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 du groupe Μῆτις et du GRIIS.

Plan

Sommaire

Le module Introduction à la théorie relationnelle

Le présent module de cours a été rédigé dans le cadre de l'exploration du thème Modélisation de données par le groupe **Μη□τις**. 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 et du *mastère* européen. Ces modules sont destinés aux étudiants des champs disciplinaires de l'informatique, du génie logiciel ou de l'informatique de gestion. Nous espérons toutefois qu'ils puissent 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 2R

Historique des révisions

version	date	auteur	description
...	2020-09-04	XX	..
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

...

1.2 Évolution du document

Le présent document tire son origine ... Les principales versions et leur contenu sont décrits ci-après :

- ◊ version 1 (2019-05-05) : ...

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.
- ◊ Faire les tableaux d'équivalence Discipulus – TD – SQL

1.4 Contenu des sections

...

1.5 Conventions

...

2 Structure relationnelle

La structure ...

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

{}

#

Opérateurs usuels

$= \neq$

$\in \notin \ni \not\ni$

$- \cup \cap \times$

$\subset \supset \not\subset \not\supset \subseteq \supseteq$

Opérateurs moins usuels

$\uplus \cupdot \biguplus \overline{\cup} \barwedge \overline{\cap}$

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 base.

\ 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'attributs

...

2.2 Couple

Le couple, aussi appelé la paire, est la formation d'une nouvelle entité (ou élément) à partir de deux autres.

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

(;)

Opérateurs usuels

@1 @2

2.3 Domaines et types

- 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 souvent l'attribut a.

- Rappels :
  - Un domaine est un ensemble *fini* de valeurs (distinctes, dans le sens ou une même valeur ne peut appartenir à deux domaines distincts).
  - Un type est un ensemble de valeurs défini par un domaine et une contrainte (l'union et l'héritage de types ne sont pas pris en compte ici).

2.4 Tuples

- Soit a_i des identifiants distincts et D_j des types, un tuple t est défini comme suit :
  - $t \triangleq (\{a_1{:}D_1, a_2{:}D_2, ..., a_n{:}D_n\}; \{(a_1,v_1), (a_2,v_2), ... (a_n,v_n)\})$
  - avec $\forall\ i : 1 \leq i \leq deg(t) \Longrightarrow val(t, a_i) \in def(t, a_i)$
- Où
  - $def(t) = \{a_1{:}D_1, a_2{:}D_2, ..., a_n{:}D_n\}$ entête de t
  - $def(t, a_i) = D_i$ type de a_i
  - $val(t) = \{(a_1,v_1), (a_2,v_2), ... (a_n,v_n)\}$ valeur de t
  - $val(t, a_i) = v_i$ valeur de a_i
  - $deg(t) = n$ degré de t
  - $id(t) = \{a_1, a_2, ..., a_n\}$ les 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.

2.5 Relations

- Soit a_i des identifiants distincts, D_j des types et t_k des tuples, une relation R est définie comme suit :
  - $R \triangleq (\{a_1:D_1, a_2:D_2, ..., a_n:D_n\}; \{t_1, t_2, ..., t_m\})$
  - avec $\forall i : 1 \leq i \leq card(R) \Longrightarrow def(R)=def(t_i)$
- Où
  - $def(R) = \{a_1:D_1, a_2:D_2, ..., a_n:D_n\}$ entête de R
  - $def(R, a_i) = D_i$ type de a_i
  - $val(R) = \{t_1, t_2, ..., t_m\}$ valeur de R
  - $deg(R) = n$ degré de R
  - $card(R) = m$ cardinalité de R
  - $id(R) = \{a_1, a_2, ..., a_n\}$ 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.

3 Opérateurs relationnels

L'algèbre relationnelle est souvent présentée à partir 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.

Note sur la projection

La dénotation de la projection en Tutorial D par la juxtaposition de la relation et de la liste d'attributs

```
R {a1, ..., an}  ≡
  R π {a1, ..., an}
```

peut être vue comme analogue à la dénotation de la multiplication en arithmétique par juxtaposition des deux opérandes.

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 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 les opérateurs de base aux seuls opérateurs minimaux 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 AS b} ≡
 ANTE a ∈ id(R) ⋀ b ∉ id(R) :
  SOIT
   E := def(R,a)
   Z := def(R) - {a:E} ∪ {b:E}
  :
   (Z ; {(Z ; val(t) - {(a,val(t,a))} ∪ {(b,val(t,a))}) | t ∈ val(R)})
```

Projection, π

```
R π w ≡ R {w0, ..., wn} ≡
 ANTE w ⊆ {a | a:D ∈ def(R)} :
  SOIT
   Z := {a:D | a:D ∈ def(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, la liste doit être explicite (une énumération d'identifiants d'attribut w_0, ..., w_n).

Jointure, ⋈, JOIN

```
R ⋈ S ≡ R JOIN S ≡
 SOIT
  Z := def(R) ∪ def(S)
  x := id(R) ∩ id(S)
 :
  ANTE ∀ a ∈ x : a:D ∈ def(R) ⋀ a:D ∈ def(S) :
   (Z ; {(Z ; val(t1) ∪ val(t2)) | t1 ∈ val(R) ⋀ t2 ∈ val(S) ⋀ (∀ a ∈ x : (a,v) ∈ val(t1) ⋀ (a,v) ∈ val(t2))})
```

Restriction, σ, WHERE

```
R σ c ≡ 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 ≡
 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 ≡
 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 ≡
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 ≡
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.

Semi-jointure, ⋉, MATCHING

R ⋉ S ≡ R MATCHING S ≡
(R ⋈ S) π id(R)

R MATCHING S ≡
(R JOIN S) {r₁, ..., rₙ}

Exercice : faire le lien avec les OUTER JOIN de SQL.

Semi-différence, ⋋, NOT MATCHING

R ⋋ S ≡ R NOT MATCHING S ≡
R – (R ⋉ S)

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

Exercice : faire le lien avec les OUTER JOIN de SQL.

Extension, ξ, `EXTEND`

$R\ \xi\ a{:}f(r_1, ..., r_n) \equiv R\ EXTEND\ a{:}f(r_1, ..., r_n) \equiv$
$ANTE\ \{r_1, ..., r_n\} \subseteq id(R) \wedge a \notin id(R) :$
SOIT
Z := def(R) ∪ {a:T}
F := (Z ; {(Z ; val(t) ∪ {(a, f(t.r_1, ..., t.r_n))}) | t ∈ val(R)})
:
F

```
EXTEND R : {a := f(r1, ..., rn)} ≡
```
ANTE {r_1, ..., r_n} ⊆ id(R) ⋀ a ∉ id(R) :
SOIT
Z := def(R) ∪ {a:T}
F := (Z ; {(Z ; val(t) ∪ {(a, f(t.r_1, ..., t.r_n))}) | 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 :

$R \circ t \equiv R(t) \equiv IMAGE\ t\ IN\ R \equiv$
$R \bowtie RELATION[t\ \pi\ (id(t) \cap id(R))]$

```
IMAGE_IN (R, t) ≡
```
SOIT
{x_1, ..., x_n} := 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, !!, `!!`

$!!R \equiv$
$R \circ TUPLE[*]$

```
!!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 : 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. :-)

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
{x_1, ..., x_n} := (id(R) – id(S)) ∪ (id(S) – id(R)) :
`(R JOIN S) {`x_1`, ..., `x_n`}`

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 domaine 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.

4 Algèbre relationnelle et logique des prédicats

Intuition

* Les ontologies permettent d'établir les prédicats lors de la déclaration de la relvar.

* Il s'ensuit que toute évaluation d'expression relationnelle peut être caractérisée par un prédicat fondé sur l'ontologie.

Préambule

* Vocabulaire : [
  assertion ::=
    contrainte | invariant ;
  contrainte ::=
    porte sur une seule relvar (et ses attributs) ;
  invariant ::=
    porte sur plusieurs relvars (et leurs attributs),
    voire aucune (donc le contexte de la BD) ;
]
* Considérer seulement les contraintes (pas les invariants).

Déclaration

R (a1, ..., ak, ak+1, ..., ak+n) key K {a1, ..., ak} constraint C1, ..., Cj :
  cas particulier :
    pred(R) = P0(K) ⋀ Pak+1(K,ak+1) ⋀ ... ⋀ Pak+n(K,ak+n) ⋀ C1 ⋀ ... ⋀ Cj
  cas général :
    pred(R) = P0(K) ⋀ P1(a1, ..., ak+n) ⋀ C1 ⋀ ... ⋀ Cj

Renommage

pred(R **ρ** x:y)
* Remplacer le nom de variable libre x par le nom y.

Projection

Cas particulier :
  soit A(bi) ∈ {ak+1, ..., ak+n} :
    pred(
      R π {A(b1), ... ,A(bm)}
      key K {a0, ..., ak} constraint C1, ..., Cj
) =
      P0(K) ⋀ P[A(b1)](K,A(b1)) ⋀ ... ⋀ P[A(bm)](K,A(bm)) ⋀ C1 ⋀ ... ⋀ Cj
Cas général :
  soit A(bi) ∈ {a1, ..., ak+n} :
    pred(

R π {A(b1), ..., A(bl), A(bl+1), ..., A(bl+m)}
key L {A(b1), ..., A(bl)} constraint C1, ..., Cj
) =
P0'(L) ⋀ P1'(A(b1), ..., A(bl+m)) ⋀ C1 ⋀ ... ⋀ Cj

* En fait, il s'agit de "retirer" les prédicats dont au moins un attribut n'est plus présent.
* Remarquer que le cas particulier est toujours possible en 6FN.

Restriction

pred(R σ C) = pred(R) ⋀ C

Union

pred(R ∪ S) = pred(R) ⋁ pred(S)
pred(R ⊎ S) = (pred(R) ⋁ pred(S)) ⋀ ¬(pred(R) ⋀ pred(S))
= (pred(R) ⋁ pred(S)) ⋀ (¬pred(R) ⋁ ¬pred(S))
pred(R ⊎ S) = (pred(R) ⋀ ¬pred(S)) ⋁ (¬pred(R) ⋀ pred(S))
= (pred(R) ⋁ pred(S)) ⋀ (¬pred(R) ⋁ ¬pred(S))

Intersection

pred(R ∩ S) = pred(R) ⋀ pred(S)

Différence

pred(R - S) = pred(R) ⋀ ¬pred(S)
pred(R ⊖ S) = pred(R) ⋀ ¬pred(S) ∧ (R ⇒ S)
= pred(R) ⋀ ¬pred(S) ∧ (¬pred(R) ⋁ pred(S))

Jointure

pred(R ⋈ S) = pred(R) ⋀ pred(S) ⋀ (∀ a ∈ id(R) ∩ id(S) : val(R.a)=val(S.a))
* La notation de la quantification manque de formalisme
(confusion relation, tuple).
* Prendre en compte les antécédents puis le résultat de la jointure elle-même
pour éliminer la quantification.
* Montrer cohérence avec intersection et produit.

Retour sur la projection

* Sans perte de généralité, supposons que tout prédicat est sous FNC.
* Donc, dans le cas général, P0, P1, C1, ..., Cj sont eux-mêmes en FNC.
* Soit X0==P0, X1==P1, X2==C1, Xj+1==Cj et

LA une liste de noms attributs de R :

$$pred(R\ \pi\ LA) = (\bigwedge Xi\ [i:0..j+1] \mid id(Xi) \subseteq LA)$$

Rappel

FNC ::= TND ('∧' TND)* ;

TND ::= PAT ('∨' PAT)* ;

PAT ::= ['¬'] ID ['(' ID (',' ID)* ')'] ;

5 Algèbre relationnelle et « logique » des intervalles

...

5.1 Opérateurs relatifs aux intervalles

Les opérateurs de Allen (opérateurs scalaires sur les intervalles) sont définis dans [1–4].

5.2 Opérateurs relatifs aux relations ayant des attributs de type intervalle

Les opérateurs portant sur les relations ayant des attributs de type intervalles (opérateurs relationnels sur les intervalles : PACK, UNPACK et USING) sont définis dans [6].

6 Langages relationnels

6.1 Tutorial D

Voir https://www.dcs.warwick.ac.uk/~hugh/TTM/Tutorial%20D%20Grammar.pdf

6.2 Discipulus

Demandez à Maxime Routhier !

6.3 SQL

Voir le standard ISO/IEC 9075:2016 (disponible à la bibliothèque des sciences).

6.3.1 Équivalences

SELECT a1, .., ak, exp1 as ak+1, ... expm, as ak+m
FROM R1
WHERE cond1
GROUP BY a1, ..., ak
HAVING cond2
===
natural join
join on
join using
===

union, intersect, minus
===
rename

6.3.2 Règles de pratique

select distinct

ne pas reposer sur l'ordre des colonnes

pas de join natural

restreindre le join on à une conjonction d'égalités d'attributs entre les deux tables

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.