Groupe de réflexion et de partage relatif à l'enseignement de l'informatique

Département d'informatique
Faculté des sciences
Université de Sherbrooke

BD010_SYN
Introduction à la théorie relationnelle

Émetteur : Luc Lavoie

Date de diffusion initiale : 2019-05-05

Objectif : Présenter l'essentiel de la théorie relationnelle et documenter une notation commune au sein des groupes Ἀκαδήμεια, Μῆτις et GRIIS.

Plan

Sommaire 2
Historique des révisions 2
1 Introduction 3
1.1 Objet et portée du document 3
1.2 Évolution du document 3
1.3 Travail en cours ou projeté 3
1.4 Contenu des sections 3
1.5 Conventions 3
2 Structure relationnelle 4
2.1 Ensembles 4
2.2 Couple 5
2.3 Types 5
2.4 Attributs 6
2.5 Tuples 6
2.6 Relations 6
3 Opérateurs relationnels 7
3.1 Opérateurs de base 8
3.2 Opérateurs usuels 10
3.3 Opérateurs spécialisés 12
3.4 Opérateurs de groupement 13
3.5 Opérateurs d'agrégation 14
3.6 Opérateurs Tutorial D déclassés 14
3.7 Exercices 14
Références 16

Sommaire

Le module Introduction à la théorie relationnelle

Le présent module a été rédigé dans le cadre de l'exploration du thème Modélisation de données par les membres du groupe **Μῆτις**. Le module présente l'essentiel de la théorie relationnelle et en fixe la notation pour les autres modules du thème et, de façon plus générale, au sein des groupes **Μῆτις** et GRIIS.

Le thème Ἀκαδήμεια comprend un ensemble de modules correspondant aux connaissances usuellement couvertes au sein du B. Sc. nord-américain, de la licence ou du *mastère* européen et de certaines écoles de génie. Ces modules sont destinés aux personnes étudiant les champs disciplinaires de l'informatique, du génie logiciel ou de l'informatique de gestion. Nous espérons toutefois qu'ils pourront être utilisés avec profit par toute personne curieuse d'apprivoiser ce champ de connaissance.

*Le groupe **Μῆτις***

Le groupe **Μῆτις** s'intéresse à la transmission des connaissances dans le domaine de l'informatique.

Groupe **Μῆτις**
Département d'informatique
Faculté des sciences
Université de Sherbrooke
Sherbrooke, QUÉBEC J1K 2R1

Historique des révisions

version	date	auteur	description
0.1.2	2021-10-24	LL	Harmonisation avec la présentation A_REVUE_BD010 préparée pour Douala.
0.1.1b	2021-03-19	LL	Introduction des dénominations « type primitif » et « sous-type » ; distinction de l'union disjointe et de la réunion (ou somme) disjointe.
0.1.1a	2021-01-26	LL	Systématisation de la présentation, correction de coquilles, éclaircissements divers, introduction des traductions SQL des opérateurs de base.
0.1.0d	2020-09-20	LL	Début de présentation de la notation relative aux paires et aux ensembles
0.1.0c	2019-07-25	LL	Correction des coquilles signalées par Maxime Routhier.
0.1.0b	2019-06-23	LL	Correction de coquilles.
0.1.0a	2019-05-01	LL	Récupération de notes diverses.

1 Introduction

1.1 Objet et portée du document

Le présent document a pour but de présenter une synthèse de la théorie relationnelle proposée par Edgar F. Codd et développée par la suite avec l'aide, notamment, de Christopher J. Date, Raymond F. Boyce et Jeffrey D. Ullman. La présentation repose sur des bases minimales :

- ◊ la logique du premier ordre,
- ◊ la théorie des ensembles,
- ◊ l'arithmétique,
- ◊ les langages rationnels.

1.2 Évolution du document

Le présent document tire son origine de l'expérience d'enseignement des auteurs. Cette présentation n'a cessé d'évoluer depuis grâce aux étudiants et auxiliaires d'enseignement qui ont participé aux cours depuis.

1.3 Travail en cours ou projeté

- ◊ Rédiger la section 2 à partir de la présentation BD010.
- ◊ Compléter la partie des opérateurs relatifs aux intervalles de la section 3.
- ◊ Compléter les exemples et exercices de la section 3.
- ◊ Rédiger les sections 4, 5 et 6.
- ◊ Faire les tableaux d'équivalence Discipulus – TD – SQL.

1.4 Contenu des sections

La section 2 présente l'essence de la théorie relationnelle et le modèle qui sera utilisé tout au long de la présentation.

La section 3 présente les opérateurs relationnels tant élémentaires que dérivés.

La section 4 explicite le lien entre la théorie relationnelle et la logique du premier en développant les correspondances entre relation et prédicat, tuple et proposition.

La section 5 intègre le constructeur de type intervalle, les opérateurs de Allen et les opérateurs relationnels proposés par Date qui en tirent parti.

La section 6 présente et compare un sous-ensemble minimal de trois langages de programmation relationnels.

1.5 Conventions

…

Nous utiliserons les opérateurs suivants pour les expressions logiques :

Tableau 1 — Opérateurs de la logique du premier ordre

Symbole	Signification
¬A	Négation de A
A→B	Implication. Si A alors B
A∧B	Conjonction. A et B.
A∨B	Disjonction (inclusive). A ou B.
A↔B	Équivalence. A est équivalent à B ; on dit aussi : A si et seulement si B.
Γ⊢A	Déduction. De l'ensemble de formules Γ on déduit A.
M⊨A	Modélisation. M est un modèle de A ; on dit aussi A est vraie dans M.
⊢A	Théorème. Notion syntaxique
⊨A	Tautologie. Notion sémantique
M⊩A	Réalisabilité. M réalise A, on dit aussi que M « force » A.

ref. : https://fr.wikipedia.org/wiki/Portail:Logique (2021-10-24)

…

2 Structure relationnelle

La théorie relationnelle peut être construite sur la seule base de la logique du premier ordre et de la théorie des ensembles. En pratique, il faut aussi ajouter un modèle de typage. Finalement, il est également utile d'y ajouter l'arithmétique et les langages rationnels afin de rendre compte de certaines propriétés fondamentales et d'intégrer les nombres et les textes aux types primitifs.

2.1 Ensembles

Dans le cadre de la théorie relationnelle, les éléments d'un ensemble sont toujours contraints par un type associé à l'ensemble lui-même – ce type sera désigné comme le « type de référence » de l'ensemble, noté typeref(e) où e est un ensemble. Cette association peut être explicite (e : SET of T ; typeref(e)=T) ou déduite grâce au contexte.

Cas général

…

Dénotation des valeurs et propriétés

{} l'ensemble vide.

{ z } le singleton comprenant la valeur dénotée par l'expression z.

{ z_1, z_2, ... z_n } l'ensemble comprenant les seules valeurs dénotées par les expressions z_i. Le point-virgule « ; » peut être utilisé indifféremment en lieu et place de la virgule « , ».

e la cardinalité de l'ensemble e.

Opérateurs usuels

= ≠

∈ ∉ ∋ ∌

− ∪ ∩ ×

⊂ ⊃ ⊄ ⊅ ⊆ ⊇

Opérateurs moins usuels

⊎ ⊍ ⊌ ⊍ ⊓ ⊓

Raccourcis

Dans la mesure où tous les ensembles ont un type de référence, la notation \e désigne le complément d'un sous-ensemble e par rapport à l'ensemble des valeurs de son type de référence.

\ e $\stackrel{\text{def}}{=}$ typeref(e) – e

Note : en Tutorial D, cet opérateur n'est utilisable que dans un contexte de dénotation explicite des valeurs, ainsi « {ALL BUT e_1,…, e_n} » dénote \{ e_1,…, e_n }. On aurait souhaité, la syntaxe plus générale suivante « ALL BUT *exp* », où *exp* peut être toute expression ensembliste. En Discipulus, la syntaxe générale est utilisée et les expressions d'ensemble (en particulier d'ensembles d'identifiants attributs) sont dénotables.

Cas des ensembles d'identifiants d'attributs

…

À développer.

…

2.2 Couple

Le couple, aussi appelé la paire, est la formation d'une nouvelle entité (ou élément) à partir de deux autres. Il serait possible de reformuler le couple en termes d'ensemble, son utilisation au titre de construction propre relève donc uniquement de la commodité.

Dans le cadre de la théorie relationnelle, les éléments d'une paire sont toujours contraints par un type associé à chacun de ceux-ci – le type de la paire elle-même sera, noté typeref(p) où p est une paire. Cette association peut être explicite (p : PAIR of T ; typeref(p)=T) ou déduite grâce au contexte.

Cas général

…

Dénotation des valeurs et propriétés

(x ; y) pour x et y des expressions dénotant des valeurs appropriées.

Note : la virgule « , » est souvent utilisée en lieu et place du point-virgule « ; » lorsqu'elle n'entraine pas d'ambigüité (e.a. en présence de nombres rationnels représentés en notation fractionnaire décimale).

Opérateurs usuels

@1(x ; y) = x

@2(x ; y) = y

2.3 Type

Si la théorie relationnelle peut s'accommoder de plusieurs modèles de typage, certaines exigences doivent être respectées. Malheureusement, il n'y a pas de consensus relativement à ces exigences (notamment, celles du comité ISO 9575 diffèrent de celles énoncées par Codd et développées par Date et Darwen). Nous nous en tiendrons donc ici à un modèle de typage très simple, également acceptable en regard des jeux d'exigences les plus courants.

Une donnée est valeur associée à une représentation (apte à être traitée par ordinateur).

Une représentation est une suite de signaux (un signal est un phénomène physique mesurable, donc suffisamment stable pour être mesuré).

Dans le cadre de la théorie minimaliste de typage utilisée dans le présent document, un type est soit un type de base, soit un sous-type.

Un type de base (appelé aussi type racine ou domaine) est un ensemble fini de valeurs propres (c'est-à-dire les valeurs d'un type de base n'appartiennent à aucun autre type de base). Une valeur est donc un élément d'un type de base.

Un sous-type (appelé aussi type dérivé) est un sous-ensemble d'un type, sous-ensemble déterminé par une contrainte explicite (qui restreint les valeurs acceptées dans le sous-ensemble).

2.4 Attribut

Un attribut est un couple formé d'un identifiant a et d'un type D, noté a:D. Par abus de langage, lorsque le contexte le permet, il est usuel de désigner l'attribut par son seul identifiant ; ainsi écrit-on l'attribut a.

2.5 Tuple

Un tuple, en tant qu'objet de la théorie relationnelle, est une « composition[1] » d'attributs. En regard de la théorie des types, il s'agit d'un constructeur de type non scalaire applicable à un ensemble d'attributs.

Soit a_i des identifiants distincts et D_j des types, un tuple t est défini comme suit :

$t \triangleq (\{a_1:D_1, a_2:D_2,\ldots, a_n:D_n\} ; \{(a_1,v_1), (a_2,v_2),\ldots (a_n,v_n)\})$

avec $\forall i : 1 \leq i \leq deg(t) \Longrightarrow val(t, a_i) \in def(t, a_i)$

où

$def(t) = \{a_1:D_1, a_2:D_2,\ldots, a_n:D_n\}$	*entête de t*
$def(t, a_i) = D_i$	*type de l'attribut a_i de t*
$val(t) = \{(a_1,v_1), (a_2,v_2),\ldots (a_n,v_n)\}$	*valeur de t*
$val(t, a_i) = v_i$	*valeur de l'attribut a_i de t*
$deg(t) = n$	*degré de t*
$id(t) = \{a_1, a_2,\ldots, a_n\}$	*ensemble des identifiants d'attributs de t*

…

La dénotation des attributs au sein d'un tuple varie beaucoup : la notation pointée « t.a », fonctionnelle « a(t) » ou TD « a FROM t » ?

Il est important, en pratique et du point de vue du génie logiciel, de différencier syntaxiquement le constructeur de types du constructeur de valeurs, ce que Tutorial D ne fait pas de façon assez explicite. Deux solutions sont généralement utilisées, utiliser des mots-clés différents (par exemple, TUPLE et TUP) ou utiliser les parenthèses différentes : TUPLE {} et TUPLE []. Discipulus a adopté cette dernière solution… pour le moment.

2.6 Relation

Une relation, en tant qu'objet de la théorie relationnelle, est un ensemble de tuples. En regard de la théorie des types, il s'agit d'un constructeur de type non scalaire applicable à un ensemble de tuples de même type.

Soit a_i des identifiants distincts, D_j des types et t_k des tuples, une relation R est définie comme suit :

$R \triangleq (\{a_1:D_1, a_2:D_2,\ldots, a_n:D_n\} ; \{t_1, t_2,\ldots, t_m\})$

avec $\forall i : 1 \leq i \leq card(R) \Longrightarrow def(R)=def(t_i)$

où

$def(R) = \{a_1:D_1, a_2:D_2,\ldots, a_n:D_n\}$	*entête de R*
$def(R, a_i) = D_i$	*type de l'attribut a_i de R*

[1] La composition est ici une extension aux attributs du produit cartésien défini sur les ensembles.

$val(R) = \{t_1, t_2, \ldots, t_m\}$	*valeur de R*
$deg(R) = n$	*degré de R*
$card(R) = m$	*cardinalité de R*
$id(R) = \{a_1, a_2, \ldots, a_n\}$	*ensemble des identifiants d'attributs de R*

…

Il est important, en pratique et du point de vue du génie logiciel, de différencier syntaxiquement le constructeur de types du constructeur de valeurs, ce que Tutorial D ne fait pas de façon assez explicite. Deux solutions sont généralement utilisées, utiliser des mots-clés différents (par exemple, RELATION et REL) ou utiliser les parenthèses différentes (par exemple, RELATION {} et RELATION []). Discipulus a adopté cette dernière solution… pour le moment.

2.7 Bases (banques) [de données]

Une base, en tant qu'objet de la théorie relationnelle, est une « composition[2] » de relations. En regard de la théorie des types, il s'agit d'un constructeur de type non scalaire applicable à un ensemble de relations.

Soit v_i des identifiants distincts, D_j des types de relation et r_k des (valeurs de) relations, une base (banque) B est définie comme suit :

$B \triangleq (\{v_1:D_1, v_2:D_2, \ldots, v_n:D_n\} ; \{r_1, r_2, \ldots, r_m\})$

avec $\forall i : 1 \leq i \leq card(B) \Longrightarrow def(B, v_i)=def(r_i)$

Où

$def(B) = \{ v_1:D_1, v_2:D_2, \ldots, v_n:D_n \}$	*entête de B*
$def(B, a_i) = D_i$	*type de ai de B*
$val(B) = \{r_1, r_2, \ldots, r_m\}$	*valeur de B*
$deg(B) = n$	*degré de B*
$id(B) = \{v_1, v2, ..., v_n\}$	*ensemble des identifiants de B*

…

3 Opérateurs relationnels

L'algèbre relationnelle est souvent présentée à l'aide de six opérateurs de base :

- ◊ renommage, R ρ a:b : la relation comprenant tous les tuples formés à partir d'un tuple de R dont l'attribut de nom a est remplacé par un attribut de nom b de même valeur, et rien d'autre ;
- ◊ restriction, R σ c : la relation comprenant tous les tuples de R satisfaisant la condition c, et rien d'autre ;
- ◊ projection, R π x : la relation comprenant tous les tuples formés à partir d'un tuple de R dont seuls les attributs dont le nom est parmi x ont été conservés, et rien d'autre ;
- ◊ jointure, R ⋈ S : la relation comprenant tous les tuples formés à partir d'un tuple de R et d'un tuple de S dont les attributs de même nom sont de même valeur, et rien d'autre ;
- ◊ union, R ∪ S : la relation comprenant tous les tuples de R et tous les tuples de S, et rien d'autre ;
- ◊ différence, R – S : la relation comprenant tous les tuples de R qui ne sont pas dans S, et rien d'autre.

[2] La composition est ici une extension aux relations du produit cartésien défini sur les ensembles.

Note sur le renommage

Le statut de l'opérateur de renommage est encore discuté. (1) Il est possible de s'en passer si on intègre la structure de catalogue à la théorie relationnelle, puisqu'il peut être exprimé à l'aide des autres opérateurs appliqués aux variables de relations appropriées du catalogue (dont la relation définissant les attributs d'une relation). (2) Il peut être considéré comme un simple artifice syntaxique, dans la mesure où seules les valeurs statiques d'identifiants sont permises (c'est-à-dire que la valeur de l'opérande représentant l'identifiant ne nécessite pas l'évaluation d'une variable). (3) Il peut être remplacé par un opérateur plus général, non relationnel, appartenant à la logique du premier ordre : le quantificateur existentiel. Le prix à payer est une complexification de la théorie et des modèles qui en découlent (1) ou une complexification des expressions relationnelles (1, 2, 3). Pour cela, il est usuel de le maintenir dans les six opérateurs de base.

Note sur la projection

Le langage Tutorial D dénote la projection par la juxtaposition d'une (dénotation de) relation et d'une liste (de dénotation) d'attributs :

$$R\ \{a_1,\ldots, a_n\} \equiv R\ \pi\ \{a_1,\ldots, a_n\}$$

Cet usage peut être vu comme analogue à la dénotation de la multiplication en arithmétique par juxtaposition des deux opérandes. Par souci de lisibilité, nous éviterons le plus souvent d'utiliser ce raccourci notionnel.

Note sur le produit cartésien

Dans ses premières publications, Codd utilisait un ensemble d'opérateurs de base comprenant le produit cartésien et ne comprenant pas la jointure. Sachant que le produit cartésien est un cas particulier de la jointure et que la jointure peut être exprimée en combinant le produit cartésien et la restriction, nous préférons la version présentée ici pour des raisons que nous exposerons bientôt.

Note sur l'algèbre minimale A

On peut réduire l'ensemble des opérateurs de base (c'est-à-dire le noyau minimal) aux seuls opérateurs suivants : ⟪NAND⟫ et ⟪REMOVE⟫, voir

https://www.dcs.warwick.ac.uk/~hugh/TTM/APPXA.pdf

Plan de la section

Les prochaines sous-sections présentent la définition formelle des opérateurs de base (3.1), puis la construction, à partir de ceux-ci, d'opérateurs usuels (3.2), spécialisés (3.3) et de regroupement (3.4).

Note de LL

Une confusion certaine est encore manifeste dans la notation utilisée : algèbre relationnelle (notation fluctuante), Tutorial D (langage défini par Date, mais toujours en évolution), Discipulus (langage en cours de définition au GRIIS). J'essaierai au cours des prochains jours, avec votre aide, d'exprimer le contenu de la présente section dans chacune des trois notations, sans mélange.

Par ailleurs, bien que le prochain module présente la mise en correspondance avec SQL, le lecteur trouvera un traitement exhaustif de la mise en correspondance de SQL avec Tutorial D dans [5]

…

3.1 Opérateurs de base

Définition relativement à la théorie des ensembles et à la représentation précédente (au modèle, à la structure).

Les équivalences sont données…

Renommage, ρ, RENAME

R ρ a:b ≡ *R RENAME {a_i AS b}* ≡ **select a_1, ..., a_i as b, ..., a_n from R** ≡
ANTE a ∈ id(R) ⋀ b ∉ id(R) :
SOIT
E := def(R,a_i)
Z := def(R) - {a_i:E} ∪ {b:E}
:
(Z ; {(Z ; val(t) - {(a_i,val(t,a_i))} ∪ {(b,val(t,a_i))}) | t ∈ val(R)})

Projection, π

R π w ≡ *R {w_0, ..., w_n}* ≡ **select w_1, ..., w_n from R** ≡
ANTE w ⊆ id(R) :
SOIT
Z := {a:D | a ∈ id (R) ⋀ a ∈ w}
:
(Z ; {(Z ; {(a,v) | (a,v) ∈ val(t) ⋀ a ∈ w}) | t ∈ val(R)})

où w est une expression ayant pour valeur une liste d'identifiants d'attribut ; en Tutorial D et en SQL, la liste doit être explicite (une énumération d'identifiants d'attribut w_0,…, w_n).

Jointure, ⋈, JOIN

R ⋈ S ≡ *R JOIN S* ≡ **select * from R natural join S** ≡
SOIT
X := id(R) ∩ id(S)
:
ANTE ∀ a ∈ X : a:D ∈ def(R) ⋀ a:D ∈ def(S) :
SOIT
Z := def(R) ∪ def(S)
:
(Z ; {(Z ; val(t_1) ∪ val(t_2)) | t_1 ∈ val(R) ⋀ t_2 ∈ val(S) ⋀ (∀ a ∈ X : (a,v) ∈ val(t_1) ⋀ (a,v) ∈ val(t_2))})

Note (1) : la jointure naturelle n'est pas commutative en SQL en raison de l'ordonnancement des attributs dans un tuple, puisque le tuple est une liste ordonnée de valeur d'attributs et non un ensemble d'attributs.

Z := { a:Dom(def(R,a),def(S,a)) | a ∈ X } ∪ {a:D ∈ def(R) | a ∉ X } ∪ { a:D ∈ def(S) | a ∉ X }

Restriction, σ, WHERE

R σ c ≡ *R WHERE c* ≡ **select * from R where c** ≡
ANTE id(c) ⊆ id(R) :
(def(R) ; {t | t ∈ val(R) ⋀ c(t)})

où *c* est une condition (expression booléenne) et *id(c)* est l'ensemble des identifiants d'attribut utilisés par *c*.

Union, ∪, UNION

R ∪ S ≡ *R UNION S* ≡ **select * from R union select * from S** ≡
ANTE def(R) = def(S) :
(def(R) ; val(R) ∪ val(S))

L'antécédent doit être aménagé en cas de relaxation de la compatibilité.

Différence, −, MINUS

R − S ≡ *R MINUS S* ≡ **select * from R except select * from S** ≡
ANTE def(R) = def(S) :
(def(R) ; val(R) − val(S))

L'antécédent doit être aménagé en cas de relaxation de la compatibilité.

3.2 Opérateurs usuels

Définis en regard des opérateurs de base.

Intersection, ∩, INTERSECT

R ∩ S ≡ R INTERSECT S ≡ **select * from R intersect select * from S** ≡

ANTE def(R) = def(S) :
R ⋈ S

R INTERSECT S ≡
ANTE def(R) = def(S) :
R JOIN S

L'antécédent doit être aménagé en cas de relaxation de la compatibilité.

Produit, ×, TIMES

R × S ≡ R TIMES S ≡ ***select * from R cross join S*** *≡* ***select * from R, S*** *≡*
ANTE id(R) ∩ id(S) = ∅ :
R ⋈ S

R TIMES S ≡
ANTE id(R) ∩ id(S) = ∅ :
R JOIN S

L'antécédent doit être aménagé en cas de relaxation de la compatibilité.

Notre (2) : le produit cartésien n'est pas commutatif en SQL.

Semi-jointure, ⋉, MATCHING

R ⋉ S ≡ R MATCHING S ≡ ***select r_1, ..., r_n from R natural join S*** *≡*
(R ⋈ S) π id(R)

R MATCHING S ≡
(R JOIN S) {r_1, ..., r_n}

Semi-différence, ▷, NOT MATCHING

R ▷ S ≡ R NOT MATCHING S ≡
R − (R ⋉ S)

R NOT MATCHING S ≡
R MINUS (R MATCHING S)

Extension, ξ, `EXTEND`

R ξ a:f(r₁, ..., rₙ) ≡ R EXTEND a:f(r₁, ..., rₙ) ≡
ANTE {r₁, ..., rₙ} ⊆ id(R) ⋀ a ∉ id(R) :
SOIT
Z := def(R) ∪ {a:T}
F := (Z ; {(Z ; val(t) ∪ {(a, f(t.r₁,…, t.rₙ))}) | t ∈ val(R)})
:
F

EXTEND R : {a := f(r₁, ..., rₙ)} ≡
ANTE {r₁, ..., rₙ} ⊆ id(R) ⋀ a ∉ id(R) :
SOIT
Z := def(R) ∪ {a:T}
F := (Z ; {(Z ; val(t) ∪ {(a, f(t.r₁,…, t.rₙ))}) | t ∈ val(R)})
:
F

où T est le type (des valeurs résultant de l'évaluation) de la fonction f. La construction de relations en lieu et place des fonctions est un élément important dans la démonstration de la complétude de l'algèbre relationnelle et pour la minimisation des opérateurs du noyau (par exemple pour le renommage, la restriction et l'extension).

Exercice : retirer les opérateurs de renommage et de restriction des opérateurs de base à l'aide de ce mécanisme.

Image, ∘, `IMAGE_IN`

Relation représentant l'image du tuple t dans R :

R ∘ t ≡ R(t) ≡ IMAGE t IN R ≡
R ⋈ RELATION[t π (id(t) ∩ id(R))]

IMAGE_IN (R, t) ≡
SOIT
{x₁,…, xₙ} := id(t) ∩ id(R) :
R JOIN RELATION{t {x₁,…, xₙ}}

Exercice : faire le lien avec la clause GROUP l'instruction SELECT de SQL.

Image du tuple courant, !!, `!!`

!!R ≡
R ∘ TUPLE[*]

!!R ≡ IMAGE_IN (R) ≡
IMAGE_IN (R, TUPLE{})*

où TUPLE[*] (RESP. TUPLE{*}) est le tuple « courant » dans une « expression évaluant tous les tuples » ; en pratique, la condition de la restriction (WHERE), les expressions dans les affectations de l'extension (EXTEND).

Exercice : faire le lien avec la clause GROUP l'instruction SELECT de SQL.

3.3 Opérateurs spécialisés

Union exclusive, ⊍ (UNION_X), `XUNION`

```
R ⊍ S ≡ R UNION_X S
  ANTE def(R) = def(S) :
    (R – S) ∪ (S – R)

R XUNION S ≡
  ANTE def(R) = def(S) :
      (R MINUS S) UNION (S MINUS R)
```

Remarque : (R – S) ∪ (S – R) = (R ∪ S) – (R ∩ S)

Union disjointe, ⊎ (UNION_D), `D_UNION`

```
R ⊎ S ≡ R UNION_D S ≡
  ANTE (def(R) = def(S)) ∧ (R ∩ S = ∅) :
    R ∪ S

R D_UNION S ≡
  ANTE (def(R) = def(S)) ∧ (R ∩ S = ∅) :
      R UNION S
```

Remarque 1 : si (R ∩ S = ∅) alors (R – S) = R et (S – R) = S, donc (R – S) ∪ (S – R) = R ∪ S. On conclut que l'union disjointe donne le même résultat que l'union exclusive, mais impose un antécédent en plus. :-)

Remarque 2 : il y a une certaine confusion quant à l'appellation de cet opérateur. En effet, l'appellation « union disjointe » est souvent donnée comme synonyme de « somme disjointe » ou de « réunion disjointe », voir par exemple https://fr.wikipedia.org/wiki/Réunion_disjointe dont les symboles sont respectivement « + » ou « ⊔ ». Cet opérateur peut être défini comme suit : A ⊔ B = ({0} × A) ∪ ({1} × B). Or nous aurons besoin de ce dernier opérateur lorsque nous étendrons note modèle de typage afin d'inclure de l'union de types.

Différence inclusive, ⊖ (MINUS_I), `I_MINUS`

```
R ⊖ S ≡ R MINUS_I S ≡
  ANTE (def(R) = def(S)) ∧ (R ⊇ S) :
    R – S

R I_MINUS S ≡
  ANTE (def(R) = def(S)) ∧ (R ⊇ S) :
      R MINUS S
```

Composition, •, `COMPOSE`

```
R • S ≡ R COMPOSE S ≡
  (R ⋈ S) π ((id(R) – id(S)) ∪ (id(S) – id(R)))

R COMPOSE S ≡
  SOIT
    {x1,…, xn} := (id(R) – id(S)) ∪ (id(S) – id(R)) :
      (R JOIN S) {x1,…, xn}
```

Fermeture positive, FP, `TCLOSE`

FP R ≡
SOIT { {a, b} := id(R) } : — corolairement #def(R)=2
ANTE def(a)=def(b) :
SOIT {S := R ∪ (R ρ a : x • R ρ b : x)} :
SI R = S ALORS R SINON FP S

`TCLOSE R ≡`
SOIT { {a, b} := id(R) } : — corolairement #def(R)=2
ANTE def(a)=def(b) :
`SOIT {S := R UNION ((R RENAME {a AS x}) COMPOSE (R RENAME {b AS x}))} :`
`CASE R = S THEN R ELSE TCLOSE S END`

Fermeture transitive, ⋆

R⋆ ≡
SOIT { {a, b} := id(R) } : — corolairement #def(R)=2
ANTE def(a)=def(b) :
FP R ∪ (R π {a} × (R π {a} ρ a:b) }

3.4 Opérateurs de groupement

Groupement par tuple

WRAP

R WRAP w AS a ≡
ANTE w ⊆ id(R) ∧ a ∉ id(R) :
(R ξ {a := TUPLE [w1:w1, ..., wn:wn]) π \w

`R WRAP {w1, ..., wn} AS a ≡`
ANTE {w1, ..., wn} ⊆ id(R) ∧ a ∉ id(R) :
`(EXTEND R : {a := TUPLE {w1 w1, ..., wn wn}) {ALL BUT w1, ..., wn}`

UNWRAP

R UNWRAP a ≡
ANTE a ∈ id(R) ∧ « a *est de type* TUPLE » *∧ id(a) ∩ id(R) = ∅ :*
SOIT
id(def(a)) = {w1, ..., wn}
:
R ξ {w1 := a π {w1},..., wn := a π {wn}} π \{a}

`R UNWRAP a ≡`
ANTE a ∈ id(R) ∧ « a *est de type* TUPLE » ∧ id(a) ∩ id(R) = ∅ :
SOIT
id(def(a)) = {w1, ..., wn}
:
`(EXTEND R : {w1 := a {w1},..., wn := a {wn}}) {ALL BUT a}`

Rappel : le type d'un nouvel attribut est (déduit de) celui de l'expression associée.

Groupement par relation

GROUP

```
R GROUP w AS a ≡
 ANTE w ⊆ id(R) ∧ a ∉ id(R) :
   R π \w ξ {a := !!R}

R GROUP {w1, ..., wn} AS a ≡
 ANTE {w1, ..., wn} ⊆ id(R) ∧ a ∉ id(R) :
    EXTEND R {ALL BUT w1, ..., wn} : {a := !!R}
```

UNGROUP

```
R UNGROUP a ≡
 Laissé en exercice !

R UNGROUP a ≡
  Laissé en exercice !
```

Rappel : le type d'un nouvel attribut est (déduit de) celui de l'expression associée.

3.5 Opérateurs d'agrégation

…

3.6 Opérateurs Tutorial D déclassés

- ◊ division, DIVIDEBY…
- ◊ sommaire, SUMMARY…

3.7 Exercices

Conversion des sélections groupées de SQL

Rappel

Les fonctions d'agrégation en Tutorial D sont de la forme g(R,a), où R est relation ayant un attribut a dont le type est compatible avec la fonction d'agrégation g.

Exemple 1

L'énoncé SQL

```
SELECT a1, ..., an, g(a) as x
FROM R
GROUP BY a1, ..., an
```

devient

```
R π {a1, ..., an} ξ {x := g(!!R, a)}

EXTEND R {a1,…, an} : {x := g(!!R, a)}
```

Remarque

En Tutorial D et en Discipulus, il n'y a pas de limitation quant au type de la fonction g.

Références

[1] James F. Allen. 1983. Maintaining Knowledge About Temporal Intervals. *Commun ACM* 26, 11 (1983), 832–843. DOI:https://doi.org/10.1145/182.358434

[2] James F. Allen. 1991. Time and time again: The many ways to represent time. Retrieved July 27, 2016 from http://citeseerx.ist.psu.edu/viewdoc/citations;jsessionid=32CC39C34FFEC1BA82E3C8BEDE2101F9?doi=10.1.1.103.8212

[3] James F. Allen and George Ferguson. 1994. *Actions and Events in Interval Temporal Logic*. The University of Rochester, Computer Science Department.

[4] James F. Allen and Patrick J. Hayes. 1990. Moments and Points in an Interval-based Temporal Logic. *Comput Intell* 5, 4 (1990), 225–238. DOI:https://doi.org/10.1111/j.1467-8640.1989.tb00329.x

[5] Chris J. Date. 2015. *SQL and relational theory: how to write accurate SQL code* (3rd ed ed.). O'Reilly, Sebastopol, Calif.

[6] Chris J. Date, Hugh Darwen, and Nikos A. Lorentzos. 2014. *Time and Relational Theory: Temporal Databases in the Relational Model and SQL*. Morgan Kaufmann, Waltham, MA.