MASTER EN SCIENCES INFORMATIQUES

Bases de Données Temporelles

Vermylen, Luc

Award date:
1992

Awarding institution:
Universite de Namur

Bases de Données Temporelles.

Luc VERMYLEN
Septembre 1992.

Mémoire de fin d'études de licence et maîtrise en informatique.

Promoteur: J.-L. Hainaut.

Remerciements:

J'aimerais remercier toutes les personnes qui ont collaboré de près ou de loin à la réalisation de ce mémoire.

J'aimerais également avoir une pensée pour mon père, qui nous a quitté inopinément ce 17 Juin 1992.

Il s'en est allé un beau matin d'été avec tous ses rêves, ses souhaits, ses projets inachevés.

Il nous aimait tous d'un amour infini.

Travailleur au courage sans fin, loyal et dévoué à tous.

Il avait toujours un mot gentil, un sourire, un bonjour pour chacun tout au long du chemin, tout au long de sa vie.

Il nous manque terriblement à tous.

Ce travail lui est dédié.

Introduction générale.

De tout temps, l'homme s'est attaché à rechercher ses origines. Depuis des siècles, anthropologues et autres scientifiques creusent, analysent les données recueillies sur notre planète ainsi qu'ailleurs afin de savoir d'où nous venons. Récemment encore, on mettait sur orbite le téléscope spatial HUBBLE qui, s'il avait bien fonctionné, aurait permis d'observer des galaxies bien plus éloignées que toutes celles connues à ce jour, et de rapporter quantités d'informations intéressantes sur le passé de l'univers, passé qui est aussi le nôtre à plus grande échelle. Plus proche de nous, notre histoire se trouve aussi cachée entre les différentes couches du sol sur lequel nous évoluons. Les archéologues parviennent à reconstituer des scènes de la vie de tous les jours sur base de fossiles et autres objets bien conservés découverts dans ce qu'ils appellent des sites archéologiques.

Ce besoin de retrouver des informations antérieures et de suivre l'évolution de ces informations au cours du temps touche aussi de près tout ceux qui utilisent des moyens informatiques notamment dans le but de pallier les déficiences de la mémoire humaine et à la lenteur de notre calculateur biologique.

C'est pourquoi, dans les systèmes d'information, on retrouve aussi le souci de "temporaliser" l'information. J'entends par là la volonté d'associer une notion de temps aux informations enregistrées dans ces systèmes. Cela permet notamment de replacer l'information dans son contexte et donc de mieux la comprendre ou l'interpréter.

Voici une vingtaine d'années, les premiers travaux sur la problématique du temps et systèmes d'information sont entamés. Les premières publications se rapportant au domaine datent des années 70. Depuis lors, on s'est forcé de poser les bases d'une théorie applicable aux bases de données. Malheureusement, les coûts de stockage étaient à l'époque très élevés, et les recherches dans ce domaine n'ont pas été des plus acharnées: peu de tentatives de réalisation concrètes ont été effectuées.

Les promesses venant du CD WORM ont fait revoir les positions sur la question. Actuellement, beaucoup de chercheurs se penchent sur le domaine et les premiers résultats concrets apparaissent cette fois au grand jour. Il s'agit essentiellement de langages d'interrogation temporels(1) et de bases de données incluant quelques fonction temporelles(2).

---

(1) et (2) *Le terme "temporel" n'a jusqu'ici aucune signification particulière mis à part le rapport au temps. Ce terme sera défini lorsqu'il prendra une signification précise.*

Objectifs du mémoire.

Les principaux travaux sur la notion de temps dans les bases de données se font sur

- ⇨ la nature du temps,
- ⇨ les différentes représentations du temps,
- ⇨ les diverses manières de raisonner sur celui-ci,
- ⇨ les extensions temporelles des modèles de données relationnels,
- ⇨ les extensions temporelles des langages d'interrogation de bases de données.

Le présent mémoire a deux objectifs principaux.

Le premier est de donner un aperçu de la situation actuelle des recherches en matière de temps lorsque celui-ci est pris en compte dans un système informatique, et plus particulièrement dans un système de gestion de bases de données.

La première partie du mémoire tentera de rencontrer cet objectif en abordant chacun des sujets évoqués ci-avant.

Le second objectif consiste à l'analyse d'un problème pratique rencontré par beaucoup d'entreprises. Ce problème est celui de la gestion informatisée des horaires et des pointages du personnel.

Plan.

Conformément aux objectifs, ce mémoire sera divisé essentiellement en deux grandes parties. La première partie sera une étude théorique. La seconde partie exposera un problème pratique lié à la gestion du temps dans un système informatique. Chacune de ces deux parties seront elles-mêmes divisées en plusieurs chapitres.

1. Première partie.

Cette première partie se composera de six chapitres abordant chacun des thèmes différents relatifs au temps.

Dans un premier temps, j'aborderai la notion de temps d'une manière axiomatique afin de mieux cerner les propriétés de ce temps que l'on ne sait ni palper ni facilement définir. Le but de ce premier chapitre n'est pas d'essayer de définir le temps. Les objectifs sont plutôt les suivants:

a) découvrir les propriétés du temps, propriétés qui peuvent être utiles à d'autres travaux en matière de temps et gestion d'informations au moyen d'un système informatique;

b) se munir d'un certain nombre de références lexicales avec lesquelles on pourra exposer les chapitres suivants de façon plus précise et plus claire;

c) se donner quelques points de repères concernant la manière dont on pourrait raisonner sur le temps si on était amené à gérer le temps dans un système informatique.

d) proposer différentes modélisations du temps pouvant être utilisées dans un système réel.

Dans le deuxième chapitre, je présenterai une **approche historique** qui se rapporte plus aux systèmes de gestion de bases de données multimédia ou orientées objets. D'un point de vue interdisciplinaire, cette approche est très intéressante étant donné les efforts consentis actuellement dans la recherche sur les bases de données orientées objets et multimédia.

Dans le chapitre 3, je présenterai à la fois une manière de caractériser les différentes approches du temps dans les bases de données, et **l'approche temporelle**. Cette caractérisation a l'avantage de pouvoir être représentée graphiquement. Elle permettra aussi d'introduire un vocabulaire plus précis.

Le quatrième chapitre apportera une dimension supplémentaire. Cette dimension est l'approche dynamique qui, combinée à une approche historique, ouvre de nouveaux horizons. Ce chapitre présentera également le modèle exploitant cette **approche combinée historique et dynamique**: le modèle REVI.

Le chapitre 5 présentera un des langages de query qui ont reçu une extension temporelle. Le langage présenté apparaît comme étant le plus complet des langages de ce type rencontrés dans la littérature à l'heure actuelle.

Le dernier chapitre apportera une conclusion de cette première partie sous forme de synthèse pratique. J'essaierai, à cet endroit, d'extraire ce qu'il faudrait retenir de ce qui précède "s'il ne fallait retenir que cela".

2. Deuxième partie.

La deuxième partie de ce mémoire procèdera à l'analyse du problème de gestion des horaires et des pointages du personnel. Cette partie sera subdivisée en sept sections.

La première section constituera une introduction; la seconde, un plan plus détaillé de la seconde partie.

La troisième section analysera le problème d'un point de vue pratique.

La quatrième section tentera de poser le problème en termes informatiques.

La cinquième section effectuera le lien avec la première partie de ce mémoire. Pour cela j'utiliserai certaines des bases théoriques introduites dans la première partie.

La sixième section comportera un ensemble de propositions visant à aider la résolution du problème exposé. Cette section fera aussi appel aux acquis théoriques de la première partie du mémoire. On y fera également l'hypothèse d'une implantation des solutions utilisant le modèle relationnel comme base et le langage SQL/400 comme outil de travail.

Table des matières

Chapitre 1.
Nature et représentation du temps.

Introduction.

Ce premier chapitre suggère qu'il existe une forme de savoir intuitif à propos du temps. En effet, la nature du temps qui passe ne semble pas être un sujet d'inquiétude pour les gens dans leurs préoccupations journalières.

Cette connaissance intuitive est de plus suffisamment riche et universelle pour permettre la communication et la coopération entre deux personnes quelconques sans qu'elles doivent se mettre d'accord sur une notion commune du temps.

Cette notion intuitive du temps est à la base de la construction de la théorie qui suit et que l'on doit à [ALLEN 87]. Cette théorie axiomatique, de par son indépendance vis-à-vis de la continuité ou de la discrétion du temps, est parfaitement adaptée pour supporter le raisonnement sur le temps commun, et la plupart des modèles.

1. Nature et composition du temps.

Intuitivement, on ne se rend pas compte que le temps n'est qu'une projection des évènements qui se déroulent et de leurs propriétés. En effet, si l'univers était figé, c'est-à-dire s'il n'y avait jamais un seul changement ou mouvement, il n'y aurait aucune notion de temps. En fait, tout évènement, toute modification est associé(e) à un "temps". L'Univers Temporel est constitué de l'ensemble des "temps".

Par la suite, et pour éviter les controverses sur les termes, on utilisera ici le terme neutre "période de temps" ou son raccourci: "période" à la place du terme "temps" d'un emploi malaisé.

Afin de mieux définir l'***Univers Temporel***, on peut dire qu'il est un ensemble d'éléments appelés périodes de temps, et dont chacun des éléments a la propriété d'avoir une durée positive (ou nulle). Il semble qu'il y ait plusieurs situations requérant de très courtes périodes. En fait, on peut distinguer deux types de périodes courtes: les ***instants*** et les ***points de temps***.

Un ***instant*** est simplement une période non décomposable. Durant un instant, rien ne peut changer. S'il y avait un changement observable durant cet instant, on pourrait le subdiviser en deux parties (avant et après le changement). Pour illustrer l'instant, on peut prendre l'exemple du stroboscope. A chaque éclair, le monde semble figé pour un bref instant. Cependant, un instant dans la vie de tous les jours

peut se révéler un processus très complexe à étudier pour un scientifique étudiant le mouvement des particules. Il n'y a donc pas de taille fixe attribuée aux instants.

Un ***point de temps*** est une période de durée nulle. On parlera de points de temps lorsqu'on raisonnera à propos des début et fin d'un évènement. Le début d'une course est le point de transition de l'état "avant le départ" à l'état "après le départ". De tels points ne peuvent avoir une durée, sinon il y aurait des "trous de vérité" (truth gaps), i.e., l'état de la course ne serait ni "avant le départ", ni "après le départ" durant une période de durée non nulle, ce qui est absurde.

2. Axiomes régissant le temps.

Considérons une classe non vide d'objets que l'on appellera périodes de temps. Ces périodes de temps sont supposées correspondre à notre notion de projection des évènements. A ce stade, on ne fait aucune hypothèse sur la décomposabilité ou non des périodes de temps. On ne fait pas non plus d'hypothèse sur la structure des périodes de temps (si elles sont continues, divisibles, ...). Le but recherché est uniquement de montrer que les périodes sont toutes contenues dans la ligne du temps (= représentation graphique de l'Univers Temporel).

La principale propriété des périodes de temps qui nous intéresse ici est que les périodes doivent pouvoir se rencontrer. Ce phénomène se produit lorsque 2 périodes ne partagent aucune période qu'elles auraient en commun et qu'elles ne sont pas non plus séparées par une tierce période de durée non nulle. Voici un exemple qui illustre cette propriété:

Soit une propriété **P**:

- ⇨ représentée par une fonction que l'on suppose partout définie sur l'axe du temps
- ⇨ ne pouvant avoir que deux valeurs distinctes: soit la valeur "vraie", soit la valeur "fausse".

Ceci revient à dire que:

- ⇨ son domaine est égal à l'axe du temps tout entier,
- ⇨ son domaine de valeurs est la paire {vrai, faux}.

Si **P** passe à un moment donné de la valeur "vraie" à la valeur "fausse", la période pendant laquelle P est vraie **rencontre** la période durant laquelle P est fausse. En effet, ces 2 périodes ne se chevauchent pas (auquel cas P serait simultanément vraie et fausse), et ne sont pas séparées par une autre période de durée non nulle (auquel cas il existerait un "trou de vérité" puisque par hypothèse P est partout définie).

On définit donc une relation **meets** (rencontre) entre les périodes, ainsi que les propriétés de cette relation. On représentera cette relation par le signe deux points ":".

De manière intuitive, on définit également les "**places de rencontre**", qui représentent la période de durée nulle entre deux périodes qui se rencontrent.

Remarques:

- Dans la suite de cet exposé, on utilisera la logique du premier ordre, et notamment, la notation (et relation) "=" (égalité), les opérateurs & (et), $\vee$ (ou), $\oplus$ (ou exclusif), $\Rightarrow$ (implication), $\equiv$ (équivalence), $\neg$ (négation), et les quantificateurs $\forall$ (universel) et $\exists$ (existentiel). La portée des opérateurs sera indiquée par l'usage des parenthèses.
- D'autre part, on considérera que les périodes de temps sont de durée non nulle sauf lorsqu'on les définira explicitement comme telles. La section 4 abordera la problématique des périodes de durée nulle (points de temps) plus en détail.

Le premier axiome est basé sur l'idée que la "place" où deux périodes se rencontrent est unique et fortement associée aux périodes de temps.

"Si deux périodes de temps rencontrent une troisième, alors toute période de temps rencontrée par l'une est aussi rencontrée par l'autre". On peut exprimer cela de manière plus formelle comme suit:

(M1) $\forall$ i,j,k,l, on a (i:j & i:k & l:j) $\Rightarrow$ l:k

Ensuite, on désire aussi que les "places de rencontre" (meeting places) soient totalement ordonnées [3], ce que l'on exprime ici:

"Si une période i rencontre une période j et si une période k rencontre une période l, alors une seule des trois affirmations suivante est vraie:

1°) i rencontre l;

2°) Il existe m tel que i rencontre m et m rencontre l;

3°) Il existe n tel que k rencontre n et n rencontre j".

(M2) $\forall$ i,j,k,l, on a (i:j & k:l) $\Rightarrow$ i:l $\oplus$ ($\exists$ m tel que i:m:l) $\oplus$ ($\exists$ n tel que k:n:j)

On a donc exactement trois cas possibles illustrés par la figure 1.1.

---

[3] *Note: de manière mathématique, l'ordre total s'exprimera sur l'axe du temps par la non-réflexivité de la relation "<", l'antisymétrie, la transitivité, et la possibilité de pouvoir comparer 2 places de rencontre quelconques par la relation "<"*

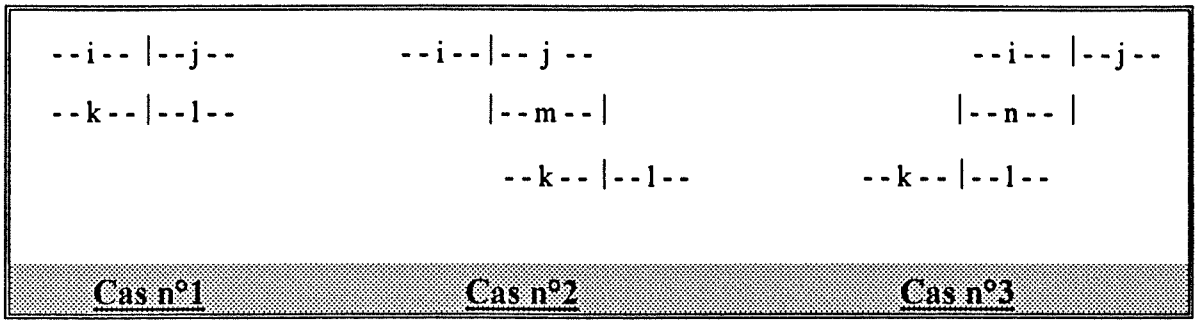


figure 1.1. Axiome M2.

Notons que si i:l, alors k:j (de par l'axiome M1). D'autre part, les "ou exclusifs" utilisés dans l'axiome M2 empêchent d'avoir des périodes circulaires. En effet, supposons que i se rencontre lui-même: alors on peut conclure que i:i et qu'il existe un m tel que i:m:i (car i:i:i). On aurait alors deux affirmations de l'axiome M2 vraies simultanément, ce qui est en contradiction avec cet axiome.

On obtient facilement les Lemmes suivants:

(ML1) $\forall$i, on a $\neg$(i:i)

(ML2) $\forall$i,j, on a i:j $\Rightarrow$ $\neg$(j:i)

(ML3) $\forall$i, on a $\neg$($\exists$m tel que i:m:i)

On a maintenant besoin de deux axiomes d'existence:

"Le temps ne peut ni commencer ni s'arrêter".

(M3) $\forall$i, $\exists$j,k tels que j:i:k

"La présence d'une période entre 2 places de rencontre est un fait de base: c'est la présence de tels intervalles [4] qui prouvent l'ordonnancement du temps.

Si deux places de rencontre sont séparées par une séquence d'intervalles, on doit pouvoir conclure que cette séquence constitue une période de durée plus longue."

(M4) $\forall$ i,j, on a i:j $\Rightarrow$ ($\exists$ k,m,n tels que m:i:j:n & m:k:n)

On peut noter que l'utilisation de temps auxiliaires (m et n) a pour but d'assurer que k et i commencent au même endroit et que k et j finissent au même endroit.

Le dernier axiome de la théorie de base des rencontres assure que la période séparant deux places de rencontre est unique:

(M5) $\forall$ i,j,k,l, on a (i:j:l & i:k:l) $\equiv$ j = k

Grâce à l'unicité apportée par l'axiome M5, on peut reformuler l'axiome M4 si on considère qu'il existe une fonction ayant pour argument deux périodes de temps adjacentes i et j (c'est-à-dire i:j), et donnant comme résultat la période k, plus longue. Cette fonction sera appelée l'union ordonnée ou somme de i et j et notée i+j. (M4) devient:

(M4.1) $\forall$ i,j, on a i:j $\Rightarrow$ ($\exists$ m,n tels que m:i:j:n & m:(i+j):n)

Il est évident que la fonction "+" est associative, ce qui permettra d'utiliser des expressions telles que i+j+k.

Ces axiomes concernant la relation rencontre sont maintenant suffisants pour pouvoir reconstruire la théorie des relations entre intervalles. En effet, on peut donner des définitions simples de toutes les relations entre intervalles en termes de rencontres.

Par exemple, on peut définir une relation BEFORE entre deux périodes (intervalles) simplement s'il existe une période qui se mesure entre eux:

i BEFORE j $\equiv$ $\exists$ k tel que i:k:j

---

[4] *Note: A partir d'ici, on emploiera également le terme "**intervalle**" pour désigner une période de temps de durée non nulle.*

Voici un tableau récapitulatif des relations, de leurs définitions, et des relations inverses (12 au total). On y a ajouté les notations abrégées.

Relation entre I & J	Définition	Relation inverse
BEFORE, b	∃k tel que I:k:J	AFTER, a
OVERLAPS, o	∃k, l, m tels que I = k + l & J = l + m	OVERLAPPED-BY, oi
STARTS, s	∃k tel que J = I + k	STARTED-BY, si
FINISHES, f	∃k tel que J = k + I	FINISHED-BY, fi
DURING, d	∃k, l tels que J = k + I + l	CONTAINS, di
MEETS, :	I:J	MET-BY, mi

figure 1.2. Relations entre intervalles.

Par exemple si I BEFORE J (relation entre I et J), alors J AFTER I (relation inverse: J par rapport à I).

La treizième relation est l'égalité. Si on ne considère pas l'égalité dans son acception logique, l'axiome M5 constitue sa définition.

On définit encore la relation IN qui résume les relations d'inclusion:

$$i \text{ IN } j \equiv i \text{ DURING } j \oplus i \text{ STARTS } j \oplus i \text{ FINISHES } j \oplus i = j$$

Par la suite, on utilisera aussi la notation abrégée de la disjonction introduite par [ALLEN 83]:

Par exemple,

j (o oi s f d) i est équivalent à j OVERLAPS i ⊕ j OVERLAPPED-BY i ⊕ j STARTS i ⊕ j FINISHES i ⊕ j DURING i.

Un opérateur d'intersection sur les périodes de temps est aussi utile. Soit I!J l'intersection de I et J. I!J est défini par les deux propriétés suivantes:

(I1) $\exists$ i tel que (i IN I) & (i IN J) $\Rightarrow$ (I!J IN I) & (I!J IN J)

"I!J est l'intervalle qui existe s'il existe un sous-intervalle commun à I et à J".

(I2) $\forall$ i on a (i IN I) & (i IN J) $\Rightarrow$ (i IN I!J)

"I!J est le plus grand sous-intervalle commun à I et à J".

Comme ces propriétés sont entièrement dérivables des axiomes M1 à M5, elles peuvent être considérées comme des axiomes de définition de cette notation.

On peut prouver que I!J est unique s'il existe (Lemme IL1), et qu'il existe quand I et J se recouvrent au sens intuitif du terme, i.e. I (o oi s si f fi d di =) J (Lemme IL2).

Le Lemme suivant (Lemme IL3) est aussi intéressant:

(IL3) "Si i = a + b et j:b, alors i!j existe et est égal à a!j.

Si i = a + b et a:j, alors i!j existe et est égal à b!j."

3. Comment raisonner en termes d'intervalles?

La question est maintenant de savoir si l'axiomatisation de la relation rencontre et des autres relations capture la logique des intervalles. C'est effectivement le cas, même s'il est fastidieux de le montrer. Avant cela, on va clarifier la signification de la table de transitivité donnée dans [ALLEN 83].

	b	a	d	di	o	oi	:	mi	s	si	f	fi
"before" b	b		b o : d s	b	b	b o : d s	b	b o : d s	b	b	b o : d s	b
"after" a		a	a oi mi d f	a	a oi mi d f	a	a oi mi d f	a	a oi mi d f	a	a	a
"during" d	b	a	d		b o : d s	a oi mi d f	b	a	d	a oi mi d f	d	b o : d s
"contains" di	b o : di fi	a oi di mi si	o oi d di =	di	o di fi	oi di si	o di fi	oi di si	di fi o	di	di si oi	di
"overlaps" o	b	a oi di mi si	o d s	b o : di fi	b o :	o oi d di =	b	oi di si	o	di fi o	d s o	b o :
"overlap-ped-by" oi	b o : di fi	a	oi d f	a oi mi di si	o oi d di =	a oi mi	o di fi	a	oi d f	oi a mi	oi	oi di si
"meets" :	b	a oi mi di si	o d s	b	b	o d s	b	f fi =	:	:	d s o	b
"met-by" mi	b o : di fi	a	oi d f	a	oi d f	a	s si =	a	d f oi	a	mi	mi
"starts" s	b	a	d	b o : di fi	b o :	oi d f	b	mi	s	s si =	d	b : o
"started-by" si	b o : di fi	a	oi d f	di	o di fi	oi	o di fi	mi	s si =	si	oi	di
"finishes" f	b	a	d	a oi mi di si	o d s	a oi mi	:	a	d	a oi mi	f	f fi =
"finished-by" fi	b	a oi mi di si	o d s	di	o	oi di si	:	si oi di	o	di	f fi =	fi

figure 1.3. Table de transitivité.

Considérons une entrée impliquant successivement la relation MEETS et la relation BEFORE. Le résultat est BEFORE, qui peut être lu dans la case correspondante de la table. On interprétera cela comme suit:

$\forall$a,b,c, on a (a MEETS b) & (b BEFORE c) $\Rightarrow$ a BEFORE c (*)

Les différentes entrées de la table ne sont pas définies par une équivalence logique ($\Leftrightarrow$, si et seulement si). En fait, forcer (*) à être une équivalence logique forcerait

tous les intervalles à être décomposables, une caractéristique explicitement refusée dans le modèle de base.

On peut prouver que, pour deux intervalles quelconques I et J, une et une seule des treize relations entre intervalles est réalisée.

On peut aussi montrer que la table de transitivité est le résultat de l'axiomatisation effectuée ci-avant. On devrait montrer cela cas par cas, et ce serait long, mais néanmoins assez simple. Les preuves peuvent être faites par applications répétées de l'axiome d'ordonnancement M2. Voici un exemple de démonstration:

Soient I, J, et K tels que I OVERLAPS J & J DURING K; on a:

∃ a,b,c,d,e tels que a:I:d:e & a:b:J:e & b:c:d

∃ f,g,h,i tels que f:g:J:h:i & f:K:i

Ces faits peuvent être représentés graphiquement (figure 1.4):

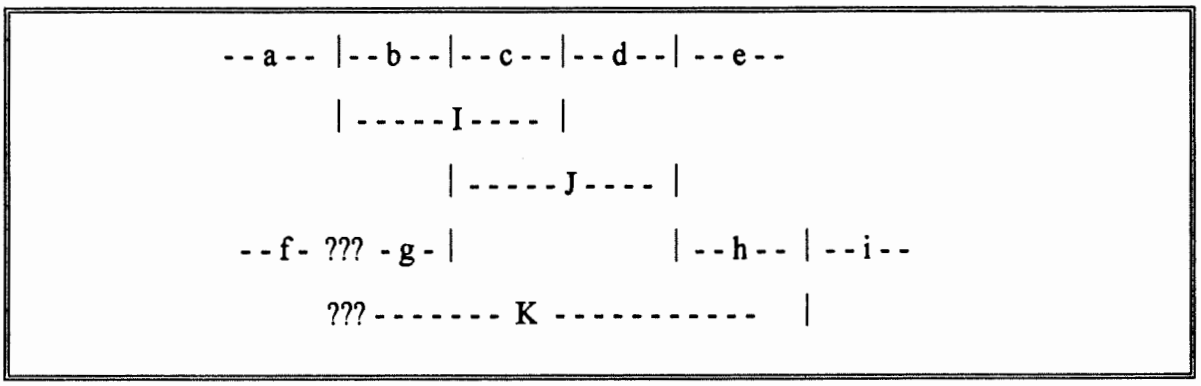


figure 1.4.

En utilisant l'axiome M2, et les faits a:b & f:g, on a trois cas:

1er cas: a MEETS g, et b = g (car a:b:J & a:g:J), on a donc

a:I:d+h:i & a:K:i,

ce qui est équivalent à dire que I STARTS K.

2ème cas: ∃ m tel que a:m:g; dans quel cas on a:

a:m:g+c:d+h:i & a:I:d+h:i & a:m:K:i,

ce qui est équivalent à dire que I OVERLAPS K.

3ème cas: ∃ n tel que f:n:b; dans quel cas on a:

f:n:I:h:i & f:K:i

ce qui est équivalent à dire que I DURING K.

En conclusion, on a que

(I OVERLAPS J & J DURING K) ⇒ I (s o d) K,

ce qui correspond bien à la case visée dans la table de transitivité de [ALLEN 83].

4. Points de temps.

On introduit ici les "**points de temps**" comme étant les places de rencontre des intervalles. Par exemple, on définit le point de temps *p* comme le point où l'intervalle associé au fait "*la porte est fermée*" rencontre l'intervalle associé au fait "*la porte est ouverte*". Par définition, les points n'ont pas de durée, et on ne peut en donner les propriétés qu'en y faisant référence indirectement, c'est-à-dire en référençant un intervalle qui, d'une certaine façon, le contient. Considérons l'exemple d'un porte s'ouvrant alors que l'éclairage fonctionne (figure 1.5). La situation est la suivante:

Soient F, la période durant laquelle la porte est fermée,

O, la période durant laquelle la porte est ouverte, et

E, la période durant laquelle l'éclairage fonctionne.

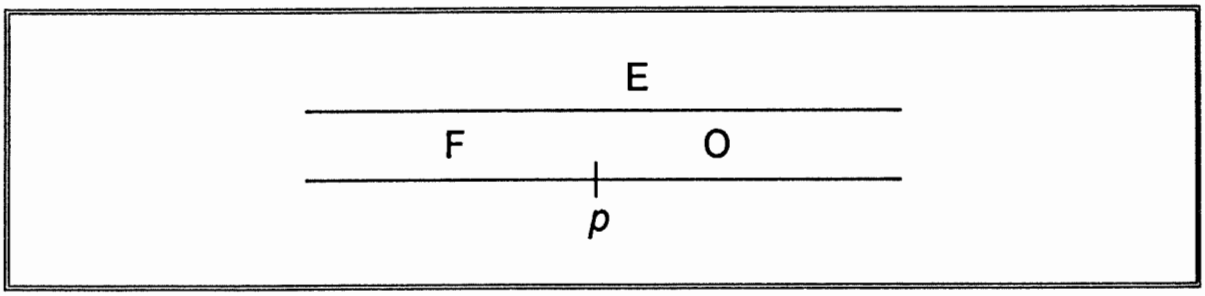


figure 1.5.

Soit p, le point où F et O se rencontrent. On peut dire que l'éclairage fonctionne au point p car p est contenu dans E. On ne peut cependant ni affirmer que la porte est fermée, ni affirmer que la porte est ouverte, car ni F, ni O ne contiennent p. Donc, il apparaîtrait qu'il existe des trous de vérité aux points de temps, mais pas dans les intervalles. En fait, on ne doit pas faire des affirmations aux points de façon "normale".

Pour satisfaire nos intuitions, l'ensemble des points devrait être totalement ordonné, et il devrait y avoir une correspondance biunivoque (fonction bijective) entre les points et les places de rencontre des intervalles, correspondance qui respecte l'ordonnancement des points, de telle manière que p BEFORE q s'il existe un intervalle ayant p et q comme extrémités. On peut formaliser cela de diverses manières, on en donnera deux ici.

* La première (notation fonctionnelle) utilise deux fonctions, **BEGIN** et **END**, ayant pour argument un intervalle et comme résultat un point: BEGIN(I) = p, END(I) = q;
* la seconde (notation relationnelle) utilise une relation ternaire **MEETS-AT** entre deux intervalles et un point: MEETS-AT(I,J,p). Ces deux formalismes peuvent être considérés équivalents moyennant certaines hypothèses dont on ne parlera pas ici.

On utilisera par la suite les lettres p, q, r, s, ... pour référencer des variables de type point. On continuera a utiliser i, j, k, ... pour référencer des variables de type intervalle.

Donnons d'abord les axiomes de base pour obtenir un ordre total pour la relation "<" sur l'ensemble des points.

Non réflexivité

(P1) $\forall$ p, on a $\neg(p < p)$

Antisymétrie

(P2) $\forall$ p, q, on a $(p < q) \Rightarrow \neg(q < p)$

Transitivité

(P3) $\forall$ p, q, r, on a $(p < q)$ & $(q < r) \Rightarrow (p < r)$

"<" est donc un ordre. Pour avoir un ordre total, il faut aussi que deux quelconques points soient comparables. De manière formelle,

(P4) $\forall$ p, q, on a $(p < q) \oplus (q < p) \oplus (p = q)$

Dans le dernier axiome, on force l'ensemble des points à n'être borné ni inférieurement, ni supérieurement.

(P5) $\forall$ p $\exists$ q, r tels que $q < p < r$

Introduisons deux fonctions sur les intervalles, qui ont pour résultat un point extrémité d'intervalle. Pour un intervalle I, BEGIN(I) est le point extrémité inférieur de I et END(I) est le point extrémité supérieur de I. On peut maintenant affirmer que seuls les points où deux intervalles se rencontrent existent:

(PBE1) $\forall$ p, $\exists$ i, j tels que p = BEGIN(j) & p = END(i)

De plus, les endroits où deux points extrémités coïncident correspondent aux places de rencontre:

(PBE2) $\forall$ i, j, on a i:j $\equiv$ (END(i) = BEGIN(j))

Enfin, la relation entre l'ordonnancement des rencontres et l'ordonnancement des points est donnée par l'axiome suivant, qui reprend l'idée selon laquelle il doit y avoir un intervalle séparant deux points de temps de telle manière que l'un soit plus tôt que l'autre:

(PBE3) $\forall$p,q on a p < q $\equiv$ ($\exists$ i tel que p = BEGIN(i) & q = END(i))

Ce dernier axiome peut être pris comme définition de "<", et il est assez aisé de dériver les axiomes P1 à P5 à partir des axiomes sur les intervalles (M1 à M5) ainsi que les axiomes PBE1 à PBE3. Par exemple, la preuve de la transitivité découle des deux Lemmes suivants:

Lemme PL1: $\forall$i,j on a i:j $\Rightarrow$ END(j) = END(i+j)

Démonstration:

De M4.1, on déduit qu'il existe un intervalle k qui est rencontré par j et par i+j. Dès lors, en appliquant PBE2, END(j) = BEGIN(k) = END(i+j), cqfd.

De la même manière, on montre que:

Lemme PL1.1: $\forall$i,j on a i:j $\Rightarrow$ BEGIN(i) = BEGIN(i+j)

Si maintenant p < q et q < r, par PBE3, il existe deux intervalles i et j, avec p = BEGIN(i), r = END(j) et i:j. Par M4.1 et les deux Lemmes ci-avant, on a p = BEGIN(i+j), et r = END(i+j). A nouveau par PBE3, on déduit que p < r, cqfd.

Le Lemme PL2 ci-après découle également des axiomes PBE1 à 3: "le début d'un intervalle quelconque précède toujours sa fin".

Lemme PL2: $\forall$i, on a BEGIN(i) < END(i)

Une autre approche peut être faite par l'intermédiaire de la relation MEETS-AT. On utilisera la notation **i;p;j** pour signifier que la relation MEETS-AT est réalisée entre les intervalles i et j, et le point p, plutôt que la notation conventionnelle MEETS-AT(i,j,p), moins intuitive. Les axiomes correspondant à PBE1-PBE3 deviennent:

(PMA1) $\forall$ p $\exists$ i, j tels que i;p;j

(PMA2) $\forall$ i, j, on a i:j $\equiv \exists$ p tel que i;p;j

La définition de l'ordonnancement du temps devient:

(PMA3) $\forall$ p, q, on a p < q $\equiv \exists$ i,j,k tels que i;p;j & j;q;k

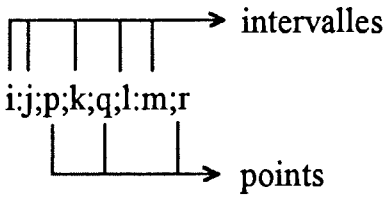
L'axiome PMA3 a pour hypothèses trois intervalles simplement par nécessité syntaxique: on ne peut écrire p;j ou j;q dans la notation introduite. Cependant, on peut imaginer une extension de cette notation, qui permet de telles expressions, afin d'exprimer dans la sémantique relationnelle des propositions telles que p = BEGIN(j) et q = END(j). Cette extension à la notation relationnelle montre directement la relation entre un intervalle et ses extrémités d'une manière semblable à la notation infixe (notation qui consiste à mettre l'opérateur entre les deux opérandes) utilisée pour la relation MEETS. Si les axiomes PBE1 à PBE3 sont réécrits avec cette extension, alors PBE1 et PBE2 deviennent respectivement identiques à PMA1 et PMA2, et PMA3 devient:

(PMA3.1) $\forall$ p, q $\exists$ i tel que p;i;q

duquel on déduit facilement PMA3. On a maintenant besoin d'un axiome d'unicité, lequel était implicite dans la notation fonctionnelle.

(PMA4) $\forall$ p, q, i, on a ((i;p & i;q) $\vee$ (p;i & q;i)) $\Rightarrow$ p = q

Par la suite, on utilisera la notation ";" pour signifier BEGIN, END et MEETS-AT, selon le contexte. Nos notations autorisent donc des expressions où les intervalles sont séparés entre eux par des ":" et séparés des points par des ";". On pourra par exemple écrire:



comme abréviation de

i:j & END(j) = p & BEGIN(k) = p & END(k) = q & BEGIN(l) = q & l:m & END(m) = r.

Notons cependant que des expressions de la forme p:q ou p:i ou i:p sont illicites, car les points ne peuvent ni se rencontrer entre eux, ni rencontrer (ou être rencontrés par) un intervalle.

Plusieurs Lemmes peuvent être démontrés, établissant des propriétés des points et des relations entre points et intervalles. Par exemple, par analogie avec M5:

Lemme PL3: $\forall i,j,p,q$ on a $(p;i;q \ \& \ p;j;q) \Rightarrow i = j$

ce qui peut encore s'écrire:

$$BEGIN(i) = BEGIN(j) \ \& \ END(i) = END(j) \Rightarrow i = j$$

Lemme PL4: $\forall i,j$ on a $i:j \Rightarrow (\forall p$ on a $i;p \equiv p;j)$

Lemme PL5: $\forall i \ \exists \ p,q$ tels que $p;i;q$

Notons qu'on ne peut toujours pas affirmer que quelque chose est vrai à un point, car le point n'est pas encore une sorte d'entité à laquelle est associée la survenance d'un fait, le fait que quelque chose soit vrai ou faux, ... On peut seulement affirmer indirectement qu'une proposition est vraie durant une période qui contient un point, comme dans l'exemple de la porte qui s'ouvre alors que l'éclairage fonctionne. On peut d'ailleurs axiomatiser la notion de point dans (appartenant à) un intervalle.

(PMA5) $\forall \ p,i$ on a $(p \in i) \equiv \exists \ k,l$ tels que $(i = k+l) \ \& \ (k;p;l)$

Notons que par cette définition, les extrémités d'un intervalle ne sont pas dans celui-ci.

5. Dérivation des points par la théorie des ensembles.

Jusqu'à présent, on a introduit les points simplement en supposant leur existence et en construisant des axiomes qui les décrivent. La littérature mathématique contient cependant plusieurs méthodes basées sur la théorie des ensembles, dans lesquelles les points peuvent être construits explicitement à partir des intervalles. Une d'entre elles, due à A.N. Whitehead, définit un point comme étant l'intersection commune d'un ensemble d'intervalles. Les axiomes M1-M5, complétés par quelques éléments de théorie des ensembles, supportent une telle construction et permettent de déduire ses principales propriétés. Un tel ensemble d'intervalles qui se recouvrent est appelé un "nid" (nest).

Dans notre cas, comme la rencontre des intervalles est la relation de base, il est naturel de grouper tous les intervalles qui partagent ou contiennent une place de rencontre. L'extrémité début d'un intervalle, par exemple, est définie par l'ensemble de tous les intervalles qui intuitivement touchent ou contiennent l'extrémité début. Il en est de même pour l'extrémité fin d'un intervalle.

$$*BEGIN(I) = \{j \text{ tels que } \exists \ a, b \text{ tels que } a:I \ \& \ a:b \ \& \ j=a+b \vee j=b \vee j=a\}$$

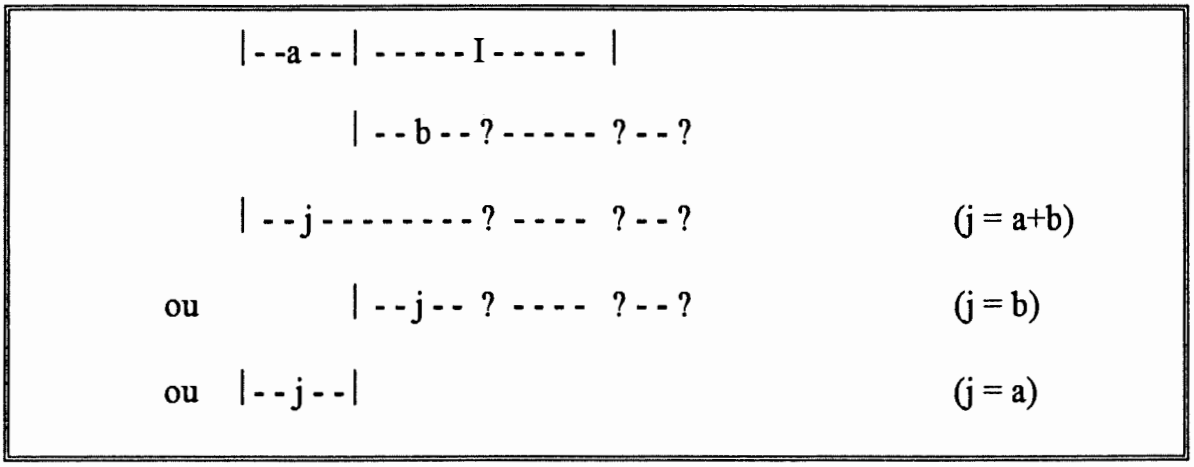


figure 1.6.

*END(I) = {j tels que ∃ b, c tels que I:c **&** b:c **&** j=b+c ∨ j=b ∨ j=c}

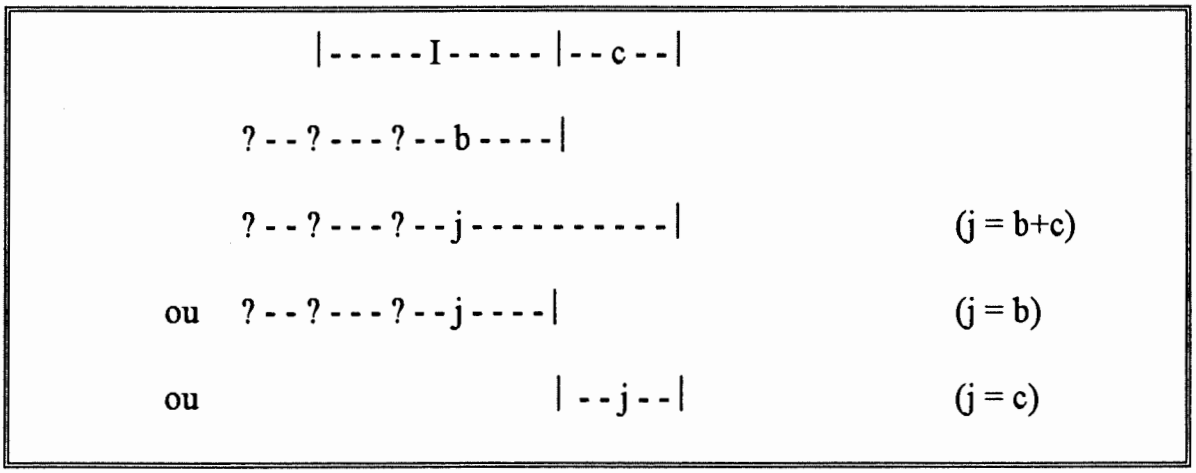


figure 1.7.

Ces ensembles, qui ne sont jamais vides, peuvent aussi être définis en utilisant les 13 relations entre intervalles définies au point 2 (figure 1.2):

*BEGIN(i) = {j tels que j (o s : fi di e si) i}

*END(i) = {j tels que i (o fi : f d e s) j}

Pour la facilité, on définira un nid comme une extrémité début ou une extrémité fin.

NEST(p) ≡ ∃ i tel que p = *BEGIN(I) ∨ p = *END(I)

Supposons que nous ayons un modèle *MM des axiomes M1 à M5. Si on ajoute tous les nids à l'univers (défini par M1-M5) et si on interprète les symboles "BEGIN" et "END" respectivement comme *BEGIN et *END, on obtient un modèle pour un ensemble plus large d'axiomes comprenant PBE1 à PBE3. Notre axiomatisation est donc cohérente avec cette construction mathématique des points à partir des intervalles.

On peut définir des relations sur les nids afin de leur donner les propriétés des points. On dira qu'un nid N est avant (before) un nid M si et seulement si il existe au moins un intervalle dans N qui est avant un intervalle dans M. Pour deux quelconques nids N et M,

$$N < M \equiv \exists\ n,m \text{ tels que } (n \in N)\ \&\ (m \in M)\ \&\ (n \text{ BEFORE } m)$$

Maintenant, on peut montrer quelques propriétés importantes des nids.

Lemme NL1: I est dans *BEGIN(I) et dans *END(I).

Démonstration:

I satisfait aux deux définitions:

en effet, si dans la définition de *BEGIN(I), j = b = I, il suffit de vérifier que:

$$\exists\ a \text{ tel que } a{:}I\ \&\ a{:}I\ \&\ (j = a{+}I \vee I = I \vee j = a)$$

Or, a:I est correct car par M3, $\forall$I, $\exists$a,b tels que a:I:b. La troisième partie du prédicat ci-avant étant trivialement réalisée, on en déduit que I fait bien partie de *BEGIN(I).

La démonstration est similaire pour *END(I).

Lemme NL2: Si i et j sont dans un nid N, alors soit i:j soit i!j existe et est dans N.

Le résultat principal de ce qui précède est que, étant donné deux nids quelconques et les relations d'ordre entre nids définies précédemment, soit les nids doivent être égaux, soit l'un des deux doit être avant l'autre. De plus, ces possibilités s'excluent mutuellement ($\oplus$), ce qui donne le théorème N1:

Théorème N1: $\forall$nids N,M on a $N < M \oplus M < N \oplus N = M$.

On peut maintenant montrer que les nids ont toutes les propriétés des points décrites dans la section précédente, en adaptant le Lemme PL1 et les axiomes PBE1 à PBE3. Une fois cela terminé, tous les autres résultats de la section précédente sont quasi-automatiques. Les variables p et q sont maintenant de type nid:

(PL1') $\forall i,j$ on a i:j $\Rightarrow$ *END(j) = *END(i+j)

$\forall i,j$ on a i:j $\Rightarrow$ *BEGIN(i) = *BEGIN(i+j)

(PBE1') $\forall p \exists i,j$ tels que p = *BEGIN(i) & q = *END(j)

(PBE2') $\forall i,j$ on a i:j $\equiv$ (*END(i) = *BEGIN(j))

(PBE3') $\forall p,q$ on a $p < q \equiv \exists$ i tel que p = *BEGIN(i) & q = *END(i)

6. Les instants.

Certains points semblent exister là où les intervalles se rencontrent, mais d'autres aussi semblent avoir assez de substance pour que des évènements se déroulent en eux. Par exemple, une ampoule qui grille lorsque le flash d'un appareil photo fonctionne. Ce genre d'évènements ressemble à de vrais évènements, lesquels occupent des points plutôt que des intervalles. Nous allons introduire la notion d'intervalle très court, appelé instant, que des évènements, tels que le bois d'une raquette usée qui se fend lorsque l'on tape trop fort sur la balle, occupent.

Un instant est un intervalle qui n'a pas de structure interne. Dès lors, il ne peut ni recouvrir un autre intervalle, ni en contenir un (Théorème MO1). On peut résumer cela en disant qu'il n'y a pas de points dedans. De manière formelle, on le définira comme suit:

$\forall$i on a TRUE-INTERVAL(i) $\equiv \exists$ a,b,c,d tels que a:i:d & a:b:c:d

$\forall$i on a MOMENT(i) $\equiv \neg$TRUE-INTERVAL(i)

Donc, un *true-interval* a au moins deux sous-intervalles (qui peuvent être des instants ou des *true-intervals*), un qui le "commence" (STARTS) et un qui le "termine" (FINISHES). Un autre manière d'établir la définition d'un true-interval serait de dire qu'il est le résultat d'une opération d'union, i.e.:

$\forall$i on a TRUE-INTERVAL(i) $\equiv \exists$ a,b tels que i = a+b

Avant de continuer, il est important de se rappeler que tous les théorèmes vus précédemment ont été démontrés sans faire de distinction entre les instants et les true-intervals, donc ils sont valables pour les deux classes: aucune preuve n'a nécessité la décomposabilité d'un intervalle.

Ces définitions nous autorisent à démontrer que deux instants ne peuvent se recouvrir d'aucune façon. Cependant, ils peuvent se rencontrer. Plus précisément,

(MO2) $\forall i,j$ on a MOMENT(i) & MOMENT(j) $\Rightarrow$ i (b m = mi a) j

Les instants sont des intervalles très simples. Bien qu'un instant soit une période où quelque chose change, rien ne change durant cette période (i.e., il n'y a pas de sous-période). Les instants sont différents des points. Ils appartiennent à des catégories différentes: la proposition "Quelque chose est vrai durant un instant" est valable, alors que la proposition "Quelque chose est vrai durant un point" n'a pas de sens. Mais dans certains aspects, les instants se comportent comme des points. Par exemple, les instants peuvent être identifiés par les extrémités des intervalles qu'ils rencontrent. En effet, on a

(MO3) $\forall i,j,k$ on a (MOMENT(i) & MOMENT(j) & k:i & k:j) $\Rightarrow$ i=j

Démonstration: Supposons que non; alors, END(i) <a END(j). De plus, par M2, il existe un k tel que END(i);k;END(j) (ou symétriquement END(j);k;END(i)). Dès lors, j = i+k (ou i = j+k), ce qui implique que moment(j) est faux (ou moment(i) faux), ce qui contredit les hypothèses selon lesquelles i et j sont des instants. Notre supposition était donc fausse, cqfd. On peut démontrer la même chose pour i:k et j:k.

Il semble prometteur d'essayer d'utiliser les instants comme des points, afin de forcer la similitude. Cependant, il existe une différence structurelle évidente entre les points et les instants: les instants peuvent rencontrer d'autres intervalles, et en plus se trouver entre ces derniers. Donc, si i;p;j pour un point p, alors i et j se rencontrent: il n'y a rien entre eux. Seulement si i:m:j pour un instant m, cela implique que i ne rencontre pas j.

7. Application de cette théorie dans les différents modèles (du temps).

Il est intéressant de voir comment cette théorie se retrouve dans les différents modèles possibles.

- Le plus simple est le modèle discret: les intervalles sont des paires d'entiers <n,m> avec n < m, et si m < k, <n,m>:<m,k>. Un instant est alors un intervalle indécomposable <n,n+1>, et les nids mettent en évidence des entiers, c'est-à-dire des points. Dans ce modèle, il y a une distinction claire entre les instants et les points.

- On peut aussi définir des modèles basés sur la droite des réels. Par exemple, les intervalles de temps peuvent être représentés par des intervalles ouverts, fermés, ou semi-ouverts.

- Un modèle (continu) plus simple, basé sur le modèle entier ci-dessus, définit les intervalles de temps comme des paires de réels <a,b>, avec <a,b>:<b,c>. Dans ce modèle-ci, les instants n'existent pas, puisque même le plus petit intervalle

imaginable est toujours décomposable. Les nids définissent des points de la droite des réels. On pourrait imaginer d'étendre le modèle en autorisant des intervalles de la forme <a,a> qui représenteraient les instants. Seulement, si on considère <a,b>, <b,b>, et <b,c>, par définition, le premier rencontre le dernier. Pourtant, le deuxième se trouve entre eux, ce qui implique que le premier devrait être strictement avant (before) le dernier, ce qui provoque une contradiction.

- Une autre tentative de modélisation consisterait à utiliser les mêmes définitions d'intervalles et de rencontre (desquelles découle tout le reste), mais en permettant d'avoir certaines parties de la ligne du temps de type discret, et d'autres parties de type continu. Intuitivement, si on n'a que des instruments de mesure pas très fins, on traitera le temps de manière discrète; si on a accès à des évènements qui peuvent affiner les distinctions, les distinctions peuvent alors faire éclater les instants en parties de plus en plus petites.

8. Représentations du temps dans les modèles de données.

Pour prendre en compte le temps au niveau d'un modèle de données, il faut tenir compte de plusieurs choses:

- le type d'application dans laquelle la gestion du temps sera intégrée,
- les restrictions en matière de place consommée par l'overhead temporel associé aux données,
- les types d'opérations que l'on veut effectuer sur les valeurs temporelles,
- ...

De plus, un SGBD[5] standard doit répondre à un large éventail d'applications; on doit dans ce cas choisir la représentation la plus souple possible et donc susceptible de répondre à la majorité des besoins courants.

On pourrait imaginer de développer un SGBD temporel spécialement adapté à chaque application particulière, mais ce serait aller à l'encontre

- de l'idée sous-jacente qui est de ne pas réinventer la roue à chaque fois, c'est-à-dire que l'on veut éviter que l'utilisateur ne soit obligé de gérer lui-même le temps dans chacune de ses applications d'une manière ad hoc,
- et de la volonté de faire faire un maximum d'opérations par le système lui-même. En effet, réécrire le SGDB en fonction de l'application revient en quelque sorte à gérer le temps au niveau applicatif et non plus au niveau système.

D'une manière générale, on utilisera un Univers Temporel que l'on représentera ici par **T** [GADIA 88].

(5) *Système de Gestion de Base de Données.*

La plus forte hypothèse que l'on puisse faire sur T est qu'il est constitué de points du temps équidistants ("equally spaced instants of time"). Cette supposition est notamment faite par [GADIA 85].

Une autre proposition qui peut être faite, moins contraignante que la précédente, est que T est un ensemble discret.

> Définition: On dit que T, sous-ensemble de R est discret s'il existe un $\varepsilon \in R$ strictement positif tel que pour tout couple de points $(t_1, t_2) \in T^2$, la distance entre t_1 et t_2 est $\geq \varepsilon$.

Cette hypothèse est faite par [CLIFFORD 85]. Attention! la notion d'ensemble discret utilisée ici n'est pas la même que la notion d'ensemble discret habituelle: elle est plus restrictive. Selon la définition qui précède, l'ensemble $\{1/n: n \geq 1\ (n \in N)\}$ est un sous-ensemble discret de R mais n'est pas discret selon la définition qui précède.

On peut encore exiger que T soit dense. Dans ce cas, pour tout couple de points $(t_1, t_2) \in T^2$, il existe toujours un point $t_3 \in T$ tel que $t_1 < t_3 < t_2$. On obtient ici une précision sans limites que l'on retrouve dans les travaux de [CLIFFORD 83].

Finalement, on peut encore supposer que T est équivalent (isomorphe) à R. On peut imaginer certaines applications où cela serait utile. Voici un exemple de ce qu'on pourrait faire: considérons la définition de tuple suivante, qui est constituée de fonctions...

F		G	
2t - 1	if $t \in [0, 5[$	sin t	if $t \in [0, 10[$
cos t	if $t \in [5, 20[$	log t	if $t \in [10, 20[$

Figure 2.1. Un tuple constitué de fonctions.

Ici les objets sont fonction du temps. Si on veut trouver un instant t où F = G, il est intéressant si pas indispensable de supposer que t est un réel. Cette approche convient fort bien à la majorité des problèmes physiques.

La solution la plus naturelle pour représenter un certain point du temps est de définir un instant de référence et une unité (souvent une seconde, une minute, une heure, ...). Par exemple, MS-DOS prend comme point de référence le 1/01/1980 à 0h, et comme unité de temps le centième de seconde. Dans cette acception, chaque évènement peut être associé à un point du temps. On interprétera une telle association comme suit: L'évènement ***e*** se produit à ***x*** unités de temps à partir de l'instant de référence. Cette solution n'est cependant pas adaptée aux applications

courantes des systèmes d'informations où la précision d'un évènement n'est pas aussi fine que celle des évènements physiques.

On préfère pour cela utiliser une représentation discrète: le calendrier. Un évènement qui se déroule à un instant donné se déroule par exemple au cours de la journée du 21/08/1991 ou au cours du mois de mai 1991.

Le calendrier le plus courant est une extension du calendrier grégorien. Un exemple d'utilisation du calendrier grégorien étendu peut être trouvé dans [ADIBA 87]. Il existe plusieurs unités différentes pour évaluer soit un point dans le temps soit une durée. Ces unités sont appelées intervalles de temps et sont de granularités décroissantes. On ne s'intéresse ici qu'à un ensemble fini d'intervalles de temps I = {I1, I2, ..., In}. Par exemple, I = {année, ..., seconde}. (1990, 8, 19, 15, 27, 43) est la représentation de l'instant "19 août 1990 à 15h27min43sec".

Lorsque le concept de calendrier est utilisé, des problèmes se posent pour manipuler des durées car les opérations sur les durées sont parfois définies sur des intervalles de temps de granularités différentes. Par exemple, combien valent (1990, 8, 19, 15, 27, 43) + 3 mois ?

Les trois prochains chapitres seront chacun consacrés à une approche particulière en matière de temps dans les modèles de données:

- ⇨ j'expose l'approche historique en premier lieu car elle est la moins évoluée des trois approches sur le plan temporel pur.
- ⇨ Le chapitre suivant exposera l'approche temporelle tout en montrant l'évolution des concepts ainsi qu'une classification des différentes extensions temporelles.
- ⇨ Le chapitre 4 étudiera une approche double qui consiste en la combinaison d'une approche historique et d'une approche dynamique.

Chapitre 2.
Un modèle historique.

Introduction.

Je base ici mon étude sur le modèle proposé par [ADIBA 87]. Cette approche a été développée comme extension à un modèle de données existant. Il s'agit du modèle de données orienté multimédia "***TIGRE***" et de son langage de définition et de manipulation "***LAMBDA***" (TIGRE et LAMBDA offrent des outils pour décrire la sémantique associée aux objets complexes).

J'ai choisi ce modèle comme base de ce chapitre car en plus d'être un modèle historique, il définit beaucoup de concepts originaux que l'on ne retrouve pas dans la plupart des autres travaux sur le même thème. Il est très intéressant de par la richesse des concepts qu'il propose.

L'introduction de la notion de temps dans le modèle de données est réalisée par l'apport de nouveaux types de données. La sémantique de ces nouveaux types est exprimée par les fonctions et les opérations qui les manipulent. La sémantique du modèle est donc aussi augmentée.

1. Types de temps.

1.1. Type de temps simple: "time".

Ce type est défini par le calendrier grégorien étendu jusqu'aux secondes. Par exemple, un type "date" est défini comme suit: **type date: time;**. La représentation externe est une chaîne de 19 caractères: par exemple, '1987/08/23 13h42:01'.

remarques:

* On peut faire référence au présent par le mot-clé 'now'. Un objet de type ***now*** prendra la valeur délivrée par la procédure "lisant" l'horloge du système.
* Les valeurs des composantes successives (année, mois, ...) satisfont aux contraintes d'intégrité liées au calendrier grégorien étendu jusqu'aux secondes. Par exemple, le mois doit être compris entre les valeurs 01 et 12
* Deux valeurs de type **time** peuvent être comparées par les opérateurs "<", "=", ... Etant donnée la représentation externe, la comparaison se fait comme entre chaînes de caractères.

1.2. Types de temps restreints.

Dans la représentation du temps, la seconde a été choisie comme granularité minimale. Pour certaines applications, une telle précision n'est pas nécessaire, et peut même être gênante. Dans ce but, on peut restreindre la précision du type **time** à l'une de ses composantes. Par exemple: **type date: time > hour;** décrit le calendrier (année, mois, jour) et peut prendre pour valeur, '1987/08/23'.

remarques:

* Les contraintes d'intégrité restent applicables.
* Les opérateurs de comparaison ne sont applicables que sur des opérandes de même précision.
* Les types de temps simples ou restreints seront appelés points du temps dans la suite de ce chapitre.

1.3. Type intervalle de temps.

Un type intervalle de temps est défini par deux points de temps de même précision dont le premier est inférieur ou égal au second. Par exemple: **type année-académique: time-interval;**, qui peut valoir '1989/09/15-1990/09/15'.

remarques:

* Les deux bornes appartiennent à l'intervalle.
* Une des bornes peut être ***now***.

1.4. Type durée.

Pour représenter une information de durée, on introduit le type 'duration'. Par exemple, on définit **type durée-travail: duration;**, qui peut valoir '8h' ou '8hour' pour la lisibilité.

1.5. Constantes de type durée.

Il est possible de définir une constante de type durée par le mot-clé **time-constant**. Par exemple, les constantes Trimestre et Semaine sont définies par

```
time-constant
        Trimestre = 3month
        Semaine = 7day
end;
```

1.6. Type période.

Cela concerne les opérations périodiques. Par exemple, le calcul des intérêts en début d'année, le calcul des salaires chaque mois. On distingue:

- des périodes instantanées: par exemple, **type intérêts: month:=1;**, pour définir la période 'tous les mois de janvier';
- des périodes intervalles: par exemple, **type calcul-salaire: periodic day:=1 - day:=4;** pour définir une période comprise entre le premier et le 4 de chaque mois (pour éviter que les opérations ne tombent pendant un week-end éventuellement prolongé d'un jour).

remarques:

* A chaque valeur de ces types, il faut associer un intervalle de référence où cette période se déroule. Par exemple, pour l'énoncé: "tous les premiers jours de chaque mois pendant l'année 1990", l'intervalle de référence peut être '1990/01/01-1990/12/01'.
* L'une des deux bornes peut être **now** ou ∞. Par défaut, l'intervalle aura la valeur (-∞, +∞).

2. Fonctions et opérations sur les types de temps.

2.1. Fonctions de sélection sur les types de temps.

Pour un argument ***t*** (point du temps), on définit les fonctions suivantes: **year(t)**, **month(t)**, **day(t)**, **hour(t)**, **minute(t)**, **second(t)**, et **week(t)**. Le résultat est un entier. Par exemple, **month(t)=10**, sauf pour **week(t)** où le résultat est une chaîne de deux caractères représentant les deux premières lettres du jour de la semaine correspondant au point du temps ***t*** argument.

D'autres fonctions sont proposées:

- **trunc-time(t, e)** où e ∈ {month, ..., second}. Par exemple, **trunc-time**('1987/08/23 13h42:01', hour) = '1987/08/23'.
- **begin(int)** et **end(int)** donnent respectivement les bornes inférieures et supérieures d'un intervalle.
- **duration(int)** donne la durée de l'intervalle.
- **multime(k, d)** multiplie une durée ***d*** par un entier ***k***.

2.2. Opérations d'addition et de soustraction.

Deux opérations d'addition sont définies: l'une entre deux durées, l'autre entre un point du temps et une durée.

Trois opérations de soustraction sont définies: une première est définie sur deux durées (le résultat est une durée), une autre sur deux points du temps (le résultat est une durée), et la dernière entre un point du temps et une durée (le résultat est du même type que celui du premier opérande).

2.3. Opérations pseudo-ensemblistes.

Sont définies:

- l'union de deux intervalles (le résultat est un intervalle);
- l'intersection de deux intervalles (le résultat est un intervalle);
- l'inclusion d'un intervalle dans un autre (le résultat est un booléen);
- l'appartenance d'un point du temps à un intervalle (le résultat est un booléen);
- la fonction **overlap** qui vérifie si deux intervalles ont un ou plusieurs points en commun (le résultat est un booléen).

Ces types de temps et ces fonctions/opérations décrits aux points 1 et 2 constituent la première extension au langage LAMBDA. La seconde extension, présentée ci-après, concerne les énoncés de manipulation. Il s'agit des clauses particulières pour la manipulation des données temporelles (= de type temps).

3. Clauses liées à la sémantique temporelle.

Soient x, y, z, des variables temporelles. Celles-ci peuvent être attributs d'une classe, définis sur un type de temps, une constante ou un paramètre de temps de document. Les classes qui suivent peuvent être utilisées dans les requêtes LAMBDA pour manipuler les données.

- x **precede** y; x **follow** y. Ces clauses expriment l'ordre "avant" et "après" entre les points de temps.
- x **longer** y; x **shorter** y. Ces clauses permettent de comparer des durées de temps.

- **x precede y by z; x follow y by z.** x et y sont des points de temps et de même type, z est une durée. La clause "precede ... by ..." signifie: vérifier que z = la valeur résultant de l'exécution de subtime(x,y). Pour "follow ... by ...", z = subtime(y,x).
- **x longer y by z; x shorter y by z.** x, y, z sont de type durée.
- **move-back x by y; move-forward x by y.** x est un point de temps et y est une durée. La première clause signifie x := addtime(x,y) et la seconde x := subtime(x,y).
- **extend x by y; reduce x by y.** x et y sont du type durée. La première clause signifie x := addtime(x,y) et la seconde x := subtime(x,y).
- **not precede, not longer, ...**

Toutes ces clauses sont utilisées dans les requêtes LAMBDA pour manipuler les données temporelles.

4. Définition des historiques et des versions d'objets: concepts de base.

Considérons un objet O de la base de données qui prend au cours du temps des valeurs successives. Notons (v,t) les couples (valeur,temps) pour cet objet: à l'instant t, O a pris la valeur v. La séquence de couples (v_i,t_i) est appelée l'historique de O. Les valeurs v_i sont appelées versions historiques de O. Les t_i sont tous distincts mais ce n'est pas forcément le cas pour les v_i.

Prenons comme exemple le salaire de l'employé X. L'intérêt de tels historiques est de pouvoir répondre à des questions du genre:

(1) Quel était le salaire de l'employé Dupont en Juin 1989 ?,

(2) Quels furent les salaires de l'employé Dupond au cours des 18 derniers mois ?,

(3) Quel est l'historique complet du salaire de l'employé Durand ?

La question (1) fait explicitement référence à un instant du passé. La question (2) fait référence à une série périodique (mois par mois). Dans la question (3), on s'intéresse à tous les couples (salaire,temps) qui correspondent à toutes les versions successives du salaire.

L'approche de [ADIBA 87] concernant la notion d'historiques est basée sur les trois notions suivantes:

1) La périodicité des t_i, c'est-à-dire la séquence des instants considérés pour caractériser les v_i. La périodicité ne doit pas nécessairement être liée à un changement de valeur.

2) La modification d'un objet dont la valeur passe de v à v'. Cette modification peut donner lieu à une nouvelle version de l'objet. C'est le concepteur de la base qui en décidera.

3) La persistance des valeurs dans la base. On peut vouloir conserver toutes les valeurs successives d'un objet ou seulement les n dernières.

Pour faciliter la compréhension de la suite, on introduit la relation suivante:

Soit la relation EMPSAL(EmpNum, Nom, Poste, Salaire) qui donne pour chaque employé identifié par son numéro EmpNum, son Nom, le Poste qu'il occupe et le Salaire qu'il gagne. On désire conserver l'historique du salaire de chaque employé.

- Une première solution ajoutera l'attribut Date. Ce sera à l'utilisateur de gérer cet attribut. Cette solution peut provoquer des incohérences dans la base de données si le système de gestion de bases de données ne gère pas explicitement le concept de temps et considère Date comme un entier ou une chaîne de caractères.

- Si le système de gestion de bases de données gère le temps en utilisant par exemple les types et opérations définis précédemment, tous les tuples de EMPSAL sont interprétés de la même manière et il n'y aura pas vraiment de notion d'historique qui permette une distinction entre la dernière version d'une donnée et les versions précédentes.

- La solution des auteurs a pour but d'offrir la notion d'historique de manière explicite au niveau du langage de définition et cela pour différents objets du schéma. Ainsi, on pourra conserver l'historique des salaires de chaque employé, mais aussi l'historique des employés eux-mêmes.

On peut donc constater que la notion d'historique dépend fortement du modèle de données et de la modélisation de la réalité que l'on souhaite. Ensuite, on définira la granularité des objets historiques et on pourra jouer sur le format et la périodicité des t_i, sur les modifications des valeurs, ...

La terminologie utilisée pour appliquer la notion d'historique à un objet de la base de données est le terme "dynamic". Pour jouer sur les notions de périodicité, de modifications et de persistance, on introduit de clauses comme "each <période>", "last n" qui sont expliquées plus loin.

Si l'on reprend l'exemple des employés, on pourra définir dans le modèle soit un attribut salaire dynamique, soit une entité dynamique.

Voici une première modélisation utilisant le langage de définition LAMBDA dans laquelle seul l'attribut Salaire est dynamique:

```
type EMPSAL: entity
EmpNum: integer;
Nom: string(15);
Poste: string(10);
Salaire: dynamic
each month
last 24
(2 000...40 000);
end;
```

Dans cet exemple, le Salaire est dynamique au sens où chaque mois ('each month'), sa valeur est recopiée dans l'historique. De plus, seules les 24 dernières valeurs ('last 24') sont conservées.

Voici un autre exemple où toute l'entité est dynamique:

```
type EMPSAL: dynamic entity
each year                          } Structure historique
last 10
with time > month
EmpNum: integer;
Nom: string(15);                   } Structure de données
Poste: string(10);
Salaire: (2 000...40 000);
end;
```

Il s'agit ici d'une entité historique. La périodicité (annuelle) est indiquée par la clause each year et la persistance (10 ans) est indiquée par la clause 'last 10'.

La première partie de la définition (contenant les mots-clés 'dynamic', 'each', 'last' et 'with') correspond à la structure historique du type EMPSAL. Le reste est la structure de données.

5. Différents types d'historiques:

Trois types d'historiques sont proposés:

Le premier type consiste à laisser l'usager décider de sauvegarder une nouvelle version d'un objet donné après qu'une ou plusieurs modifications aient eu lieu. Dans ce cas, on parlera d'Historiques à Versions Manuelles (HVM). Une commande GENERATE-VERSION permet de le faire. Dans ce cas, l'historique de l'objet contient toutes les versions successives que l'usager a décidé de garder.

Si on désire un traitement automatique des historiques, on utilisera la clause 'each' pour se référer à une notion de périodicité. Par exemple, chaque jour ou chaque mois. Dans ce cas, on parlera d'Historiques à Versions Périodiques (HVP). Pour un HVP, toutes les modifications sur un objet sont effectuées sur la version courante et les nouvelles versions (historiques) ne sont générées dans la base qu'à la fin de chaque période par la copie de la version courante.

Le troisième type d'historique est utilisé lorsque chaque modification donne lieu à une recopie préalable de l'ancienne valeur dans l'historique. Il s'agit alors d'Historiques à Versions Successives (HVS). Dans ce cas, l'historique contient toutes les valeurs successives d'un objet. On peut toutefois restreindre cet historique au moyen de la clause 'last' si nécessaire.

On remarquera ici que le temps associé aux données historiques est le temps interne du système, c'est-à-dire le temps physique.

6. Historiques et modèle de données.

Dans le modèle relationnel à structure plate, la notion d'historique n'a de sens qu'en considérant qu'elle s'applique au niveau d'une relation. Cela signifie qu'il faut être capable de (re)construire toutes les instances qu'une relation R a eu au cours du temps. Le passage d'une instance à la suivante se fait par insertion, suppression et modification de tuples dans R. Dans le système de gestion de bases de données TIGRE, dont le modèle est une extension du modèle Entité-Association, la notion d'historique est vue autrement. Chaque occurrence d'une entité possède un identificateur interne (surrogate) et éventuellement plusieurs constituants qui sont typés. En rendant certains types historiques, on peut aussi conserver pour une occurrence d'entité, l'historique des valeurs prises par tel ou tel constituant.

Dans TIGRE, on peut ainsi appliquer la notion d'historique aux types construits (renommé, liste, enregistrement, document) mais aussi aux classes (Entités et Associations).

7. Propagation d'historicité.

Si on considère à nouveau l'exemple de l'entité historique EMPSAL, on a un historique à versions périodiques (HVP). Si on retire la clause 'each year', on obtient un historique à versions successives (HVS). Dans ce cas, chaque modification d'un employé entraîne une recopie dans la zone historique. Ainsi, on aura toutes les valeurs successives de tous les attributs de tous les employés. On dit ici que "l'historicité" de l'entité s'est propagée sur ses types composants.

Pour le concepteur de la base de données, le choix de placer la notion d'historique sur une entité (ou une association) ou sur un type composant est un problème de modélisation qui dépend de l'application envisagée. Il s'agit de savoir s'il est utile de "propager" l'historicité sur tous les composants en "historisant" l'entité (ou l'association). Il ne faut pas oublier que la propagation de l'historicité impose les mêmes caractéristiques d'un type historique à ses composants, c'est-à-dire la même périodicité, la même persistance, les mêmes versions.

Remarques:

1°) On dit qu'un type est défini explicitement comme un type historique s'il est défini dans le schéma conceptuel avec le mot-clé 'dynamic'.

2°) On dit qu'un type est défini implicitement comme un type historique s'il est devenu historique par propagation de l'historicité.

Quels sont les principes de propagation de l'historicité s'appliquant au modèle TIGRE ? (Pour les détails, voir [ADIBA 86])

a) Historique et types de base: un type de base n'est pas historique.

b) Historique et types construits: -- Les types construits (renommé, liste, enregistrement, document) peuvent être historiques; -- les types composants d'un enregistrement ou d'une liste historique sont statiques; -- les types composants d'un document non historique peuvent être historiques. En effet, on n'a pas toujours besoin de stocker des anciennes valeurs pour tout un document mais seulement pour certaines de ses parties; -- les noeuds d'un document historique défini comme type en LAMBDA sont historiques (définition implicite).

c) Historique et types classe: les types composants d'une classe historique le sont aussi (définition implicite); une association historique ne lie que des entités statiques. Si une association est historique, il faut propager son historicité aux entités liées à cette association; une spécialisation d'un historique l'est aussi (définition implicite); une intersection (union) des historiques l'est aussi (définition implicite) si leurs historicités sont les mêmes. Par contre, s'il existe un composant statique de l'intersection (union), le type résultat est statique.

Dans certains cas, la propagation des historiques est une chaîne de propagations. Par exemple, si un agrégat associatif est historique, il faut propager son historicité à l'association et aux entités qui le produisent. Pour chacune d'elles la propagation continue pour leurs types construits composants. Ce n'est pas un problème simple.

8. Historiques et mise à jour.

Les mises à jour sur un type historique sont différentes de celles faites sur un type statique. L'insertion d'une occurrence sur un type historique est toujours faite sur la version courante avec le temps associé.

La modification sur un HVS est effectuée en deux phases:

1°) Les tuples à modifier sont copiés avec le temps associé dans une zone d'historiques.

2°) Les nouveaux tuples sont insérés dans la version courante.

La modification sur un HVP se passe comme si l'on n'avait pas de notion d'historiques: la version courante est copiée périodiquement (clause 'each') dans la zone d'historiques.

Pour la suppression, il faut considérer les cas suivants:

1°) Si un tuple d'une classe historique est supprimé, il est enregistré dans la zone d'historiques avec l'indication de suppression.

2°) La suppression d'un tuple d'une classe historique doit être suivie par la suppression sur ses types composants qui sont définis comme historiques d'une façon implicite.

3°) Si un tuple d'une classe non historique contenant des attributs historiques est supprimé, on supprime pour chacun de ceux-ci leur chaîne d'historiques.

8.1. Persistance.

Chaque fois qu'un objet est inséré dans la zone d'historiques on doit vérifier sa persistance. Cette notion est importante parce qu'elle permet de contrôler la taille de l'espace de stockage réservé pour l'historique. Soit, par exemple, une persistance p. A partir de la $(p+1)^{ième}$ insertion de tel ou tel objet, sa version la plus ancienne doit être supprimée de la zone d'historiques.

8.2. Corrections et modifications.

Par définition, on ne peut pas modifier les données de la zone d'historiques. Celle-ci est sémantiquement différente, car l'insertion, la modification et la suppression sont

toujours effectuées sur les versions courantes. Par conséquent, elle ne pourra être déclenchée que par des usagers dûment autorisés, l'administrateur de la base de données par exemple.

9. Historiques et langage de manipulation.

Pour pouvoir manipuler les données historiques, il faut étendre le language de manipulation (LAMBDA ici). On peut rappeler qu'on ne peut effectuer des mises à jour que sur la version courante d'une variable historique de la base de données. Sur les versions historiques, on ne peut qu'interroger et exceptionnellement corriger.

Pour formuler des interrogations sur les données historiques, le temps peut intervenir de deux façons:

a) Sous forme de temps explicite (ou absolu): par exemple, «Donner la valeur de O à l'instant t_0, dans l'intervalle $[t_1,t_2]$ ou après (avant) l'instant t_0».

b) Sous forme de temps implicite (ou relatif): par exemple, «Donner la dernière version de O, les trois dernières (premières) versions de O ou toutes les versions de O».

De ce fait, on emploie le mot-clé 'version' pour construire les requêtes sur les différents types d'historiques.

Les extensions à LAMBDA concernant la manipulation des données temporelles ont déjà été introduites. Voici les extensions à LAMBDA concernant les données historiques.

Le schéma de la base de données qui sert d'exemple est le suivant:

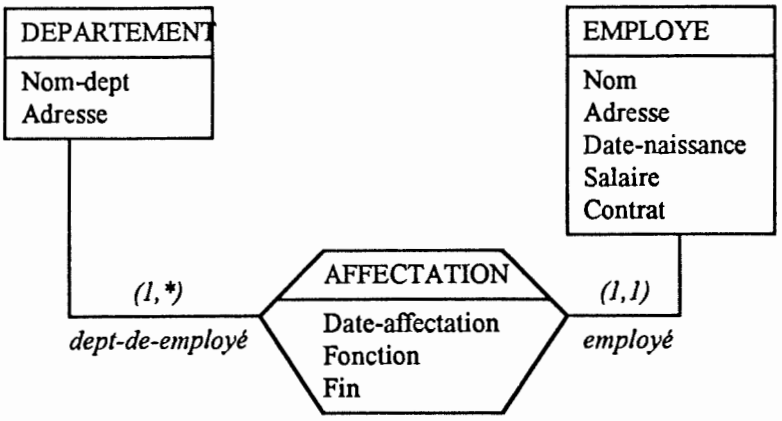


Figure 2.1. Schéma de la base de données.

On définit l'entité EMPLOYE comme une entité dynamique, ce qui correspond à un historique à versions successives (HVS). Le contrat de travail est un document sur lequel se propage la dynamicité.

```
type EMPLOYE : dynamic entity
Nom : string(20);
Adresse : adresse;
Date-naissance : date;
Salaire : argent;
Contrat : t-contrat;
end;
```

Exemple: On s'intéresse à la dernière adresse de l'employé Dupont:

```
SELECT  last version of e.Adresse
FROM  EMPLOYE  e
WHERE  e.Nom = 'Dupont';
```

Pour les données historiques, on associe par convention à chaque version un entier ou une fonction last-k: 1 à la première version, 2 à la deuxième version,..., last-1 à l'avant-dernière version, et last à la dernière version (c'est-à-dire la version courante).

Cette requête est donc équivalente à:

```
SELECT  e.Adresse
FROM  EMPLOYE  e
WHERE  e.Nom = 'Dupont';
```

Exemple: On s'intéresse aux trois derniers salaires de l'employé Dupont:

```
SELECT  3 last version of e.Salaire
FROM  EMPLOYE  e
WHERE  e.Nom = 'Dupont';
```

Exemple: On s'intéresse à toutes les versions des premières clauses des contrats de l'employé Dupont:

```
SELECT  all version of clause 1 of e.Contrat
FROM  EMPLOYE  e
WHERE  e.Nom = 'Dupont';
```

Exemple: On s'intéresse à l'adresse de l'employé Dupont au 30/10/89:

```
SELECT  version at '1989/10/30' of e.Adresse
FROM  EMPLOYE  e
WHERE  e.Nom = 'Dupont';
```

On aura remarqué que le mot-clé 'version at' est employé pour interroger les données historiques avec le temps explicite. La valeur de temps après ce mot doit avoir une granularité plus grande ou égale à celle du type de temps associé à l'historique interrogé.

Exemple: On s'intéresse aux salaires de l'employé Dupont durant la période 1978-1988:

```
SELECT  version during '1978-1988' of e.Salaire
FROM  EMPLOYE  e
WHERE  e.Nom = 'Dupont';
```

Le mot-clé during doit être suivi par une valeur de type intervalle. En remplaçant during '1978-1988' par after '1988', on obtient les salaires de Dupont après 1988.

Remarque: Pour un historique à VP quelconque H, on lui associe une série périodique de temps $T = \{t_1,t_2,...,t_n\}$ pour laquelle on stocke les valeurs de H. Supposons que l'on s'intéresse à la valeur dans H à l'instant t_0. Si t_0 appartient à T, la réponse est claire. Par contre, dans le cas où t_0 n'appartient pas à T, une solution simpliste consiste à retrouver t_i de T tel que t_i soit le plus proche de t_0 et le résultat (approximatif) serait la valeur dans H à l'instant t_i. Mais il peut exister t_i et t_{i+1} de T "équidistants" de t_0. Dans ce cas, on peut prendre comme réponse, la valeur à l'instant t_{i+1} qui nous semble la plus récente.

La base de données historiques peut gérer des données éventuellement supprimées. L'exemple suivant le montre:

Exemple: On s'intéresse aux derniers salaires des employés licenciés.

```
SELECT  last version of e.Salaire
FROM  EMPLOYE e
WHERE  end of e precede 'now';
```

'end of e' indique que la variable e désigne des employés déjà supprimés de la base. La sémantique de precede 'now' signifie que le moment de suppression précède le moment actuel. Si dans cet exemple, on ne veut considérer que les employés licenciés avant le 1er janvier 1988, il faudra remplacer now par cette date.

Jusqu'ici, toutes les requêtes étaient des requêtes de consultation. En voici maintenant une qui effectue une correction. Attention (rappel): une correction ne pourra être effectuée que par un utilisateur ayant autorité pour le faire.

Exemple: On désire corriger le code postal de l'avant dernière adresse de l'employé Dupont.

```
CORRECT  last-1 version of e.Adresse.code-postal = 7090
FROM  EMPLOYE e
WHERE e.Nom = 'Dupont';
```

Remarques:

La notion d'historique peut également s'appliquer au schéma d'une base de données dans la mesure où celui-ci évolue dynamiquement au cours du temps. Il faut alors "historiser" les catalogues de la base. Ceci pose des problèmes non triviaux de cohérence entre les données historiques et les schémas.

10. Photographies dynamiques, albums et films.

10.1. Les photographies: un type de données.

Capturer et stocker indépendamment l'état de tout ou partie de la base à un instant précis peut permettre à certaines applications de travailler sur des données pendant que d'autres travaillent à mettre à jour les données "sources".

La notion de photographie ou de 'vue matérialisée' a été introduite dans les bases de données relationnelles dans [ADIBA 80]. Cela permet de stocker le résultat d'une requête comme une relation accessible seulement en lecture mais sur laquelle

l'opération de rafraîchissement est possible. Cette option s'est avérée très utile en particulier dans le cadre des bases de données réparties.

Les auteurs introduisent d'abord le type photo (= 'snapshot') dans le modèle TIGRE. Ensuite, les notions historiques vues précédemment seront appliquées au type photo. Ceci permettra de mettre en évidence différents types de photos dynamiques selon la périodicité des rafraîchissements et le report des modifications entre données sources et photos.

10.2. Définition des photos.

On considère les photos comme un type de données en introduisant l'opérateur photographie (snapshot) comme un opérateur de dérivation. Il permet de dériver de nouvelles informations à partir de celles présentes dans la base. Une photo TIGRE est définie à l'aide d'une requête d'interrogation en LAMBDA.

Exemple: définir la photo EMPDEP donnant, pour chaque employé, son nom, son salaire, son contrat, le département dans lequel il travaille et la date d'affectation à ce département.

```
type EMPDEP: snapshot AS
SELECT  e.Nom, e.Salaire, e.Contrat, d.Nom-dept, a.Date-affectation
FROM  AFFECTATION a of EMPLOYE e of DEPARTEMENT d;
```

D'une façon générale, on dénote par REQ(T1, T2, ..., Tn) une requête d'interrogation en LAMBDA où Ti ($1 \leq i \leq n$) sont de types classes. L'énoncé de définition d'une classe peut alors être formellement décrit comme suit:

type (nom-de-photo) : snapshot (<liste-des-attributs>) AS REQ(T1, T2, ..., Tn)

Sémantiquement, un type photo est un type classe et les attributs d'une photo sont définis par la requête associée. Le type de chaque attribut hérite donc du type correspondant dans la clause SELECT de la requête définissant la photo. Une fois la photo définie, le système de gestion de bases de données TIGRE évalue la requête (si elle est correcte), génère l'espace de stockage pour la photo et y stocke le résultat de l'évaluation. A partir de ce moment, on peut interroger la photo.

Exemple: Donner la liste des employés affectés au département comptable depuis le 1er janvier 1988.

```
SELECT  ed.Nom, ed.Date-affectation
FROM  EMPDEP ed
WHERE  ed.Nom-dept = 'Comptabilité'
        and ed.Date-affectation follow '1988/01/01';
```

10.3. Rafraîchissement des photos.

On peut créer, détruire, interroger une photo mais on peut également la rafraîchir. Cette opération est la seule qui permette d'en modifier le contenu. Pour cela, on dispose en LAMBDA de deux commandes:

1°) REFRESH (nom-de-photo)

2°) REFRESH (nom-de-photo) AT <point-de-temps>

Dans le premier cas, il s'agit d'un rafraîchissement immédiat. Le système de gestion de bases de données réévalue la requête de définition de la photo et remplace le contenu de la photo par le résultat de cette évaluation.

Pour la deuxième commande, expliquons le terme <point-de-temps> t :

a) **t appartient au passé.** On veut voir l'état de la photo à un instant t. Le système de gestion de bases de données évalue immédiatement la requête de définition si tous les T1, T2, ..., Tn de cette requête sont historiques et si les faits correspondant à ce moment-là sont stockés dans la base. Dans le cas contraire, le système retourne un message d'erreur à l'usager.

b) **t appartient au futur.** Il s'agit d'un rafraîchissement différé. Au moment précisé, le système évaluera la requête de définition de la photo et remplacera son contenu par le résultat de cette évaluation.

Exemple: Rafraîchir la photo EMPDEP demain à 7h00.

```
REFRESH  EMPDEP  AT '1990/08/16 07h';
```

11. Photos dynamiques.

11.1. Notion d'album et de film.

Jusqu'ici, les photos étaient considérées comme statiques. En fait, les structures d'historiques développées précédemment peuvent également s'appliquer aux photos.

L'opération de rafraîchissement est la seule qui permet de modifier le contenu d'une photo. Ce rafraîchissement reflète l'état d'une portion des données à un instant donné. Jusqu'au rafraîchissement suivant, le contenu de la photo ne change pas en dépit des changements éventuels de la base de données. De nombreuses applications requièrent un rafraîchissement périodique ou occasionnel de la photo, par exemple, chaque jour, chaque mois, aujourd'hui etc.

Ceci met en évidence le fait que la structure d'historique peut parfaitement s'appliquer aux photographies. Il s'agit alors de photos dynamiques. Elles seront caractérisées, comme tous les historiques, par la périodicité du temps de rafraîchissement, par le type de version (manuelle, périodique, ou successive), par la persistance dans la base et par le format du temps associé.

Exemple: redéfinir la photo EMPDEP comme une photo dynamique D-EMPDEP à versions successives.

```
type D-EMPDEP  dynamic snapshot AS
SELECT  e.Nom, e.Salaire, e.Contrat, d.Nom-dept, a.Date-affectation
FROM  AFFECTATION a of EMPLOYE e of DEPARTEMENT d;
```

Conformément aux définitions données précédemment, cette photo est un historique à versions successives avec persistance illimitée. Chaque fois que l'on ajoute (supprime) un employé ou que l'on modifie les affectations ou les départements, on doit les reporter immédiatement sur D-EMPDEP.

Compliquons maintenant encore un peu les définitions de photos.

Exemple: Définir une photo sur les 10 derniers salaires moyens de chaque année pour chacun des départements:

```
type SALAIRE-MOYEN: dynamic snapshot
each year
last 10
with time > month
AS      SELECT  d.Nom, AVG[e.Salaire] by d
        FROM  EMPLOYE e of DEPARTEMENT d;
```

SALAIRE-MOYEN est un historique à versions périodiques (HVP) avec une périodicité annuelle de rafraîchissement et une persistance de 10 ans. AVG est une fonction qui donne ici comme résultat la valeur moyenne des salaires pour chaque département pour l'année écoulée.

D'une façon générale, la syntaxe des énoncés de définition de photos dynamiques est la suivante:

type <nom-photo> : dynamic snapshot
<liste-des-attributs>
<structure-historique>
AS REQ(T1, T2, ..., Tn)

Ici, les types T1, T2, ..., Tn peuvent être statiques ou historiques mais la requête de définition porte toujours sur les versions courantes de T1, T2, ..., Tn. Autrement dit, une photo historique peut être définie sur d'autres historiques. Ceci est tout à fait différent des données historiques de base où la définition historique des historiques est interdite. De plus, il n'y a pas de propagation d'historicité pour les types sources d'une photo dynamique.

Pour une photo dynamique à version manuelle (HVM), une nouvelle version est générée par la commande GENERATE-VERSION. Le résultat de cette commande est de copier le contenu courant de la photo dans la zone d'historiques. La commande REFRESH peut modifier le contenu de la photo, mais, comme toute modification sur les HVM, elle peut ne pas générer une nouvelle version.

Pour une photo dynamique à version périodique (HVP), la périodicité de rafraîchissement est indiquée par la clause 'each' qui peut être mensuelle, annuelle, etc. S'il n'y a pas de 'each' dans l'énoncé de définition, il s'agit d'une photo dynamique à versions successives (HVS) pour laquelle le rafraîchissement doit être effectué après chaque modification sur T1, T2, ..., Tn.

Comme pour d'autres historiques, la clause 'last' indique la persistance de la photo dynamique. Pour les données "non photo", une persistance nulle correspond à un type statique et toute modification est effectuée sur la version courante sans garder des versions dans la zone d'historiques. Cependant, on traite aussi le cas de persistance nulle pour les photos dynamiques à versions successives et à version périodique. Par exemple, on peut demander au système de rafraîchir une photo chaque mois. Cette opération est effectuée sur la version courante de la photo sans générer d'informations historiques. On a alors une photo dynamique mais avec une persistance nulle, et on emploie le mot-clé 'last only' pour définir une telle photo. En fait, cela revient à avoir un rafraîchissement qui se fait automatiquement mais en remplaçant l'ancienne valeur par le résultat de ce rafraîchissement.

Exemple: Définir la photo sur les salaires moyens des employés par département avec un rafraîchissement annuel.

```
type SAL-DEP: dynamic snapshot
each year
last only
AS   SELECT  d.Nom, AVG[e.Salaire] by d
     FROM  EMPLOYE e of DEPARTEMENT d;
```

D'une façon imagée, une photo dynamique à version périodique est une collection de photos figées à différents instants. Elle fournit un album automatique sur une partie de la base de données. De la même façon, une photo dynamique à version manuelle est un album manuel. Quant à une photo dynamique à versions successives, elle reflète tous les changements de la base de données sur les types sources T_1, T_2, ..., T_n, ce qui correspond à un film sur la base. La figure 2.2 montre l'évolution des notions: vue, photo, album et film.

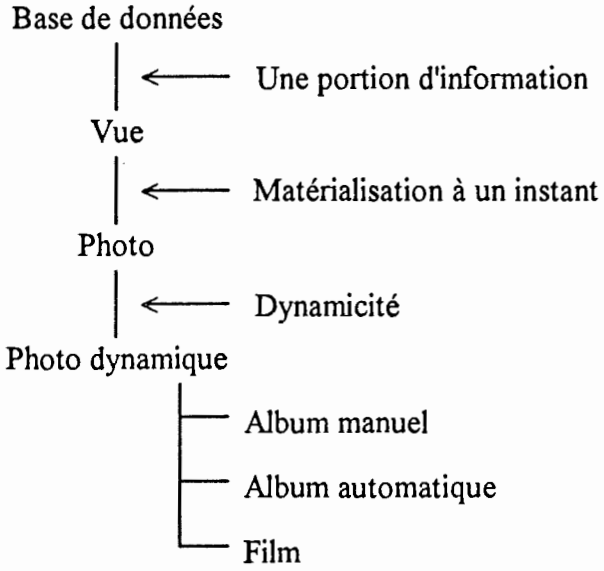


Figure 2.2. Evolution des notions.

Le mécanisme de gestion des historiques décrit précédemment reste le même pour les photos dynamiques. Plus exactement, l'espace de stockage de la version courante est séparé de celui des anciennes versions. Il y a donc aussi une zone d'historiques pour les photos dynamiques. A chaque rafraîchissement d'une photo dynamique, et dans le cas d'une persistance non nulle, au lieu de détruire l'ancienne version de la photo, on la recopie dans la zone d'historiques. Le temps associé est joint à la version courante pour respecter la sémantique de la photo.

Exemple: On s'intéresse au salaire moyen du département 'comptabilité' en 1989.

```
SELECT  version at '1989' of s.SALAIRE-MOYEN
FROM SALAIRE-MOYEN s
WHERE s.Nom-dept = 'Comptabilité';
```

11.2. Rafraîchissement de films.

L'implantation du mécanisme de rafraîchissements de photos dynamiques à versions successives (films) pose des problèmes car un rafraîchissement est en général une opération coûteuse. La méthode brutale consiste à détruire le contenu de la photo à rafraîchir, et à la recréer par réévaluation complète de la requête de définition.

Un mécanisme de rafraîchissement de photos dynamiques doit tenir compte des modifications sur les types sources ne changeant pas le contenu de la photo à rafraîchir, car, dans ce cas, le rafraîchissement n'a pas lieu d'être. Dans le cas contraire, il faut élaborer une méthode permettant d'éviter le plus possible la réévaluation globale de la requête de définition.

Une méthode dite différentielle a été proposée [KOENIG 81] pour éviter la réévaluation globale des données dérivées. L'idée de cette méthode consiste à remplacer n'importe quelle transaction sur les relations sources ainsi que sur les relations dérivées, où T' est équivalente à la séquence {T, réévaluation des relations dérivées}.

Les auteurs proposent une première méthode pour détecter les transactions ne modifiant pas le contenu d'une photo dynamique à VS. Dans ce cas-là, le rafraîchissement n'est pas déclenché. Ils proposent ensuite une deuxième méthode pour effectuer, de manière économique, l'opération de rafraîchissement, et ceci sous certaines conditions portant sur la requête de définition. Voici les principes généraux de ces deux méthodes (pour les détails, voir [ADIBA 86] et [ADIBA 87]).

1°) Détection des transactions ignorées.

Une transaction Tr est dite ignorée pour la photo P si, après la fin de Tr, le rafraîchissement de P ne change pas le contenu de celle-ci.

Par exemple, un changement d'adresse d'un employé est une transaction ignorée pour la photo EMPDEP, parce que l'attribut adresse de l'entité EMPLOYE n'entre pas dans la liste des attributs de EMPDEP. Grâce à des vecteurs de bits, on peut exprimer l'indépendance ou la dépendance de la photo P par rapport aux attributs. En comparant ces vecteurs, on peut déterminer si la transaction Tr est ignorée pour P ou non.

2°) Notion de U-filtre et critère de U-additivité.

Si on considère une transaction insérant seulement l'affectation d'un employé à un département donné, cette modification n'ajoute qu'un tuple à la photo EMPDEP. Le rafraîchissement de EMPDEP doit pouvoir se faire de manière économique. C'est la raison pour laquelle une méthode est proposée pour éviter la réévaluation globale des requêtes de définition de photo. L'idée principale est: au lieu de rafraîchir, ne peut-on pas rafraîchir que ce qui est ajouté (supprimé) ?

Pour faciliter la compréhension, les auteurs utilisent ici le vocabulaire relationnel.

Soit P, un film défini par la requête REQ(R, R1, R2, ..., Rn) où R et les Ri sont des relations. On dénote par R.v et R.n les ensembles de tuples de R respectivement avant et après une transaction quelconque Tr. Considérons d'abord le cas de l'insertion de tuples dans la relation R. Soit R.a, l'ensemble de tuples ajoutés par Tr.

Après l'exécution de Tr, le contenu de P est le résultat de l'évaluation de:

$$REQ(R.v \cup R.a, R1, R2, ..., Rn). \qquad (*)$$

Montrons que sous certaines conditions, la réévaluation de (*) peut être remplacée par l'évaluation de REQ(R.a, R1, R2, ..., Rn) auquel on ajoute l'ancienne valeur de P. Dans ce cas, on dit que le U-filtre (pour dénoter l'union) a été utilisé pour rafraîchir la photo P.

Commençons par l'examen de la propriété d'additivité des opérateurs algébriques relationnels. Soit OP(A1, A2, ..., An), un opérateur ensembliste n-aire sur les ensembles A1, A2, ..., An, dont le résultat est aussi un ensemble.

Cet opérateur est dit U-additif pour le i-ème opérande si l'égalité:

$$OP(A1, ..., Ai-1, B1 \cup B2, Ai+1, ..., An) =$$
$$OP(A1, ..., Ai-1, B1, Ai+1, ..., An)$$
$$\cup \quad OP(A1, ..., Ai-1, B2, Ai+1, ..., An)$$

est vérifiée pour tout couple B1, B2 du i-ème opérande.

Un opérateur est U-additif s'il est U-additif pour tous ses opérandes.

On peut considérer en général l'opérateur de photographie PH(R1, R2, ..., Rn) portant sur des relations Ri comme un opérateur ensembliste. Cette photo est alors U-additive pour R1 si, pour tout couple (A,B) de R1, on a l'égalité:

$$PH(A \cup B, R2, ..., Rn) = PH(A, R2, ..., Rn) \cup PH(B, R2, ..., Rn).$$

Dans le cas d'une insertion dans R1, si la photo est U-additive, on a l'égalité:

$$PH(R1.v \cup R1.a, R2, ..., Rn) = PH(R1.v, R2, ..., Rn) \cup PH(R1.a, R2, ..., Rn).$$

Mais ici, PH(R1.v, R2, ..., Rn) et PH(R1.a, R2, ..., Rn) représentent respectivement l'ancien contenu de la photo avant insertion et celui de la photo ne s'appliquant seulement qu'à l'ensemble des nouveaux tuples de R1. De ce fait, si une photo est U-additive, le U-filtre peut être utilisé. La liste des propriétés de U-additivité pour les opérateurs relationnels est donnée en annexe de ce chapitre.

Un résultat intéressant est que, selon l'expression algébrique qui compose la requête de définition d'une photo, on peut effectivement utiliser cette notion de U-filtre ou non. Soit P une photo dynamique à VS (= un film) dont la requête de définition est REQ(R, R1, R2, ..., Rn) et soit Tr une transaction pour R. Sous certaines conditions, le nouveau contenu de P après la transaction Tr est égal à l'union de l'ancien contenu avec le résultat de REQ(R.a, R1, R2, ..., Rn). C'est-à-dire:

$$\begin{aligned} REQ(R.n, R1, R2, ..., Rn) &= REQ(R.v \ U \ R.a, R1, R2, ..., Rn) \\ &= REQ(R.v, R1, R2, ..., Rn) \\ &\cup REQ(R.a, R1, R2, ..., Rn). \end{aligned}$$

Or, REQ(R.n, R1, R2, ..., Rn) est exactement le nouveau contenu de P après Tr et REQ(R.v, R1, R2, ..., Rn) est son ancien contenu.

3°) Notion de D-filtre.

D'une façon analogue, on peut introduire la notion de D-filtre pour la suppression de tuples dans une relation source d'une photo dynamique à VS. L'idée de base reste la même: au lieu de recalculer toute la photo, on calcule seulement

REQ(R.s, R1, R2, ..., Rn)

où R.s est l'ensemble des tuples supprimés par une transaction Tr, et on retire ce résultat de l'ancien contenu de P. Dans ce cas, on dit que le D-filtre est utilisé pour rafraîchir P immédiatement après la transaction Tr. La démarche est identique à la précédente.

12. Photo de photos.

On n'a considéré jusqu'ici que des photos pour lesquelles les relations sources n'étaient pas des photos. Si toutes les relations sources d'une photo ne sont pas des photos, on dit que P est une photo du premier ordre. La définition d'une photo dynamique non du premier ordre peut porter sur des photos sources dynamiques ou statiques. Or, une photo porte toujours sur les versions courantes des éléments sources. Pour un film P non de premier ordre, chaque rafraîchissement de l'une des sources provoque un rafraîchissement de P, car le rafraîchissement est une opération de mise à jour. De ce fait, si P porte elle-même sur d'autres films, une mise à jour sur la base peut provoquer des rafraîchissements en cascade, ce qui peut dégrader les performances du système. Ici, la méthode de détection des transactions ignorées et celle de U-filtre et de D-filtre peuvent être appliquées afin d'écourter la durée des transactions de ce type. Ces aspects demandent cependant des études plus poussées.

13. Notes: Approche historique de [GADIA 88]

L'approche de [ADIBA 87] modélise les données de manière à associer la partie historique à chaque tuple. Dans ce cas, une modification à un des items du tuple provoque l'enregistrement d'un nouveau tuple où seule la valeur de l'item modifié a changé mise à part la période de validité. Cette solution peut être coûteuse en espace de stockage puisqu'on enregistre la totalité du tuple une nouvelle fois en ne changeant qu'une petite partie du tuple.

Une autre solution existe. Elle consiste à associer la période de validité à chaque item historique(6) du tuple. Cette solution est notamment exposée dans [GADIA 88].

(6) *Un item sera défini comme historique si on veut garder la trace des valeurs antérieures de cet item. Par exemple, dans une base de données où l'on enregistre l'évolution de la carrière des employés d'une entreprise, on définira l'item fonction comme historique, de telle sorte qu'on enregistrera pour chaque employé des couples (fonction, période pendant laquelle l'employé exerce cette fonction). Un item sera défini comme non-historique si cet item ne doit pas refléter une histoire. Par exemple, l'adresse d'un employé sera définie comme non-historique. En effet, on ne s'intéresse dans la plupart des cas qu'à l'adresse courante d'une personne. La valeur de l'item "adresse" sera donc écrasée à chaque changement d'adresse de l'employé, comme dans une base de données classique.*

Lorsque l'on utilise cette dernière modélisation, les items historiques d'un tuple sont alors décomposables et répétitifs.

Les items historiques sont décomposables car on leur adjoint une ou deux valeurs temporelles (au sens large du terme) en plus de leur valeur initiale.

Ces mêmes items sont également répétitifs puisque "toutes autres choses restant égales dans le temps, ces items peuvent avoir plusieurs valeurs associées chacune à une période de validité".

Annexe: propriétés de U-additivité pour les principaux opérateurs relationnels.

Les opérateurs: sélection, projection, jointure, produit cartésien, union et intersection sont U-additifs. La différence et la division sont U-additives pour la première opérande. On a en effet les égalités suivantes:

E1: $(A1 \cup A2)[X] = A1[X] \cup A2[X]$ (projection sur X)

E2: $(A1 \cup A2) : E = (A1 : E) \cup (A2 : E)$ (sélection par condition E)

E3: $(A1 \cup A2) * B = (A1 * B) \cup (A2 * B)$ (jointure)

$A * (B1 \cup B2) = (A * B1) \cup (A * B2)$

E4: $(A1 \cup A2) \times B = (A1 \times B) \cup (A2 \times B)$ (produit cartésien)

$A \times (B1 \cup B2) = (A \times B1) \cup (A \times B2)$

E5: $(A1 \cup A2) \cup B = (A1 \cup B) \cup (A2 \cup B)$ (union)

$A \cup (B1 \cup B2) = (A \cup B1) \cup (A \cup B2)$

E6: $(A1 \cup A2) \cap B = (A1 \cap B) \cup (A2 \cap B)$ (intersection)

$A \cap (B1 \cup B2) = (A \cap B1) \cup (A \cap B2)$

E7: $(A1 \cup A2) \setminus B = (A1 \setminus B) \cup (A2 \setminus B)$ (différence)

E8: $(A1 \cup A2) \div B = (A1 \div B) \cup (A2 \div B)$ (division)

Chapitre 3.
Bases de données temporelles.

Introduction.

Ce troisième chapitre présente une nouvelle grille de différenciation en matière de bases de données incluant des notions de temps. Cette grille permet de créer une classification des divers modèles. Elle permet également de resituer le modèle historique vu au chapitre précédent par rapport aux autres modèles possibles.

Chacun des modèles identifiés est étudié de manière à faire ressortir les principales différences existant entre ceux-ci.

Enfin, une liste de critères permettant d'évaluer la qualité, la simplicité, la performance potentielle d'un modèle quelconque incluant la gestion du temps est proposée.

1. Opposition temps logique - temps physique.

Les divers auteurs ont presque toujours différencié les différentes notions de temps vis-à-vis des bases de données par une séparation temps logique - temps physique. La littérature a d'ailleurs donné plusieurs définitions pour chacune des deux notions. Ces notions ne sont pas toujours reprises telles quelles. En effet, le temps logique a aussi été appelé "*event time*" [COPELAND 84], "*effective time*" [BEN-ZVI 82], "*state*" [CLIFFORD 83], "*valid time*" [SNODGRASS 84], et "*start/end time*" [JONES 79], [JONES 80]. De la même façon, le temps physique a aussi été appelé "*transaction time*" [COPELAND 84], "*registration time*" [BEN-ZVI 82], "*data-valid-from/to*" [MUELLER 83]. Chacun des auteurs définit les termes de manière légèrement différente. Malgré le peu de consensus sur les détails, un accord général peut être admis sur leurs définitions.
Les différences entre temps logique et temps physique identifiées par les auteurs peuvent être caractérisées en termes de trois attributs. Dans les trois prochaines sections, on examinera ces attributs et leurs contributions aux concepts de temps logique et de temps physique. Le tableau qui suivra reprendra en les classifiant les différentes notions de temps utilisées dans la littérature.

1.1. Opposition réalité - représentation.

La correspondance du modèle enregistré dans la base de données avec la réalité est un des aspects utilisés pour distinguer le temps logique du temps physique. Le temps logique est caractérisé par le moment où un évènement se déroule; le temps

physique est caractérisé par le moment où les données concernant l'évènement sont enregistrées dans la base de données. Des exemples simples peuvent être trouvés. Citons les prévisions météorologiques enregistrées dans une bases de données: l'évènement "temps orageux" est enregistré mais n'est pas encore une réalité; une augmentation de salaire annoncée dans le cours du mois n'est enregistrée qu'à la fin du mois dans la base de données, ...

1.2. Flexibilité de mise à jour.

Une autre distinction peut être faite au niveau de la possibilité de mise à jour d'une valeur temporelle (= valeur de type "temps", par exemple, une date).

Une valeur temporelle physique peut être ajoutée à une base de données, mais elle ne peut être modifiée par après. On évoque souvent le concept d'une horloge avançant indéfiniment pour indiquer comment les valeurs temporelles de ce type sont générées.

Par contre, les valeurs temporelles logiques sont toujours sujettes à des changements. Par exemple, on n'enregistre pas toujours une valeur temporelle logique correspondant à la réalité, ceci pour diverses raisons. Cela mène parfois à des incohérences et il faut alors corriger les valeurs erronées.

Cette distinction consiste donc à soit permettre uniquement des ajouts, soit permettre toute modification.

1.3. Dépendance vis-à-vis de l'application.

Le troisième attribut permettant de distinguer le temps logique du temps physique est celui de la dépendance vis-à-vis de l'application.

Le temps logique est généralement caractérisé dans la littérature par sa dépendance vis-à-vis de l'application, alors que le temps physique est considéré comme indépendant vis-à-vis de l'application. Cette distinction est la plus difficile à définir précisément.

En pratique, cette distinction se marque par le contrôle que l'utilisateur a sur le domaine temporel de la valeur enregistrée dans la base de données. Si la valeur peut être calculée automatiquement par le gestionnaire de la base de données, la valeur sera nécessairement indépendante vis-à-vis de l'application et elle aura une sémantique simple. L'exemple le plus simple est le moment où l'information est enregistrée dans la base de données: cette valeur peut être fournie par l'horloge du système tout à fait indépendamment de l'application. Si par contre le domaine de valeurs des valeurs temporelles est définie par l'utilisateur de manière explicite,

- ⇨ les valeurs doivent aussi être spécifiées par l'utilisateur,
- ⇨ l'utilisateur doit gérer lui-même l'intégrité de ces valeurs,

- ⇨ les valeurs temporelles doivent pouvoir être modifiées par l'utilisateur lorsque des "*décalages*" sont découverts entre la réalité et le contenu la base de données.

Dans ce cas, les valeurs temporelles (logiques) sont dépendantes de l'application.

1.4. Discussion.

Les deux premiers attributs définis aux points 1.1 et 1.2 sont des concepts relativements précis. Ils sont aussi fortement liés l'un l'autre de par le fait qu'une valeur qui enregistre le moment où les données sont stockées ne peut être modifiée par après.

Le troisième attribut (point 1.3) par contre n'est malheureusement pas applicable de manière rigoureuse. Il force certaines conclusions parmi lesquelles la plus cruciale est que toutes les actions exécutées par le gestionnaire de la base de données sont indépendantes vis-à-vis de l'application. Cette affirmation n'est pas acceptable. D'une part, le schéma de la base de données, qui oriente la plupart des actions du gestionnaire est en toute logique dépendant de l'application. D'autre part, la plupart des systèmes de gestion de bases de données permettent à l'utilisateur de spécifier des contraintes d'intégrité qui sont aussi dépendantes de l'application bien qu'interprétées automatiquement par le système sans intervention de l'utilisateur.

Référence	Terminologie	Ajout seulement	Dépendance par % application	Représentation
[ARIAV 82]	Time	Oui	Oui	Représentation
[BEN-ZVI 82]	Registration Effective	Oui Non	Oui Oui	Représentation Réalité
[CLIFFORD 83]	State	Non	Oui	
[COPELAND 84]	Transaction Event (1)	Oui Non	Oui Non	Représentation Réalité
[DADAM 84] & [LUM 84]	Physical Logical (1)	(2) Non	Oui Non	Représentation Réalité
[JONES 79] & [JONES 80]	Start/End User Defined	(2) Non	Oui Non	Réalité Réalité
[MUELLER 83]	Data-Valid- Time-From/To	(3)	Oui	Représentation (4)
[REED 78]	Start/End	Oui	Oui	Représentation
[SNODGRASS 84]	Valid Time	Non	Oui	Réalité

Notes:
(1) Pas supporté par le système.
(2) Seules les corrections sont permises.
(3) Seules les modifications concernant le futur sont acceptées.
(4) La réalité n'est indiquée que dans le futur.

Figure 3.1. Types de temps.

2. Nouvelle classification et modèle temporel.

2.1. Introduction.

Richard Snodgrass [Snodgrass 85] présente une nouvelle manière de caractériser les différents types de bases de données incluant la notion de temps. Cette nouvelle caractérisation n'est plus basée sur l'opposition temps logique/temps physique. Elle

est plutôt basée sur l'opposition réalité/représentation, et peut être aisément interprétée graphiquement, ce qui en facilite la compréhension.

2.2. Bases de données statiques.

Bien que le monde réel soit dynamique, les bases de données conventionnelles modélisent le monde réel par une photo unique associée à un point particulier dans le temps. Un **état** (ou une **instance**) d'une base de données est son contenu courant, contenu qui ne reflète pas nécessairement l'état courant du monde réel.

La mise à jour de l'état d'une base de données est effectuée en utilisant des opérations de manipulation telles que insertion, suppression ou remplacement, ne prenant effet que lorsque la manipulation entière a été entérinée (c'est-à-dire, au "commit transaction"(7)). Avec ce processus, les états antérieurs de la base de données sont exclus et complètement oubliés. Ces bases de données sont appelées 'statiques'.

Dans le modèle relationnel, une base de données est une collection de relations. Chaque relation consiste en un ensemble de tuples ayant chacun le même ensemble d'attributs. Une relation est couramment représentée par un tableau à deux dimensions. Si des changements se produisent dans le monde réel, les changements sont répercutés dans cette table.

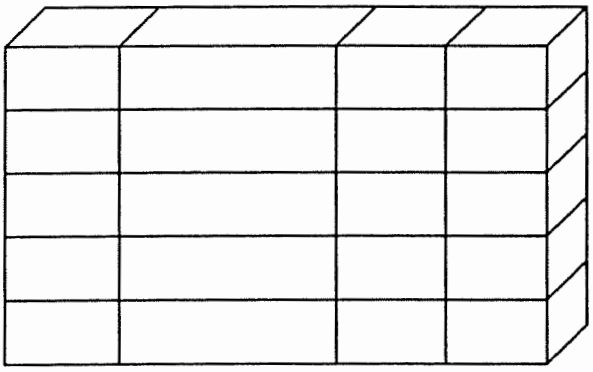


figure 3.2. Une relation statique.

(7) *Ce "commit transaction" est dans certains cas explicite, dans d'autres implicite.*

Par exemple, une instance de la relation "**Faculté**" peut être à un certain moment:

Nom	Fonction
Jean	Professeur
Pierre	Chargé de cours

figure 3.3. Une instance de la relation **Faculté**.

Une requête demandant l'affectation de Jean en Quel (language de query basé sur le "tuple calculus" développé pour le SGBD INGRES) serait:

```
range of f is Faculté
retrieve (f.fonction)
        where f.nom = "Jean"
```

Requête 3.1.

et produirait comme résultat:

Nom
Jean

figure 3.4. Résultat de la requête 3.1.

Seulement, dans bien des situations, une telle relation statique basée sur les photos est inadaptée. Par exemple, on ne saura pas répondre à des questions comme:

Quel était le statut de Jean il y a deux ans? (requête historique)

Comment le nombre d'employés du département a-t-il évolué au cours des 5 dernières années? (analyse de tendance, d'évolution).

On ne saura pas non plus enregistrer des faits tels que:

> Dupont a été promu chef de service avec effet à partir du mois passé. (modification rétroactive)
>
> Durand rejoindra le département commercial le mois prochain. (modification dans le futur ou "postactive").

Sans aide de la part du système, beaucoup d'applications doivent maintenir et traiter l'information temporelle d'une manière ad hoc.

2.3. Bases de données statiques Rollback.

> Avertissement: Avant d'aller plus loin, il est judicieux de bien situer le terme "***rollback***". En effet, celui-ci n'a pas ici la même acception que celle qu'il a lorsque l'on parle de "***commitment control***". Dans ce dernier domaine, l'opération "rollback" sert à annuler l'effet d'une transaction plus ou moins complexe lorsqu'un problème empêche la bonne fin de celle-ci. Dans le domaine du temps, l'opération "rollback" permet de faire référence à un état antérieur d'une base de données. Ceci sera expliqué en détails ci-après.

Une approche pour résoudre les déficiences ci-dessus est de stocker tous les états passés de la base de données statique, indexés sur le temps. Une telle approche requiert une représentation du temps transactionnel, c'est-à-dire le moment où l'information a été stockée dans la base de données. Une relation sous cette approche peut être illustrée en trois dimensions (figure 3.5) avec le temps transactionnel comme troisième axe. La relation peut être perçue comme une séquence de relations statiques indexées sur le temps. En se déplaçant le long de l'axe du temps, et en prenant une tranche verticale du parallélépipède, il est possible de sélectionner une photo de la relation et effectuer des requêtes sur celle-ci. L'opération qui consiste à prendre une tranche verticale est appelée "rollback", et une base de données supportant cela est appelée "base de données statique rollback". Les modifications sur une telle base de données ne peuvent être faites que sur l'état statique le plus récent.

La relation illustrée dans la figure 3.5 a été l'objet de trois transactions: en partant de la relation "vide" ou "nulle",

- l'addition de trois tuples,
- l'addition d'un tuple,
- la suppression d'un tuple (entré dans la première transaction) et l'addition d'un autre tuple.

Chaque transaction a pour résultat une nouvelle relation statique étant ajoutée sur la face frontale du parallélépipède. Une fois la transaction terminée, les relations dans la relation statique rollback ne peuvent plus être altérées.

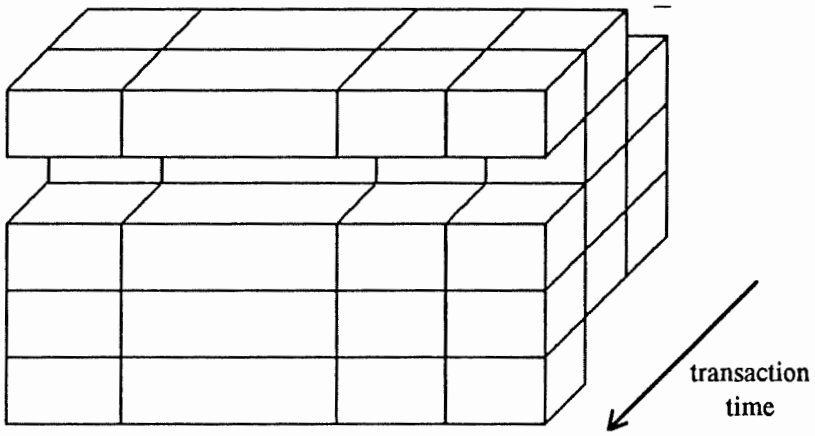


figure 3.5. Une relation statique rollback

Une limite du fait d'utiliser le temps transactionnel est que l'on enregistre l'histoire de la base de données plutôt que l'histoire du monde réel. Un tuple devient valide aussitôt qu'il entre dans la base de données (comme dans une base de données statique). Il n'y a pas non plus moyen d'enregistrer des modifications dans le passé ou dans le futur, ni de corriger des erreurs dans des tuples "passés". Les erreurs peuvent parfois être corrigées (si elles se trouvent dans l'état courant), mais elles ne peuvent pas être oubliées.

Implémenter une relation statique rollback de cette manière n'est pas pratique car il y a beaucoup trop de données dupliquées: les tuples qui ne changent pas entre les états doivent être recopiés dans le nouvel état.

Une autre approche ajoute les temps de début et de fin de la transaction à chaque tuple, indiquant par là le moment où le tuple faisait partie de la base de données. Un exemple typique de cette approche est illustré dans la figure 3.6. La double barre verticale sépare les domaines non-temporels des domaines temporels maintenus par la base de données. Ces derniers domaines n'apparaissent pas dans le schéma de la relation, mais doivent plutôt être considérés comme une partie de l'overhead associé à chaque tuple. Notons le fait que Jean était précédemment "chargé de cours", un fait que l'on ne pouvait pas exprimer dans une base de données statique.

Nom	Fonction	Transaction time (start)	(stop)
Jean	Chargé de cours	25/08/84	15/12/89
Jean	Professeur	15/12/89	∞
Pierre	Chargé de cours	07/12/89	∞
Michel	Assistant	10/01/90	25/02/91

figure 3.6. Une relation statique rollback.

N'importe quel langage de query peut être converti de manière à pouvoir effectuer des requêtes sur une base de données statique rollback en ajoutant une clause effectuant le rollback. TQuel (Temporal QUEry Language) [SNODGRASS 84] [SNODGRASS 87], une extension de Quel pour les bases de données temporelles, ajoute la clause "as of" à la requête afin de spécifier le temps transactionnel. La requête TQuel

```
range of f is Faculté
retrieve (f.fonction)
        where f.name = "Jean"
        as of "10/12/89"    (5 jours avant sa nomination)
```

Requête 3.2.

sur la relation "Faculté" de la figure 3.6 trouvera la fonction de Jean au 10/12/89:

Fonction
Chargé de cours

figure 3.7. Résultat de la requête 3.2.

Notons que le résultat d'une requête sur une base de données statique rollback est une pure relation statique.

Le concept de temps transactionnel est apparu dans plusieurs systèmes, par exemple: GemStone [COPELAND 84], MDM/DB (Model Data Management / Database) [ARIAV 82].

2.4. Bases de données historiques.

Alors que les bases de données statiques rollback enregistrent une séquence d'états statiques, les bases de données historiques enregistrent un seul état historique par relation. Lorsque des erreurs sont découvertes, elles sont corrigées en modifiant la base de données. Les états précédents ne sont pas retenus, de telle manière qu'il n'est pas possible de voir la base de données comme elle était à un moment précis du passé. Il n'y a aucune trace des corrections enregistrées non plus. Dans ce sens, les bases de données historiques sont similaires aux bases de données statiques. Les bases de données historiques doivent représenter le temps "valide", c'est-à-dire le temps où l'information stockée modélise la réalité. Les bases de données historiques peuvent aussi être illustrées en trois dimensions.

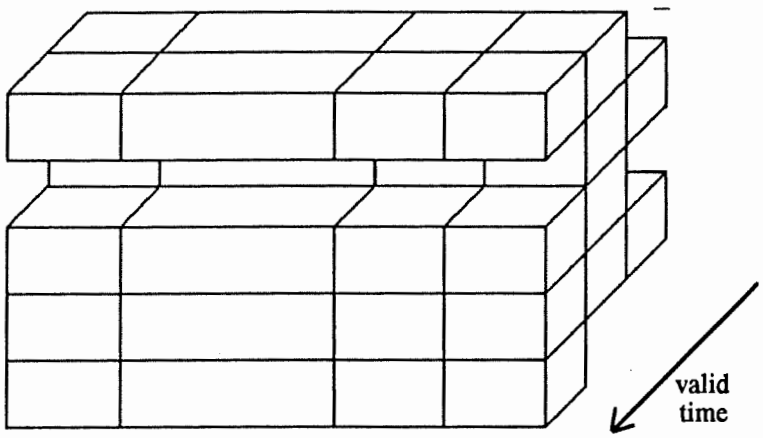


figure 3.8. Une relation historique.

La même séquence de transactions qui a donné comme résultat la relation statique rollback de la figure 3.5 donne également comme résultat la relation historique de la figure 3.8. Cependant, une transaction ultérieure (impossible sur une relation statique rollback) a supprimé un tuple incorrect inséré lors de la première transaction.

Bien que son illustration semble fort similaire à celle d'une base de données statique rollback, l'intitulé de l'axe du temps a été modifié, et sa sémantique est maintenant plus proche de la réalité (au lieu de l'historique de mise à jour). Pour cela, des opérations plus sophistiquées (que celles relatives aux bases de données statiques

rollback) sont nécessaires pour manipuler de manière adéquate la sémantique complexe du temps valide.

Une seconde distinction entre bases de données historiques et statiques rollback est que les gestionnaires de bases de données historiques acceptent des modifications arbitraires, alors que les gestionnaires de bases de données statiques rollback n'autorisent que l'ajout d'états statiques supplémentaires. La même séquence de transactions qui donnait la relation statique rollback de la figure 3.5 donne quasiment le même résultat dans la relation historique de la figure 3.8. Seulement, une transaction supplémentaire (impossible dans une base de données statique rollback) a supprimé un tuple erroné qui avait été inséré dans la première transaction. Un gestionnaire de bases de données statiques rollback peut revenir à une précédente relation statique incorrecte; un gestionnaire de bases de données historiques sait enregistrer la connaissance courante à propos du passé.

Les bases de données historiques incorporent aussi la notion de temps "user-defined", dont on parlera dans le contexte des bases de données temporelles. Aussi bien le temps valide que le temps "user-defined" concernent la modélisation de la réalité, et il est donc normal qu'ils apparaissent ici.

Les bases de données historiques requièrent des langages de query plus évolués. LEGOL 2.0 [JONES 79], basé sur l'algèbre relationnelle, et TQuel [SNODGRASS 84], basé sur Quel sont deux des langages les plus connus développés pour effectuer des requêtes sur des bases de données historiques. TQuel permet l'expression de requêtes historiques en ajoutant à la requête une clause "valid" pour spécifier comment doit être calculé le domaine du temps implicite, et un prédicat "when" pour spécifier la relation temporelle des tuples participant à une dérivation. Ces constructions traitent des relations temporelles complexes telles que "*start of*", "*precede*" et "*overlap*".
Comme pour les bases de données statiques rollback, l'implémentation d'une relation historique comme décrite précédemment n'est pas pratique. Il existe également (voir bases de données statiques rollback) une alternative: en ajoutant les débuts et fins de temps valide à chaque tuple, on indique les points du temps où le tuple modélise effectivement la réalité (figure 3.9). De manière analogue au temps transactionnel dans les bases de données statiques rollback, le temps valide n'est pas inclus dans le schéma de la relation.

Nom	Fonction	Valid time (from)	(to)
Jean	Chargé de cours	01/09/84	01/12/89
Jean	Professeur	01/12/89	∞
Pierre	Chargé de cours	05/12/89	∞
Michel	Assistant	01/01/90	01/03/91

figure 3.9. Une relation historique.

La requête TQuel demandant la fonction de Jean lorsque Pierre est arrivé,

```
range of f1 is faculté
range of f2 is faculté

retrieve(f1.fonction)
        where f1.nom = "Jean"
        and f2.nom = "Pierre"
        when f1 overlap start of f2
```

Requête 3.3.

sur la relation historique 'faculté' de la figure 3.9 a pour résultat:

Fonction	Valid time (from)	(to)
Assistant	01/01/90	01/03/91

figure 3.10. Résultat de la requête 3.3.

Notons que la relation dérivée est aussi une relation historique, qui peut être réutilisée pour des requêtes ultérieures. Alors que la requête ci-dessus et l'exemple donné pour la relation statique rollback semblent demander la fonction de Jean au 05/12/89, les réponses sont différentes. La raison est que Jean a été promu professeur au 01/12/89 mais que cette information n'a été enregistrée dans la base de données que 2 semaines plus tard. La base de données était donc incohérente avec la

réalité durant ces deux semaines. Dans la base de données historique, l'erreur a été corrigée, mais il n'est pas possible de déterminer que, même pour une courte période, la base de données était incohérente (vis-à-vis de la réalité).

Beaucoup de recherches ont été effectuées sur les bases de données historiques. Entre autres résultats, citons CSL (Conceptual Schema Language) [BREUTMANN 79], TERM (Time-extended Entity Relationship Model) [KLOPPROGGE 81], la logique intentionnelle (IL) [CLIFFORD 83].

2.5. Bases de données temporelles.

Les bénéfices des deux précédentes approches peuvent être combinés afin d'obtenir un modèle supportant à la fois le temps transactionnel et le temps valide.

Une base de données statique rollback voit les tuples valides à un moment donné par rapport à ce moment là, c'est-à-dire sans tenir compte des corrections enregistrées après. Une base de données historique voit les tuples valides à un moment donné par rapport au moment présent.

Un gestionnaire de bases de données temporelles permet de voir les tuples valides à un moment donné par rapport à un quelconque autre moment, prenant ainsi en compte l'historique des modifications rétroactives et postactives.

On utilise le terme base de données temporelle pour bien marquer le double besoin des temps transactionnel et valide dans le traitement de l'information relative au temps.

Comme on utilise ici deux temps distincts, on aura besoin de deux axes du temps distincts, et on aura des illustrations en quatre dimensions (figure 3.11).

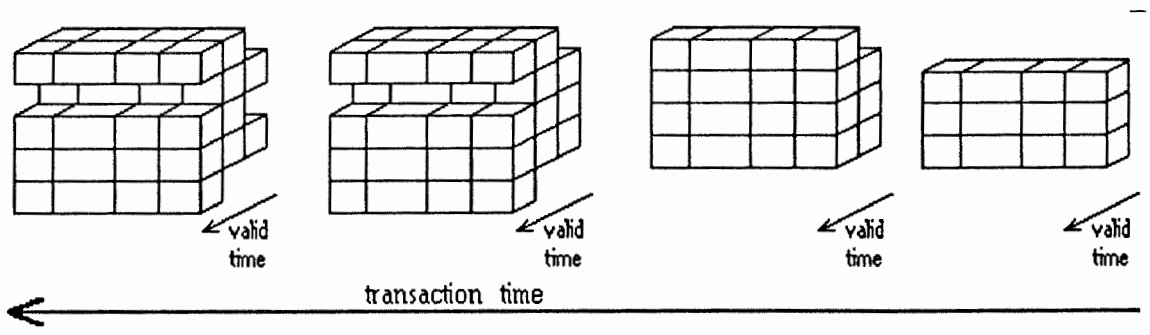


Figure 3.11. Une relation temporelle.

Une relation temporelle peut être considérée comme une séquence d'états historiques, chacun de ces états étant lui même une relation historique complète. L'opération de rollback sur une relation temporelle sélectionne un état historique particulier, sur lequel on peut effectuer des requêtes à caractère historique (c'est-à-dire que l'on peut considérer l'état historique particulier comme une simple relation historique).

Chaque transaction a pour effet de créer un nouvel état historique. Dès lors, les relations temporelles sont "append only". La relation de la figure 3.11 est le résultat de quatre transactions partant de la relation vide:

(1) trois tuples ont été ajoutés,

(2) un tuple a été ajouté,

(3) un tuple a été ajouté et un autre (préexistant) a été effacé, et

(4) un tuple (préexistant) a été effacé (car il ne devait probablement pas se trouver là lors de la première transaction).

La relation de la figure 3.9 ressemblera à la relation de la figure 3.12 après ajout du temps transactionnel. Cette dernière montre que:

* Jean a commencé à travailler le 01/09/84, information qui a été enregistrée dans la base de données le 25/08/84 (modification postactive). Jean a été promu professeur le 01/12/89, mais l'information n'a été enregistrée que le 15/12/89.

* L'arrivée de Pierre a été enregistrée dans la base de données le 01/12/89. A cette date, on lui a attribué (dans la base de données) le statut de professeur; le fait qu'il soit chargé de cours et non pas professeur a été enregistré le 07/12/89.

* Michel a quitté la faculté le 01/03/91, fait qui a été enregistré le 25/02/91.

Nom	Fonction	Valid time (from)	Valid time (to)	Transaction time (start)	Transaction time (stop)
Jean	Chargé de cours	01/09/84	∞	25/08/84	15/12/89
Jean	Chargé de cours	01/09/84	01/12/89	15/12/89	∞
Jean	Professeur	01/12/89	∞	15/12/89	∞
Pierre	Professeur	05/12/89	∞	01/12/89	07/12/89
Pierre	Chargé de cours	05/12/89	∞	07/12/89	∞
Michel	Assistant	01/01/90	∞	10/01/90	25/02/91
Michel	Assistant	01/01/90	01/03/91	10/01/90	∞

figure 3.12. Une relation temporelle.

La requête TQuel

```
range of f1 is faculté
range of f2 is faculté

retrieve(f1.fonction)
        where f1.nom = "Jean"
        and f2.nom = "Pierre"
        when f1 overlap start of f2
        as of "10/12/89"
```

Requête 3.4.

sur cette relation détermine la fonction de Jean quand Pierre est arrivé par rapport à l'état de la base de données au 10/12/89. Le résultat est:

Fonction	Valid time (from)	Valid time (to)	Transaction time (start)	Transaction time (stop)
Chargé de cours	01/09/84	∞	25/08/84	15/12/89

Figure 3.13. Résultat de la requête 3.4.

Cette relation dérivée est aussi une relation temporelle, de sorte que d'autres relations temporelles peuvent en être déduites. Si la requête avait spécifié "as of

20/12/89", la réponse aurait été "professeur", car la modification a eu un effet rétroactif.

TRM (Time Relational Model) [BEN-ZVI 82] est un exemple de base de données temporelle.

2.6. Le temps "user-defined"

Le temps "user-defined" [JONES 80] est nécessaire quand de l'information temporelle, non traitée par le temps transactionnel ni par le temps valide, est stockée dans la base de données.

Par exemple, considérons la relation promotion de la figure 3.14. Comme c'est une relation enregistrant des évènements, un seul temps valide est nécessaire. La colonne "date effective" enregistre la date à laquelle la promotion devient effective. Le temps valide est la date à laquelle la lettre de promotion a été signée, le temps transactionnel ("transaction start time") est la date à laquelle l'information concernant la promotion a été enregistrée dans la base de données.

Nom	Fonction	Date effective	Valid time (at)	Transaction time (start)	(stop)
Jean	Chargé de cours	01/09/84	25/08/84	25/08/84	∞
Jean	Professeur	01/12/89	11/12/89	15/12/89	∞
Pierre	Professeur	05/12/89	05/12/89	01/12/89	07/12/89
Pierre	Chargé de cours	05/12/89	07/12/89	07/12/89	∞
Michel	Assistant	01/01/90	01/01/90	10/01/90	∞
Michel	Départ	01/03/91	25/02/91	25/02/91	∞

figure 3.14. Une relation temporelle "évènement" (event relation).

Ce temps "user-defined" est spécifique à l'application considérée. Les valeurs des domaines temporels "user-defined" ne sont pas interprétées par le gestionnaire de la base de données, et sont donc les plus faciles à supporter; tout ce qui est nécessaire, ce sont d'une part une représentation interne et d'autre part des fonctions d'input/output.
Une version expérimentale de INGRES [OVERMYER 82] supporte la notion de temps "user-defined".

3. Critères d'appréciation d'un modèle incluant le temps.

L.E. McKenzie & R. Snodgrass [MCKENZIE 91] proposent une liste de critères destinée à évaluer les extensions temporelles aux modèles dits "snapshot". En voici les principaux:

⇨ "Toutes les valeurs d'attributs d'un tuple sont définies pour le(s) même(s) intervalle(s) (de temps)".

Cela signifie que si un attribut quelconque du tuple possède une valeur pour un instant particulier, alors, tous les atrributs du tuple doivent posséder aussi une valeur associée à ce même instant.

⇨ "Une extension temporelle doit être stable par rapport au modèle "snapshot" dont elle est issue".

Autrement dit, le modèle étendu doit être au moins aussi puissant que le modèle de base.

⇨ "Le modèle temporel supporte le concept de périodicité".

En d'autres mots, il devrait être capable de représenter des phénomènes périodiques tels que "tous les mois", "chaque lundi à 8h30", "tous les jours entre 8h30 et 17h" sans devoir spécifier chacune des occurrences. Il devrait également proposer des opérateurs capables de manipuler directement de telles données périodiques.

⇨ "Tout groupe de valeurs d'attributs valides tirés des domaines de valeurs appropriés forme un tuple valide".

Ou encore: dans un tuple, la valeur ou la composante "temps valide" de l'attribut ne devrait pas restreindre arbitrairement la valeur ou la composante "temps valide" d'un autre attribut d'un tuple.

⇨ "Tout ensemble de tuples valides forme une relation valide".

⇨ "La sémantique formelle est bien définie".

Chacun des objets et opérations sont définis mathématiquement de manière concise (Le modèle "snapshot" possède une telle sémantique formelle.

⇨ "Le modèle supporte des timestamps multidimentionnels".

Par exemple, on peut vouloir enregistrer la date et l'heure auxquelles le décollage d'un avion est prévue ainsi que la date et l'heure effectives du décollage.

⇨ "Le modèle est réductible au modèle "snapshot"".

Pour tout opérateur temporel, la relation "snapshot" obtenue en appliquant l'opérateur temporel à une relation temporelle et en prenant une photo (snapshot) du résultat devrait être équivalente à la relation obtenue en prenant une photo de la relation temporelle et en y appliquant l'opérateur snapshot analogue.

⇨ "Le modèle restreint les relations à la première forme normale"

⇨ "Le modèle supporte les relations des quatres classes définies au chapitre 3, à savoir snapshot, rollback, historique et temporelle".

⇨ "Le modèle supporte le concept de schémas multiple pour une même relation".

Si le schéma de la relation évolue au cours de l'exploitation de la base de données, il doit être possible d'accéder aux données anciennes sur base du schéma de la relation en vigueur durant la période concernée par la requête.

⇨ "Le modèle supporte les attributs statiques".

Une relation doit pouvoir être constituée à la fois d'attributs dont la validité est restreinte par le temps et d'attributs dont la validité n'est pas restreinte par le temps.

⇨ "Les traitements des temps valides et temps transactionnels doivent être tout à fait indépendants".

Une opération impliquant un des deux aspects ne doit pas affecter arbitrairement l'autre aspect.

⇨ "L'estampille est donnée au niveau du tuple plutôt qu'au niveau des attributs".

⇨ "Une représentation unique doit être d'application"

Ces critères constituent à mon sens un des moyens les plus poussés permettant de comparer les différentes approches en matière de temps dans les bases de données relationnelles. Dans un travail ultérieur, il serait intéressant de comparer différentes approches sur base de ces critères.

Chapitre 4.
TEMPS + DYNAMIQUE + REFERENTIEL = HISTOIRE.

Introduction.

Dans ce chapitre, j'analyse un modèle qui combine deux approches différentes qui intègrent chacune des notions temporelles (au sens large du terme). Chacune des deux approches constitue une extension d'un système de gestion de base de données traditionnel. La première, l'approche historique, intègre le temps dans le modèle de données, tandis que la seconde, l'approche dynamique, spécifie les interactions entre les traitements et les données. Ces deux approches ont leurs limites respectives. L'intérêt revient néanmoins lorsqu'on se rend compte que ces deux approches peuvent être complémentaires. En effet, en intégrant les deux approches, on parvient à faire sauter leurs limitations. C'est dans ce cadre que le modèle REVI [BURSENS 87], ainsi que son langage d'interrogation ont été étudiés.

Parmi les travaux sur le temps dans les bases de données, le modèle REVI apparaît actuellement comme un des modèles les plus évolués, si pas le plus évolué. Malheureusement, il est le seul modèle de ce type. On ne possède donc pas de point de comparaison avec d'autres modèles apparentés.

1. L'approche historique.

Elle a déjà fait l'objet d'une étude approfondie et ne sera plus introduite ici.

2. L'approche dynamique.

2.1. Les modèles de traitements.

Cette seconde approche se démarque nettement de celle qui conduit aux bases de données historiques. En effet, elle s'intéresse aux interactions entre les données, les traitements, et les contraintes d'intégrité au cours du temps. Le développement de l'approche dynamique a pour objectif l'amélioration de la spécification de la base de données, ainsi que la vérification et la validation de cette dernière.

Lors de la conception d'une application base de données, l'approche dynamique permet de spécifier les contraintes d'utilisation appartenant au champ d'application considéré. Le concepteur peut ainsi obtenir des réponses aux questions suivantes:

- Quel est l'intervalle de disponibilité de chaque donnée ?
- Quelles sont les données consommées et produites par les traitements ?
- Quelles sont les contraintes de synchronisation entre les traitements, et celles-ci ne mènent-elles pas à des situations d'interblocage ?
- Quelles sont les périodes pendant lesquelles il est pertinent de valider les contraintes d'intégrité et d'autoriser les traitements ?

Pour répondre à ces questions, les modèles de données (relationnel ou entité-association), même s'ils sont étendus, ne suffisent pas. Par exemple, deux bibliothèques pourraient avoir une modélisation identique de leurs données, mais des contraintes complètement différentes, dues à des règlements et à une manière de traiter l'information propres à chacune de ces bibliothèques. De plus, l'étude des questions précédentes nécessite l'introduction d'une notion temporelle.

Les travaux effectués dans ce domaine se sont orientés dans diverses directions. Un des travaux est notamment la méthode vue au cours de F. BODART, qui a été élaborée par F. BODART et Y. PIGNEUR [BODART 83]; méthode débouchant sur le projet IDA, implémentant celle-ci. La méthodologie présentée ici est MTG [GUYOT 86]. Ce modèle intègre des notions de temps, de synchronisation des traitements et un concept d'état dynamique. Pour spécifier et vérifier les différentes contraintes de synchronisation, ce modèle utilise les réseaux de Pétri.

Voici un bref résumé expliquant les cinq concepts du modèle (ce sera utile pour la compréhension de la suite):

- Les relations définissent les objets de l'application qui doivent être mémorisés dans la base de données.
- Les contraintes d'intégrité permettent d'énoncer les propriétés régissant les objets de l'application.
- Les traitements sont associés à la modification des valeurs des objets.
- Les évènements décrivent les conditions de synchronisation des traitements.
- Les périodes définissent l'exploitation de la base de données. Chacune est caractérisée par un ensemble de contraintes d'intégrité et d'évènements qui lui est propre.

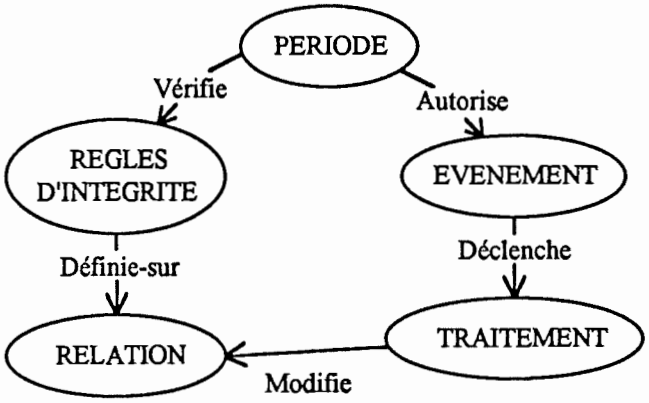


figure 4.1. Les concepts de MTG et leurs interactions.

2.2. Les états dynamiques.

Le modèle MTG permet d'associer un état dynamique à une entité d'une relation. C'est ce concept qui est développé ci-après.

Par exemple, un contribuable fait un recours contre l'imposition dont il est l'objet. Il doit être reconnu par le système comme suspendu temporairement (par rapport au service d'encaissement des impôts) tant que son recours n'est pas tranché. Avec les SGBD classiques, le concepteur doit déclarer un constituant "ETAT" qui lui permettra de marquer et de suivre l'évolution dynamique d'une entité dans le système d'information. Dans MTG, ceci est rendu implicite par l'utilisation d'états dynamiques qui sont propres à un schéma de relation mais qui indiquent, pour chaque entité, son état spécifique.

Ces états sont implicitement définis lors de la spécification des traitements. Dans l'entête de la déclaration d'un traitement, le concepteur spécifie les états des entités qui sont consommées en entrée par le traitement ainsi que les états des entités qui sont produites en sortie. Le traitement enregistrant le recours d'un contribuable est défini de la façon suivante:

```
TRAITEMENT Enregistrement_de_recours
   (ENTREE Recours_possible : contribuable;
   SORTIE En_recours : contribuable)

   (* fait passer le contribuable de l'état Recours_possible
         à l'état En_recours. *)
   . . .
FIN DESCRIPTION
```

Recours_possible et En_recours sont des ensembles d'entités associés aux états précités ayant pour schéma de relation 'Contribuable'. Pour une relation donnée, on peut rechercher toutes les entités qui sont dans un état donné. Ensuite, en examinant les entrées et les sorties des traitements, il est possible de construire le graphe dynamique d'une relation; c'est un réseau de Pétri dont les places sont des états d'une même relation et dont les transitions sont des traitements. Une place est en entrée/sortie d'une transition si l'état spécifique est en entrée/sortie du traitement.

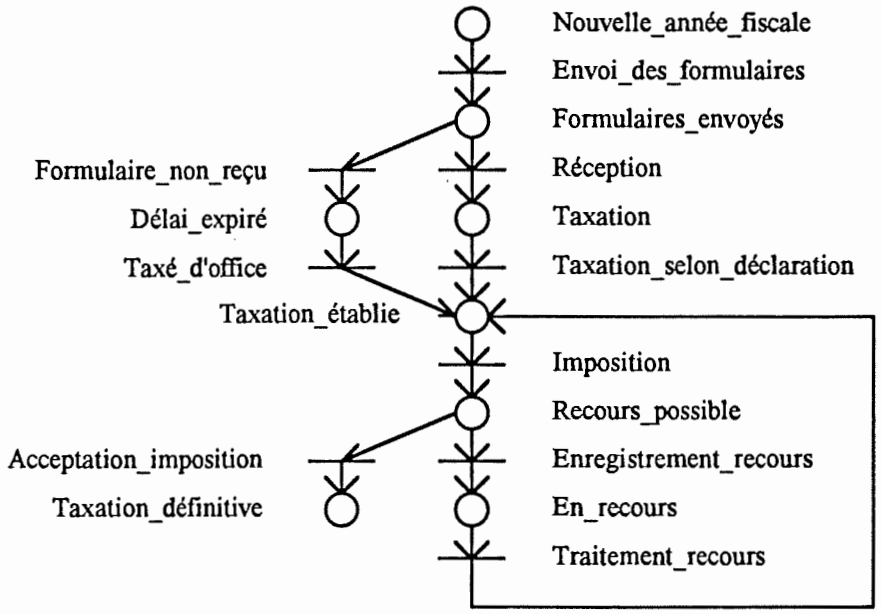


figure 4.2. graphe d'états dynamiques pour la relation Contribuable.

2.3. Utilité des états dynamiques.

L'état dynamique indique donc l'avancement d'une entité dans un processus de traitement. La notion temporelle qui est attachée à l'entité par l'intermédiaire de son état dynamique n'est plus absolue par rapport à une date (comme c'est le cas dans les bases de données historiques), mais relative aux processus de traitement des informations d'une organisation.

Quelles peuvent être les utilisations possibles des états dynamiques:

- Les états dynamiques simplifient la spécification des traitements; en effet, il est souvent plus simple de désigner un sous-ensemble d'entités devant subir un même traitement par leur état dynamique que par une requête de sélection portant sur leur constituants.

- La déclaration automatique de constituants d'états dynamiques évite au concepteur de rajouter des constituants remplissant le même rôle durant la modélisation des données.
- L'utilisateur de la base de données peut aussi s'en servir pour répondre à des questions du type "Il y a deux mois, j'ai fait telle démarche administrative; depuis, je n'ai pas eu de nouvelles; où en est mon dossier ?". Dans une base de données classique, il est très difficile de répondre à de telles questions. On trouvera les données du dossier, mais aucun indicateur d'avancement de celui-ci dans les procédures administratives. Vraisemblablement, il faudra s'adresser à la personne qui est en charge du dossier.

L'implantation des états dynamiques se fait en ajoutant un constituant d'état à chaque relation. Le domaine du constituant est défini par les différents états explicités dans le graphe d'états dynamique de cette relation.

3. Limitations du modèle historique.

Le modèle historique permet la représentation, dans le temps, d'une succession identifiable de valeurs qui ont été prises par une entité donnée. Cependant, avec cette représentation, on ne peut interpréter, pour une entité, les causes de transition d'une valeur prise au temps ti à celle prise au temps ti+1. A un instant donné, les seuls états dynamiques représentés dans une base de données historique sont:

① L'entité n'a jamais existé (elle n'a pas été créée).

② L'entité existe et est active actuellement.

③ L'entité a cessé d'exister (détruite logiquement).

Ces états dynamiques se réfèrent aux primitives [créer], [détruire] du SGBD historique, mais ne permettent pas d'exprimer les causes des changements de valeurs de ces entités.

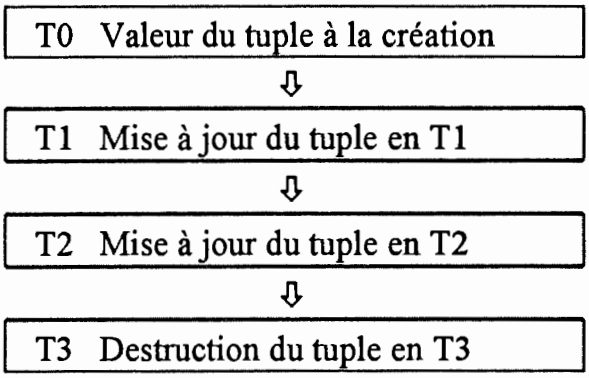


figure 4.3. Les valeurs successives dans un modèle historique.

4. Limitation du modèle dynamique.

Dans un modèle dynamique, la base de données exprime, par son contenu, le référentiel d'exploitation pour la période courante, ainsi que l'état dynamique actuel de chaque entité. Néanmoins, ceci présente les désavantages suivants:

- on ne peut pas expliquer pourquoi une entité se trouve dans un état dynamique donné, et quels sont les traitements qu'ils l'ont modifiée.
- Plusieurs séquences de traitements sont susceptibles d'être utilisées pour atteindre un état dynamique donné. L'état dynamique d'une entité ne détermine pas l'histoire de celle-ci. En fait, les entités dynamiques contrôlent les comportements futurs des entités par rapport au processus de traitement de l'application.

Remarques:

Dans les deux modèles, il est impossible de tenir compte de l'évolution de l'exploitation. En effet, dans l'hypothèse où une base de données historique ou une base de données dynamique sont utilisées sur une longue période, le champ d'application évolue et par conséquent, les relations, les contraintes d'intégrité ainsi que les traitements sont modifiés. L'évolution de l'application doit être associée aux entités si l'on veut interpréter correctement les valeurs de celle-ci. Par exemple, la vue permettant de trouver les contribuables non imposables peut changer en fonction des règlementations ou d'un barème annuel.

Cette notion d'évolution va être introduite dans REVI.

5. REVI.

5.1. Les concepts du modèle REVI.

Valeur: Elles sont prises par une entité pour les différents constituants.

Instant: C'est la date à partir de laquelle une instance d'entité est définie par une valeur ou un ensemble de valeurs propre à l'instant. Une relation d'ordre stricte est définie sur les dates.

Etat: Il s'agit de l'état d'une entité par rapport aux processus de traitement de l'application qui forment les places des graphes des relations considérées.

Référentiel: Un référentiel correspond à une modélisation de l'application et porte sur l'ensemble des concepts du modèle: Période, évènements, règles d'intégrité, relations, traitements.

Une interprétation cohérente des entités est réalisée en se basant sur le référentiel auquel ces dernières appartiennent. Dès qu'une partie de la structure est modifiée, le référentiel change. Il est donc indispensable d'associer à chaque entité, son référentiel d'exploitation.

Le référentiel ne doit pas être confondu avec la période: cette dernière caractérise l'exploitation de la base de données en associant un ensemble de règles d'intégrité et d'évènements qui lui est propre pour un intervalle de temps spécifique. Le référentiel définit l'application par rapport à l'ensemble des concepts {période, évènements, règles d'intégrité, relations, traitements} tant que ces derniers restent inchangés.

Un nouveau référentiel est défini dès qu'une modification est apportée aux modélisations basées sur les concepts précédents. Avec le référentiel, on pourra déterminer une interprétation cohérente des valeurs historiques, tandis qu'avec la période, on pourra définir les états et les transformations cohérentes de la base de données dans le référentiel applicable.

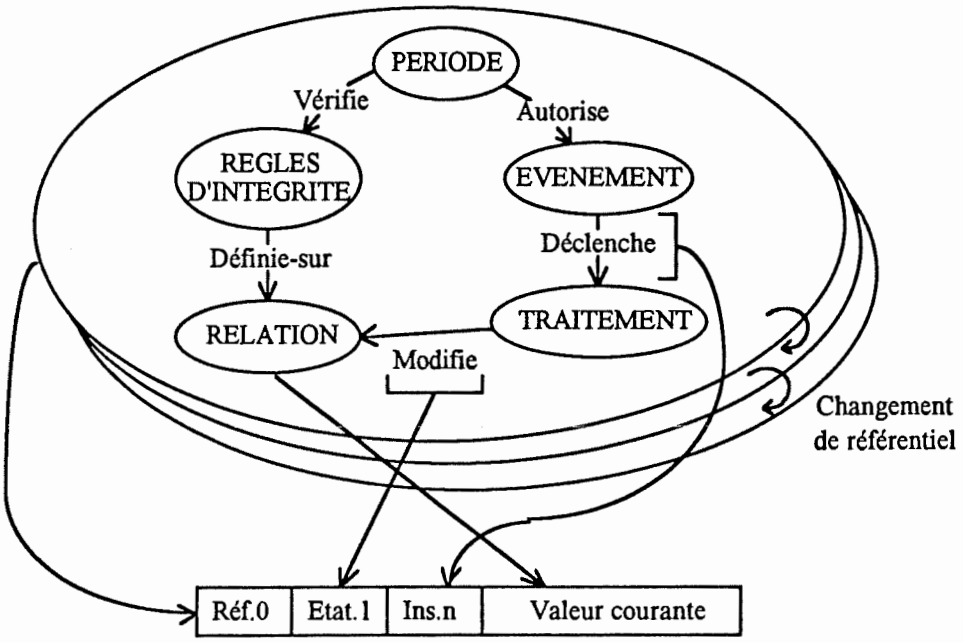


figure 4.4. Les concepts du modèle REVI.

5.2. Les estampilles temporelles.

Schématiquement, l'estampille se compose de trois constituants:

- L'instant, qui joue le même rôle que dans une base de données historique.
- L'état, qui est similaire à la définition vue dans l'approche dynamique.
- Le référentiel d'exploitation d'une instance d'entité.

Si un concepteur a spécifié un schéma de relation R = (K, X) où K est la clé de la relation et X un ensemble de constituants, la relation REVI sera:

- R_{revi}(Ref, Etat, Inst, K, T, X)

  dont la $clé_{revi}$ est : {Ref, Etat, Inst, K}

  dont la $clé_{interne}$ est : {K, T}

Les notions d'entité et d'instance d'entité utilisées précédemment peuvent alors être définies de la façon suivante:

- Une instance d'entité de la relation R_{revi} à l'instant i est un tuple (r, e, i, k, t, -) de cette relation.
- Une entité de la relation R_{revi} ayant pour clé k est l'ensemble des instances 'occ' d'entités telles que occ.K = k.
- L'instance courante d'une entité de la relation R_{revi} ayant pour clé K est le tuple (r, e, i, k, t1, -) tel qu'il n'existe pas de tuple (r, e, i, k, t2, -) où t1 < t2.

Grâce à ces quatre concepts, nous sommes capables de répondre aux questions : quand, pourquoi et dans quel référentiel les informations ont été traitées.

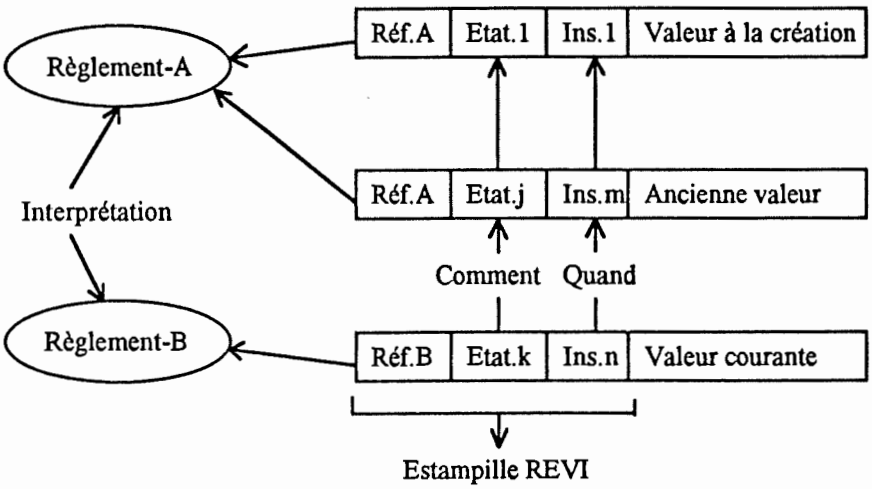


figure 4.5. Evolution d'une entité dans un modèle REVI.

5.3. Langage d'interrogation de REVI.

Le langage d'interrogation doit être adapté à la manipulation et à la sélection des entités stockées dans la base de données. Les base de données historiques ont étendu les langages d'interrogation en permettant d'effectuer des requêtes portant sur une estampille temporelle. On procédera de même pour les deux nouveaux concepts introduits dans l'estampille REVI. Les auteurs décrivent ci-après pour chaque concept soit le langage d'interrogation auquel ils font référence, soit les extensions qu'ils définissent.

V: Valeur:

SQL permet d'énoncer des requêtes portant uniquement sur les valeurs.

Exemple:

```
SELECT  Nombre_a
FROM  Contribuable
WHERE  Nombre_a = count(Class-Impo = 'A')
```

Cette requête dénombre les contribuables de la classe d'imposition "A".

I: Instant:

LAMBDA (déjà décrit précédemment dans ce travail) permet l'expression des requêtes par rapport aux instants; les sélections portent sur des notions temporelles (intervalles, durée, à partir, jusqu'à, ...).

Exemple:

```
SELECT  version during '1985-1989' of Nombre_a
FROM  Contribuable cont
WHERE  Nombre_a = count(cont.Class-Impo ='A')
```

Cette requête dénombre les contribuables de la classe d'imposition "A" pendant l'intervalle 1985-1989.

R: Référentiel:

Une application est définie pour l'ensemble des référentiels qui ont été établis durant l'exploitation de la base de données. Pour effectuer une requête, il est suffisant d'invoquer la liste des référentiels qui forment la portée de la requête.

Exemple:

SELECT Nombre_a belonging to {REGLEMENT-A}

FROM Contribuable cont

WHERE Nombre_a = count(cont.Class-Impo ='A')

Cette requête dénombre les contribuables de la classe d'imposition "A" taxés selon le référentiel {REGLEMENT-A}. Il est évidemment possible d'effectuer une requête portant sur plusieurs référentiels (belonging to {ref_1, ref_2, ..., ref_n}).

On peut également construire un critère de sélection en utilisant simultanément des contraintes référentielles et temporelles.

Exemple:

SELECT version during '1989' of Nombre_a

belonging to {REGLEMENT-A}

FROM Contribuable cont

WHERE Nombre_a = count(cont.Class-Impo ='A')

Cette requête dénombre les contribuables de la classe d'imposition "A" taxés selon le référentiel {REGLEMENT-A}, et ayant été imposés en 1989.

E: Etat:

Les états définissent le comportement des entités par rapport aux processus de traitements de l'application. On doit donc pouvoir effectuer des sélections portant sur les comportements des entités. On exprime cela par un "patron" d'états. Lors de l'exécution de la requête, la succession des états pris par les instances de cette entité sera comparée au patron de sélection.

La forme générale d'un patron est:

[$état_j$, $état_{j+1}$, $état_{j+2}$, ..., $état_{j+n}$] où l'$état_j$ précède l'$état_{j+1}$.

Si un état est défini, il est alors une constante du graphe d'état dynamique de la relation, sinon, il est indéfini.

Exemple:

```
SELECT  Nombre_a
FROM  Contribuable cont
WITH  cont  MATCHING[Délai_expiré,Recours_possible,Taxation_définitive]
WHERE  Nombre_a = count(cont.Class-Impo ='A')
```

Cette requête dénombre les contribuables de la classe d'imposition "A" ayant été taxés d'office et qui n'ont pas fait de recours.

Dans certains cas, il est intéressant de pouvoir récupérer les différentes instances d'une entité afin d'examiner les valeurs de leurs attributs. La déclaration des variables associées à une instance se fait implicitement dans le patron de la sélection:

```
WITH cont MATCHING[e1:Formulaire_envoyé,Recours_possible]
```

Les valeurs d'instances sont désignées par la notation suivante: cont[e1].Class_Impo. Dans ce cas, on désigne la classe d'imposition pour l'entité cont dans l'état désigné par e1.

Exemple:

```
SELECT  New_Nombre_a
FROM  Contribuable cont
WITH  cont MATCHING
        [e1:Taxation_établie,En recours,e2:Taxation_établie]
WHERE  New_Nombre_a =
        count(cont[e1].Class-Impo <> 'A' AND cont[e2].Class-Impo = 'A')
```

Cette requête sélectionne les contribuables ayant fait recours et dénombre parmi ces derniers ceux qui, par leur recours, sont entrés dans la classe d'imposition "A".

Remarque:

Les estampilles REVI peuvent être référencées dans la condition de sélection comme les autres constituants de l'entité, et sont accessibles à partir du langage d'interrogation comme les autres constituants. Les noms de ces constituants sont respectivement:

- **.Ref** pour le référentiel,
- **.Etat** pour l'état,
- **.Ins** pour l'instant.

6. Intérêts du modèle REVI.

La juxtaposition du référentiel, de l'état et de l'instant permet à l'utilisateur de poser de nouvelles questions. De même, l'administrateur et le concepteur peuvent résoudre certains problèmes difficilement solubles avec les autres approches. En voici quelques exemples:

6.1. Dans le domaine sémantique:

Afin d'obtenir une image cohérente de certains invariants du champ d'application, l'histoire des valeurs n'est pas suffisante; il faut également posséder le référentiel des valeurs.

Exemple: donner la liste des personnes qui n'étaient pas imposables durant ces dix dernières années.

SELECT version during '1979-1989' of nimp.Nom

FROM Non_Imposable nimp

"Non_Imposable" est défini comme une relation calculée qui change selon les échelles d'imposition minimum propre à chaque référentiel. Grâce à l'estampille REVI associée à chaque instance d'entité, le système peut appliquer la relation calculée qui lui correspond.

Exemple: donner les adresses courantes des personnes qui ont été taxées d'office dans les années 1984-1985.

```
SELECT  cont1.Nom, cont1.Adresse
FROM  Contribuable cont1
WHERE  cont1.Nocont IN
        SELECT  version during '1984-1985' of cont2.Nocont
        FROM  Contribuable cont2
        WITH  cont2 matching[Délai_expiré,Taxation_établie]
```

Cette dernière requête est particulièrement délicate; en effet, la réponse doit être recherchée dans l'instance courante de l'entité (on écrit aux gens à leurs adresses actuelles!) mais la condition de sélection est restreinte à l'intervalle 1984-1985. Pour contourner cette difficulté, on emploie la notion de requête imbriquée de SQL. Le numéro du contribuable permet la composition entre les instances de la réponse et les instances de la sélection. Cette requête ne comporte pas de clause dans la sélection la plus imbriquée car le critère de sélection porte uniquement sur le patron.

6.2. Dans le domaine organisationnel:

En mémorisant l'historique des états et en connaissant la dynamique des traitements, il est possible de construire une image du fonctionnement de l'organisation au cours du temps.

Exemple: pour le "REGLEMENT-B", quel est le temps moyen pour traiter une demande de recours ?

```
SELECT  Value
FROM  Contribuable cont
BELONGING  TO  {REGLEMENT-B}
WITH  cont MATCHING[e1:En_recours,e2:Taxation_établie]
WHERE  Value = average(delai = cont[e2].instant - cont[e1].instant)
```

Exemple: pour les dix dernières années, quel est le nombre de contribuables ayant fait plus de trois recours dans la même année ?

SELECT version during '1979-1989' of Nombre_C

FROM Contribuable cont

WITH cont MATCHING[e1:En_recours,*,En_recours,*,e2:En_recours]

WHERE Nombre_C = count (cont[e2].instant - cont[e1].instant < 1 year)

Ce type de requête permet de faire un audit du fonctionnement de l'organisation et de l'utilisation des services, afin d'effectuer ensuite des changements pertinents dans celle-ci.

6.3. Dans le domaine de la gestion des informations:

En spécifiant des règles d'archivage basées sur les états et les instants, il est possible de gérer la sauvegarde des données automatiquement chaque année pour certaines catégories d'information.

Exemple: il faut archiver la déclaration de chaque contribuable dès que sa taxation est définitive.

SELECT version during 'Today[year] - 1 year' of Decl.all

FROM Déclaration decl, Contribuable cont

WHERE decl.Nocont = cont.Nocont

AND cont.état = 'Taxation_définitive' INTO Archive

L'application automatique d'une telle règle permet de copier les instances sélectionnées de la base de données sur un support externe. Le traitement des informations est ainsi défini par rapport à un état spécifique de l'organisation (Taxation_définitive) et non plus seulement en termes de nombre d'occurrences ou d'ancienneté.

Chapitre 5.
Description d'un langage de query temporel.

Introduction.

J'ai choisi de présenter ici le langage de query temporel développé par R. Snodgrass [Snodgrass 84], [Snodgrass 87]. Quelles sont les raisons qui m'ont fait choisir ce langage plutôt qu'un autre?

* Ce langage, nommé TQuel (Temporal QUEry Language), est un des seuls langages réellement temporel au sens où lui même le définit dans un autre article;
* TQuel est une extension à un langage de query (Quel) pour une base de données relationnelle conventionnelle n'incluant pas la gestion du temps (SGBD INGRES);
* TQuel permet d'effectuer des queries "classiques", "rollback", "historiques" et "temporels", car il supporte les quatre notions (ces quatre notions ont été développées plus tôt dans ce travail et sont en correspondance avec celles utilisées ici).

1. Aperçu de TQuel:

TQuel est un sur-ensemble de Quel. Le choix de R. Snodgrass s'est porté sur Quel pour plusieurs raisons:

* Quel est bien connu et il en existe plusieurs implantations;
* Quel est particulièrement simple tout en étant assez puissant;
* Quel a une sémantique simple et bien définie.

TQuel est une extension à Quel dite minimale, et cela à la fois syntaxiquement et sémantiquement. Cette caractéristique a plusieurs implications (et raisons):

* Tous les statements Quel sont aussi des statements TQuel valides;
* De tels statements ont une sémantique identique dans Quel et dans TQuel lorsque le domaine temporel est fixé;

* Les constructions additionnelles définies dans TQuel pour manipuler le temps ont des constructions correspondantes dans Quel.

La figure suivante montre des relations classiques (c'est-à-dire ne faisant pas intervenir le temps).

Faculté (Nom, Fonction):

Nom	Fonction
Jean	Professeur
Michel	Chargé de cours
Pierre	Chargé de cours

Publication (Auteur, Périodique):

Auteur	Périodique
Jean	CACM
Michel	CACM
Michel	TODS
Pierre	VLDB

Figure 5.1. Deux relations statiques

La représentation des relations temporelles dans les trois dimensions est élégante conceptuellement parlant, mais n'est pas pratique pour afficher le contenu de ces relations.

Pour la lisibilité, on présentera ici les relations temporelles comme des relations classiques en y adjoignant deux attributs temporels. La valeur du premier attribut spécifie le **temps valide**, cet attribut spécifie la période pendant laquelle ou l'instant auquel le tuple est valide. Cet attribut contient soit une, soit deux valeurs (valid at ...; valid from ... to ...).

Pour des relations contenant des tuples représentant des occurrences instantanées, on utilisera valid at ... . De telles relations sont appelées "event relations". Pour des relations contenant des tuples représentant un état valide durant un intervalle de temps, on utilisera valid from ... to ... où "from" et "to" représentent les valeurs délimitant l'intervalle. Ces relations sont appelées "interval relations".

Le second attribut temporel spécifie le **temps transactionnel**. Deux valeurs sont toujours associées à ce temps transactionnel: le moment où le tuple est inséré dans la base de données (transaction start time), et le moment où le tuple est enlevé de la base de données (transaction stop time). Dès lors, les données sont "courantes" à

partir du start time jusque juste avant le stop time, car à partir du stop time, les données ne sont plus courantes. La figure suivante illustre les relations Faculté et Publications étendues et devenues respectivement une "interval relation" et une "event relation".

Faculté (Nom, Fonction):

Nom	Fonction	Valid time		Transaction time	
		(from)	(to)	(start)	(stop)
Jean	Assistant	09/78	12/83	09/78	∞
Jean	Chargé de cours	12/83	11/87	12/83	∞
Jean	Professeur	11/87	∞	10/87	∞
Michel	Assistant	09/84	12/89	08/84	∞
Michel	Chargé de cours	12/89	∞	12/89	∞
Pierre	Chargé de cours	09/82	∞	08/82	10/82
Pierre	Assistant	09/89	12/89	10/82	∞
Pierre	Chargé de cours	12/89	∞	11/89	∞

Publication (Auteur, Périodique):

Auteur	Périodique	Valid time	Transaction time	
		(at)	(start)	(stop)
Jean	CACM	11/86	11/86	∞
Michel	CACM	09/85	09/85	∞
Michel	TODS	05/86	05/86	∞
Pierre	VLDB	12/89	12/89	∞

Figure 5.2. Deux relations temporelles.

2. Le statement retrieve.

2.1. Le statement de base.

Si on exécute le query Quel 5.1 sur les relations de la figure 5.1

range of f **is** Faculté
retrieve into Chargés-de-cours (Nom = f.Nom)
where f.Fonction = "Chargé de cours"

Requête 5.1. Liste des chargés de cours.

on obtient la relation:

Nom
Michel
Pierre

Figure 5.3. Résultat de la requête 5.1.

Remarques:

* Pour la compréhension de la suite, il est utile de noter que le statement retrieve de Quel est constitué de 2 composants de base: la "target list", qui spécifie comment les attributs de la relation dérivée sont obtenus au départ des attributs des relations sous-jacentes, et la clause "where" qui spécifie quels sont les tuples qui participent à la dérivation.
* La granularité des valeurs temporelles est arbitraire. Ici, elle est fixée à 1 mois. On suppose également que le système de gestion de la base de données affiche les valeurs temporelles approchées au mois le plus proche.
* Les intervalles pour le temps valide et le temps transactionnel sont des intervalles fermés à gauche et ouverts à droite. Enfin, les tuples sont supposés regroupés: les tuples ayant des valeurs identiques pour les attributs explicites (appelés tuples "value-equivalent") ne se chevauchent pas et ne sont pas adjacents dans le temps.

Si en septembre 90, on exécute le même query sur les relations de la figure 5.2, on obtient la relation suivante:

Nom	Valid time		Transaction time	
	(from)	(to)	(start)	(stop)
Jean	12/83	11/87	09/90	∞
Michel	12/89	∞	09/90	∞
Pierre	12/87	∞	09/90	∞

Figure 5.4. Résultat de la requête 5.1.

Le temps transactionnel spécifie alors le moment de création de la relation.

Comme les attributs temporels additionnels sont un artifice pour emboîter une relation temporelle dans une relation conventionnelle (appelée par métaphore relation "snapshot"), ils sont rendus implicites et sont manipulés par le langage de query. C'est pour cela que les attributs temporels sont mis entre parenthèses et qu'ils sont séparés des autres attributs par une double ligne verticale. Pour manipuler ces attributs, TQuel étend le statement "retrieve" avec 3 composants.

2.2. La clause "when".

La clause when est le correspondant temporel à la clause where de Quel. Cette clause est constituée du mot clé "when" suivi d'un prédicat temporel sur les variables tuples. Le prédicat peut contenir des opérateurs tels que overlap, extend, precede, begin of, end of, and, or, not. Voici quelques exemples d'utilisation de la clause when:

range of c **is** Chargés-de-cours
retrieve into Chargés-de-cours-88 (Nom = c.Nom)
when c **overlap** "1988"

Requête 5.2. Liste des chargés de cours en 88.

range of c **is** Chargés-de-cours
range of p **is** Publication
retrieve into Publications-CC (Nom = p.Auteur, Périodique = p.Périodique)
where c.Nom = p.Auteur
when p **overlap** c

Requête 5.3. Liste des publications écrites par des chargés de cours.

range of f1 **is** Faculté
range of c **is** Chargés-de-cours
retrieve into Professeur (Nom = f1.Nom)
where c.Nom = "Pierre" **and** f1.Fonction = "Professeur"
when f1 **overlap begin of** c

Requête 5.4. Qui était Professeur quand Pierre fut promu chargé de cours?

2.3. La clause "valid".

La clause valid a la même utilité que la "**target list**": spécifier la valeur d'un attribut dans la relation dérivée. Seulement ici, l'attribut en question est l'attribut temporel implicite **temps valide**. Les deux variantes (valid at, valid from to) ont pour effet que la relation dérivée sera une event relation ou une interval relation. La clause valid peut contenir les opérateurs begin of, end of, overlap, extend et precede. Voici un exemple d'utilisation de la clause valid:

range of c **is** Chargés-de-cours
retrieve into Promotion-CC (Nom = c.Nom)
valid at begin of c

Requête 5.5. Quand furent promus les chargés de cours à cette fonction?

2.4. La clause as-of.

Les clauses when et valid concernent l'aspect historique du statement. Pour exprimer l'aspect "rollback", la clause as-of est utilisée.

La clause as-of permet de retrouver la base de données dans l'état où elle était à un moment donné du passé. Le query utilise alors uniquement l'information connue à ce moment là. Cela implique que les insertions et les corrections faites après ce moment là n'ont aucune influence sur la relation résultante du query. La plupart du temps, on s'intéressera à l'information la plus complète (dans le sens "la plus up-to-date"), et on ne spécifiera pas la clause "as-of now" qui sera utilisée par défaut. On utilisera la clause as-of A through B pour examiner une séquence de transactions sur une période précise.

Jusqu'à présent, on n'a parlé que du statement retrieve. Quel reconnaît quatorze statements. TQuel en étend cinq d'entre eux: les statements create, retrieve, append, delete et replace. Les statements append, delete et replace n'admettent pas de clause

as-of, car le temps transactionnel est calculé (fourni) automatiquement par le système de gestion de base de données temporel: c'est le temps courant.

3. Le statement create.

Le statement create définit une nouvelle relation et fournit un schéma pour celle-ci. Le statement suivant crée la relation Faculté de la figure 5.2. (sans son contenu).

create persistent interval Faculté (Nom = c20, Fonction = c15)

Requête 5.6. Un statement **create**.

Le statement create de Quel ne reconnaît pas les mots-clés "persistent", "interval" et "event". Ces mots-clés sont facultatifs en TQuel. Si le mot-clé "persistent" est spécifié, on obtient une relation soit rollback, soit temporelle. La clause as-of peut dès lors être spécifiée dans un query. Si le mot-clé "interval" ou "event" est spécifié, on obtient soit une relation historique, soit une relation temporelle. Les clauses when et valid peuvent alors être utilisées dans un query. Si aucun de ces mots-clés n'est spécifié, on obtient une relation conventionnelle (dite relation snapshot).

Le statement create est donc polyvalent et permet de créer un des quatre types de base de données suivantes: snapshot, rollback, historique, temporelle.

4. Remarques.

4.1. Le statement append.

Un statement append ajoute un ou plusieurs tuples à la relation existante. Cet ajout ne peut pas se faire aveuglément car il se peut qu'une partie de l'information supplémentaire soit redondante. Il faut donc l'éliminer et ne garder en ajout que l'information nécessaire. Par exemple, si en septembre 92, on exécute le statement suivant:

append to Faculté (Nom = "Michel", Fonction = "Assistant")
valid from "08/84" **to** "12/89"

Requête 5.7. Michel a rejoint le département un mois plus tôt.

on devrait obtenir le tuple suivant comme ajout:

Nom	Fonction	Valid time		Transaction time	
		(from)	(to)	(start)	(stop)
Michel	Assistant	08/84	09/84	09/92	∞

Figure 5.5. Ajout de la requête 5.7.

car on savait que ce même tuple était déjà valide de septembre 84 à décembre 89. NB: on suppose que les contraintes d'intégrité ont été vérifiées auparavant.

Le schéma suivant montre les différents cas de figure qui peuvent se présenter.

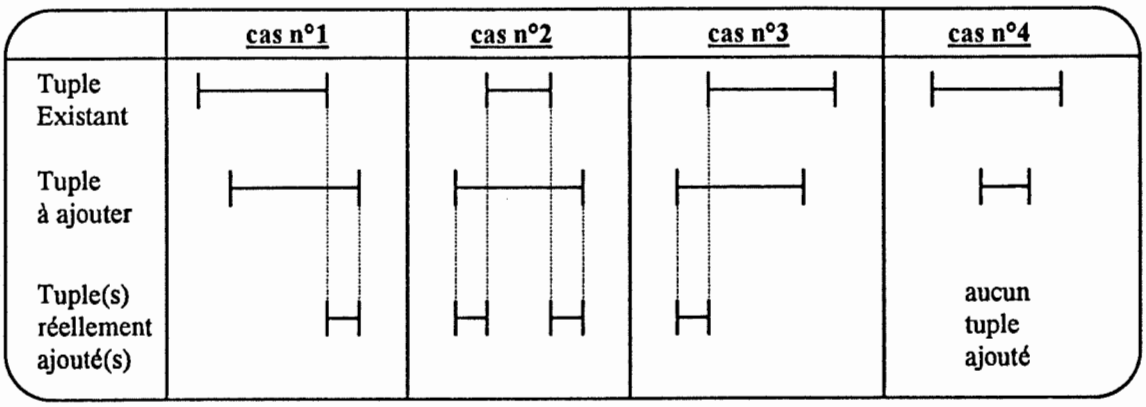


Figure 5.6. Calcul du temps valide dans un statement append.

4.2. Le statement delete.

Le statement delete connaît le même genre de problèmes que le statement append. L'exemple suivant illustre une situation possible. Soit le statement:

range of f **is** Faculté
delete f
where f.Nom = "Jean"
valid from "03/88"

Requête 5.8. Jean quitta la faculté en Mars 88.

(Jean quitta la faculté en Mars 88: fait enregistré en septembre 92). Le résultat de ce statement est la modification d'un tuple existant et l'ajout d'un nouveau tuple (figure 5.7).

Nom	Fonction	Valid time (from)	Valid time (to)	Transaction time (start)	Transaction time (stop)
Jean	Professeur	11/87	∞	10/87	09/92
Jean	Professeur	11/87	03/88	09/92	∞

Figure 5.7. Modifications apportées par la requête 5.8.

La figure suivante illustre les différents cas possibles.

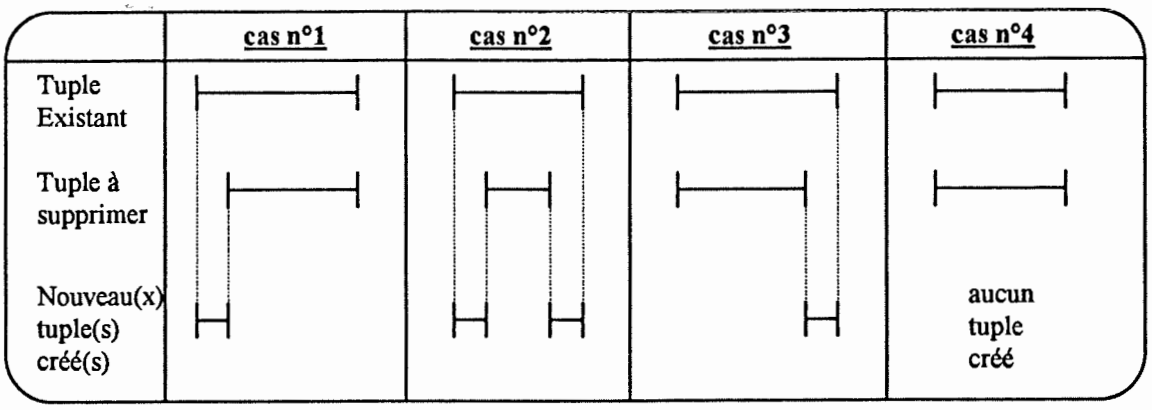


Figure 5.8.

Le statement replace a une sémantique similaire à celle d'un statement delete suivi d'un statement append. Ce n'est toutefois pas une équivalence. Pour le détail, voir [Snodgrass 87].

5. Sémantique de TQuel.

5.1. Introduction.

Les statements TQuel manipulent l'information d'une base de données temporelle qui est composée d'une séquence d'états historiques indexés par le temps transactionnel. Chaque état historique est lui-même constitué d'une séquence de photos (snapshots) indexées par le temps valide. On a donc une structure à quatre dimensions.

La sémantique de TQuel doit spécifier comment une relation temporelle est modifiée par un update ou comment une relation temporelle est créée par un "retrieve". Deux solutions sont possibles:

- soit étendre la sémantique du modèle relationnel en incorporant directement la notion du temps;
- soit utiliser le modèle relationnel conventionnel et étendre le langage de query.

c'est la deuxième alternative qui a été choisie par R. Snodgrass. En effet, le modèle relationnel classique est simple et est basé sur des formalismes de théorie des ensembles et de calcul des prédicats bien connus, alors que les modèles incorporant directement la gestion du temps reposent sur des mécanismes beaucoup plus complexes et moins bien connus. De plus, une base de données temporelle basée sur le modèle relationnel peut être directement implantée dans un système de gestion de base de données relationnel existant.

5.2. Comment emboîter une relation temporelle dans une relation snapshot:

Le problème est d'emboîter une relation quadri-dimensionnelle dans une relation snapshot bidimensionnelle. La sémantique des opérations sur une relation temporelle quadri-dimensionnelle sera spécifiée en établissant leurs effets sur une relation snapshot bidimensionnelle. De cette façon, la sémantique peut être exprimée par le formalisme classique du "tuple calculus".

Cet emboîtement peut être effectué de différentes manières:

* une première façon consiste à ajouter deux attributs aux attributs de l'utilisateur. Chacun de ces deux attributs contient une seule valeur temporelle spécifiant respectivement le valid et le temps transactionnel pour chaque tuple.

Nom	Fonction	Valid time	Transaction time
Jean	Assistant	09/78	09/78
Jean	Assistant	09/78	08/82
...	...	...	...
Jean	Assistant	09/82	08/82
Pierre	Chargé de cours	09/82	08/82
Jean	Assistant	09/78	10/82
...	...	...	...
Jean	Assistant	09/82	10/82
Pierre	Assistant	09/82	10/82

Jean	Assistant	09/78	12/83
...	...	...	...
Jean	Assistant	09/82	12/83
Pierre	Assistant	09/82	12/83
...	...	...	...
Jean	Chargé de cours	12/83	12/83
Pierre	Assistant	12/83	12/83
Jean	Assistant	09/78	08/84
...	...	...	...
Jean	Assistant	09/82	08/84
Pierre	Assistant	09/82	08/84
...	...	...	...
Jean	Chargé de cours	12/83	08/84
Pierre	Assistant	12/83	08/84
...	...	...	...
Jean	Chargé de cours	09/84	08/84
Pierre	Assistant	09/84	08/84
Michel	Assistant	09/84	08/84

figure 5.9. Emboîtement d'une relation temporelle (version 1).

La figure ci-avant montre une partie de la relation temporelle de la figure 5.2 sous ce formalisme.

- Les tuples englobant un état historique à un temps transactionnel particulier sont séparés par des lignes horizontales.
- Les tuples englobant une "snapshot slice" à un temps valide particulier sont séparés par des pointillés.

La figure 5.9 contient une relation temporelle comprenant 5 états historiques (associés chacun à un temps transactionnel). Chacun des états historiques est constitué de snapshot slices (associés chacun à un temps valide). L'entièreté de la relation comprendrait 8 états historiques (résultant de 8 transactions) et un total de 102 tuples. Les relations historiques de CLIFFORD et WARREN sont similaires à ce formalisme.

* Une autre manière d'emboîter une relation temporelle dans une relation snapshot est d'ajouter deux attributs, chacun contenant deux valeurs temporelles, dénotant des intervalles de temps valide et de temps transactionnel. C'est cette représentation qui est utilisée par R. Snodgrass (cf. chapitre 3). Cette même représentation est utilisée

par Ariav [ARIAV 86] dans son TODMS (Temporal Oriented Data Management System). Son principal avantage est de permettre un prototypage rapide d'une base de données temporelles "au-dessus" d'une base de données snapshot.

* Une troisième manière de procéder est d'ajouter cinq attributs:

- le moment où le tuple devient valide (T_{es}: "time-effective-start");
- le moment où T_{es} est enregistré dans la base de données (T_{rs}: "time-registration-start");
- le moment où le tuple devient invalide (T_{ee}: "time-effective-end");
- le moment où T_{ee} est enregistré dans la base de données (T_{re}: "time-registration-end");
- le moment où le tuple entier est retiré de la base de données, car il n'est plus correct (T_d: "time-deletion").

Cette représentation est proposée par BEN-ZVI [Ben-Zvi 82] dans son Time Relational Model. Elle permet d'économiser quelques tuples par rapport à la représentation de R. Snodgrass lorsque des tuples sont "effacés" de la base de données.

La figure 5.10 représente la figure 5.2 avec le troisième formalisme.

Nom	Fonction	T_{es}	T_{ee}	T_{rs}	T_{re}	T_d
Jean	Assistant	09/78	12/83	09/78	12/83	-
Jean	Chargé de cours	12/83	11/87	12/83	10/87	-
Jean	Professeur	11/87	-	10/87	-	-
Michel	Assistant	09/84	12/89	08/84	12/89	-
Michel	Chargé de cours	12/89	-	12/89	-	-
Pierre	Chargé de cours	09/82	-	08/82	-	10/82
Pierre	Assistant	09/82	12/87	10/82	11/87	-
Pierre	Chargé de cours	12/87	-	11/87	-	-

Figure 5.10. Emboîtement d'une relation temporelle (version 3).

* Un quatrième formalisme consiste à associer des valeurs temporelles aux attributs-mêmes. A l'intérieur d'un tuple, les attributs peuvent individuellement prendre plusieurs valeurs à différents moments. Ce formalisme a pour conséquence que la relation snapshot n'est plus en première forme normale.

Nom		Fonction	
Jean	[09/78, ∞[	Assistant	[09/78, 12/83[
		Chargé de cours	[12/83, 11/87[
		Professeur	[11/87, ∞[
Michel	[09/84, ∞[	Assistant	[09/84, 12/89[
		Chargé de cours	[12/89, ∞[
Pierre	[09/82, ∞[	Assistant	[09/82, 12/87[
		Chargé de cours	[12/87, ∞[

Figure 5.11. Emboîtement d'une relation temporelle (version 4).

* Une dernière représentation propose de n'enregistrer que les transactions. Cette représentation est la plus efficace en matière d'économie de place. Un autre avantage de cette méthode est la facilité de modification. Le désavantage de cette méthode est que pour déterminer les tuples valides à un moment donné, il faut passer en revue les transactions depuis le début. La figure 5.12 illustre cette représentation.

Type	Transaction time	Nom	Fonction	Valid time
Ajout	09/78	Jean	Assistant	09/78
Ajout	08/82	Pierre	Chargé de cours	09/82
Modification	10/82	Pierre	Assistant	09/82
Modification	12/83	Jean	Chargé de cours	12/83
Ajout	08/84	Michel	Assistant	09/84
Modification	10/87	Jean	Professeur	11/87
Modification	11/87	Pierre	Chargé de cours	12/87
Modification	12/89	Michel	Chargé de cours	12/89

Figure 5.12. Emboîtement d'une relation temporelle (version 5).

Le développement de la sémantique formelle de TQuel peut se trouver dans [SNODGRASS 87].

Remarque: Une base de données temporelle définie en utilisant le formalisme de R. Snodgrass n'est pas tout à fait "append-only". En effet, dans certains cas, une transaction ne donne pas lieu à l'ajout d'un nouveau tuple. Un statement delete ou replace peut ne modifier que le transaction-stop-time d'un tuple. Seulement, cette modification ne s'effectuera jamais qu'une seule fois par tuple (au maximum), et la valeur avant modification est toujours infinie. Il n'y a donc pas d'ambiguité possible. D'autre part, cette mise au point n'empêche pas l'utilisation de la technologie des disques optiques WORM. Il suffit d'associer la valeur correspondant à une zone intacte sur le disque avec la valeur infinie qui caractérise le transaction-stop-time d'un tuple non-effacé. On dira donc d'une telle base de données temporelle qu'elle n'est pas append-only, mais qu'elle a la propriété de non-effacement.

Chapitre 6.
Conclusion de la première partie.

Au terme de cette étude théorique, il apparaît clairement que 3 voies de recherche sur la notion de temps dans les bases de données se dessinent:

- Une voie se dirige vers la notion de base de données historique (cf. chapitre 2).
- Une voie se dirige vers la notion de base de données temporelle (cf. chapitre 3).
- Une voie se dirige vers la combinaison des notions d'historique et de dynamique des traitements (cf chapitre 4).

Chacune de ces trois approches du temps dans les bases de données a ses qualités, comme on a pu s'en rendre compte.

Que doit-on retenir de ces différentes approches? Quels sont les éléments importants de chacune d'elles?

1. L'approche historique.

L'approche historique est l'approche la plus courante et celle qui répond le mieux à une grande majorité d'applications. En effet, elle permet d'associer une période de validité à des données. Cette période de validité peut être associée soit au niveau du tuple complet, soit au niveau de chaque item dont on désire garder l'historique.

La première solution maintient la relation en première forme normale pour autant qu'elle le soit indépendamment du temps. Cela implique que l'overhead destiné à recevoir les informations de la structure historique que l'on rajoute à la structure d'un tuple est assez restreint. L'espace de stockage supplémentaire est de plus proportionnel au nombre de tuples stocké dans la relation. L'overhead total pour tous les tuples de la relation peut être calculé facilement: il suffit de multiplier le nombre de tuples par la taille de l'overhead d'un tuple. Par contre, le fait de stocker les informations de la structure historique au niveau du tuple a pour conséquence un gaspillage considérable de place étant donné que lorsque l'on désire enregistrer une modification à un tuple, on doit recopier entièrement la valeur précédente du tuple dans l'historique, alors que le tuple après modification n'a peut-être qu'un seul de ses items qui a été modifié. On obtient donc une copie exacte du tuple avant modification mis à part l'item modifié et la structure historique.

La seconde solution a pour conséquence que la relation n'est plus en première forme normale. Cette dernière solution a l'avantage de ne stocker que les items qui ont été modifiés. Par contre, l'overhead entraîné par l'adjonction d'une structure historique à chaque item historique est multiplié par le nombre d'items historiques que contient le tuple. L'avantage de cette solution peut donc devenir un inconvénient dans certaines circonstances. Par exemple, si la structure du tuple est composée de beaucoup d'items historiques de petite taille, on va créer un overhead total pour le tuple très important. De plus, si certains items historiques sont modifiés de manière simultanée, on enregistrera les mêmes données historiques plusieurs fois. Enfin, le fait d'enregistrer séparément l'évolution des différents items augmente la complexité, et diminue la performance du gestionnaire de la base de données. En effet, le travail de recherche des différents morceaux d'un tuple peut prendre un certain temps si ceux-ci ne sont pas stockés au même endroit sur le disque.

Dans la modélisation de [ADIBA 87], la structure historique est rattachée au tuple entier. Or, le modèle TIGRE est destiné à stocker des informations multimédia. Ces informations multimédia sont en général très coûteuses en espace de stockage. Il est donc intéressant dans de telles bases de données de reconsidérer la solution consistant à ajouter la structure historique au niveau des items. En effet, dans ce cas, les items sont de taille suffisante que pour justifier un overhead à leur niveau.

L'approche historique développée par [ADIBA 87] est également intéressante du point de vue de l'originalité des concepts qu'elle propose. Il s'agit notamment de la différenciation entre divers types d'historiques (historiques à versions manuelles, historiques à versions périodiques, historiques à versions successives), de la notion de persistance, de la notion de photo. Il s'agit aussi de l'application des notions d'historique à ces photos (notions d'album et de film). Ces album et film peuvent être vus comme des photos de second degré.

Les différents types d'historique, ainsi que la notion de persistance sont destinés à économiser l'espace de stockage. On n'enregistre les données que lorsqu'on le juge utile et on purgera automatiquement la base de données des données trop anciennes devenues inutiles. Cette idée de purger la base de données est mise en oeuvre de telle sorte que la purge est faite à chaque ajout d'un nouveau tuple. Lors de l'ajout, on vérifie si l'ajout d'un tuple n'entraîne pas l'obsolescence d'un autre tuple. Si c'est le cas, le tuple obsolète est éliminé. De cette manière, on évite de voir la base de données grossir inconsidérément, et on évite de devoir effectuer de grosses réorganisations lorsque l'espace de stockage vient à manquer.

2. L'approche temporelle.

D'un autre côté, l'approche historique n'est pas toujours suffisante pour répondre à des besoins particuliers.

Dans certains cas, une approche temporelle au sens de [SNODGRASS 85] et [SNODGRASS 87] s'impose. Il est possible dans une telle approche de ressortir un état de la base de données équivalent à ce qu'il était à un moment bien précis du passé, et d'effectuer des requêtes historiques au départ de cet état. Le fait de retourner ainsi dans le passé de la base de données constitue l'opération de rollback dont on a défini le sens exact dans le chapitre 4. Lorsqu'on effectue un rollback, on annule momentanément toute correction qui aurait été faite après l'instant de référence indiqué dans l'opération de rollback. On dispose donc dans le modèle temporel d'une dimension supplémentaire par rapport au modèle historique.

Le modèle temporel de [SNODGRASS 85] est le plus complet du point de vue de la souplesse de modélisation des données. Il permet de travailler sur une base de données temporelle comme si c'était soit une base de données statique, soit une base de données statique rollback, soit une base de données historique, soit une base de données temporelle. Le mécanisme qui permet cela est le langage de query associé au modèle de données temporel (TQuel en l'occurrence). Une requête TQuel de base est une requête dans laquelle on ne fait aucune allusion au concept du temps. Cette requête est alors identique à celle utilisée si la base de données était statique. Si on ajoute une clause de rollback à cette requête, on obtient une requête travaillant comme sur une base de données statique rollback. Si on ajoute une clause historique à cette requête, on obtient une requête travaillant comme sur une base de données historique. Si on ajoute à la fois une clause de rollback et une clause historique, on obtient une requête travaillant sur une base de données temporelle. Dans la pratique, toutes les requêtes sont temporelles car le gestionnaire de la base de données ajoute toujours les clauses de rollback et historique si elles ne sont pas spécifiées par l'utilisateur. Seulement, les clauses par défaut sont telles qu'elles n'ont aucune influence sur le caractère initial de la requête.

Ces dernières réflexions peuvent être rassemblées dans un tableau qui donne, en fonction des différents modèles de requêtes, la catégorie de base de données correspondante.

	Rollback impossible	**Rollback possible**
Requêtes statiques	Statique	Statique Rollback
Requêtes historiques	Historique	Temporelle

Figure 7.1. Catégories de bases de données.

Quels sont les autres enseignements que l'on peut retirer du modèle temporel ?

1°) Trois types de temps bien définis remplacent les notions de temps physique et temps logique qui étaient trop vaguement définies. Il s'agit du **temps transactionnel**, du **temps valide** et du **temps user-defined**.

2°) Les trois types de temps peuvent être caractérisés par les trois critères de classification développés par [SNODGRASS 85].

* Le premier critère mesure la correspondance du temps du modèle enregistré dans la base de données avec la réalité.
* Le deuxième critère dénote la flexibilité de mise à jour de la base de données. Dans certains cas, toutes les modifications sont permises, dans d'autres, seuls les ajouts sont admis.
* Le troisième critère sépare les types de temps indépendants vis-à-vis de l'application des types de temps dépendants vis-à-vis de l'application.

Le résultat de la classification est donné dans le tableau suivant (figure 7.2). Cette nomenclature est suffisante pour être également utilisée afin de classifier n'importe quel modèle de données incluant une quelconque notion de temps.

Terminologie	Réalité - Représentation	Flexibilité de mise à jour	Dépendance vis-à-vis application
Transactionnel	Représentation	Append-only	Oui
Valide	Réalité	Oui	Oui
User-defined	Réalité	Oui	Non

Figure 7.2. Résultats de la classification.

3°) Ces types de temps sont supportés ou non par les différentes catégories de base de données (figure 7.3).

* Pour supporter l'opération rollback, un modèle de données doit incorporer la notion de temps transactionnel.
* Pour supporter les requêtes historiques, un modèle de données doit incorporer la notion de temps valide.
* On peut résumer dans le tableau suivant les différents types de temps qui peuvent ou doivent être supportés par les différentes catégories de bases de données.

	Transactionnel	Valide	User-defined
Statique			
Statique Rollback	√		
Historique		√	√
Temporelle	√	√	√

Figure 7.3. Types de temps - types de bases de données.

3. L'approche du modèle REVI.

La dernière approche analysée dans ce travail est l'approche combinée historique et dynamique. L'approche historique et l'approche dynamique sont deux approches intégrant des notions de temps, et toutes deux sont des extensions à des systèmes de gestion de base de données traditionnels. L'approche historique intègre le temps dans le modèle de données tandis que l'approche dynamique spécifie les interactions entre les traitements et les données.

Le modèle historique, comme on l'a déjà vu, permet la représentation, dans le temps, d'une succession identifiable de valeurs qui ont été prises par un tuple donné. Cependant elle possède d'autres limites que celles identifiées lors de l'étude du modèle temporel. Avec cette représentation, on est dans l'incapacité d'interpréter pour un tuple les causes des transitions d'une valeur prise au temps t_i à celle prise au temps t_{i+1}.

Le modèle dynamique pris comme exemple est MTG, développé par [GUYOT 86]. Il permet d'associer un état dynamique à un tuple d'une relation. Les états dynamiques sont implicitement définis lors de la spécification des traitements. Ils indiquent l'avancement d'un tuple dans un processus de traitement. La notion temporelle qui est attachée au tuple par l'intermédiaire de son état dynamique n'est pas absolue par rapport à un instant de référence mais relative aux processus de traitement des informations d'une organisation. Parmi les avantages des états dynamiques, citons la simplification de la spécification des traitements, la déclaration automatique de la partie constituant l'état dynamique du tuple, ... Le modèle dynamique a aussi ses limites: on ne peut pas expliquer pourquoi un tuple se trouve dans un état dynamique donné, et quels sont les traitements qui l'ont modifié. Plusieurs séquences de traitements sont susceptibles d'être utilisées pour atteindre un état dynamique donné.

Les limitations des deux modèles séparés disparaissent dans un modèle intégrant les deux approches. C'est sur cette base qu'a été développé le modèle REVI [Bursens 87]. Ce modèle est basé sur quatre concepts: Valeur, Instant, Etat et Référentiel. La valeur constitue la partie classique du tuple, l'instant représente

l'extension historique, l'état représente l'état dynamique du modèle dynamique et le référentiel correspond à une modélisation de l'application et porte sur l'ensemble des concepts du modèle: période, évènements, règles d'intégrité, relations, traitements.

Le modèle REVI, de par les concepts associés à chaque tuple est plus complet pour analyser l'histoire des informations d'une organisation. Chaque information du passé est localisable:

- dans le temps, grâce aux instants;
- dans son avancement par rapport aux processus de traitement, grâce aux états;
- dans son référentiel d'exploitation, la rendant ainsi plus cohérente.

Pour rendre implicite cette localisation des informations, l'approche demande de spécifier l'application de manière globale et de la mémoriser de manière continue, en fonction de son évolution.

L'approche de [BURSENS 87], [BURSENS 89] ne tient malheureusement pas compte des mises au point en matière de temps faites par [SNODGRASS 85]. Le modèle REVI est basé sur l'approche historique et non pas temporelle. Il y a d'ailleurs une certaine imprécision sur les termes utilisés. Qu'à cela ne tienne, il nous est amplement loisible de remplacer la notion d'historique utilisée par la notion temporelle de [SNODGRASS 85]. Les adaptations nécessaires ne représentent pas un remaniement complet du modèle présenté. On a alors un modèle tout à fait complet, tant au niveau temporel, qu'au niveau dynamique, ce qui, compte tenu du référentiel, permet de satisfaire les besoins de n'importe quelle application dans laquelle le suivi et l'évolution sont des notions omniprésentes.

Deuxième partie

1. Introduction.

La deuxième partie de ce mémoire présente un problème concret de gestion d'une base de données enregistrant l'activité du personnel d'une entreprise. Le problème rencontré est le fait que si l'on travaille avec une base de données classique, tout retour en arrière dans le temps provoque des difficultés de recalcul de certaines valeurs. Il apparaît évident que les travaux sur la problématique du temps dans les bases de données peuvent constituer des solutions à ce type de problèmes. C'est de cela qu'il sera question dans cette seconde partie.

2. Plan de cette partie:

Cette deuxième partie tentera dans un premier temps de poser le problème. Ce premier point constituera une analyse tout à fait générale des problèmes qu'une entreprise quelconque peut avoir dans ce domaine (section 3).

Dans un deuxième temps, j'essaierai de poser le problème d'un point de vue informatique (section 4).

Dans un troisième temps (section 5), je ferai appel aux acquis théoriques exposés dans la première partie de ce mémoire. La question sera "quels sont les travaux susceptibles d'aider à une résolution partielle ou complète du problème posé?"

J'élaborerai ensuite quelques propositions personnelles (section 6). Enfin, je ferai quelques propositions ayant trait aux travaux futurs dans cette lignée.

3. Analyse du problème.

Beaucoup d'entreprises désirent contrôler les prestations de leurs ouvriers ou employés. Pour cela, elles utilisent divers systèmes, dont beaucoup sont basés sur des terminaux de pointage reliés à un système informatique. Les ouvriers ou employés disposent chacun d'un badge qui leur permet non seulement d'accéder à leur lieu de travail (système de contôle d'accès), mais aussi d'enregistrer les moments auxquels ils entrent et sortent de l'entreprise.

Le système informatique le plus moderne et le plus approprié à ce genre de chose est un gestionnaire de bases de données. On voudrait enregistrer des informations dans une base de données de manière à

- avoir la possibilité de régir automatiquement les prestations du personnel,
- être capable de rémunérer le personnel selon ses prestations et en fonction des règles définies par l'état et/ou l'entreprise,

- donner les informations nécessaires à chacun des terminaux de pointage afin qu'ils puissent remplir leurs fonctions de contrôle d'accès et d'enregistrement des pointages.

Que veut-on enregistrer dans la base de données destinée à gérer à la fois les terminaux de pointage et les prestations du personnel?

Tout d'abord, il faut enregistrer les différentes données permettant d'identifier chacune des personnes travaillant dans l'entreprise. Ces informations constituent le signalétique du personnel et comprennent notamment des indications sur chacune des personnes, sur leur fonction, sur leur position hiérarchique dans l'entreprise, sur leurs autorisations spéciales si elles en possèdent, sur leur lieu de travail exact dans le bâtiment (ex: n° du local), etc.

Ensuite, on veut enregistrer des informations concernant l'activité du personnel. Ces informations peuvent être réparties en deux groupes: d'une part, les informations qui planifient et règlementent l'activité future du personnel, et d'autre part, les informations recueillies sur l'activité réelle du personnel.

Enfin, sur base de toutes les informations dont on dispose, on désire rémunérer le personnel.

Tout irait pour le mieux dans le meilleur des mondes si toutes les informations que l'on enregistre étaient correctes, complètes et enregistrées à temps. De la sorte, tous les traitements à effectuer fourniraient des résultats corrects du premier coup eux aussi. On n'aurait pas non plus besoin de stocker les informations des périodes précédentes. En effet, dès la période courante clôturée, on pourrait démarrer la période suivante avec une image correcte de la réalité.

Malheureusement, la réalité est de loin beaucoup plus complexe à traiter. Les informations que l'on désire enregistrer dans un système informatique sont souvent incorrectes, incomplètes et arrivent fréquemment en retard[8]. Dès lors, toute une série de problèmes en découlent:

- Est-il possible de corriger et/ou compléter les informations incorrectes et/ou incomplètes? Si oui, comment effectuer ces modifications?
- Que faut-il faire lorsqu'un traitement doit être exécuté alors que les données dont il a besoin ne sont pas sûres?
- Que faut-il faire lorsqu'un traitement est déjà terminé et que l'on veut corriger ou compléter une donnée utilisée par ce traitement?
- ...

[8] *Les raisons de ces problèmes ne seront pas discutées ici.*

La plupart du temps, la réponse est simple: on ne peut pas attendre d'être certain que tout soit correct et complet pour effectuer les traitements nécessaires. Il faut, avec un délai fixé, exécuter les traitements nécessaires avec les informations dont on dispose, même si elles sont sujettes à modifications ultérieures. Il faut stocker les informations durant un certain temps, de sorte que l'on puisse corriger les erreurs, compléter les informations enregistrées partiellement. De plus, et c'est là que la complexité atteint son maximum, il faut parfois tenir compte des changements opérés et recommencer certains traitements.

4. Formalisation informatique du problème.

Dans cette section, j'essaierai de formaliser le problème en utilisant le "langage informatique". Pour cela j'utiliserai un exemple de modélisation des informations enregistrées dans un système de gestion des prestations du personnel d'une entreprise.

Il est bien entendu que cet exemple sera simplifié à l'extrême afin de ne pas s'encombrer de notions inutiles. On utilisera le formalisme du modèle relationnel pour la simplicité et la facilité de représentation.

On ne dispose à l'heure actuelle d'aucun outil à caractère temporel: aucun système de gestion de bases de données historiques ou temporelles n'existe (à ma connaissance). Il faut donc essayer d'utiliser ce qui existe, à savoir des gestionnaires de bases de données classiques. Je dispose actuellement d'un accès à un système IBM AS/400. Je prendrai donc comme hypothèses de travail l'utilisation de l'AS/400 et du langage de définition et de manipulation de bases de données SQL/400.

Enoncé de l'exemple:

Soit une entreprise X désireuse de contrôler et gérer les prestations du personnel qu'elle emploie.

- L'entreprise X est constituée d'un certain nombre de services. Chaque employé fait partie d'un et un seul service.
- Chaque employé est caractérisé par ses coordonnées personnelles et ses coordonnées propres à sa position dans l'entreprise. Ses coordonnées personnelles sont, entre autres, son nom, sa situation familiale, etc. Ses coordonnées concernant sa position dans l'entreprise sont notamment sa fonction (comptable, réceptionniste, directeur général, ...), son statut (employé, cadre, ...), sa date d'entrée en service, son numéro de badge, etc.
- Les employés travaillent suivant des horaires souples. Les horaires sont attribués à un employé par ses supérieurs ou par lui-même s'il y est autorisé. Un horaire est représenté par une heure de début, une heure de fin et un nombre d'heures à prester.

- Un employé dispose d'un certain nombre de jours de congés qu'il peut prendre quand il le veut.
- On enregistre les prestations des employés via les pointages à l'entrée et à la sortie de l'entreprise. Chaque employé dispose à cet effet d'un badge qui l'identifie dans le système d'information.
- Les employés peuvent effectuer des heures supplémentaires qui peuvent soit être récupérées soit être rémunérées. Un certain nombre de règles régissent ces prestations supplémentaires afin d'éviter les abus.
- On enregistre également les absences et leurs motifs.
- Un certain nombre de traitements peuvent être effectués sur les données enregistrées dans le système. Par exemple, on désire augmenter la rémunération d'un employé, changer son statut, ...
- On désire rémunérer le personnel avec ce système. De plus, on désire également pouvoir l'interroger pour toutes sortes de choses. Par exemple, pour savoir combien d'heures l'employé A a-t-il déjà travaillé ce mois-ci?, L'employé B vient-il travailler demain? si oui à quelle heure? ...

On peut essayer de modéliser cet exemple de la manière suivante: les relations statiques (*snapshot* dans [SNODGRASS 85]) représentées ci-après regroupent les informations.

Soient les relations Employés, Horaires, Pointages, Compteurs.

- La relation Employés enregistre les données personnelles d'un employé et les données propres à sa position dans l'entreprise.
- La relation Horaires enregistre les informations concernant l'horaire d'un employé pour une journée, ceci pour chaque employé et chaque jour.
- La relation Pointages enregistre les pointages des employés (un tuple de la relation Pointage enregistre la date, l'heure, le numéro du badge, et le type de pointage (entrée ou sortie).
- La relation Compteurs enregistre les valeurs de tous les compteurs dont on a besoin. On enregistre de cette manière:
  - ⇨ le nombre d'heures prestées depuis le début de la semaine et depuis le début du mois,
  - ⇨ le nombre d'heures supplémentaires prestées depuis le début du mois,

⇨ le nombre de jours de congés restant,

⇨ le nombre de jours entiers prestés depuis le début du mois (pour les chèques repas, par exemple).

- La relation Absences enregistre les absences et leurs motifs.

Quels sont les types de donées dont on dispose pour modéliser les informations à enregistrer dans les tables relationnelles ?

Types	Description
char(n)	chaîne de caractères de longueur n fixe.
varchar(n)	chaîne de caractères de longueur ≤ n.
smallint	petit nombre entier (codé sur 16 bits)
integer	nombre entier (codé sur 32 bits)
numeric(m,n)	nombre décimal ayant m positions dont n décimales
decimal(m,n)	équivalent à numeric(m,n) mais codé en "packed"
real	nombre en virgule flottante (simple précision - 32 bits)
float	nombre en virgule flottante (double précision - 64 bits)
date	date (plusieurs formats sont disponibles)
time	heure (plusieurs formats sont disponibles)
timestamp	valeur composée de 7 parties: l'année, le mois, le jour, l'heure, les minutes, les secondes et les microsecondes. Par exemple, "1992-05-21-20:35:10.123456" est l'estampille représentant le 21 mai 1992 à 20 heures 35 minutes 10 secondes et 123456 microsecondes.

On peut créer ici les tables correspondant aux relations ci-avant:

1) la relation Employés.

```
CREATE TABLE LVTEST.EMPLOYES
  (BADGE          CHAR(3),          (numéro du badge)
   NOM            VARCHAR(30),
   PRENOM         VARCHAR(20),
   RUE            VARCHAR(50),
   NUMERO         CHAR(4),
   CODE_POSTAL    INTEGER,
   COMMUNE        VARCHAR(30),
   DATE_NAISS     DATE,             (date de naissance)
   DATE_ENTREE    DATE,             (date d'entrée dans la société)
   DEPARTEMENT    VARCHAR(20),      (département)
   FONCTION       VARCHAR(40),      (fonction)
   GRADE          CHAR(2),
   LOCAL          DECIMAL(3,0))     (numéro du local occupé par cet employé)
```

2) la relation Horaires.

```
CREATE TABLE LVTEST.HORAIRES
 (BADGE          CHAR(3),          (numéro du badge)
  DATE_HORAIRE   DATE,             (date concernée)
  HEURE_DEBUT    TIME,             (heure d'arrivée théorique)
  HEURE_FIN      TIME,             (heure de départ théorique)
  TOTAL_HEURES   DECIMAL(3,0))     (nombre d'heures à prester)
```

3) la relation Pointages.

```
CREATE TABLE LVTEST.POINTAGES
 (BADGE          CHAR(3),          (numéro du badge)
  DATE_PTAGE     DATE,             (date du pointage)
  HEURE_PTAGE    TIME,             (heure du pointage)
  TYPE_PTAGE     CHAR(1))          (type du pointage)
```

4) la relation Compteurs.

```
CREATE TABLE LVTEST.COMPTEURS
 (BADGE          CHAR(3),          (numéro du badge)
  MOIS           SMALLINT,
  SEMAINE        SMALLINT,
  HRS_PREST_SEMAINE
                 DECIMAL(4,2),     (heures prestées depuis le début de la semaine)
  HRS_PREST_MOIS
                 DECIMAL(5,2),     (heures prestées depuis le début du mois)
  HRS_SUPPL_MOIS
                 DECIMAL(4,2),     (heures supplémentaires depuis le début du mois)
  CONGES         DECIMAL(3,1),     (nombre de jours de congé restants)
  CHEQ_REPAS     SMALLINT)         (jours entiers prestés depuis le début du mois)
```

5) la relation Absences.

```
CREATE TABLE LVTEST.ABSENCES
 (BADGE          CHAR(3),          (numéro du badge)
  DATE_ABSENCE   DATE,             (date de l'absence)
  TYPE_ABSENCE   SMALLINT)         (numéro représentant un motif d'absences)
```

Ces relations sont des relations statiques au plein sens du terme. Si on utilise cette modélisation, toute modification d'un tuple d'une de ces relations entraîne la perte de l'information modifiée.

Montrons quelques faiblesses de cette modélisation statique:

- Lorsque l'on arrive à la fin du mois, le nombre d'heures prestées depuis le début du mois doit être utilisé pour calculer le salaire et ensuite remis à zéro pour le mois suivant. dès que le salaire a été calculé, on n'a plus accès à ce nombre. De plus, on n'a pas la possibilité de le recalculer de manière certaine car il n'est pas toujours mis à jour de manière automatique. En effet, il peut faire l'objet de modifications manuelles. Par exemple, le chef de service ajoute une heure au compte d'un de ses employés.

  Il est donc impossible de revenir en arrière sans risques de perdre des informations.

La solution qui consiste à garder les informations du mois précédent celui en cours ne fait que déplacer le problème sans toutefois y apporter une solution solide.

- Les règles de calcul sont traduites en portions de code source des traitements. Si les règles changent, le code source est adapté, recompilé, et le nouveau code objet remplace l'ancien en l'écrasant. Par exemple: si un traitement journalier est modifié le 15 du mois, et si le 20 du même mois, on modifie les données du 10 et que l'on décide de retraiter les données à partir du 10, on ne disposera plus de la version du programme utilisée jusqu'au 14. Le référentiel des données [BURSENS 87], [BURSENS 89] enregistrées avant le 15 ne sera plus le même.

5. Quelles sont les notions théoriques de la première partie susceptibles d'apporter des éléments de solutions.

5.1. Les choix possibles.

Les trois approches étudiées apportent chacune des réponses. Laquelle convient le mieux au problème qui nous occupe?

- L'approche historique permet de mémoriser l'ensemble des valeurs prises par un type de données et d'y associer chaque fois le temps valide.
- L'approche temporelle permet également de mémoriser l'ensemble des valeurs prises par un type de données et d'y associer le temps valide et le temps transactionnel.
- L'approche combinée historique et dynamique permet de mémoriser l'ensemble des valeurs prises par un type de données et d'y associer le temps valide, un état dynamique et un référentiel décrivant la modélisation utilisée pour l'exploitation de la base de données.

5.2. Argumentation.

Dans un premier temps, j'éliminerai l'approche combinée historique et dynamique car cette approche n'est pas intéressante pour la modélisation d'une telle application, et cela pour deux raisons:

- Je ne vois pas en quoi le concept d'état dynamique pourrait être utile ici.
- L'approche historique possède une sémantique moins riche que l'approche temporelle.

Un seul argument est favorable à l'utilisation d'une telle approche. Il s'agit du concept de référentiel d'exploitation. Dans une telle application, le schéma de la

base de données est assez stable. Par contre, les traitements sont fréquemment mis à jour, selon le désir de l'employeur ou en fonction des nouvelles législations. Cet argument n'est pas suffisant pour faire oublier les deux autres. Par contre, l'idée sous-jacente au concept lui-même peut être utile. En effet, on peut imaginer de garder les différentes versions d'un traitement afin de pouvoir traiter les données avec la version du traitement qui était d'application au moment de l'enregistrement de celles-ci.

Dans un deuxième temps, j'éliminerai l'approche historique pour la raison déjà évoquée ci-avant. Elle n'est pas aussi souple que l'approche temporelle.

L'approche temporelle fait intervenir à la fois le temps valide et le temps transactionnel: cette combinaison a plusieurs avantages par rapport à l'approche historique:

- On obtient un suivi des corrections très précis.
- Lorsque l'on veut réeffectuer un traitement, on peut se resituer dans la base de données, de manière à travailler sur les mêmes données que lors de la précédente exécution du traitement (notion de Rollback).

5.3. Le choix adopté.

L'approche que j'adopterai pour proposer des solutions au problème posé est donc l'approche temporelle. Cette approche est, à mon sens, la plus appropriée pour la modélisation des données d'une application de gestion des prestations du personnel d'une entreprise.

6. Eléments de solutions.

Essayons de faire évoluer cette base de données ([SNODGRASS 87]) vers ce qui constituerait un embryon de base de données temporelle telle que définie dans [SNODGRASS 85].

Quelles sont les possibilités d'évolutions ?

1. Au niveau de la structure des tables relationnelles:

Dans SQL/400, il n'est pas possible d'adjoindre des attributs (colonnes) temporels implicites. Il n'est par contre pas interdit de rajouter des attributs (colonnes) de type date, time, ou timestamp explicites. L'aspect implicite sous-entend que la gestion de ces attributs est assurée par le SGBD.

Dans le cas des attributs explicites, aucune aide n'est fournie par le SGBD. Il faut donc se contenter des outils de base offerts par le SGBD, en l'occurrence les facilités du langage SQL/400.

2. Au niveau du langage de définition et de manipulation:

Les fonctions, les opérateurs arithmétiques et les opérateurs de comparaison ayant pour argument et/ou résultat des valeurs temporelles (date, time, timestamp) fournis par SQL/400 ne sont pas très puissants.

Les clauses TQuel telles que WHEN, VALID, AS-OF n'ont pas d'équivalentes dans SQL/400. Il faut donc travailler avec l'unique clause de sélection: WHERE. SQL/400 ne propose pas non plus de type intervalle de temps. Toutes les opérations sur les données de type intervalle de temps sont donc également absentes.

Le langage SQL/400 ne permet donc pas à lui seul d'aider dans le processus visant à introduire une sémantique temporelle dans la base de données. Le seul moyen d'action dont on dispose est donc la définition des tables.

On peut ajouter des colonnes supplémentaires à chacune des tables en utilisant la méthode élaborée par R. Snodgrass (cf. 1ère partie, chapitre 5: "Comment emboîter une relation temporelle dans une relation snapshot, version 2") [SNODGRASS 87]. Selon cette méthode, on rajoutera soit 3 champs (⇨ relation évènement) soit 4 (⇨ relation intervalle) à la structure de chacune des tables pour obtenir des relations temporelles emboîtées dans les relations "snapshot".

A chacune des tables EMPLOYES et COMPTEURS, on adjoint les quatres colonnes suivantes:

```
VTFROM      TIMESTAMP,      (valid time from ...)
VTTO        TIMESTAMP,      (valid time to ...)
TTSTART     TIMESTAMP,      (transaction start time)
TTSTOP      TIMESTAMP,      (transaction stop time)
```

De cette manière, on obtient 2 relations intervalles. Les trois autres relations sont des relations enregistrant des occurrences d'évènements tels qu'un pointage, une absence, les informations sur l'horaire d'une journée particulière. Il est donc logique de les faire évoluer vers des relations évènements

A chacune des tables HORAIRES, POINTAGES et ABSENCES, on adjoint les deux colonnes suivantes:

```
TTSTART     TIMESTAMP,      (transaction start time)
TTSTOP      TIMESTAMP,      (transaction stop time)
```

Par ailleurs, les colonnes DATE_HORAIRE et DATE_ABSENCE sont chacune remplacées par une colonne

```
VTAT        TIMESTAMP,      (valid time at ...)
```

Les colonnes DATE_PTAGE et HEURE_PTAGE sont fusionnées et remplacées par une colonne

```
VTAT        TIMESTAMP,        (valid time at ...)
```

Que peut-on attendre d'une telle modélisation?, ou en d'autres mots, quelles sont les questions auxquelles on peut répondre sur base des données enregistrées dans une telle structure?

Prenons pour cela des exemples de requêtes du type TQuel et essayons de les "traduire" en SQL/400[9]:

Exemple 1: Quelle est l'adresse de l'employé Mr Dupont?

```
range of e is emp
retrieve into Adresse   (Rue = e.Rue, Numero = e.Numero,
                         Codpos = e.Codpos, Commun = e.Commun)
where e.Nom = "Dupont"
```

Dans cette requête, les valeurs par défaut pour la clause ***when*** et la clause ***as of*** sont d'application: on a ici:

```
valid from begin of e to end of e
when e overlap "now"
as of "now"
```

Cette requête, bien qu'élémentaire, présente déjà quelques difficultés lors de sa réécriture en SQL/400 sur une base de données statique:

```
create view Adresse
as  select Rue, Numero, Codpos, Commun
    from emp
    where   (Nom = 'Dupont')
            and (current date between Vtfrom and Vtto)
            and (Ttstop is null)
```

[9] *Le but de cet exercice est de voir s'il est possible de simuler un langage d'interrogation temporel en utilisant un langage d'interrogation classique et les "emboîtements" définis plus avant.*

Remarques:

- J'utilise le concept de vue pour rester général, c'est-à-dire pour ne pas forcer le résultat à contenir un seul tuple au maximum comme ce serait le cas si j'utilisais le statement "**select into**".
- Les valeurs infinies sont ici remplacées par la valeur particulière "**null**" qui a la propriété remarquable d'être supérieure à toute autre valeur lors de comparaisons.
- Le fait de tester Ttstop par rapport à la valeur "null" oblige l'adresse à être la dernière et à être toujours valable, c'est-à-dire que le tuple n'a pas été effacé de la base de données.

Exemple 2: Liste des comptables en 1991?

```
range of e is emp
retrieve into Comptables (Nom = e.Nom)
where e.Fonct = "Comptable"
when e overlap "1991"
```

Cette requête utilise la clause when et l'opérateur overlap, qui n'existent pas dans le SQL standard.

```
create view Comptables
as  select Nom
    from emp
    where   Fonct = 'Comptable'
            and (year(Vtfrom) <= 1991)
            and (year(Vtto) >= 1991)
            and (Ttstop is null)
```

Exemple 3: Liste des absences des comptables en janvier 1992?

```
range of e is emp
range of a is abs
retrieve into Abs_Compta (Nom = e.Nom, Vtat = a.Vtat)
where  a.badge = e.badge
       and e.Fonct = 'Comptable'
when a overlap "Janvier 1991"
```

Cette requête peut être réécrite comme suit:

```
create view Abs_Compta
as  select Nom, Vtat
    from emp, abs
    where   abs.badge = emp.badge
            and Fonct = 'Comptable'
            and date(Vtfrom) <= date('31/01/1991')
            and date(Vtto) >= date('01/01/1991')
            and (year(Vtat) = 1991)
            and (month(Vtat) = 1)
            and abs.Ttstop is null
            and emp.Ttstop is null
```

Remarque:

On voit ici que lorsque l'on effectue des requêtes sur plusieurs tables à la fois, la complexité de la clause "where" dans le statement SQL/400 augmente considérablement.

Exemple 4: Que savait-on le 15 juillet 1992 à propos du nombre d'heures supplémentaires prestées par le personnel durant le mois de juin 1992?

```
range of e is emp
range of c is cpt
retrieve into Heures_sup (Nom = e.Nom, Hsumoi =  c.Hsumoi)
where  c.badge = e.badge
when c overlap "Juin 1992"
as of "15 juillet 1992"
```

Cette requête peut être réécrite comme suit:

```
create view Heures_sup
as  select Nom, Hsumoi
    from emp, cpt
    where   cpt.badge = emp.badge
            and date(cpt.Vtfrom) >= date('01/06/1992')
            and date(cpt.Vtto) <= date('31/06/1992')
            and date(cpt.Ttstart) <= date('15/07/1992')
            and date(cpt.Ttstop) > date('15/07/1992')
```

Remarque:

Une fois de plus, on remarque la complexité de la clause "where". Le statement TQuel n'est déjà pas évident à écrire, mais lorsqu'il s'agit de transposer sa sémantique dans une requête SQL/400, cela devient très ardu.

Comme il apparaît dans ce qui précède, il est possible d'obtenir une connaissance approfondie sur l'évolution des données enregistrées dans une base de données. Il est également possible de retrouver ces données en utilisant des requêtes appropriées.

Un autre axe de réflexion peut être dégagé de la section 5: il s'agit de l'évolution des traitements appliqués aux données d'une telle base de données. Il semble évident qu'une version d'un traitement doit faire partie de l'environnement, du référentiel [BURSENS 87], [BURSENS 89] des données qu'il est susceptible de consulter ou de modifier. Il faudrait donc établir un système permettant de ne pas perdre l'association "données - versions des traitements".

Une fois ces deux objectifs atteints, il faut encore considérer les problèmes de recalculs. En effet, lorsque l'on décide de modifier quelque chose dans le passé, que ce soit des données ou des traitements, il faut pouvoir décider des actions à entreprendre afin de remettre l'entièreté du système "en phase" avec les modifications déjà effectuées. Cela amène le système devant des situations auxquelles il ne connaît pas toujours la parade. Il appartient donc au développeur d'application de trouver tous les cas d'exception pouvant survenir et de codifier tous ces cas ainsi que les attitudes à avoir face à ceux-ci. Dans ce domaine, il sera fortement question d'intelligence artificielle. Mais ceci dépasse de loin les ambitions limitées du présent mémoire.

7. Travaux futurs.

Il serait intéressant de pouvoir appliquer les quelques résultats obtenus au moyen de systèmes intégrant une gestion du temps (dès que de tels systèmes seront disponibles sur le marché ...).

En attendant, le développement d'une application réelle de gestion des horaires de personnel se voulant résolument temporelle apparaît être un développement assez important mais il en vaut la peine étant donné le nombre de clients potentiel pour ce type d'applications. Il est certain qu'une grande partie des concepts en matière de temps pourraient être repris de cette application afin de créer d'autres applications requérant une gestion du temps.

Sur base de plusieurs développements tels que ceux-ci, on pourrait alors jeter les bases d'un SGBD temporel "universel", c'est-à-dire convenant à une majorité d'applications ayant les mêmes besoins en matière de temps.

Il serait d'autre part utile de soumettre les souhaits concernant l'apport possible de l'intelligence artificielle à une recherche plus profonde.

◆◆◆◆◆

Bibliographie

[ADIBA 80] M. Adiba, B. Lindsay: ***Databases snapshots.*** Proceedings of the 6th VLDB Montréal, Canada, 1980.

[ADIBA 86] M. Adiba, N. Bui Quang: ***Historical multi-media databases.*** Proceedings of the twelfth international conference on VLDB, Kyoto 1986.

[ADIBA 87] M. Adiba, N. Bui Quang, C. Collet: ***Aspects temporels, historiques et dynamiques des bases de données.*** Technique et science informatiques, vol.6, n°5, Grenoble 1987.

[ALLEN 83] J.F. Allen: ***Maintaining knowledge about temporal intervals.*** Communications of the ACM, 1983 vol.26 n°11.

[ALLEN 87] J.F. Allen, P.J. Hayes: ***Moments and points in an interval-based temporal logic.*** Technical report 180, University of Rochester Computer Science 1987.

[ARIAV 82] G. Ariav, H.L. Morgan: ***MDM: Embedding the time dimension in information systems.*** Technical report 82-03-01. Department of decision sciences, The Warton School, University of Pennsylvania 1982.

[ARIAV 83] G. Ariav, J. Clifford, M. Jarke: ***Time and databases.*** ACM-SIGMOD international conference on management of data, San Jose, Californie, Mai 1983.

[ARIAV 86] G. Ariav: ***A temporally oriented data model.*** ACM transactions on database systems, 1986, vol.11, n°4.

[BEN-ZVI 82] J. Ben-Zvi: ***The time relational model.*** Ph. D. Dissertation, University of California, Los Angeles, 1982.

[BHARGAVA 90] G. Bhargava, S.K. Gadia: ***The concept of error in a database: an application of temporal databases.*** Proceedings of 1990 international conference on management of data.. N. Prakash, Ed Tata McGraw-Hill, New Delhi.

[BODART 83] F. Bodart, Y. Pigneur: ***Conception assistée des applications informatiques. 1. Etude d'opportunité et analyse conceptuelle.*** Masson 1983.

[BREUTMANN 79] B. Breutmann, E.F. Falkenberg, R. Mauer: ***CSL: A language of defining conceptual schemas.*** Data Base Architecture, Amsterdam 1979.

[BURSENS 87] R. Bursens, J. Guyot: ***Temps + dynamique + référentiel = histoire.*** Conférence de Port-Camargue 1987.

[BURSENS 89] R. Bursens: ***Bases de données historiques et temporelles, dépendances temporelles inter-contextes.*** Notes établies à la suite d'une collaboration avec le CERN, Novembre 1989.

[CLIFFORD 83] J. Clifford, D.S. Warren: ***Formal semantics for time in databases.*** ACM transactions on database systems 1983, vol.8, n°2.

[CLIFFORD 85] J. Clifford, A.U. Tansel: ***On an algebra for histiorical relational databases: two views.*** Proceedings of ACM SIGMOD international conference on management of data, S Navathe ed., ACM Press, Austin, Texas, mai 1985.

[COPELAND 84] G. Copeland, D. Maier: ***Making Smalltalk a database system.*** Proceedings of the SIGMOD '84 conference, Juin 1984.

[DADAM 84] P. Dadam, V. Lum, H.D. Werner: ***Integration of time versions into a relational database system.*** Proceedings of the conference on VLDB, U.Dayal, G. Schlageter, L.H. Seng Eds. Singapore, Août 1984.

[GADIA 85] S.K. Gadia, J.H. Vaishnav: ***A query language for a homogeneous temporal database.*** Proceedings of the ACM Symposium on principles of database systems, ACM Press, Mars 1985.

[GADIA 88] S.K. Gadia: ***A homogeneous relational model and query languages for temporal databases.*** ACM transactions on database systems, ACM Press, vol.13, n°4, Décembre 1988.

[GUYOT 86] J. Guyot: ***Un modèle de traitements pour les bases de données.*** Thèse de la faculté des sciences de l'université de Genève, Concept Moderne / Editions, octobre 1986.

[JONES 79] S. Jones, P.J. Mason, R. Stamper: ***LEGOL 2.0: a relational specification language for complex rules.*** Information systems, 4, n°4, Novembre 1979.

[JONES 80] S. Jones, P.J. Mason: ***Handling the time dimension in a database.*** Proceedings of the international conference on databases, S.M. Deen, P. Hammersley Eds, British Computer Society, University of Aberdeen: Heyden, Juillet 1980.

[KLOPPROGGE 81] M.R. Klopprogge: ***TERM: An approach to include the time dimension in the entity-relationship model.*** Proceedings of the second international conference on entity relationship approach, Octobre 1981.

[KOENIG 81] S. Koenig, R. Paige: ***A transformational framework for the automatic control of derived data.*** Proceedings of 7th VLDB, Cannes, Septembre 1981.

[LUM 84] V. Lum, P. Dadam, R Erbe, J. Guenauer, P. Pistor, G. Walch, H Werner, J. Woodfill: ***Designing DBMS support for the temporal dimension.*** Proceedings of the Sigmod '84 conference, Juin 1984.

[MCKENZIE 86] L.E. McKenzie: ***Bibliography: Temporal databases.*** ACM SIGMOD RECORD, vol.15, n°4, Décembre 1986

[MCKENZIE 90] L.E. McKenzie, R. Snodgrass: ***Schema evolution and the relational algebra.*** Information systems, vol.15, n°2, Juin1990.

[MCKENZIE 91] L.E. McKenzie, R. Snodgrass: ***Evaluation of relational algebras incorporating the time dimension in databases.*** ACM Computing surveys, vol.23, n°4, Décembre 1991.

[MUELLER 83] T. Mueller, D. Steinbauer: ***Eine Sprachschnittstele zur versionenkontrolle in CAM-datanbanken.*** Informatik-Fachberichte, Berlin, springer 1983.

[OVERMYER 82] R. Overmyer, M. Stonebraker: ***Implementation of a time expert in a database system.*** SIGMOD RECORD, vol.12, n°3, Avril 1982.

[REED 78] D. Reed: ***Naming and synchronization in a decentralized computer system.*** Ph. D. Dissertation M.I.T., Septembre 1978.

[SNODGRASS 84] R. Snodgrass: ***The temporal Query Language TQuel.*** Proceedings of 3rd ACM SIGACT-SIGMOD Symposium on principles of database systems, Waterloo, Canada, Avril 1984.

[SNODGRASS 85] R. Snodgrass, I. Ahn: ***A taxonomy of time in databases.*** Proceedings of ACM SIGMOD, Mai 1985.

[SNODGRASS 87] R. Snodgrass: ***The temporal query language TQuel.*** ACM transactions on database systems, vol.12, n°2, Juin 1987.

[TAIS 87] Temporal Aspects in Information Systems. Conférence TAIS Sophia-Antipolis, 12-15 Mai 1987.

[VELEZ 85] F. Velez: ***LAMBDA: An entity-relationship based language for the retrieval of structured documents.*** Proceedings of the 4th international conference on entity relationship approach, Chicago, Octobre 1985.