Projet ROSES
Programme MDCO – Edition 2007

Livrable no D2.2
Modèle et algèbre ROSES

Identification

Acronyme du projet	ROSES
Numéro d'identification de l'acte attributif	ANR-07-MDCO-011-01
Coordonnateur	Paris 6
Rédacteur (nom, téléphone, email)	Dan Vodislav et al. 0134252814 dan.vodislav@u-cergy.fr
No. et titre	D2.2 : Modèle et algèbre ROSES
Version	v0.1
Date de livraison prévue	Date / t0+12
Date de livraison	Février 2009 (t0+12)

Résumé

Ce livrable présente en première partie le modèle logique pour les flux ROSES (notions de temps, données et flux), ainsi que l'algèbre logique d'opérateurs sur flux (filtrage, map, union, jointure, etc.), qui permet d'exprimer des requêtes continues sur les flux ROSES. Sur la base du modèle logique, Le modèle logique permet d'établir des équivalences d'expressions algébriques utilisées à l'optimisation des requêtes – les principales équivalences sont également présentées dans cette première partie.

La seconde partie présente un modèle physique orienté événements, plus proche de l'implémentation et une algèbre physique qui permet de traduire dans ce modèle les opérateurs logiques. Les règles de réécriture des expressions de l'algèbre logique en algèbre physique sont également présentées. Enfin, cette partie spécifie une implémentation événementielle de l'algèbre physique, qui servira à la réalisation d'un évaluateur de requêtes continues dans le système ROSES.

Table des matières

Modèle logique de flux

Pourquoi un modèle logique ?

Nous avons défini un modèle logique des flux pour :

- pouvoir définir formellement la sémantique des opérateurs de composition de flux,
- pouvoir prouver des équivalences algébriques des opérateurs
- et ainsi pouvoir définir des règles de réécriture des plans logiques d'évaluation de requêtes et pouvoir réaliser des optimisations au niveau logique.

Notions

Le temps

Supposons un domaine infini $\mathbb{T}$ (pour temps) totalement ordonné, c'est-à-dire, muni d'une relation binaire $O_{\mathbb{T}} \subseteq \mathbb{T} \times \mathbb{T}$, tel que $O_{\mathbb{T}}$ est une relation d'ordre total sur $\mathbb{T}$:

- $O_{\mathbb{T}}$ est antisymétrique, transitive et totale :
  - **antisymétrie** : $\forall t_1, t_2 \in \mathbb{T} : t_1 \, O_{\mathbb{T}} \, t_2 \wedge t_2 \, O_{\mathbb{T}} \, t_1 \Rightarrow t_1 = t_2$
  - **transitivité** : $\forall t_1, t_2, t_3 \in \mathbb{T} : t_1 \, O_{\mathbb{T}} \, t_2 \wedge t_2 \, O_{\mathbb{T}} \, t_3 \Rightarrow t_1 \, O_{\mathbb{T}} \, t_3$
  - **totalité (ou linéarité)** : $\forall t_1, t_2 \in \mathbb{T}$: soit $t_1 \, O_{\mathbb{T}} \, t_2$, soit $t_2 \, O_{\mathbb{T}} \, t_1$
- $O_{\mathbb{T}}$ est discrète :
  $\forall t_1, t_2 \in \mathbb{T}$, le nombre d'éléments (t) appartenant à $\mathbb{T}$, tels que $t_1 \, O_{\mathbb{T}} \, t$ et $t \, O_{\mathbb{T}} \, t_2$, est fini

Nous n'imposons plus de restrictions sur $\mathbb{T}$ que celles-ci, ainsi $\mathbb{T}$ pourrait être l'ensemble des naturels $\mathbb{N}$ ou l'ensemble $\mathbb{DATETIME}$.

Nous n'imposons non plus des restrictions par rapport à la signification des éléments de $\mathbb{T}$, c'est-à-dire, un élément de $\mathbb{T}$ peut représenter un tic d'horloge ou l'arrivée d'un nouvel item dans le flux.

Nous définissons une durée sur $\mathbb{T}$ comme la distance entre deux éléments de $\mathbb{T}$.

Les données

Supposons maintenant un ensemble infini $\mathbb{I}$ (pour item), qui contient des items.

Un item est une donnée en format xml. Ainsi, pour des sources de données qui ne soient pas en format xml (une table relationnelle, une boîte de messagerie…), une fonction de transformation (xmlisation) de la source de données devrait être définie.

Il existe un item vide que nous représentons par λ. Nous notons l'union entre l'ensemble $\mathbb{I}$ et λ par $\mathbb{I}_{\lambda}$

$$\mathbb{I}_{\lambda} = \mathbb{I} \cup \{\lambda\}$$

Supposons également que nous avons un ensemble de fonctions définies sur l'ensemble $\mathbb{I}$. Il existe trois types de fonctions : observers, accessors et mutators.

Les fonctions observers sont des fonctions qui nous permettent vérifier des propriétés sur les items, par exemple, l'égalité entre deux items, ou si un item valide un certain prédicat. Ainsi, les observers sont des fonctions de la forme :

obs : $\mathbb{I} \rightarrow \mathbb{B}$ (la description de l'item contient le mot Obama ?) ou
obs : $\mathbb{I} \times \mathbb{I} \rightarrow \mathbb{B}$ (deux items sont égaux ?)

Les fonctions mutators (ou transformers) nous permettent modifier les items, par exemple, ajouter une nouvelle balise à un item, ou fusionner deux items en un de seul. Ainsi donc, il s'agit des fonctions :

mut : $\mathbb{I} \rightarrow \mathbb{I}$ ou
mut : $\mathbb{I} \times \mathbb{I} \rightarrow \mathbb{I}$

Exemples :

Etant donné un item sous la forme :

```
<item>
      <title>...</title>
      <link>...</link>
      <description>...</description>
      <pubDate>...</pubDate>
</item>
```

un exemple de mutator qui ajoute une balise avec la traduction du titre serait (syntaxe à la xquery) :

```
return
      <item>
            { $i/title, $i/link, $i/description, $i/pubDate }
            <title-fr>{ translate($i/title, "en", "fr") }</title-fr>
      </item>
```

ou un exemple de fonction mutator qui génère un seul item à partir de deux items :

```
return
      <item>
            { $i1/title, $i1/link, $i1/description, $i1/pubDate }
            <related-data>{ $i2/title, $i2/link }</related-data>
      </item>
```

Finalement, les fonctions accessors nous permettent accéder à un sous-ensemble des données contenues dans un item. Il s'agit du résultat obtenu par une expression xpath. Ces sont des fonctions :

acc : $\mathbb{I} \rightarrow \mathbb{I}$

Le flux

Nous définissons un flux logique F comme un sous-ensemble du produit cartésien entre $\mathbb{T}$ et $\mathbb{I}_\lambda$, tel que (1) pour tout $t \in \mathbb{T}$, l'ensemble des items tels que $(t, i) \in F$ est fini, et (2) si $(t, i) \in F$, $(t', i') \in F$ pour tout $t' \; O_{\mathbb{T}} \; t$:

$F \subseteq \mathbb{T} \times \mathbb{I}_\lambda$ tel que

- $\forall t \in \mathbb{T} : \{i \mid (t, i) \in F\}$ est fini et
- $\forall t \in \mathbb{T} : \exists i \in \mathbb{I}_\lambda : (t, i) \in F$

Exemple :

$F = \{\ldots(-2, \lambda), (-1, \lambda), (0, \lambda), (1, a), (1, b), (1, c), (2, a), (2, d), (3, \lambda), (4, \lambda), (5, e), (5, f), (6, \lambda), (7, \lambda)\ldots\}$

Nous représentons graphiquement les flux de la façon suivante :

$\mathbb{T}$	F
1	a, b, c
2	a, d
3	
4	
5	e, f
6	
7	

C'est pourquoi, nous pouvons aussi définir un flux logique comme suit :

$F \subseteq \mathbb{T} \times \wp(\mathbb{I})$ tel que

- $(t_1, I_1), (t_2, I_2) \in F \wedge t_1 = t_2 \Rightarrow I_1 = I_2$,
- $\forall (t, I) \in F : I$ est fini et
- $\forall t \in \mathbb{T} : \exists I \in \wp(\mathbb{I}) : (t, I) \in F$

Ainsi, pour l'exemple précédent :

$F = \{\ldots(-2, \{\}), (-1, \{\}), (0, \{\}), (1, \{a, b, c\}), (2, \{a, d\}), (3, \{\}), (4, \{\}), (5, \{e, f\}), (6, \{\}), (7, \{\})\ldots\}$

Nous appelons la première représentation comme plate (unnested) et la deuxième, imbriquée (nested).

Pour simplifier, nous pouvons définir un flux logique comme une fonction totale :

$F : \mathbb{T} \rightarrow \wp(\mathbb{I})$ tel que

- $\forall t \in \mathbb{T} : F(t)$ est fini

Nous avons deux opérations, nest et unnest, qui nous permettent passer d'une représentation à l'autre. Si $\mathbb{F}_u$ est un ensemble qui contient tous les flux plats et $\mathbb{F}_n$, tous les flux imbriqués, nous

définissons les opérations nest et unnest comme ci-dessous :

$$nest : \mathbb{F}_u \rightarrow \mathbb{F}_n$$
$$nest(F_u) = \{(t, I) \in \mathbb{T} \times \wp(\mathbb{I}) \mid I = \{i \in \mathbb{I} \mid (t, i) \in F_u\}\}$$

$$unnest : \mathbb{F}_n \rightarrow \mathbb{F}_u$$
$$unnest(F_n) = \{(t, i) \in \mathbb{T} \times \mathbb{I}_\lambda \mid ((t, I) \in F_n \wedge i \in I) \vee ((t, \{\}) \in F_n \wedge i = \lambda)\}$$

L'opération nest est l'opération inverse d'unnest et vice-versa :

$$nest^{-1}(F) = unnest(F)$$

Autrement dit, la composition de l'opération nest et unnest (ou unnest et nest) est idempotente :

$$nest(unnest(F)) = F$$
$$unnest(nest(F)) = F$$

Modèle sequence-based

Si le domaine $\mathbb{T} = \mathbb{N}$ et un flux est défini comme une fonction partielle :

$F : \mathbb{N} \rightarrow \mathbb{I}$ tel que

- $F(t)$ est défini $\wedge$ t' $O_{\mathbb{T}}$ t $\Rightarrow F(t')$ est défini

nous avons est-ce qu'on appelle un modèle sequence-based ou événementiel.

Dans ce cas, chaque item dans le flux est affecté à un $n \in \mathbb{N}$ différent. Ainsi donc, on connaît la position de chaque item dans le flux.

Modèle time-based

Dans le cas général, où plusieurs items peuvent être affectés à un même $t \in \mathbb{T}$, nous disons que nous avons un modèle time-based.

Dans ce cas, on ne connaît pas la position de tous les items, mais nous avons une position approximative des items dans le flux.

Opérateurs logiques

Nous définissons un opérateur logique comme une fonction totale :

$$op : \mathbb{F}_u^{\,n} \rightarrow \mathbb{F}_u \qquad \text{où, } n \in \{1, 2\}$$

c'est-à-dire, un opérateur peut avoir un ou deux flux en entrée mais un seul flux en sortie.

Nous représentons graphiquement un opérateur logique comme suit :

$\mathbb{T}$	F	op(F)
1	a, b, c	a', b', c'
2	d, e	d', e'
3	f, g, h, i	f', g', h', i'
4		

Ainsi, $\mathbb{T}$ représente le temps de présence d'un item dans le flux.

1. **Sélection ou filtrage**

$$\sigma_{pred}(F) = \{(t, i) \mid (t, i) \in F \wedge pred(i)\}$$

où,

$$pred : \mathbb{I}_\lambda \rightarrow \mathbb{B}$$

$\mathbb{T}$	F	$\sigma_{pred}(F)$
1	a, b, c	a, c
2	d, e	e
3	f, g, h, i	f, i
4		

Par exemple : on s'intéresse seulement aux items dont la page web référencée par son url contient les mots sarkozy et obama.

2. **Map, transformation ou projection**

$$\mu_{map}(F) = \{(t, i') \mid (t, i) \in F \wedge i' = map(i)\}$$

où,

$$map : \mathbb{I}_\lambda \rightarrow \mathbb{I}_\lambda$$

$\mathbb{T}$	F	$\mu_{map}(F)$
1	a, b, c	a', b', c'
2	d, e	d', e'
3	f, g, h, i	f', g', h', i'
4		

Par exemple, si un item contient un ensemble de paires attribut-valeur :

- enlever des paires attribut-valeur,
- changer le nom à certains attributs,
- ajouter des nouveaux paires contenant le résultat d'une expression arithmétique sur les valeurs d'autres attributs de l'item, ou
- traduire les valeurs de certains attributs.

3. **Union**

$$F_1 \cup F_2 = \{(t, i) \mid (t, i) \in F_1 \vee (t, i) \in F_2\}$$

$\mathbb{T}$	F_1	F_2	$F_1 \cup F_2$
1	a_1, b_1, c_1	a_2, b_2, c_2, d_2	$a_1, b_1, c_1, a_2, b_2, c_2, d_2$
2	d_1, e_1	e_2, f_2	d_1, e_1, e_2, f_2
3	f_1, g_1, h_1, i_1	g_2, h_2, i_2	$f_1, g_1, h_1, i_1, g_2, h_2, i_2$
4		j_2, k_2	j_2, k_2
5			

4. Fenêtrage

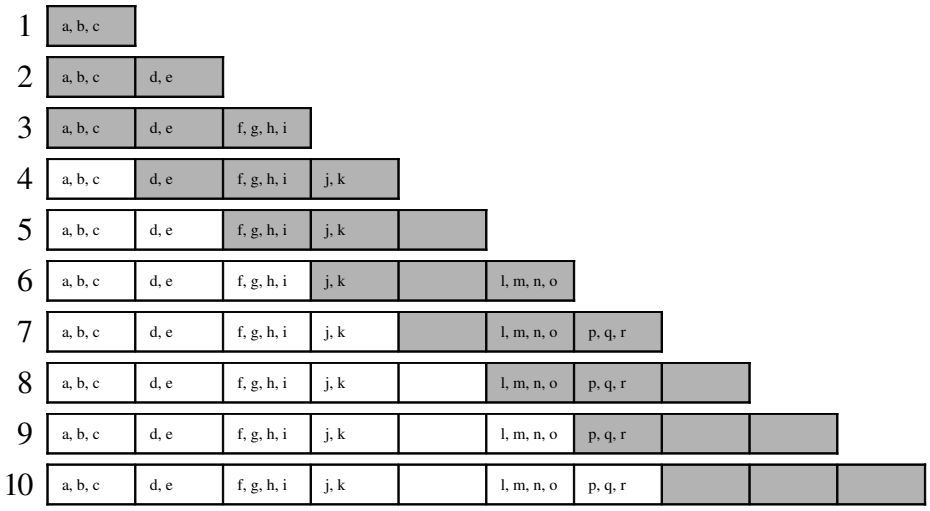
Nous avons deux types de fenêtres : des fenêtres time-based et des fenêtres count-based. Les fenêtres time-based nous permettent définir des fenêtres sur une certaine période de temps ; par exemple : tous les items publiés dans le flux pendant les dernières 24 heures. Tandis que, les fenêtres count-based (ou item-based) nous permettent définir des fenêtres sur un nombre déterminé d'items ; par exemple : les 25 derniers items publiés dans le flux (indépendamment de la date de publication).

4.1. Fenêtres time-based

$$\omega_{time,\,w}(F) = \{(t', i) \mid (t, i) \in F \wedge t \leq t' \leq t + w - 1\}$$

où, w est la taille de la fenêtre, c'est-à-dire, une durée sur $\mathbb{T}$.

Par exemple, si on définit une fenêtre de taille 3 sur le flux suivant, les zones sombrées représentent la fenêtre à chaque instant de temps :



$\mathbb{T}$	F	$\omega_{time,\,3}(F)$
1	a, b, c	a, b, c
2	d, e	a, b, c, d, e
3	f, g, h, i	a, b, c, d, e, f, g, h, i
4	j, k	d, e, f, g, h, i, j, k
5		f, g, h, i, j, k
6	l, m, n, o	j, k, l, m, n, o
7	p, q, r	l, m, n, o, p, q, r

8		l, m, n, o, p, q, r
9		p, q, r
10		

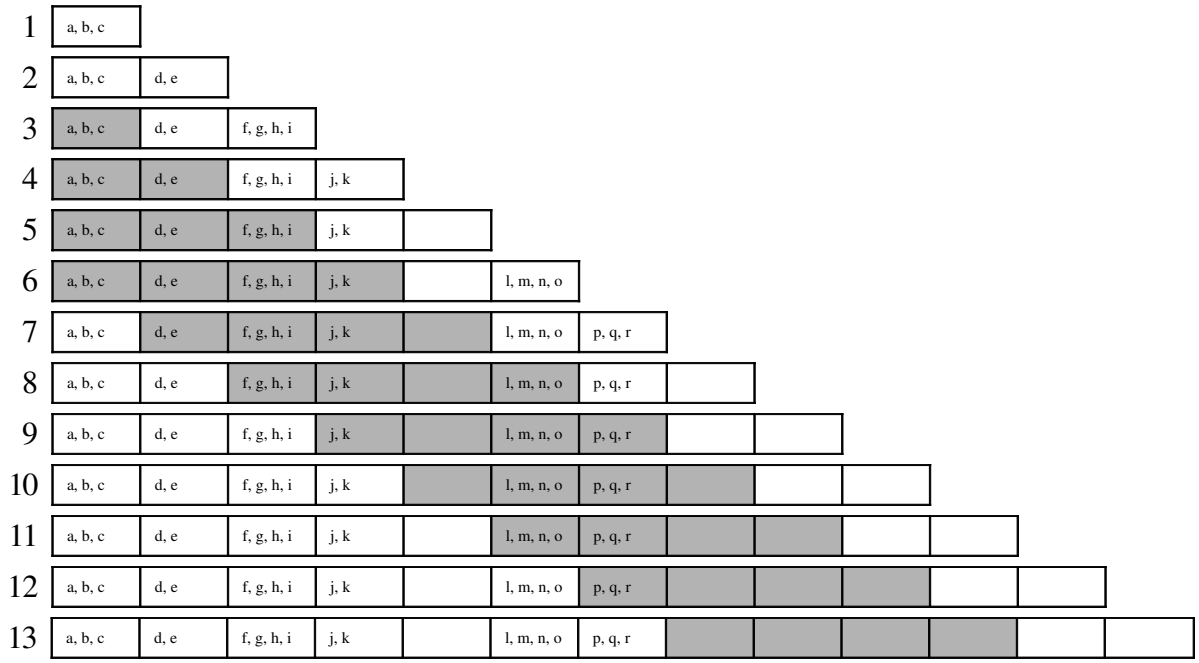
On appelle ce type de fenêtres des fenêtres sliding.

Décalage

Nous avons un autre paramètre d, qui nous permet définir le délai de la fenêtre par rapport à l'instant courant. Par exemple, si nous voulons une fenêtre avec les items publiés hier et avant-hier, mais pas aujourd'hui, nous définirons une fenêtre de taille 48 heures avec un délai de 24 heures.

$$\omega_{time,\, w,\, d}(F) = \{(t', i) \mid (t, i) \in F \wedge t + d \leq t' \leq t + d + w - 1\}$$

Fenêtre de taille 4 et délai 2 sur le flux F :



$\mathbb{T}$	F	$\omega_{time,\, 4,\, 2}(F)$
1	a, b, c	
2	d, e	
3	f, g, h, i	a, b, c
4	j, k	a, b, c, d, e
5		a, b, c, d, e, f, g, h, i
6	l, m, n, o	a, b, c, d, e, f, g, h, i, j, k
7	p, q, r	d, e, f, g, h, i, j, k
8		f, g, h, i, j, k, l, m, n, o
9		j, k, l, m, n, o, p, q, r
10		l, m, n, o, p, q, r
11		l, m, n, o, p, q, r
12		p, q, r
13		

Fréquence

Finalement, nous avons un paramètre f, qui nous permet définir la fréquence de rafraîchissement de la fenêtre, c'est-à-dire, chaque combien instants de temps on rafraîchit la fenêtre. Par exemple, nous pouvons créer une fenêtre de taille 24 heures qui se mette à jour chaque 6 heures.

$$\omega_{time,\, w,\, d,\, f,\, t0}(F) = \{(t', i) \mid (t, i) \in F \wedge f_s(d, f, t_0, t) \leq t' \leq f_e(w, d, f, t_0, t)\}$$

où,

- t_0 est l'instant de temps où la fenêtre a été calculée pour la première fois, normalement, l'instant où la requête dont la fenêtre fait partie a été lancée,
- $f_s(d, f, t_0, t)$ est l'instant de temps où un item publié à l'instant t dans le flux d'entrée apparaîtra dans la fenêtre et
- $f_e(w, d, f, t0, t)$ est le dernier instant de temps où un item publié à t sera dans la fenêtre.

$$f_s(d, f, t_0, t) = \begin{cases} t + d & \text{si } t_x = t \\ t_x + f + d & \text{si } t_x \neq t \end{cases}$$

$$t_x = \left[\frac{t - t_0}{f} \right] * f + t_0$$

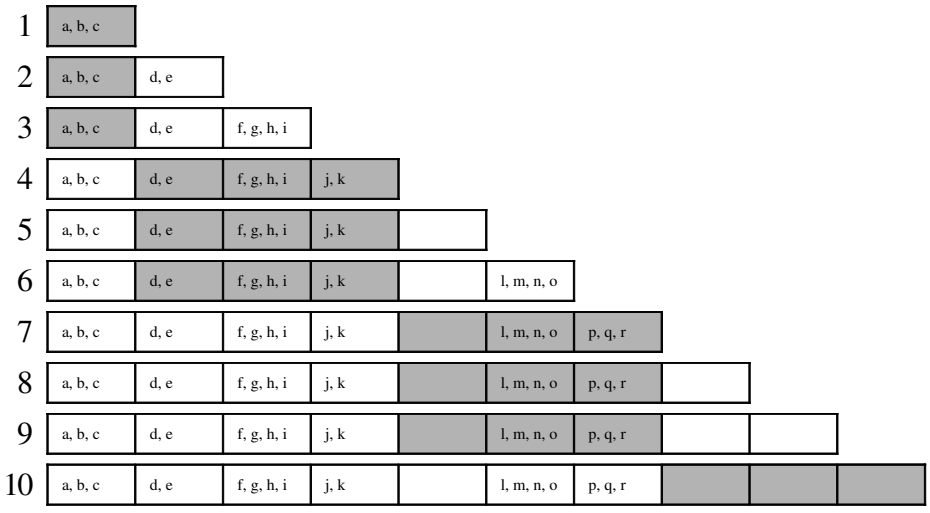
t_x est le dernier instant de temps où la fenêtre a été rafraîchie par rapport à t.

De façon alternative, nous pouvons définir f_s comme ci-dessous :

$$f_s(d, f, t_0, t) = \left[\frac{t - t_0}{f} \right] * f + t_0 + \left[\frac{f - t_0 + (t - t_0)\%f}{f} \right] * f + d$$

Exemples :

- Fenêtre de taille 3 et fréquence 3 sur le flux F

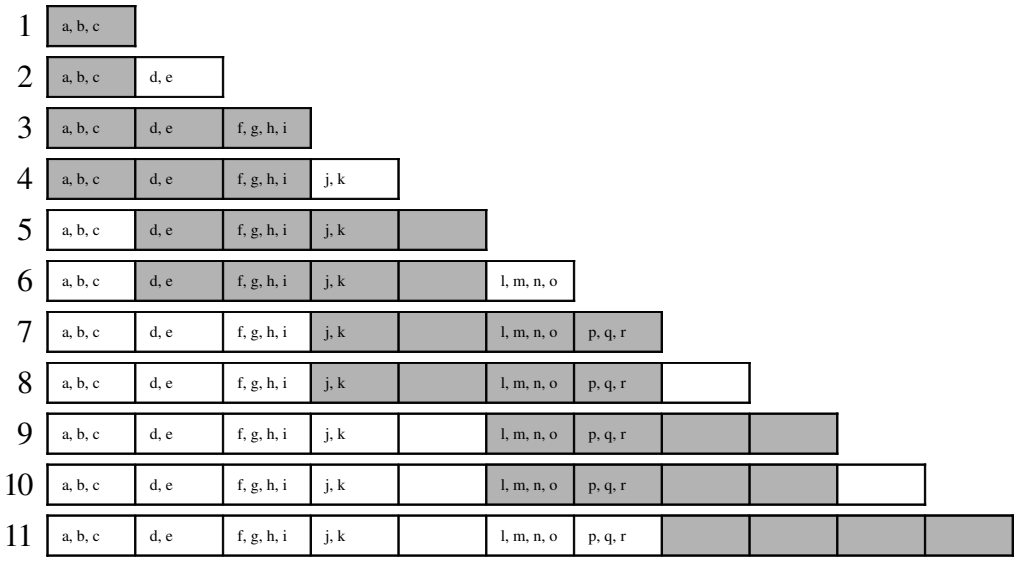


$\mathbb{T}$	F	$\omega_{time,\, 3,\, 0,\, 3,\, 1}(F)$
1	a, b, c	a, b, c

2	d, e	a, b, c
3	f, g, h, i	a, b, c
4	j, k	d, e, f, g, h, i, j, k
5		d, e, f, g, h, i, j, k
6	l, m, n, o	d, e, f, g, h, i, j, k
7	p, q, r	l, m, n, o, p, q, r
8		l, m, n, o, p, q, r
9		l, m, n, o, p, q, r
10		

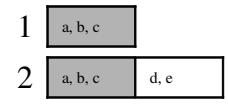
Ce type de fenêtres est appelé tumbling windows.

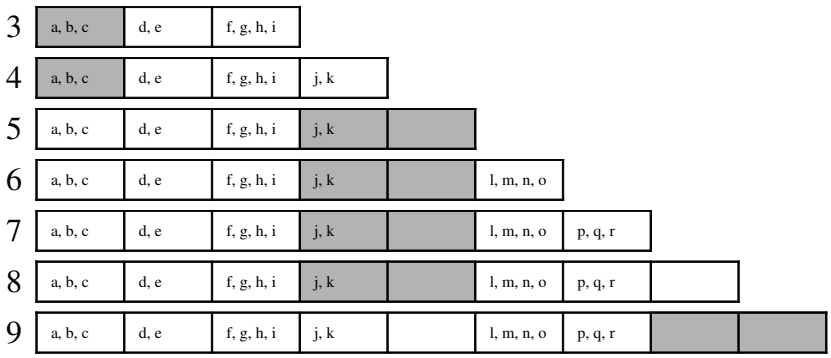
- Fenêtre de taille 4 et fréquence 2 sur F



$\mathbb{T}$	F	$\omega_{time, 4, 0, 2, 1}(F)$
1	a, b, c	a, b, c
2	d, e	a, b, c
3	f, g, h, i	a, b, c, d, e, f, g, h, i
4	j, k	a, b, c, d, e, f, g, h, i
5		d, e, f, g, h, i, j, k
6	l, m, n, o	d, e, f, g, h, i, j, k
7	p, q, r	j, k, l, m, n, o, p, q, r
8		j, k, l, m, n, o, p, q, r
9		l, m, n, o, p, q, r
10		l, m, n, o, p, q, r
11		

- Fenêtre de taille 2 et fréquence 4 sur F





$\mathbb{T}$	F	$\omega_{time,\ 2,\ 0,\ 4,\ 1}(F)$
1	a, b, c	a, b, c
2	d, e	a, b, c
3	f, g, h, i	a, b, c
4	j, k	a, b, c
5		j, k
6	l, m, n, o	j, k
7	p, q, r	j, k
8		j, k
9		

Ainsi, le rapport entre w et f nous permet définir trois types de fenêtres (par rapport à son contenu) :

- w > f : fenêtre sliding
- w = f : fenêtre tumbling
- w < f : fenêtre avec des trous

4.2. Fenêtres count-based ou item-based

$\omega_{count,\ N}(F) = \{(t', i) \mid (t, i) \in F \wedge t \leq t' \leq t_e \wedge$
$t_e = \max\{t_e' \in \mathbb{T} \mid |\{(t'',\ i'') \in F \mid t \leq t'' \leq t_e'\}| \leq N\} + 1\}$

$\mathbb{T}$	F	$\omega_{count,\ 8}(F)$
1	a, b, c	a, b, c
2	d, e	a, b, c, d, e
3	f, g, h, i	a, b, c, d, e, f, g, h, i
4	j, k	d, e, f, g, h, i, j, k
5		d, e, f, g, h, i, j, k
6	l, m, n, o	f, g, h, i, j, k, l, m, n, o
7	p, q, r	j, k, l, m, n, o, p, q, r
8		j, k, l, m, n, o, p, q, r
9		j, k, l, m, n, o, p, q, r

5. Produit cartésien

$F_1 \times F_2 = \{(t, i) \mid (t, i_1) \in F_1 \wedge (t, i_2) \in F_2 \wedge i = i_1 \cdot i_2\}$

où, · est une fonction de composition entre deux items :

$$\cdot : \mathbb{I}_\lambda \times \mathbb{I}_\lambda \to \mathbb{I}_\lambda$$

$\mathbb{T}$	F_1	F_2	$F_1 \times F_2$
1	a_1, b_1, c_1	a_2, b_2, c_2	$a_1 \cdot a_2, a_1 \cdot b_2, a_1 \cdot c_2, b_1 \cdot a_2, b_1 \cdot b_2, b_1 \cdot c_2, c_1 \cdot a_2, c_1 \cdot b_2, c_1 \cdot c_2$
2	d_1, e_1		d_1, e_1
3	f_1, g_1, h_1, i_1	d_2, e_2, f_2, g_2	$f_1 \cdot d_2, f_1 \cdot e_2, f_1 \cdot f_2, f_1 \cdot g_2, g_1 \cdot d_2, g_1 \cdot e_2, g_1 \cdot f_2, g_1 \cdot g_2, h_1 \cdot d_2, h_1 \cdot e_2, h_1 \cdot f_2, h_1 \cdot g_2, i_1 \cdot d_2, i_1 \cdot e_2, i_1 \cdot f_2, i_1 \cdot g_2$
4		h_2, i_2	h_2, i_2
5	j_1, k_1, l_1	j_2, k_2, l_2	$j_1 \cdot j_2, j_1 \cdot k_2, j_1 \cdot l_2, k_1 \cdot j_2, k_1 \cdot k_2, k_1 \cdot l_2, l_1 \cdot j_2, l_1 \cdot k_2, l_1 \cdot l_2$
6			

6. Jointure

6.1. Inner join

$$F_1 \bowtie_{\text{inner, pred}} F_2 = \{(t, i) \mid (t, i_1) \in F_1 \wedge (t, i_2) \in F_2 \wedge \text{pred}(i_1, i_2) \wedge i = i_1 \cdot i_2\}$$

où,

$$\text{pred} : \mathbb{I}_\lambda \times \mathbb{I}_\lambda \to \mathbb{B}$$

$\mathbb{T}$	F_1	F_2	$F_1 \bowtie_{\text{inner, pred}} F_2$
1	a_1, b_1, c_1	a_2, b_2, c_2	$a_1 \cdot c_2, b_1 \cdot a_2, b_1 \cdot b_2, c_1 \cdot a_2$
2	d_1, e_1		
3	f_1, g_1, h_1, i_1	d_2, e_2, f_2, g_2	$g_1 \cdot d_2, g_1 \cdot e_2, g_1 \cdot f_2, h_1 \cdot d_2, i_1 \cdot f_2$
4		h_2, i_2	
5	j_1, k_1, l_1	j_2, k_2, l_2	$j_1 \cdot j_2, j_1 \cdot k_2, l_1 \cdot k_2$
6			

6.2. Left outer join

$$F_1 \bowtie_{\text{left outer, pred}} F_2 = \{(t, i) \mid c_1 \vee c_2\}$$

$$c_1 = (t, i_1) \in F_1 \wedge (t, i_2) \in F_2 \wedge \text{pred}(i_1, i_2) \wedge i = i_1 \cdot i_2$$

$$c_2 = (t, i_1) \in F_1 \wedge (\forall (t, i_2) \in F_2 : \neg \text{pred}(i_1, i_2)) \wedge i = i_1$$

$\mathbb{T}$	F_1	F_2	$F_1 \bowtie_{\text{left outer, pred}} F_2$

1	a_1, b_1, c_1	a_2, b_2, c_2	$a_1 \cdot c_2$, $b_1 \cdot a_2$, $b_1 \cdot b_2$, $c_1 \cdot a_2$
2	d_1, e_1		**d_1, e_1**
3	f_1, g_1, h_1, i_1	d_2, e_2, f_2, g_2	**f_1**, $g_1 \cdot d_2$, $g_1 \cdot e_2$, $g_1 \cdot f_2$, $h_1 \cdot d_2$, $i_1 \cdot f_2$
4		h_2, i_2	
5	j_1, k_1, l_1	j_2, k_2, l_2	$j_1 \cdot j_2$, $j_1 \cdot k_2$, **k_1**, $l_1 \cdot k_2$
6			

6.3. Full outer join

$F_1 \bowtie_{\text{full outer, pred}} F_2 = \{(t, i) \mid c_1 \vee c_2 \vee c_3\}$

$c_1 = (t, i_1) \in F_1 \wedge (t, i_2) \in F_2 \wedge \text{pred}(i_1, i_2) \wedge i = i_1 \cdot i_2$

$c_2 = (t, i_1) \in F_1 \wedge (\forall (t, i_2) \in F_2 : \neg \text{pred}(i_1, i_2)) \wedge i = i_1$

$c_3 = (t, i_2) \in F_2 \wedge (\forall (t, i_1) \in F_1 : \neg \text{pred}(i_1, i_2)) \wedge i = i_2$

$\mathbb{T}$	F_1	F_2	$F_1 \bowtie_{\text{full outer, pred}} F_2$
1	a_1, b_1, c_1	a_2, b_2, c_2	$a_1 \cdot c_2$, $b_1 \cdot a_2$, $b_1 \cdot b_2$, $c_1 \cdot a_2$
2	d_1, e_1		d_1, e_1
3	f_1, g_1, h_1, i_1	d_2, e_2, f_2, g_2	f_1, $g_1 \cdot d_2$, $g_1 \cdot e_2$, $g_1 \cdot f_2$, $h_1 \cdot d_2$, $i_1 \cdot f_2$, **g_2**
4		h_2, i_2	**h_2, i_2**
5	j_1, k_1, l_1	j_2, k_2, l_2	$j_1 \cdot j_2$, $j_1 \cdot k_2$, k_1, $l_1 \cdot k_2$, **l_2**
6			

7. Delta

7.1. Delta insert

$\delta^+(F) = \{(t, i) \mid (t, i) \in F \wedge (t - 1, i) \notin F\}$

$\mathbb{T}$	F	$\delta^+(F)$
1	a, b, c	a, b, c
2	a, b, c, d, e	d, e
3	a, b, c, d, e, f, g, h, i	f, g, h, i
4	d, e, f, g, h, i, j, k	j, k
5	f, g, h, i, j, k	
6	j, k, l, m, n, o	l, m, n, o
7	l, m, n, o, p, q, r	p, q, r
8	l, m, n, o, p, q, r	
9	p, q, r	
10		

7.2. Delta delete

$\delta^{-}(F) = \{(t, i) \mid (t, i) \notin F \wedge (t - 1, i) \in F\}$

$\mathbb{T}$	F	$\delta^{-}(F)$
1	a, b, c	
2	a, b, c, d, e	
3	a, b, c, d, e, f, g, h, i	
4	d, e, f, g, h, i, j, k	a, b, c
5	f, g, h, i, j, k	d, e
6	j, k, l, m, n, o	f, g, h, i
7	l, m, n, o, p, q, r	j, k
8	l, m, n, o, p, q, r	
9	p, q, r	l, m, n, o
10		p, q, r
11		

8. Agrégation

$\alpha_{aggr}(F) = \{(t, i) \mid i = aggr(F(t))\}$

où,

$aggr : \wp(\mathbb{I}) \rightarrow \mathbb{I}_{\lambda}$

$\mathbb{T}$	F	$\alpha_{aggr}(F)$
1	a, b, c	j
2	d, e	k
3	f, g, h, i	l
4		

9. Group by

$\gamma_{group}(F) = \{(t, i) \mid i \in group(F(t))\}$

où,

$group : \wp(\mathbb{I}) \rightarrow \wp(\mathbb{I})$

$\mathbb{T}$	F	$\gamma_{group}(F)$
1	$a_1, a_2, a_3, b_1, b_2, c_1, c_2, c_3, c_4$	a, b, c
2	$d_1, d_2, e_1, e_2, e_3, e_4$	d, e
3	$f_1, f_2, f_3, g_1, g_2, h_1, h_2, h_3, h_4, h_5, i_1, i_2, i_3$	f, g, h, i
4		

L'opérateur de split

Pour pouvoir définir l'opérateur de split, nous avons besoin de pouvoir identifier les flux du système, ainsi donc, nous avons étendu le modèle logique.

Supposons un ensemble infini $\mathbb{M}$ (pour naMe), qui contient des noms. Nous définissons un flux nommé NF (named flow) comme une fonction partielle :

$$NF : \mathbb{M} \rightarrow \mathbb{F}_u$$

Si $\mathbb{NF}$ est l'ensemble qui contient tous les flux nommés, nous définissons l'opérateur de split comme suit :

$$\psi_{split}(NF) = \{(name', F') \in \mathbb{NF} \mid NF = (name, F) \wedge F' = \{(t, i) \in F \mid split(i) = name'\}\}$$

où,

$$split : \mathbb{I} \rightarrow \mathbb{M}$$

Ainsi,

$$\psi : \mathbb{NF} \rightarrow \wp(\mathbb{NF})$$

Exemples :

$$split(i) = \begin{cases} politique & si\ i.description\ contains\ ("sarkozy","royal","obama") \\ sport & si\ i.description\ contains\ ("lyon","psg","benzema") \\ autres & autrement \end{cases}$$

$$split(i) = pays(i)$$

où, pays(i) retourne le nom du pays en fonction des balises geo:lat et geo:long de l'item i

$\mathbb{T}$	NF	$\psi_{split}(NF)$		
	x	politique	sport	autres
1	a_p, b_p, c_p, d_s, e_s, f_s, g_s, h_a, i_a	a_p, b_p, c_p	d_s, e_s, f_s, g_s	h_a, i_a
2	j_p, k_p, l_s, m_s, n_s, o_s, p_s	j_p, k_p	l_s, m_s, n_s, o_s, p_s	
3	q_s, r_s, s_s, t_s, u_a, v_a, w_a		q_s r_s, s_s, t_s	u_a, v_a, w_a
4				

Le nombre de flux en sortie d'un opérateur split peut être fixe ou inconnu, ça dépend de la fonction de split. Par exemple, nous pouvons splitter un flux en trois flux comme dans la première fonction, mais nous pouvons aussi le splitter en fonction de la valeur d'un certain attribut des items, ainsi nous ne connaissons pas a priori combien des flux nous aurons, car il y aura autant des flux comme des valeurs différentes de l'attribut.

Ainsi, nous pouvons redéfinir un opérateur logique comme une fonction totale :

$$op : \mathbb{NF}^n \rightarrow \mathbb{NF}^m \qquad \text{où, } n \in \{1, 2\} \text{ et } m \in \mathbb{N}$$

c'est-à-dire, il peut avoir un ou deux flux en entrée et plusieurs flux en sortie.

Généralisation des opérateurs logiques

Comme nous avons vu un opérateur logique a la forme suivante :

$$op(F) = \{(t', i') \in \mathbb{T} \times \mathbb{I} \mid (t, i) \in F \wedge t' \in op_t(t) \wedge i' \in op_i(i)\}$$

Par exemple :

1. pour l'opérateur de sélection

   $\sigma_{pred}(F) = \{(t, i) \mid (t, i) \in F \wedge pred(i)\}$

   $$op_t(t) = \{t\}$$

   $$op_i(i) = \begin{cases} \{i\} & si\ pred(i) \\ \{\} & autrement \end{cases}$$

2. pour l'opérateur de map

   $\mu_{map}(F) = \{(t, i') \mid (t, i) \in F \wedge i' = map(i)\}$

   $$op_t(t) = \{t\}$$

   $$op_i(i) = \{map(i)\}$$

3. pour l'opérateur de fenêtrage

   $\omega_{time, w}(F) = \{(t', i) \mid (t, i) \in F \wedge t \leq t' \leq t + w - 1\}$

   $$op_t(t) = \{t' | t \leq t' \leq t + w - 1\}$$

   $$op_i(i) = \{i\}$$

Ainsi donc, si $op_t(t) = \{t\}$ le résultat de l'opérateur dépend seulement de la composante item des flux et nous appelons ces opérateurs indépendants du temps (sélection, map, union…).

Si, au contraire, $op_i(i) = \{i\}$, le résultat dépend juste de la composante temps et nous appelons ces opérateurs indépendants des items (fenêtrage…).

Snapshot-reducibility

Les opérateurs indépendants du temps ont des opérateurs analogues dans le model relationnel et nous pouvons les définir au moyen d'un langage de requêtes (Query Language) et en utilisant la représentation imbriquée des flux.

Ainsi, par exemple, nous pouvons définir l'opérateur de sélection comme ci-dessous :

$$\sigma_{pred}(F) = \{(t, I') \in \mathbb{T} \times \wp(\mathbb{I}) \mid (t, I) \in F \wedge I' = QL_{pred}(I)\}$$

où, $QL_{pred}(I)$ est l'ensemble d'n-uplets retournés par la requête suivante :

```
SELECT *
FROM I
WHERE pred
```

Un opérateur logique est snapshot-reducible si et seulement si pour tout $t \in \mathbb{T}$, l'ensemble d'items résultant d'appliquer son opérateur analogue relationnel sur l'ensemble d'items en entrée à l'instant t est égal à l'ensemble d'items à l'instant t résultant d'appliquer l'opérateur logique :

$$F' = op(F) \wedge \forall t \in \mathbb{T} : F'(t) = op_R(F(t)) \Leftrightarrow op \text{ est snapshot-reducible}$$

où, op_R est l'opérateur homologue d'op dans le modèle relationnel.

Tous les opérateurs indépendants du temps sont snapshot-reducibles.

Propriétés des opérateurs logiques

Sélection

$$F' = \sigma_{pred}(F) \Rightarrow F' \subseteq F$$

Commutativité : $\sigma_{pred1}(\sigma_{pred2}(F)) = \sigma_{pred2}(\sigma_{pred1}(F))$
Sélection en cascade : $\sigma_{pred1 \wedge \ldots \wedge predn}(F) = \sigma_{pred1}(\ldots\sigma_{predn}(F))$
Décomposition : $\sigma_{pred1 \vee \ldots \vee predn}(F) = \sigma_{pred1}(F) \cup \ldots \cup \sigma_{predn}(F)$

Map

Commutativité avec l'union : $\mu_{map}(F_1 \cup F_2) = \mu_{map}(F_1) \cup \mu_{map}(F_2)$

Union

$$F_1 \cup F_2 = F_1 \cup_{set} F_2$$

Commutativité : $F_1 \cup F_2 = F_2 \cup F_1$
Associativité : $(F_1 \cup F_2) \cup F_3 = F_1 \cup (F_2 \cup F_3)$
$F_1 \cup \ldots \cup F_n = (F_1 \cup \ldots) \cup F_n$

Fenêtrage

Commutativité avec la sélection : $\sigma_{pred}(\omega_{time,\ w,\ d,\ f,\ t0}(F)) = \omega_{time,\ w,\ d,\ f,\ t0}(\sigma_{pred}(F))$

mais, $\sigma_{pred}(\omega_{count,\ N}(F)) \neq \omega_{count,\ N}(\sigma_{pred}(F))$

Commutativité avec le map :

- $\omega_{time,\ w,\ d,\ f,\ t0}(\mu_{map}(F)) = \mu_{map}(\omega_{time,\ w,\ d,\ f,\ t0}(F))$
- $\omega_{count,\ N}(\mu_{map}(F)) = \mu_{map}(\omega_{count,\ N}(F))$

Distributivité avec l'union : $\omega_{time,\ w,\ d,\ f,\ t0}(F_1 \cup F_2) = \omega_{time,\ w,\ d,\ f,\ t0}(F_1) \cup \omega_{time,\ w,\ d,\ f,\ t0}(F_2)$

mais, $\omega_{count,\ N}(F_1 \cup F_2) \neq \omega_{count,\ N}(F_1) \cup \omega_{count,\ N}(F_2)$

Commutativité : $\omega_{time,\ w1}(\omega_{time,\ w2}(F)) = \omega_{time,\ w2}(\omega_{time,\ w1}(F))$

Fenêtrage en cascade : $\omega_{time,\ w1}(\omega_{time,\ w2}(F)) = \omega_{time,\ w1\ +\ w2\ -\ 1}(F)$

Décomposition :

- $\omega_{time,\ w}(F) = \omega_{time,\ w1}(F) \cup \ldots \cup \omega_{time,\ wn}(F)$ où, $w = \max\{w_1, \ldots, w_n\}$
- $\omega_{count,\ N}(F) = \omega_{count,\ N1}(F) \cup \ldots \cup \omega_{count,\ Nn}(F)$ où, $N = \max\{N_1, \ldots, N_n\}$

$\delta^+(\omega_{time,\ w}(F)) = \delta^+(F)$
$\delta^-(\omega_{time,\ w}(F)) = \delta^-(\omega_{time,\ w\ -\ 1}(\delta^-(F)))$

$\delta^+(\omega_{count,\ N}(F)) = \delta^+(F)$

$\delta^+(\omega_{time,\ w1}(F_1) \bowtie_{pred} \omega_{time,\ w2}(F_2)) = (\omega_{time,\ w1}(F_1) \bowtie_{pred} F_2) \cup (F_1 \bowtie_{pred} \omega_{time,\ w2}(F_2))$

Jointure

$$F_1 \bowtie_{inner,\ pred} F_2 = \sigma_{pred}(F_1 \times F_2)$$

Modèle et algèbre physiques

L'algèbre physique s'appuie sur un modèle de données plus proche de l'implémentation (comparé au modèle logique) et propose un ensemble d'opérateurs capables d'exprimer les opérateurs de l'algèbre logique.

La différence majeure entre le niveau physique et le niveau logique est que *les flux ne sont pas définis pour toutes les valeurs consécutives possibles des estampilles temporelles, mais seulement pour les estampilles où il y a au moins un item présent dans le flux.* Ceci correspond à une vision plus « physique » des flux, où le « traitement » des items est guidé par leur insertion dans le flux et non pas par les signaux d'une horloge.

Pour résumer, *le modèle physique est orienté événements*, tandis que *le modèle logique est orienté temps*.

Nous présentons ensuite une implémentation possible pour les opérateurs de l'algèbre physique basée sur un traitement événementiel (à la volée).

Une autre implémentation est également envisagée (mais pas présentée ici), basée sur le stockage des flux dans une base de données et des requêtes sur cette base déclenchées par les événements.

Notions

a. Temps

Au niveau physique, le temps est représenté, comme au niveau logique, par des *estampilles temporelles* utilisées pour annoter les données.

Def. ***Une estampille temporelle*** est une valeur dans un ensemble *TStamp* infini, discret, totalement ordonné et sans valeur maximale.

Def. ***Une durée*** est une valeur dans un ensemble infini et totalement ordonné *TDuration*.
La différence entre deux estampilles temporelles est une durée : si $ts1, ts2 \in TStamp$, $ts1 < ts2$, alors $ts2\text{-}ts1=d \in TDuration$. La valeur de durée notée 0 correspond à la différence entre deux estampilles identiques $ts\text{-}ts=0$.
La somme entre une estampille temporelle et une durée est une estampille temporelle :
si $ts1 \in TStamp$, $d \in TDuration$ alors $ts1+d=ts2 \in TStamp$ et $ts2 - ts1 = d$.
La différence entre une estampille temporelle et une durée est une estampille temporelle (si elle est définie) : si $ts1 \in TStamp$, $d \in TDuration$ et $\exists\ ts2 \in TStamp$ tel que $ts1 - ts2 = d$ alors $ts1 - d = ts2$, sinon $ts1 - d$ n'est pas défini.

On considère sur chaque site RoSeS un ***gestionnaire du temps*** qui produit des estampilles temporelles croissantes. Il offre une fonction

now : *void* ➠ *TStamp*

qui produit *l'estampille courante*, qui correspond au moment de l'appel de cette fonction.

Chaque site crée des flux d'items estampillés produits par un ***gestionnaire d'événements*** local, sur

la base d'événements détectés par celui-ci : mise à jour de flux RSS externe, notification de réception de mail, appel de service reçu de l'extérieur, etc. Tout item ainsi produit par le gestionnaire d'événements recevra une estampille produite par le gestionnaire du temps.

b. <u>Événements, séquences et flux d'événements, sources de données</u>

Def. ***Un événement*** de type *T* est représenté par un couple *e*=[*ts*, *data*], où *ts* est une estampille temporelle du et *data* est une donnée (item) de type *T* transportée par l'événement. Le type d'un événement de type *T* est noté *E(T)*.

Notation : Pour un événement *e*=[*t*, *d*], on note ***ts(e)***=*t* l'estampille temporelle de l'événement et ***item(e)***=d la donnée qui l'accompagne.

Def. ***Une séquence d'événements*** de type *T* est une séquence *eseq*, où chaque élément de la séquence est un événement de type *E(T)* et qui est *ordonnée en ordre croissant des estampilles temporelles*.

$eseq = <e \mid e \in E(T), \forall e1, e2 \in eseq, \; e1 < e2 \Rightarrow ts(e1) \leq ts(e2) >$

Le type d'une séquence d'événements de type *T* est < *E(T)* >.

Notation :

- Pour un événement $e \in eseq$, on note *pos(i)* l'entier représentant la position de l'item dans la séquence. La valeur de *pos(i)* est comprise entre 1 et *length(seq)*.
- Pour deux événements *e1*, *e2* ∈ *eseq*, on note *e1* < *e2* le fait que *e1* soit avant *e2* dans *eseq*, donc $e1 < e2 \Leftrightarrow pos(e1) < pos(e2)$. De manière similaire on définit les comparateurs ≤, > et ≥ entre événements d'une même séquence d'événements.

Def. ***Un flux d'événements*** de type *T* est une séquence d'événements *f* qui change dans le temps *par concaténation* de nouveaux événements.
On peut voir le flux d'événements comme une fonction

$f : TStamp \rightarrow <E(T)>$

où *f(t)* est la valeur de la séquence qui correspond au moment d'estampille *t*. Le type d'un flux d'événements de type *T* est *F(T)*.

Propriétés : a) $\forall\, t \in TStamp, \forall\, e \in f(t), ts(e) \leq t$ (les événements accumulés au moment *t* se sont produits avant ce moment).
b) $\forall\, t1 < t2$, *f(t1)* est un préfixe de *f(t2)* (accumulation par concaténation).

<u>Remarque</u> : On peut facilement passer de la représentation physique à celle logique d'un flux, en ignorant l'ordonnancement des événements de même estampille. Inversement, une représentation logique peut produire plusieurs représentations physiques en fonction de l'ordre choisi pour les événements d'une même estampille.

On peut définir une ***relation d'équivalence*** entre deux flux physiques *f1* et *f2* :

$f1 \equiv f2$ ssi les représentations logiques de deux flux sont identiques.

Def. ***Une source de données*** *ds*=[*src*, *q*, *T*] est définie par une base de données *src*, une requête *q* sur cette base et un type de données *T* produit par la requête. La source de données produit par exécution de la requête une séquence d'items de type *T*. Interrogée à des moments différents, la source produit des séquences d'items potentiellement différentes.

On note ***data(ds, t)***$\in$<*T*> la séquence d'items produite par la source de données *ds* interrogée au moment d'estampille *t*.
Soit ***DS*** l'ensemble de toutes les sources de données disponibles dans le système.

Def. ***Une séquence variable d'items*** est une séquence d'items dont la valeur peut être différente à des moments de temps différents. Plus généralement, une séquence variable d'items de type *T* est une fonction

$v : TStamp \rightarrow <T>$

qui associe à toute estampille temporelle une séquence d'items *v(t).*

Une séquence variable d'items permet de représenter le résultat d'un fenêtrage sur un flux ou le contenu d'une source de données *à n'importe quel moment*. Le type d'une séquence variable d'items de type *T* est noté *V(T)*.

c. Fenêtres

Au niveau physique, une fenêtre sur un flux définit *à tout moment* (donc pour une estampille temporelle donnée) une sous-séquence du flux dont on ne retient que les items, donc une *séquence d'items*. En conclusion, une fenêtre sur un flux produit une *séquence variable d'items*.

On garde la notation utilisée pour le fenêtrage dans le modèle logique ($\omega_{time,w}$, $\omega_{count,N}$, etc.), mais pour simplifier on va noter $\omega(f,t)$ la séquence d'items produite par la fenêtre ω sur le flux *f* au moment d'estampille *t* . Autrement dit, $\omega(f)$ est une séquence variable d'items et $\omega(f,t)$ est sa valeur au moment *t*.

Dans la suite, on considère que toutes les séquences d'items produites par fenêtrage préservent l'ordre des événements d'origine du flux auquel on applique la fenêtre.

- Fenêtre time-based :
  $\omega_{time,w}(f,t) = <item(e) \mid e \in f(t) \wedge$ (si *t-w* défini alors $ts(e) \in [t-w, t]$ sinon $ts(e) \leq t) >$
  Tous les items du flux dont l'estampille est entre *t-w* (s'il est défini) et *t*.

- Fenêtre count-based :
  $\omega_{count,N}(f, t) =$ si $length(f(t)) <$ N alors $f(t)$
  sinon $<e \in f(t) \mid e \geq min\{e' \in f(t) \mid ts(e')=ts(b), pos(b)=length(f(t))-N+1\}>$
  Les derniers *N* items du flux, plus tous ceux qui ont la même estampille que le premier de cette sous-séquence.

Algèbre physique

a. Opérateurs correspondant directement à des opérateurs logiques

- ***Filter*** $_{P}$: $F(T) \rightarrowtail F(T)$
  Sélection des événements *e* pour lesquelles le prédicat P est satisfait, P(*item(e)*)=true.

  Sémantique *: Filter* $_{P}(f) = f'$
  (contenu) $\forall t \in TStamp,\ e \in f(t),\ P(item(e))=true \Rightarrow e \in f'(t)$
  (ordre) $\forall e1,e2 \in f'(t)$ (donc $e1,e2 \in f(t)$), si $e1<e2$ dans $f(t)$ alors $e1<e2$ dans $f'(t)$

- ***Map*** $_{Tr}$: $F(T) \rightarrowtail F(T')$
  Transforme un flux en flux éventuellement d'un autre type, en appliquant la transformation *Tr* à tous les items du flux d'entrée.

  Sémantique *: Map* $_{Tr}(f) = f'$
  (contenu) $\forall t \in TStamp,\ e \in f(t) \Rightarrow [ts(e), Tr(item(e))] \in f'(t)$
  (ordre) $\forall e1,e2 \in f(t),\ e1<e2 \Rightarrow [ts(e1), Tr(item(e1))] < [ts(e2), Tr(item(e2))]$ dans f'

- ***Union*** : $F(T) \times \ldots \times F(T) \rightarrowtail F(T)$
  Union ordonnée par estampille de tous les événements des flux d'entrée.

  Sémantique *: Union* $(f_1, \ldots, f_n) = f$
  (contenu) $\forall t \in TStamp,\ e \in f_1(t) \cup \ldots \cup f_n(t) \Rightarrow e \in f(t)$
  (ordre) $\forall i \in \{1,\ldots,n\}$, $\forall e1,e2 \in f_i(t)$, si $e1<e2$ dans $f_i(t)$ alors $e1<e2$ dans $f(t)$
  $\forall i,j \in \{1,\ldots,n\},\ i \neq j,\ \forall e_i \in f_i(t),\ \forall e_j \in f_j(t),\ ts(e_i)=ts(e_j)$, alors l'ordre entre e_i et e_j dans le flux résultat dépend de l'implémentation du système

  Remarque : Les résultats possibles en fonction de l'ordre donné aux événements de même estampille venant de flux différents *sont équivalents* (même représentation logique)

- ***Delta***$^{+}$: $F(T) \rightarrowtail F(T)$
  Produit pour un flux donné un autre flux constitué des items nouveaux à chaque estampille par rapport aux items de l'estampille précédente du flux.

  Sémantique *: Delta*$^{+}(f) = f'$
  (contenu) $\forall t \in TStamp,\ e \in f(t)$,
  soit $fp = <e' \in f(t) |\ ts(e') < ts(e)>$ la séquence d'événements ayant des estampilles < $ts(e)$,
  alors si $fp = <>$ alors $e \in f'(t)$
  sinon soit $tp = ts(\text{last}(fp))$ la dernière estampille avant $ts(e)$
  alors si $\forall e' \in f(t),\ ts(e')=tp \Rightarrow item(e') \neq item(e)$ alors $e \in f'(t)$
  (ordre) $\forall e1,e2 \in f'(t)$ (donc $e1,e2 \in f(t)$), si $e1<e2$ dans $f(t)$ alors $e1<e2$ dans $f'(t)$

- ***Delta***$^{-}$: $F(T) \rightarrowtail F(T)$
  Produit pour un flux donné un autre flux constitué des items disparus à chaque estampille par rapport aux items de l'estampille précédente du flux.

  Sémantique *: Delta*$^{-}(f) = f'$
  (contenu) $\forall t \in TStamp,\ e \in f(t)$,
  soit $fp = <e' \in f(t) |\ ts(e') < ts(e)>$ la séquence d'événements ayant des estampilles < $ts(e)$,
  alors si $fp \neq <>$ et $tp = ts(\text{last}(fp))$ la dernière estampille avant $ts(e)$
  alors $\forall e' \in f'(t),\ ts(e')=tp$,

si $\forall e'' \in f(t), ts(e'')=ts(e) \Rightarrow item(e') \neq item(e'')$ alors $[ts(e), item(e')] \in f'(t)$

(ordre) $\forall$[ts, item($e1$)],[ts, item($e2$)]$\in f'(t)$ (donc $e1,e2 \in f(t)$), si $e1<e2$ dans $f(t)$ alors [ts, item($e1$)] < [ts, item($e2$)] dans $f'(t)$

b. **Opérateurs d'appel** (« pull »)

A la différence des opérateurs précédents, dont les résultats correspondent à des événements du (des) flux d'entrée (opérateurs « push »), ces opérateurs produisent leur résultat suite à un appel *à n'importe quel moment*. Ils produisent une séquence d'items à chaque appel, donc le résultat est une séquence variable d'items.

- ***WindowCall*** $_{\omega}$: $F(T) \twoheadrightarrow V(T)$
  Pour un flux d'événements et une fenêtre ω d'un des types présentés ci-dessus, retourne la séquence d'items produite par la fenêtre sur le flux au moment de l'appel (estampille produite par l'appel à *now()*).

  Sémantique : $WindowCall_{\omega}(f) = \omega(f)$ suivant le type de fenêtre, donc $WindowCall_{\omega}(f)$ appelé au moment d'estampille t produit $\omega(f, t)$.

- ***Data*** : $DS \twoheadrightarrow V(T)$
  Pour une source de données *ds* de type *T* et une estampille temporelle quelconque, produit une séquence d'items de type *T,* constituée des items résultats de la requête sur *ds* à ce moment.

  Sémantique : $Data(ds) = data(ds, now())$ au moment de l'appel.

c. **Autres opérateurs physiques**

- ***Nest*** : $F(T) \twoheadrightarrow F(<T>)$
  Regroupement de tous les items de même estampille pour produire un flux de séquences d'items (un événement de sortie par estampille distincte dans l'entrée). Correspond à l'opérateur logique **nest**, à la différence qu'il produit un flux de séquences et qu'il n'y a pas de séquence vide pour les estampilles absentes du flux d'entrée.

  Sémantique *: Nest* $(f) = f'$
  (contenu) $\forall t \in TStamp,$ soit $TS(t)=\{ts(e) | e \in f(t)\} \Rightarrow$
  $[ts, <item(e) | ts \in TS(t) \wedge e \in f(t) \wedge ts(e)=ts>] \in f'(t)$
  (ordre) Dans le flux il est donné par les estampilles, qui sont toutes distinctes. Dans les séquences d'items d'une estampille, il est donné par l'ordre des événements d'origine dans *f*

- ***Unnest*** : $F(<T>) \twoheadrightarrow F(T)$
  Inverse de *Nest*.

  Sémantique *: Unnest* $(f) = f'$
  (contenu) $\forall t \in TStamp, e=[ts, I] \in f(t) \Rightarrow \forall i \in I, [ts, i] \in f'(t)$
  (ordre) Pour une même estampille, l'ordre des items dans les séquences *I* est préservé.

- ***Window*** $_{\omega}$: $F(T) \twoheadrightarrow F(T)$
  Correspond au fenêtrage logique sur un flux, à la différence que la fenêtre n'est calculée que sur les événements du flux d'entrée. Peut être exprimé à l'aide de *WindowCall* (suite d'appels sur les

événements du flux d'entrée) et de *Unnest* (aplatissement des résultats dans le flux de sortie).

Sémantique : $Window_{\omega}(f) = f'$
(contenu) $\forall t \in TStamp,\ TS(t)=\{ts(e) |\ e \in f(t)\} \Rightarrow \forall ts \in TS(t),\ \forall e \in \omega(f)(ts),\ [ts, item(e)] \in f'(t)$
(ordre) Pour une même estampille *ts*, on garde l'ordre de $\omega(f)(ts)$.

- ***Semijoin*** $_{P}$: $F(T) \times V(T') \rightarrowtail F(T)$
  Semi-jointure entre un flux et une séquence variable d'items (résultat d'un fenêtrage ou d'un appel à une source de données) sur un autre flux. Le résultat est synchronisé sur le flux d'entrée. Le prédicat P s'applique à un couple d'items (ex. égalité/inégalité de titre/date/identifiant…).

  Sémantique : $Semijoin_{P}(f, v) = f'$
  (contenu) $\forall t \in TStamp,\ e \in f(t),\ \exists i \in v(ts(e)),\ P(item(e), i)=\text{true} \Rightarrow e \in f'(t)$
  (ordre) $\forall e1,e2 \in f'(t)$ (donc $e1,e2 \in f(t)$), si $e1<e2$ dans $f(t)$ alors $e1<e2$ dans $f'(t)$

- ***Join*** $_{P,Tr}$: $F(T) \times V(T') \rightarrowtail F(T'')$
  Jointure entre un flux et une séquence variable d'items – similaire à la semi-jointure, mais produit *des items composés* à partir des items du flux et de ceux de la séquence variable d'items.

  Sémantique : $Join_{P,Tr}(f, v) = f'$
  (contenu) $\forall t \in TStamp,\ e \in f(t),\ i \in v(ts(e)),\ P(item(e), i)=\text{true} \Rightarrow$
  $[ts(e), Tr(item(e), i)] \in f'(t)$
  (ordre) $\forall e1',e2' \in f'(t),\ e1'=[ts(e1), Tr(item(e1), i1)],$
  $e2'=[ts(e2), Tr(item(e2), i2)],\ e1,e2 \in f(t),\ i1,i2 \in v(t)$
  alors $e1'<e2' \Leftrightarrow e1<e2 \vee (e1=e2 \wedge i1<i2)$

- ***Diff*** $_{P}$: $F(T) \times V(T') \rightarrowtail F(T)$
  Différence (généralisée) entre un flux et le résultat d'un fenêtrage sur un autre flux. A l'opposé de la semi-jointure, la différence produit tous les événements du flux *qui ne peuvent pas être mis en correspondance* à travers le prédicat P avec les items de la séquence variable.

  Sémantique : $Diff_{P}(f, v) = f'$
  (contenu) $\forall t \in TStamp,\ e \in f(t),\ \forall i \in v(t),\ P(item(e), i)=\text{false} \Rightarrow e \in f'(t)$
  (ordre) $\forall e1,e2 \in f'(t)$ (donc $e1,e2 \in f(t)$), si $e1<e2$ dans $f(t)$ alors $e1<e2$ dans $f'(t)$

Réécriture de l'algèbre logique vers l'algèbre physique

Les opérateurs logiques ont chacun un correspondant direct dans l'algèbre physique à part le fenêtrage et la jointure.

Le fenêtrage logique a deux opérateurs disponibles au niveau physique:

Window: opérateur synchrone produisant un flux

WindowCall: opérateur asynchrone, produisant à chaque appel une séquence d'items – utilisé pour les jointures

On considère que la jointure logique se fait sur des flux après fenêtrage. Un flux logique quelconque peut être vu comme le résultat d'un fenêtrage avec une fenêtre time-based de durée 0, notons-là ω_0.

Réécriture de la jointure logique

- $\omega1(f1)$ ◗❖$_{\text{inner, P}}$ $\omega2(f2)$ ➠

$Union(\ Join_{P,\cdot}(Window_{\omega 1}(f1),\ WindowCall_{\omega 2}(f2)),$
$Join_{P,\cdot}(Window_{\omega 2}(f2),\ WindowCall_{\omega 1}(f1)))$

- $\omega 1(f1)$ ◗❖$_{\text{left outer, P}}$ $\omega 2(f2)$ ➠
  $Union(\ Join_{P,\cdot}(Window_{\omega 1}(f1),\ WindowCall_{\omega 2}(f2)),$
  $Join_{P,\cdot}(Window_{\omega 2}(f2),\ WindowCall_{\omega 1}(f1)),$
  $Diff_{P}(Window_{\omega 1}(f1),\ WindowCall_{\omega 2}(f2)))$

- $\omega 1(f1)$ ◗❖$_{\text{full outer, P}}$ $\omega 2(f2)$ ➠
  $Union(\ Join_{P,\cdot}(Window_{\omega 1}(f1),\ WindowCall_{\omega 2}(f2)),$
  $Join_{P,\cdot}(Window_{\omega 2}(f2),\ WindowCall_{\omega 1}(f1)),$
  $Diff_{P}(Window_{\omega 1}(f1),\ WindowCall_{\omega 2}(f2)),$
  $Diff_{P}(Window_{\omega 2}(f2),\ WindowCall_{\omega 1}(f1)))$

Remarque : $Window_{\omega 0}(f) = f$, donc pour un flux non fenêtré on peut remplacer tous les *Window* par le flux lui-même.

Implémentation événementielle

On considère que chaque opérateur réagit aux événements reçus dans les flux d'entrée (nouvel événement rajouté à la fin d'un flux).

- $Filter_{P}(f) = f'$
  on *f.e* **if** P(*item(f.e)*) **then produce** *f.e*

  Signification : Lorsque l'événement *e* se produit dans (est rajouté à) *f*, si P(*item(e)*) est vrai, alors *e* est aussi rajouté à *f'*.

- $Map_{Tr}(f) = f'$
  on *f.e* **produce** [*ts(f.e)*, Tr(*item(f.e)*)]

- $Union(f_1, \ldots, f_n) = f$
  on $f_1.e_1$ **produce** $f_1.e_1$
  …
  on $f_n.e_n$ **produce** $f_n.e_n$

- $WindowCall_{\omega}(f) = v$ (pour fenêtres alignées sur la fin du flux)
  On maintient W un objet séquence d'événements pour la fenêtre ω (initialisé à < >)
  Soit *fn = Nest(f)*
  on *fn.e*
  if some *i* **in** *item*(*fn.e*) **satisfies** [*ts*(*fn.e*), *i*] ∈ ω(*f*)(*ts*(*fn.e*)) **then**
  W = *concat* (W, *item*(*fn.e*))
  end if
  éliminer les événements « expirés » de W ;
  on call return $compute_{\omega}$(W)
  (sur appel on retourne une séquence calculée sur la base de W, par exemple pour une fenêtre time-based, des événements qui faisaient partie de W à la dernière mise-à-jour peuvent être « expirés » au moment de l'appel)

- *Nest* (*f*) = *f'*
  On maintient *C*, une séquence d'items venant d'événements de même estampille (initialisée à < >) et *tc* la plus grande estampille vue jusqu'ici dans le flux
  on *f.e*
    if *ts(f.e)>tc* **then**
      produce [*tc, C*]
      tc = ts(f.e)
      C = < >
    else *C = concat(C, item(f.e))*

- *Unnest* (*f*) = *f'*
  on *f.e* **foreach** *i* **in** *item(f.e)* **produce** [*ts(f.e), i*]

- *Window* $_{\omega}$ (*f*) = *f'*
  Soit *fn = Nest(f)*
  on *fn.e*
    I **=** *WindowCall* $_{\omega}$(*f*)(*ts(fn.e)*) //liste d'items produite par la fenêtre à ce moment
    foreach *i* **in** *I* **produce** [*ts(fn.e), i*]

- *Semijoin* $_{P}$ (*f*, *v*) = *f'*
  on *f.e* **if some** *i* **in** *v*(*ts(f.e)*) **satisfies** P(*item(f.e)*, *i*) **then produce** *f.e*

- *Join* $_{P,Tr}$ (*f*, *v*) = *f'*
  on *f.e* **foreach** *i* **in** *v*(*ts(f.e)*) **if** P(*item(f.e)*, *i*) **then produce** [*ts*(*f.e*), Tr(*item(f.e)*, *i*)]

- *Diff* $_{P}$ (*f*, *v*) = *f'*
  on *f.e* **if every** *i* **in** *v*(*ts(f.e)*) **satisfies** not P(*item(f.e)*, *i*) **then produce** *f.e*

- *Delta*$^{+}$ (*f*) = *f'*
  Soit *fn = Nest(f)*
  On maintient *L*, la dernière séquence d'items produite par *fn* (initialisée à < >)
  on *fn.e*
    produce [*ts(fn.e), item(fn.e)-L*] (différence entre la nouvelle et l'ancienne séquence)
    L = item(fn.e)

- *Delta*$^{-}$ (*f*) = *f'*
  Soit *fn = Nest(f)*
  On maintient *L*, la dernière séquence d'items produite par *fn* (initialisée à < >)
  on *fn.e*
    produce [*ts(fn.e), L-item(fn.e)*] (différence entre l'ancienne et la nouvelle séquence)
    L = item(fn.e)