les fonctions arithmétiques, par exemple des fonctions écrites dans un langage externe. On aboutit ainsi à une généralisation de l'algèbre relationnelle aux fonctions [Zaniolo85].

7.2 Support des agrégats

Les expressions d'attributs permettent d'effectuer des opérations de calcul en ligne, sur des attributs de relations. En pratique, il est nécessaire d'effectuer des opérations de calcul en colonnes, sur des tuples de relations, cela par exemple afin de sommer des dépenses, etc. Le concept d'**agrégat** permet de telles opérations.

Notion VI.28 : Agrégat (*Aggregat*)

Partitionnement horizontal d'une relation en fonction des valeurs d'un groupe d'attributs, suivi d'un regroupement par application d'une fonction de calcul sur ensemble.

Les fonctions de calcul sur ensemble les plus souvent proposées sont :

- `SOMME (SUM)` permettant de calculer la somme des éléments d'un ensemble ;
- `MOYENNE (AVG)` permettant de calculer la moyenne des éléments d'un ensemble ;
- `MINIMUM (MIN)` permettant de sélectionner l'élément minimum d'un ensemble ;
- `MAXIMUM (MAX)` permettant de sélectionner l'élément maximum d'un ensemble ;
- `COMPTE (COUNT)` permettant de compter les éléments d'un ensemble.

La figure VI.30 illustre le concept d'agrégat. La table `VINS` est enrichie d'un attribut `QUANTITE`. L'agrégat représenté calcule la somme des quantités par `CRU`. L'opération générique s'écrit :

R = AGREGAT(<RELATION> ; <ATTRIBUT1> ; <FONCTION>**{**<ATTRIBUT2>**})**

<Attribut1> représente un ou plusieurs attributs utilisés pour le partitionnement. Fonction est la fonction d'ensemble appliquée à l'attribut <Attribut2>. Par exemple, on écrit :

RESULTAT = AGREGAT(VINS ; CRU ; SUM{QUANTITE})

pour l'agrégat illustré figure VI.30. Notez qu'un agrégat peut s'effectuer sans partitionnement ; on peut par exemple calculer simplement la moyenne de tous les degrés comme indiqué figure VI.30.

RESULTAT = AGREGAT(VINS ; ; AVG{DEGRE}**)**.

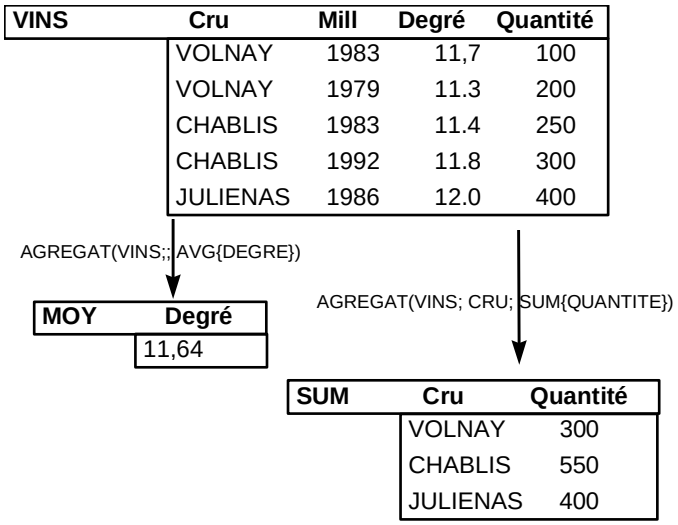


Figure VI.30 — Exemples de calcul d'agrégats

8. CONCLUSION

Dans ce chapitre, nous avons introduits les concepts essentiels aujourd'hui supportés par le modèle relationnel dans les grands systèmes industriels tels que ORACLE, INGRES, DB2, SYBASE, etc. Ces concepts constituent la base du langage SQL, le langage des systèmes relationnels. Nous allons étudier ce langage et ses extensions dans les chapitres qui suivent.

Le modèle relationnel fait aujourd'hui autorité dans l'industrie. Issu de la théorie des relations, il est à l'origine une remarquable construction de la recherche. Il a su progressivement intégrer des concepts de plus en plus riches, tels que l'intégrité référentielle, les réflexes, etc. Le modèle en est aujourd'hui à intégrer les concepts de l'objet pour fonder l'objet-relationnel, comme nous le verrons dans la troisième partie de cet ouvrage. Il a encore un bel avenir devant lui, bien qu'il soit parfois contesté par les tenants de l'objet.

9. BIBLIOGRAPHIE

[Berstein81] Bernstein P., Goodman N., « The Power of Natural Semijoins », *Siam Journal of Computing*, Vol. 10, N° 4, décembre 1981, pp. 751-771.

Une discussion de la puissance de l'opérateur de semi-jointure. Après une définition de la semi-jointure, Phil Bernstein et Nathan Goodman montrent que cet opérateur permet d'exprimer un grand nombre de questions avec jointures, plus spécifiquement toutes celles dont le graphe des jointures est un arbre.

[Chen76] Chen P.P., « The Entity-Relationship Model - Towards a Unified View of Data », *ACM Transactions on Database Systems*, Vol. 1, N° 1, mars 1976.

L'article de base sur le modèle entité-association. Il introduit ce modèle pour décrire la vue des données d'une entreprise. En particulier, les diagrammes de Chen sont présentés. Il est montré que le modèle permet d'unifier les différents points de vue et de passer simplement à une implémentation relationnelle. Ce dernier point explique le fait que beaucoup d'outils d'aide à la conception de bases de données relationnelles utilisent une variante du modèle de Chen.

[Childs68] Childs D.L., « Feasibility of a Set-Theoretic Data Structure — A General Structure Based on a Reconstituted Definition of a Relation », *Congrès IFIP, Genève*, 1968, pp. 162-172.

Un des premiers articles proposant un modèle basé sur le concept de relation et des opérateurs ensemblistes. La proposition de Codd s'est inspirée des travaux de Childs.

[Codd70] Codd E.F., « A Relational Model for Large Shared Data Banks », *Communications de l'ACM, Vol. 13, N° 6*, juin 1970, pp. 377-387.

L'article de base proposant le modèle relationnel. Il introduit les concepts essentiels de relation, domaine, attribut et clé. L'algèbre relationnelle est proposée comme langage de manipulation des relations.

[Codd79] Codd E.F., « Extending the Relational Model to Capture More Meaning », *ACM TODS, Vol. 4, N° 4*, décembre 1979, pp. 397-433.

Une proposition d'extension du modèle relationnel pour mieux décrire la sémantique des applications. L'idée de base est de distinguer différents types de relations (entité, association, généralisation, etc.) et d'étendre les opérateurs relationnels en conséquence. Un traitement des valeurs nulles avec une logique trivaluée est aussi proposé. L'ensemble du modèle étendu, appelé RM/T, est décrit par un métamodèle relationnel.

[Date81] Date C.J., « Referential Integrity », *7e Very Large Data Bases*, Cannes, France, 1981, IEEE Ed.

L'article proposant le support de l'intégrité référentiel au sein du modèle relationnel. Chris Date introduit les contraintes référentielles et les contraintes d'entité comme contraintes de base à intégrer au modèle relationnel pour un meilleur support de la sémantique des données. Un langage de déclaration de contraintes est aussi esquissé.

[Delobel83] Delobel C., Adiba M., *Bases de Données et Systèmes Relationnels*, livre, Dunod Informatique, 1983.

PDF-XChange
Click to buy NOW!
www.docu-track.com

Un des premiers livres français sur les bases de données relationnelles. Alors que le livre Bases de Données - Les Systèmes et Leurs Langages *de Georges Gardarin paru à la même époque propose une vue simplifiée des différents concepts, ce livre offre une vision plus formelle, souvent fondée sur les travaux de l'université de Grenoble. Des opérateurs relationnels spécifiques et une approche originale à la normalisation des relations sont notamment développés.*

[Fagin80] Fagin R., « A Normal Form for Relational Databases that is Based on Domains and Keys », *ACM TODS, Vol. 6, N° 3*, septembre 1981, pp. 387-415.

Un des articles prosant une forme ultime de normalisation des relations. Une relation est en 5e forme normale si elle ne peut plus être décomposée par projection sur différents sous-schémas, de sorte à obtenir la relation de départ par jointures naturelles des sous-relations. Fagin introduit une méthode de normalisation fondée sur les domaines et les clés. Il montre qu'il s'agit là de la forme ultime de normalisation par projection et jointure. Il introduit aussi un opérateur d'éclatement qui permet d'envisager d'autres formes de normalisation horizontale.

[Gardarin89] Gardarin G., Cheiney J.P., Kiernan J., Pastre D., « Managing Complex Objects in an Extensible DBMS », *15th Very Large Data Bases International Conference*, Morgan Kaufman Pub., Amsterdam, Pays-Bas, août 1989.

Une présentation détaillée du support d'objets complexes dans le SGBD extensible Sabrina. Ce système est dérivé du SGBD relationnel SABRE et supporte des types abstraits comme domaines d'attributs. Il a aujourd'hui évolué vers un SGBD géographique (GéoSabrina) et est commercialisé par INFOSYS.

[ISO89] International Organization for Standardization, « Information Processing Systems - Database Language SQL with Integrity Enhancement », *International Standard ISO/IEC JTC1 9075* : 1989(E), 2e édition, avril 1989.

La norme SQL aujourd'hui en vigueur. Ce document de 120 pages présente la norme SQL1 : concepts de base, éléments communs, langage de définition de schémas, définition de modules de requêtes, langage de manipulation. Toutes la grammaire de SQL1 est décrite en BNF. Ce document résulte de la fusion du standard de 1986 et des extensions pour l'intégrité de 1989.

[Maier83] Maier D., *The Theory of Relational Databases*, livre, Computer Science Press, 1983.

Le livre synthétisant tous les développements théoriques sur les bases de données relationnelles. En 600 pages assez formelles, Maier fait le tour de la théorie des opérateurs relationnels, des dépendances fonctionnelles, multivaluées et algébriques, et de la théorie de la normalisation.

[Stonebraker87] Stonebraker M., « The Design of the POSTGRES Storage System », *13th Very Large Databases International Conference*, Morgan & Kauffman Ed., Brighton, Angleterre, 1987.

Une description assez complète du système POSTGRES, successeur d'INGRES développé à Berkeley. M. Stonebraker présente la conception du noyau de stockage du système POSTGRES. Outre un contrôle de concurrence original permettant le support de déclencheurs (ou réflexes), ce sytème intégre les types abstraits au modèle relationnel. Un type abstrait permet de définir un type d'attribut ou de tuple. Le système POSTGRES a ainsi permis de prototyper les fonctionnalités orientées objets intégrées à la version 7 d'INGRES.

[Ullman88] Ullman J.D., *Principles of Database and Knowledge-base Systems*, livres, volumes I et II, Computer Science Press, 1988.

Deux volumes très complets sur les bases de données, avec une approche plutôt fondamentale. Jeffrey Ullman détaille tous les aspects des bases de données, des méthodes d'accès aux modèles objets en passant par le modèle logique. Les livres sont finalement centrés sur une approche par la logique aux bases de données. Les principaux algorithmes d'accès, d'optimisation de requêtes, de concurrence, de normalisation, etc. sont détaillés.

[Zaniolo83] Zaniolo C., « The Database Language GEM », *ACM SIGMOD Conférence*, San José, Ca., ACM Ed., 1983.

Une extension du modèle relationnel vers le modèle entité-association et du langage QUEL pour supporter un tel modèle. Carlo Zaniolo propose d'utiliser les entité pour spécifier les domaines lors de la définition des associations. Il propose alors une extension de QUEL avec une notation pointée pour naviguer depuis les associations vers les attributs des entités. L'ensemble constitue le langage GEM, qui a été implémenté à Bell Labs au-dessus de la machine bases de données IDM.

[Zaniolo85] Zaniolo C., « The Representation and Deductive Retrieval of Complex Objects », *11th Very Large Data Bases International Conference*, Morgan Kaufman Pub., Stockholm, Suède, août 1985.

Une extension de l'algèbre relationnelle au support de fonctions. Cet article présente une extension de l'algèbre relationnelle permettant de référencer des fonctions symboliques dans les critères de projection et de sélection, afin de manipuler des objets complexes. Des opérateurs déductifs de type fermeture transitive étendue sont aussi intégrés.