Université de Namur (FUNDP)
Laboratoire d'ingénierie des applications de bases de données
Institut d'Informatique
Rue Grandgagnage, 21
B-5000 Namur
http://www.info.fundp.ac.be/libd

TimeStamp

Understanding
Developing
Processing Temporal databases

Responsable : Jean-Luc Hainaut
Chercheurs : Virginie Detienne, Didier Roland, Thomas Lieutenant

Volume 3
Documents techniques

Deuxième partie
Documentations techniques des outils

Financé par le Ministère de la Région Wallonne (contrat 9713563)

Laboratoire d'ingénierie des applications de bases de données

TimeStamp

Comprendre, développer et exploiter les données temporelles

Volume3

Documents techniques

Deuxième partie

Documentations techniques des outils

Janvier 2002

Le projet TimeStamp a été financé par le Ministère de la Région Wallonne (contrat 9713563)

Le volume "Documents techniques" analyse diverses matières en profondeur. Il décrit les différentes pistes de réflexion qui ont été abordées, justifie les choix effectués, donne des explications techniques,... Il est destiné aux personnes désireuses d'approfondir certains sujets évoqués dans les Volumes 1 et 2, ainsi que celles qui souhaitent avoir des informations techniques concernant les outils.

Le volume est divisé en deux parties. La première comprend des rapports techniques de recherches, tandis que la seconde offre la documentation technique des outils.

Rapports techniques de recherche

Divers thèmes sont abordés dans la première partie.

Les données bitemporelles sont d'abord étudiées. On analyse la façon de représenter des données bitemporelles non valides, passées, courantes et futures.

La structuration des données est ensuite analysée. Les données temporelles peuvent être localisées dans les tables selon différentes stratégies, de façon à optimiser des critères tels que l'occupation d'espace mémoire, la performance des requêtes,... Différentes structures de données sont analysées et trois d'entre elles sont sélectionnées pour la suite des travaux. Ce sont ces trois structures qui sont gérées par l'outil de génération de bases de données temporelles de DB-Main.

Les propriétés temporelles des éléments d'un schéma conceptuel doivent être conservées lors des transformations. L'héritage du caractère temporel des éléments d'un schéma est analysé pour différentes transformations.

Pour être cohérent, un schéma conceptuel temporel doit répondre à certains critères qui sont, notamment, liés à l'aspect temporel des données. Ces différentes règles sont décrites et justifiées.

Un schéma logique temporel peut comprendre des tables d'entités et des tables d'associations. Les historiques de ces deux types de tables ont des propriétés différentes et ne se gèrent pas de la même manière. Les règles concernant les historiques d'associations sont discutées et définies.

Différents modèles ont été décrits dans le cadre du projet TimeStamp. Ils sont comparés avec d'autres modèles répertoriés dans la littérature.

Les langages classiques n'ont pas été conçus pour accéder aux bases de données temporelles. Dans le cadre du projet TimeStamp, un nouveau langage a été spécifiquement élaboré pour consulter ce type de bases de données (mini-TSQL2). De plus, de nouvelles fonctions de type ODBC (T-ODBC) ont été implémentées afin de permettre l'exécution des requêtes formulées au moyen de ce langage. Les sémantique et syntaxe de mini-TSQL2 sont définies. L'amélioration d'ODBC en T-ODBC est justifiée et décrite. Des opérateurs temporels sont également spécifiés.

Documentations techniques des outils

Les outils de génération de bases de données temporelles sont d'abord décrits. Des patterns de génération de triggers destinés à la gestion de l'aspect temporel des bases de données sont proposés. Une étude sur les performances de ces bases de données générées est ensuite effectuée.

Les outils d'exploitation des bases de données temporelles sont à leur tour documentés.

On décrit d'abord un générateur de scripts SQL de population d'une base de données temporelles, qui est utilisé dans le but d'effectuer des tests sur des bases de données temporelles volumineuses.

Des tests de performance de requêtes d'exploitation types sont ensuite réalisés.

Enfin, la documentation technique de l'API T-ODBC est fournie. Elle décrit les caractéristiques des différents modules.

Table des matières

Volume 1 - Introduction aux bases de données temporelles

Volume 2 - Méthodologie et exploitation des bases de données temporelles

Volume 3 - Documents techniques

TimeStamp

Volume 3 - Documents techniques

Deuxième partie

Documentations techniques des outils

Les outils de génération de bases de données temporelles

DB-Main Manual Series

Projet TimeStamp

PERFORMANCES DES BASES DE DONNÉES GÉNÉRÉES PAR DB-MAIN

VIRGINIE DETIENNE

MARS 2002

The University of Namur - LIBD
Institut d'Informatique
Rue Grandgagnage, 21 B-5000 Namur
http://www.info.fundp.ac.be/libd

TimeStamp

Performances des bases de données temporelles générées par DB-Main

Version 1

Mars 2002

Virginie Detienne

1. Introduction

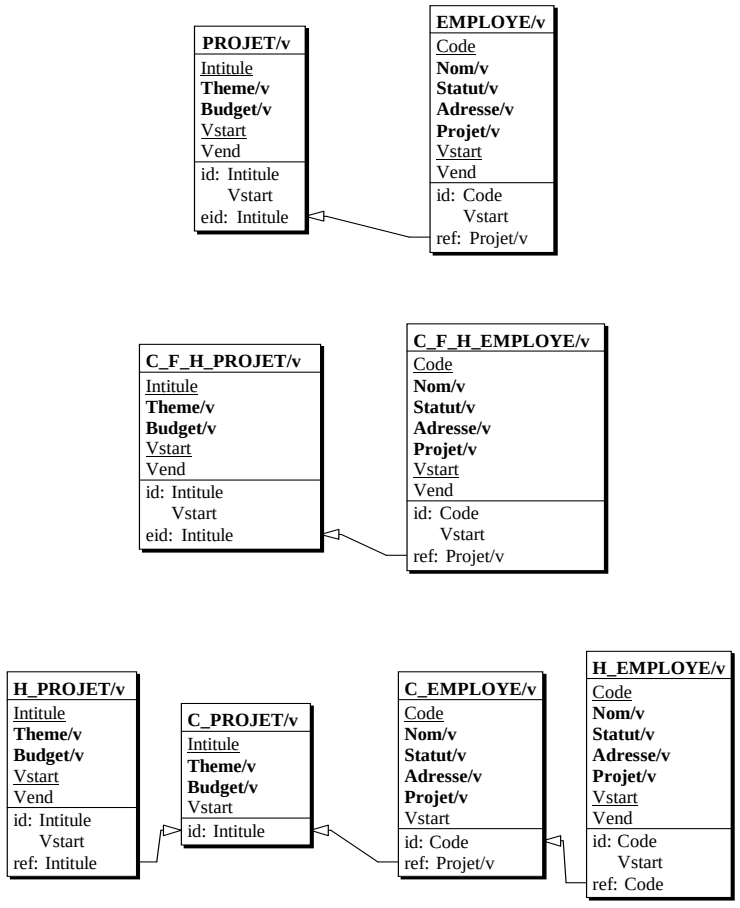
L'outil CASE BD-Main permet de générer automatiquement du code SQL (Oracle8) à partir d'un schéma physique temporel, et ce pour trois configurations de données différentes. Ce code comprend la définition des tables, mais également des triggers qui gèrent l'aspect temporel des données. Toutes les opérations nécessaires sont ainsi automatiquement réalisées lors de chaque INSERT, UPDATE ou DELETE.

Nous allons analyser, pour ces trois configurations de données, les performances des bases de données générées. Nous allons également discuter la présence de différents index destinés à améliorer les temps d'exécution des requêtes de gestion de données.

2. Bases de données de tests

Les tests ont été effectués sur deux bases de données : une monotemporelle et une bitemporelle. La gestion de ces dernières données étant plus complexe, ces bases de données sont généralement moins performantes.

A chaque schéma logique d'une de ces bases de données, correspondent trois schémas physiques représentant les trois structures de données différentes (Figure 1 et Figure 2). Dans la première configuration, tous les états d'un objet sont localisés dans la même table. Dans la deuxième, les états courants sont stockés dans une table et les états passés et non valides dans une autre. Dans la troisième structure, tous les états sont groupés dans une même table et il y une a copie des états courants dans une table spécifique.



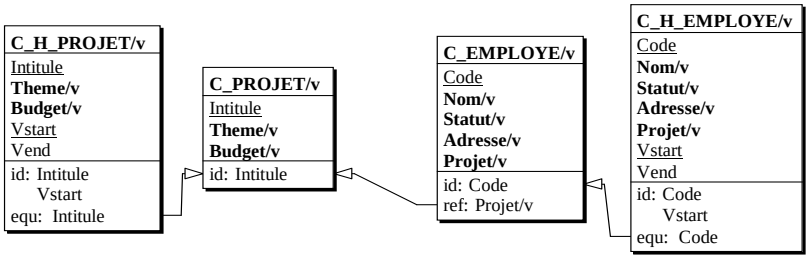
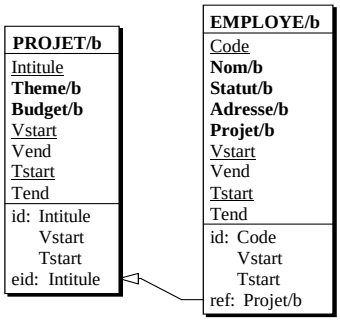
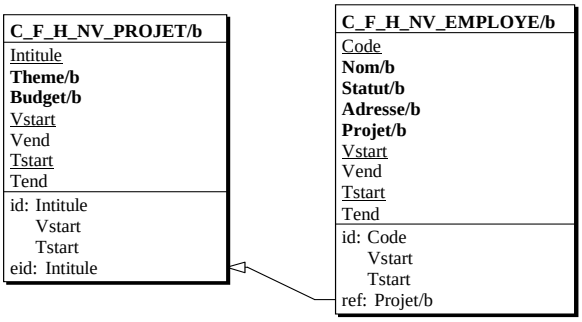
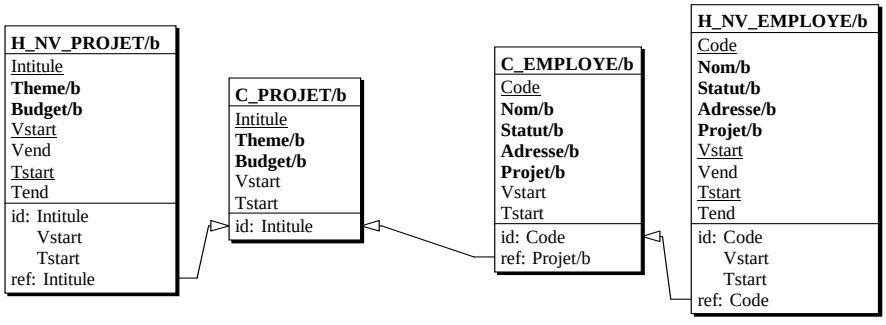


Figure 1 : Base de données monotemporelles
De haut en bas : schéma logique, première configuration du schéma physique, deuxième configuration du schéma physique et troisième configuration du schéma physique







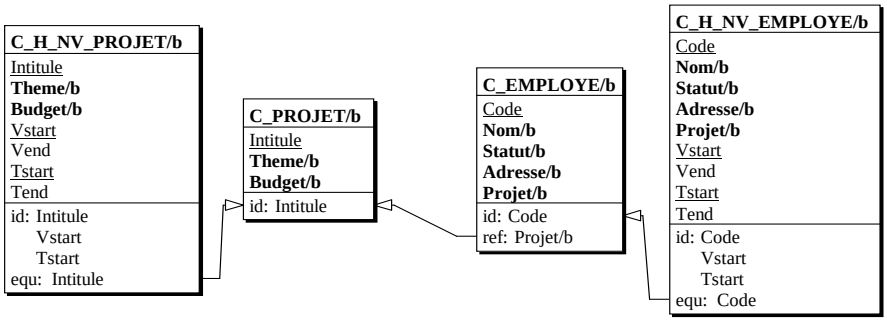


Figure 2 : Base de données bitemporelles
De haut en bas : schéma logique, première configuration du schéma physique, deuxième configuration du schéma physique et troisième configuration du schéma physique

Les deux schémas logiques sont composés de deux tables et d'une clé étrangère temporelle. Les opérations sont réalisées sur le schéma logiques et traduites au niveau physique par les triggers.

Les opérations testées sont des opérations de gestion, c'est-à-dire des insertions, des mises-à-jour (évolutions essentiellement) et des suppressions d'entités.

3. Choix des index

Différents index vont être testés.

Lorsqu'on analyse les triggers de mise-à-jour et de gestion des clés étrangères, on s'aperçoit qu'ils contiennent plusieurs requêtes dont les conditions comprennent les identifiants d'entités et les dates, ainsi que les clés étrangères et les dates.

A partir de cette analyse, trois stratégies d'utilisation d'index ont été déterminées.

De manière globale on peut dire que :

- Dans la première configuration, les identifiants primaires de chaque table sont des index.
- Dans la deuxième, les identifiants primaires, les identifiants d'entités additionnés des dates de fin, et les clés étrangères additionnées des dates de début et de fin, sont des index.
- Dans la troisième, les identifiants primaires, les identifiants d'entités additionnés des dates de fin, et les clés étrangères additionnées des dates de fin, sont des index.

Tous les index sont décrits dans un tableau détaillant chaque configuration et chaque dimension temporelle des tables. Les symboles ont les significations suivantes :

ide : identifiant d'entité

fk : clé étrangère

Vs, Ve, Ts, Te : Vstart, Vend, Tstart, Tend

	Valid Time		Transaction Time		Bitemporel	
Configuration 1	**Table C_F_H_**		**Table C_H_**		**Table C_F_H_NV_**	
Ind 1	ide,Vs		ide,Ts		ide,Vs, Ts	
Ind 2	ide, Vs ide, Ve fk, Vs, Ve		ide, Ts ide, Te fk, Ts,TVe		ide, Vs, Ts ide, Ve, Te fk, Vs, Ve, Ts, Te	
Ind 3	ide, Vs ide, Ve fk, Ve		ide, Ts ide, Te fk, Te		ide, Vs, Ts ide, Ve, Te fk, Ve, Te	
Configuration 2	**Table H_**	**Table C_**	**Table H_**	**Table C_**	**Table H_NV_**	**Table C_**
Ind 1	ide,Vs	ide	ide,Ts	ide	ide,Vs, Ts	ide
Ind 2	ide, Vs ide, Ve fk, Vs, Ve	ide fk	ide, Ts ide, Te fk, Ts, Te	ide fk	ide, Vs, Ts ide, Ve, Te fk, Vs, Ve, Ts, Te	ide fk
Ind 3	ide, Vs ide, Ve fk, Ve	ide fk	ide, Ts ide, Te fk, Te	ide fk	ide, Vs, Ts ide, Ve, Te fk, Ve, Te	ide fk
Configiguration 3	**Table C_H_**	**Table C_**	**Table C_H_**	**Table C_**	**Table C_H_NV_**	**Table C_**
Ind 1	ide,Vs	ide	ide,Ts	ide	ide,Vs, Ts	ide
Ind 2	ide, Vs ide, Ve fk, Vs, Ve	ide fk	ide, Ts ide, Te fk, Ts, Te	ide fk	ide, Vs, Ts ide, Ve, Te fk, Vs, Ve, Ts, Te	ide fk
Ind 3	ide, Vs ide, Ve fk, Ve	ide fk	ide, Ts ide, Te fk, Te	ide fk	ide, Vs, Ts ide, Ve, Te fk, Ve, Te	ide fk

4. Analyse des résultats

Les tests ont été effectués sur les neuf bases de données Valid Time (3 configurations de données X 3 stratégies d'index) et les neufs bitemporelles.

Le point de départ était à chaque fois une base de données vide. Des séries de 200, 1000, 10000, et 200000 opérations ont été testées.

La Figure 3 donne, pour chacun des tests, le nombre moyen de requêtes exécutées par seconde.

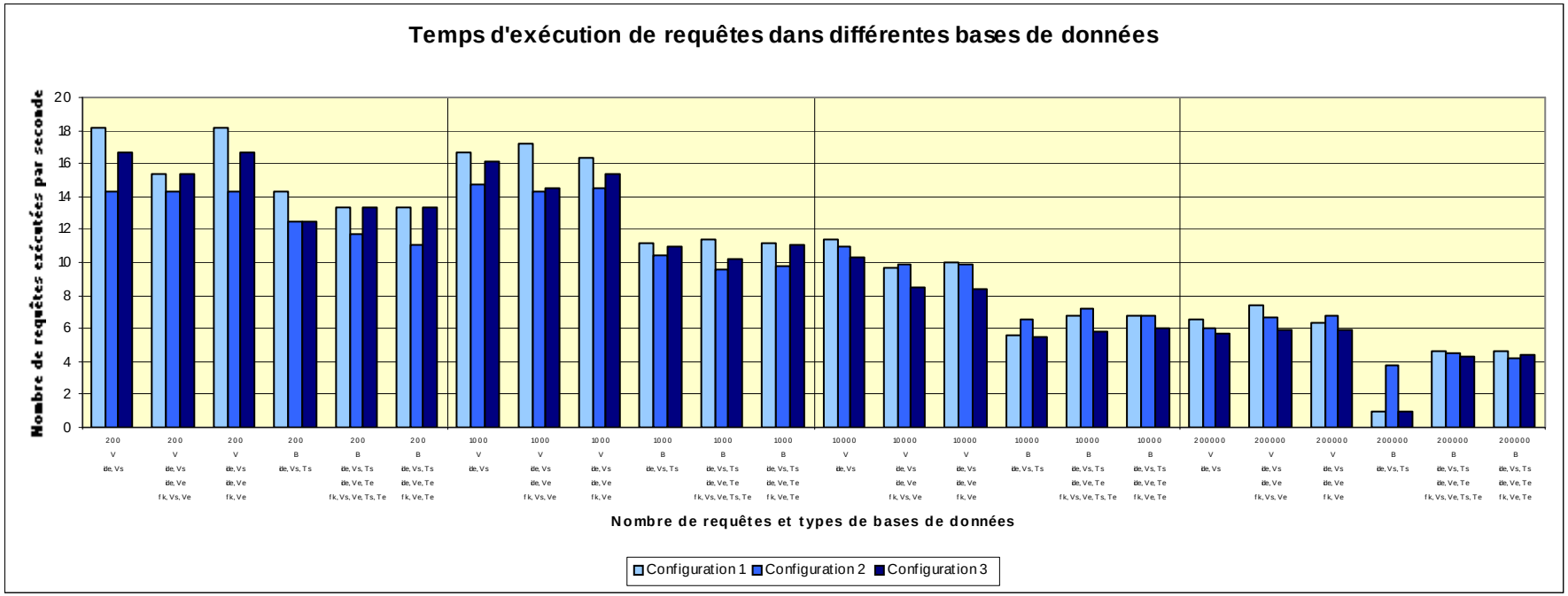


Figure 3 : Résultats des tests

On notera tout d'abord que certains triggers ont pour objectif de gérer les contraintes référentielles lors de chaque opération. Ces vérifications atténuent les performances des requêtes.

On observe que les performances moyennes diminuent lorsque la taille de la base de données augmente. On passe ainsi d'une moyenne de 18 opérations par seconde pour une table monotemporelle de 100 lignes, à une opération par seconde pour une table de 180000 lignes.

Les performances des bases de données bitemporelles sont inférieures aux monotemporelles. Ceci est dû aux faits qu'il y a plus d'opérations à effectuer pour gérer les données bitemporelles, et que les tables sont plus volumineuses.

Une comparaison entre les différentes configurations montre que dans les petites bases de données (moins de 4000 lignes par table), la première configuration est la plus performante alors que la deuxième offre les moins bons résultats.

Lorsque la taille de la base de données augmente (plus de 4000 lignes), la deuxième configuration obtient des résultats équivalents, voire superieurs aux deux autres structures de données, surtout s'il s'agit de bases de données bitemporelles. Cela s'explique par le type des opérations effectuées dans les tests. Il s'agit en effet d'opérations concernant les états courants. Dans la deuxième configuration, ces états sont stockés dans une table spécifique de petite taille, alors que pour les deux autres structures, les états courants doivent être retrouvés dans des tables reprenant l'ensemble des états. Si les opérations testées avaient été en majorité des corrections d'états passés, les résultats de cette configuration n'égaleraient pas ceux des deux autres structures de données. La troisième configuration est ici la moins performante. Sa gestion est en effet équivalente à la gestion de la première configuration, additionnée de la gestion de la table des états courants.

Dans les petites et moyennes bases de données (moins de 80000 lignes), les trois ensembles d'index testés donnent des performances comparables. Dans les bases de données monotemporelles de plus grande taille (plus de 80000 lignes), il semble que le deuxième index offre des résultats légèrement supérieurs aux deux autres. Pour les bases de données bitemporelles de grande taille (180000 lignes), il est fortement conseillé de faire usage des index des stratégies 2 et 3. On observe en effet que les performances de la stratégie 1 pour les configuration de données 1 et 3 sont extrêment faibles (moyenne d'une opération par seconde).

Pour des analyses plus poussées des tests, le lecteur peut se référer aux graphiques situés en annexes. Pour chacun des tests, on donne la durée moyenne d'un nombre déterminé d'opérations, ainsi que l'évolution de la durée de l'ensemble des opérations.

5. Conclusions

Nous avons testé les performances de différentes bases de données sur lesquelles on effectue des opérations de gestion.

Cette analyse indique que suivant la taille des tables, certaines configurations sont plus ou moins performantes que d'autres. Ainsi pour des tables de moins de 4000 lignes, les temps d'execution des requêtes sont plus courts pour les configurations 1 et 3. Lorsque la taille des tables augmente, on préférera les configurations 1 et 2.

Certains index permettent d'obtenir des résultats nettement meilleurs que d'autres. C'est le cas notamment de l'index 2 pour les configurations 1 et 3 dans tables de grandes tailles (180000 lignes). Il permet en effet d'obtenir des temps d'exécution 4 fois plus rapides.

6. Annexes

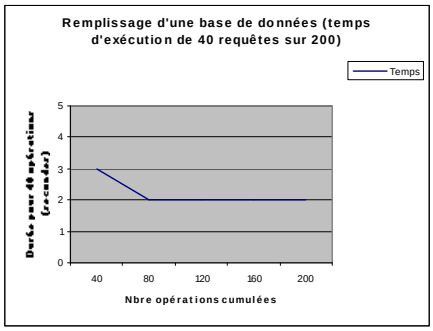
6.1. Configuration de données 1

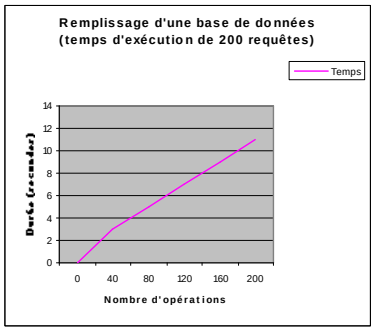
6.1.1. 200 opérations

6.1.1.1. Base de données monotemporelles

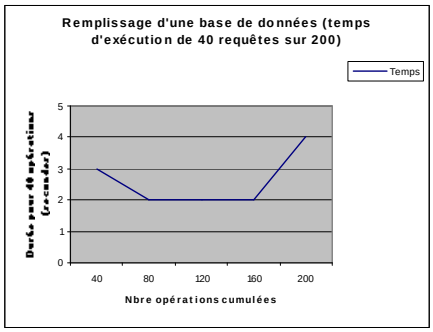
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_F_H_EMPLOYE	68	10	6,8
C_F_H_PROJET	103	12	8,6

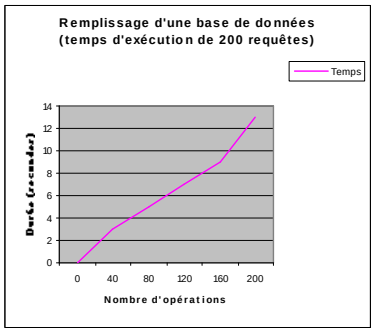
Index : ide,Vs



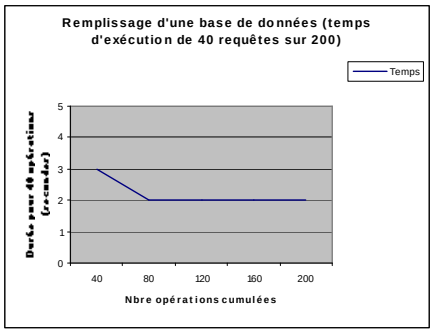


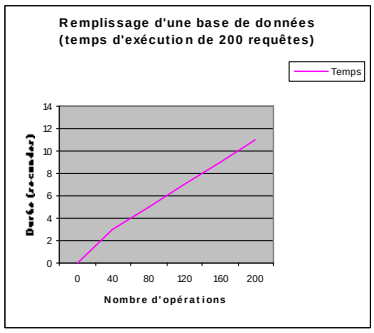
Index : ide,Vs - ide,Ve - fk,Vs,Ve





Index : ide,Vs - ide,Ve - fk,Ve

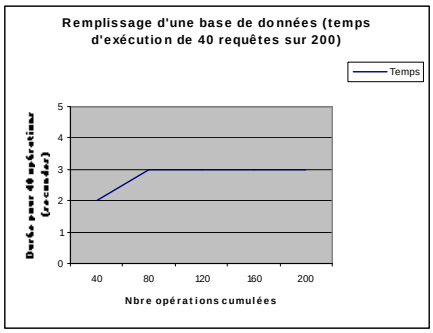


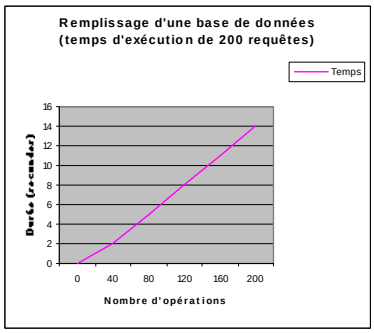


6.1.1.2. Base de données bitemporelles

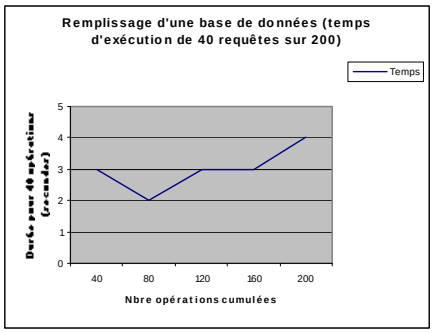
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_F_H_NV_EMPLOYE	206	12	17,2
C_F_H_NV_PROJET	163	10	16,3

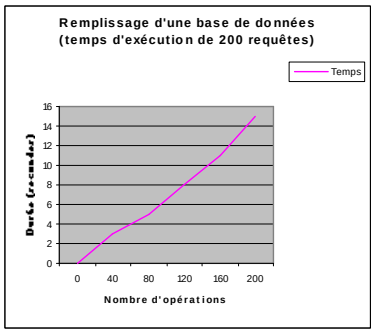
Index : ide,Vs,Ts



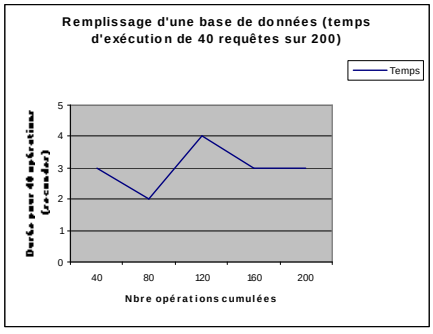


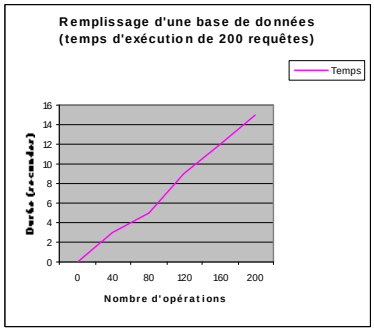
Index : ide,Vs,Ts - ide,Ve,Te - fk,Vs,Ve,Ts,Te





Index : ide,Vs,Ts - ide,Ve,Te - fk,Ve,Te



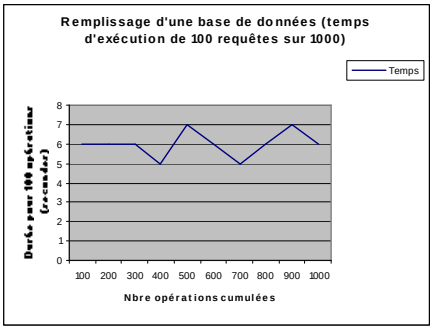


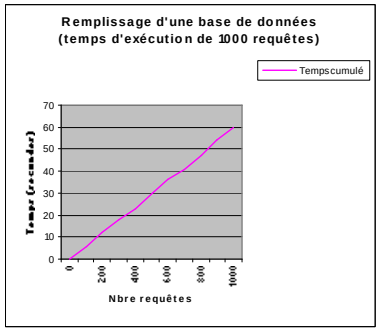
6.1.2. 1000 opérations

6.1.2.1. Base de données monotemporelles

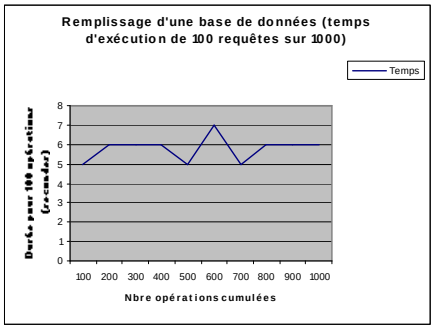
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_F_H_EMPLOYE	527	45	11,7
C_F_H_PROJET	334	41	8,1

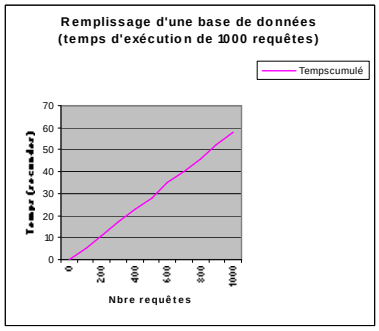
Index : ide,Vs



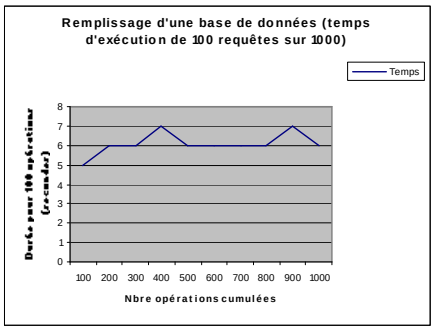


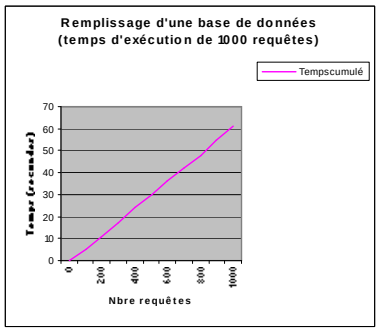
Index : ide,Vs - ide,Ve - fk,Vs,Ve





Index : ide,Vs - ide,Ve - fk,Ve

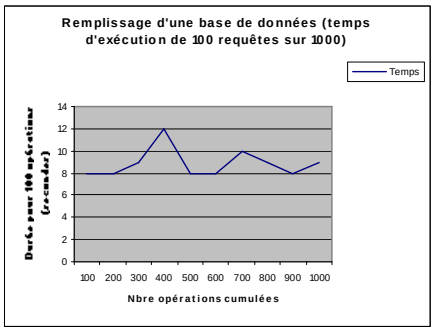


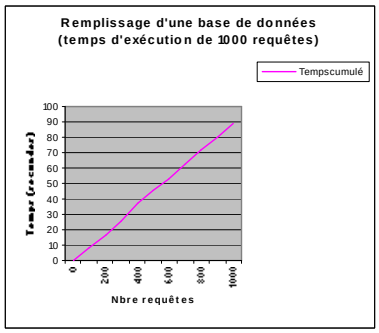


6.1.2.2. Base de données bitemporelles

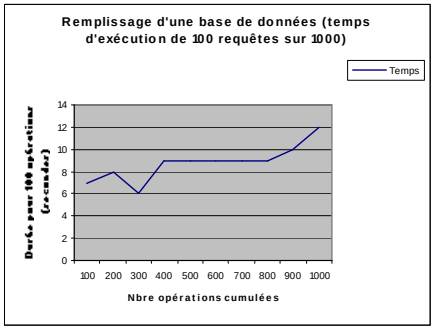
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_F_H_NV_EMPLOYE	1093	45	24,3
C_F_H_NV_PROJET	776	41	18,9

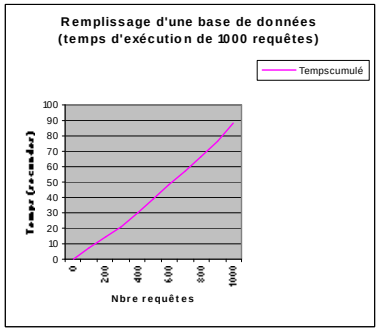
Index : ide,Vs,Ts



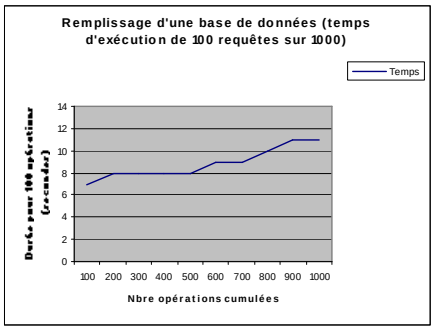


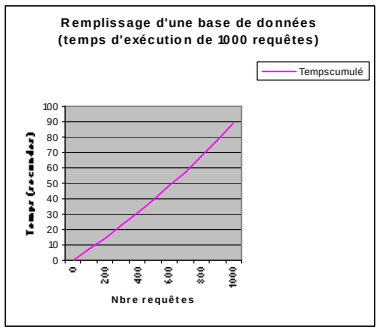
Index : ide,Vs,Ts - ide,Ve,Te - fk,Vs,Ve,Ts,Te





Index : ide,Vs,Ts - ide,Ve,Te - fk,Ve,Te



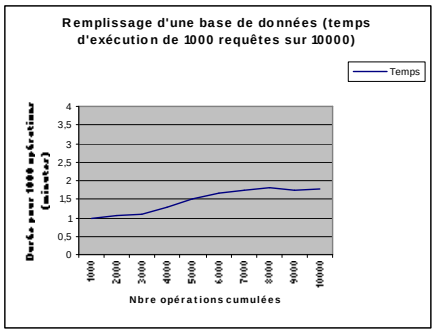


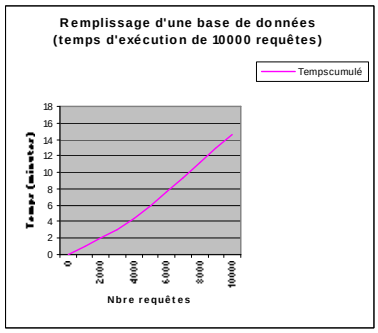
6.1.3. 10000 opérations

6.1.3.1. Base de données monotemporelles

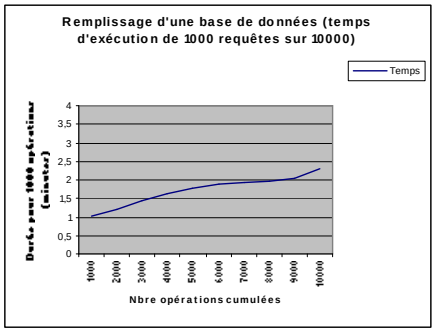
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_F_H_EMPLOYE	5433	482	11,3
C_F_H_PROJET	3384	339	9,9

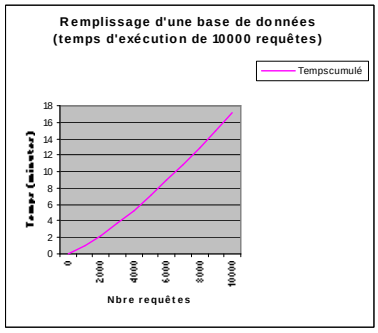
Index : ide,Vs



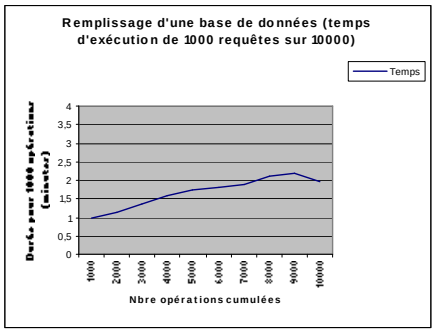


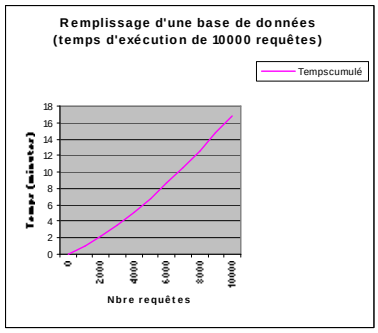
Index : ide,Vs - ide,Ve - fk,Vs,Ve





Index : ide,Vs - ide,Ve - fk,Ve

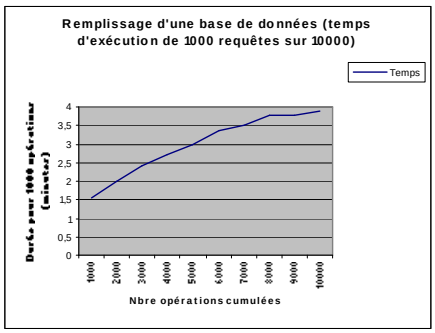


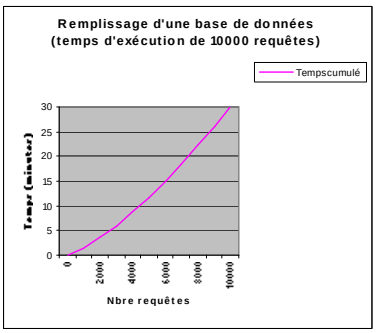


6.1.3.2. Base de données bitemporelles

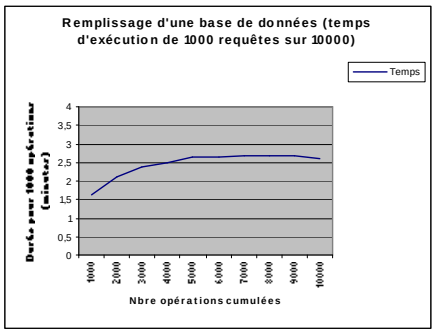
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_F_H_NV_EMPLOYE	10779	482	22,4
C_F_H_NV_PROJET	7686	339	22,7

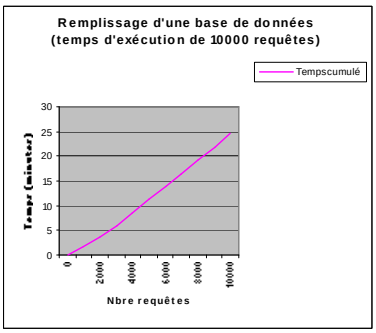
Index : ide,Vs,Ts



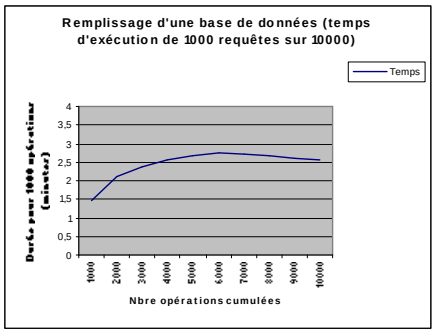


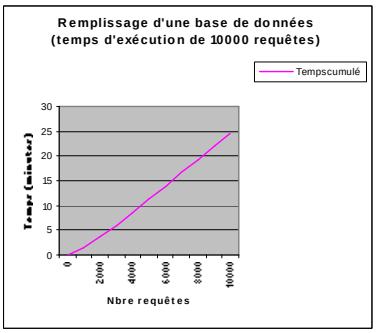
Index : ide,Vs,Ts - ide,Ve,Te - fk,Vs,Ve,Ts,Te





Index : ide,Vs,Ts - ide,Ve,Te - fk,Ve,Te



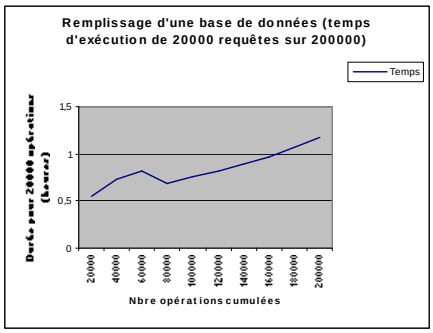


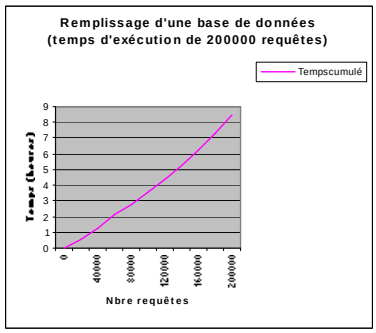
6.1.4. 200000 opérations

6.1.4.1. Base de données monotemporelles

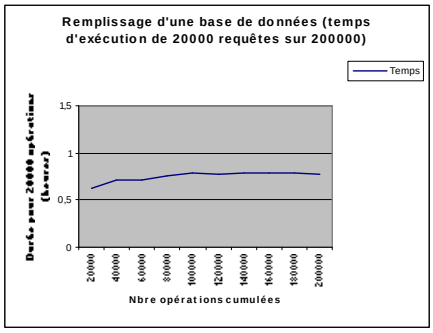
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_F_H_EMPLOYE	107867	3471	31,1
C_F_H_PROJET	66028	2000	33

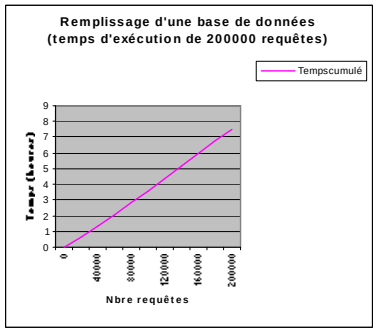
Index : ide,Vs



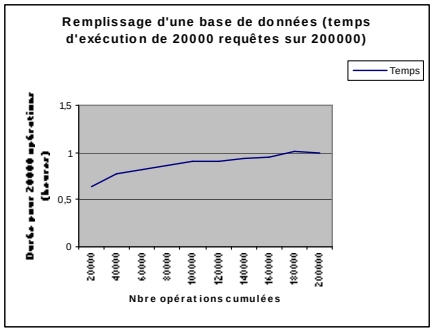


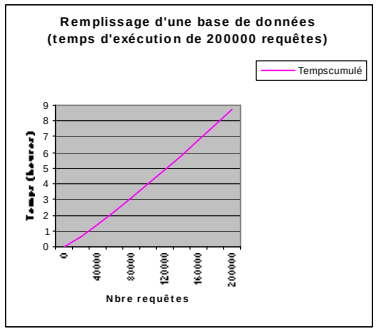
Index : ide,Vs - ide,Ve - fk,Vs,Ve





Index : ide,Vs - ide,Ve - fk,Ve

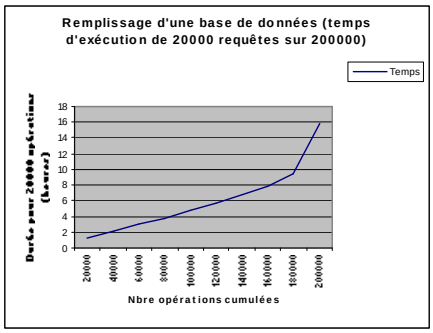


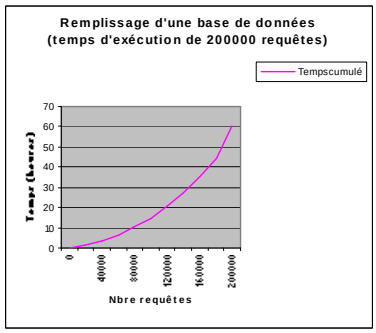


6.1.4.2. Base de données bitemporelles

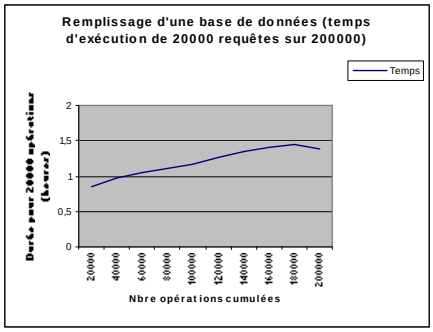
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_F_H_NV_EMPLOYE	218649	3471	63
C_F_H_NV_PROJET	158141	2000	79,1

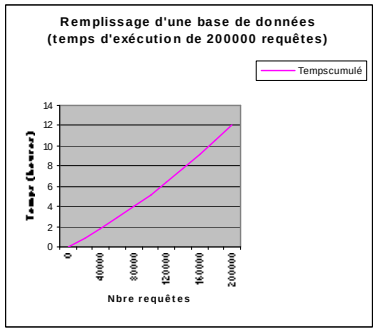
Index : ide,Vs,Ts



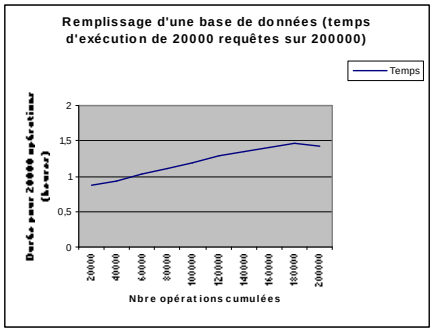


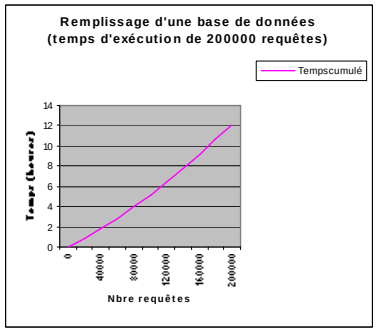
Index : ide,Vs,Ts - ide,Ve,Te - fk,Vs,Ve,Ts,Te





Index : ide,Vs,Ts - ide,Ve,Te - fk,Ve,Te





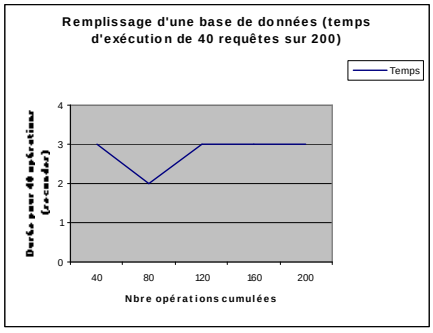
6.2. Configuration de données 2

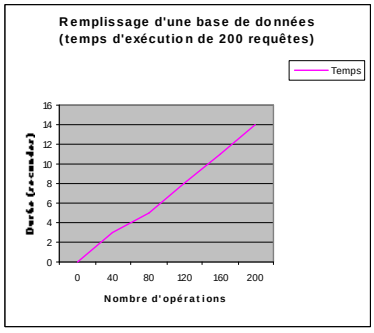
6.2.1. 200 opérations

6.2.1.1. Base de données monotemporelles

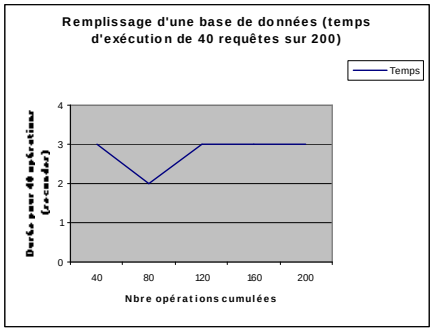
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
H_EMPLOYE	92	11	8,4
C_EMPLOYE	11	11	1
H_PROJET	59	10	5,9
C_PROJET	9	9	1

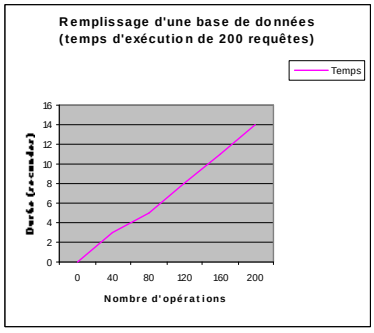
Index : ide,Vs



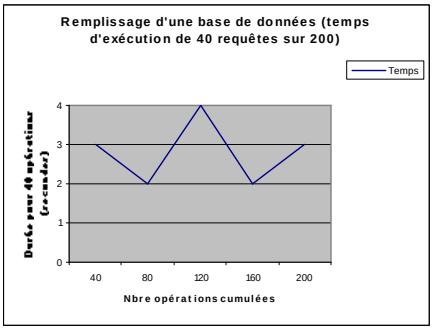


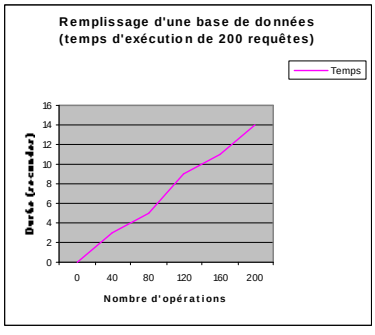
Index : ide,Vs - ide,Ve - fk,Vs,Ve





Index : ide,Vs - ide,Ve - fk,Ve

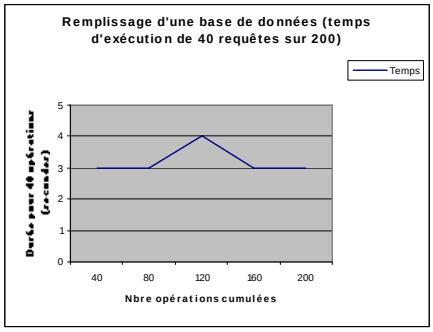


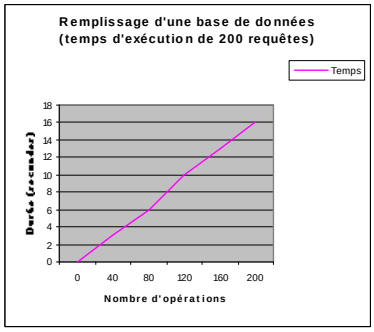


6.2.1.2. Base de données bitemporelles

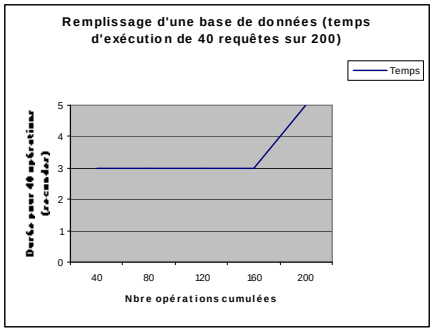
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
H_NV_EMPLOYE	195	11	17,7
C_EMPLOYE	11	11	1
H_NV_PROJET	154	10	15,4
C_PROJET	9	9	1

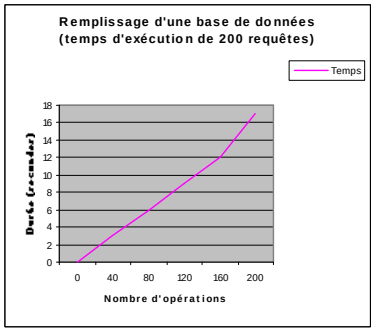
Index : ide,Vs,Ts



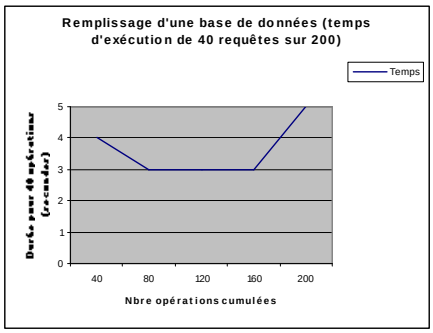


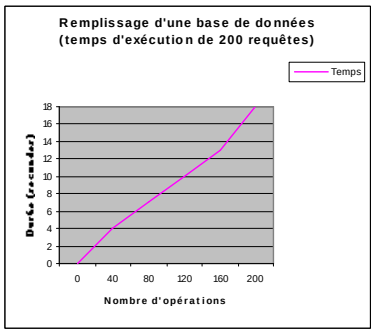
Index : ide,Vs,Ts - ide,Ve,Te - fk,Vs,Ve,Ts,Te





Index : ide,Vs,Ts - ide,Ve,Te - fk,Ve,Te



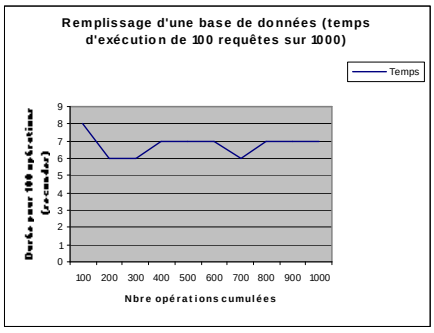


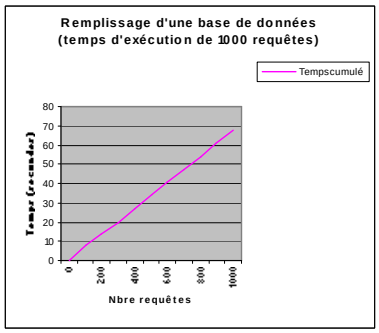
6.2.2. 1000 opérations

6.2.2.1. Base de données monotemporelles

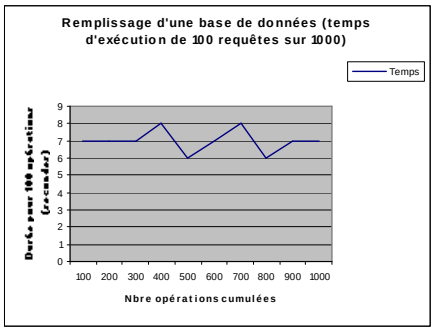
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
H_EMPLOYE	494	45	11
C_EMPLOYE	33	33	1
H_PROJET	298	38	7,8
C_PROJET	36	36	1

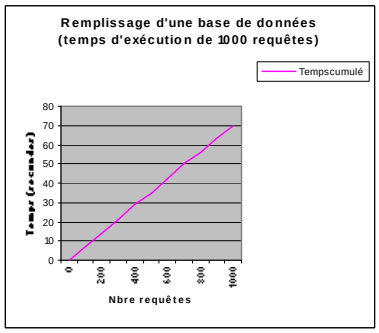
Index : ide,Vs



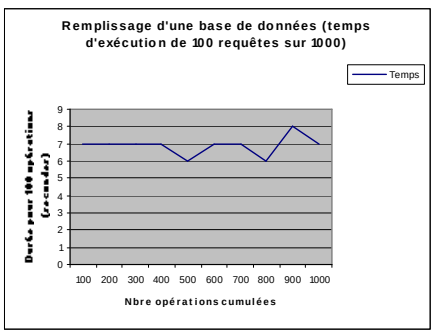


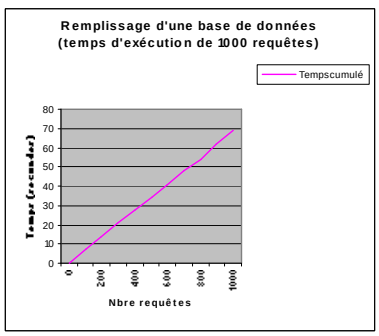
Index : ide,Vs - ide,Ve - fk,Vs,Ve





Index : ide,Vs - ide,Ve - fk,Ve

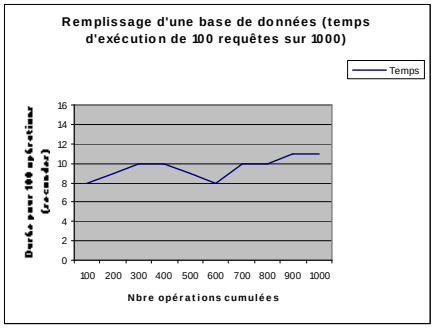


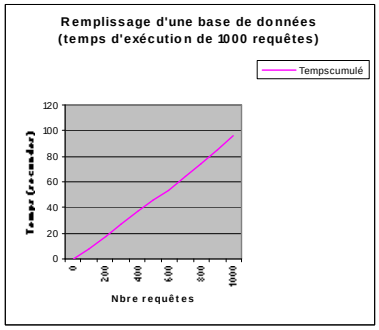


6.2.2.2. Base de données bitemporelles

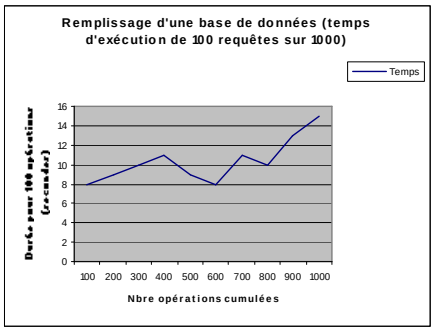
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
H_NV_EMPLOYE	1060	45	23,6
C_EMPLOYE	33	33	1
H_NV_PROJET	740	39	19
C_PROJET	36	36	1

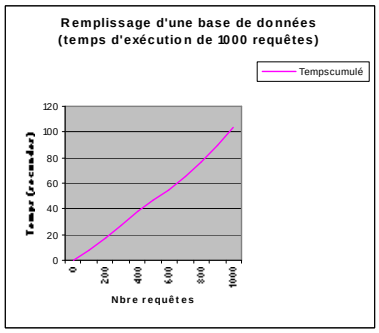
Index : ide,Vs,Ts



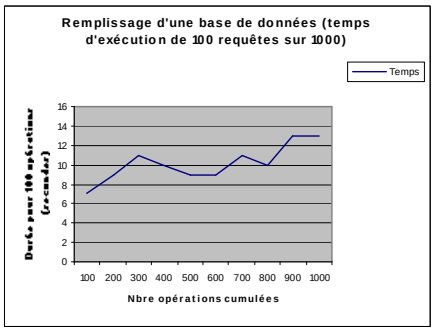


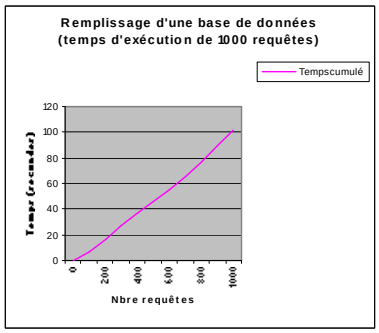
Index : ide,Vs,Ts - ide,Ve,Te - fk,Vs,Ve,Ts,Te





Index : ide,Vs,Ts - ide,Ve,Te - fk,Ve,Te



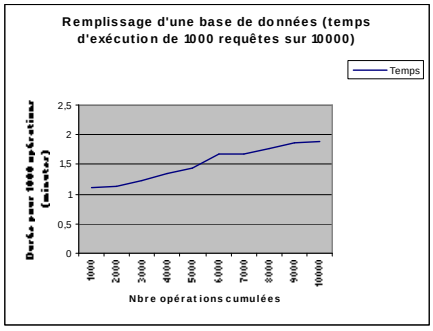


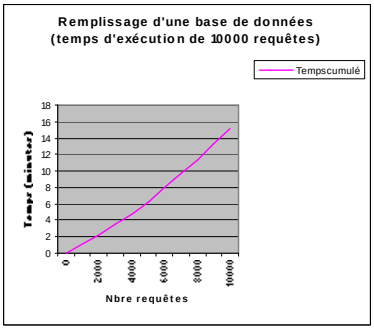
6.2.3. 10000 opérations

6.2.3.1. Base de données monotemporelles

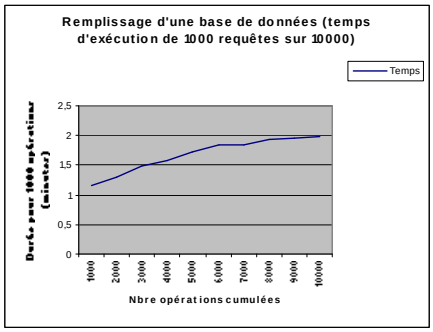
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
H_EMPLOYE	5074	435	11,7
C_EMPLOYE	359	359	1
H_PROJET	3070	310	9,9
C_PROJET	314	314	1

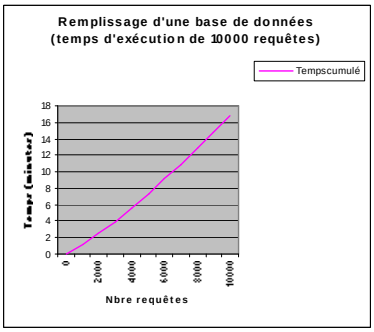
Index : ide,Vs



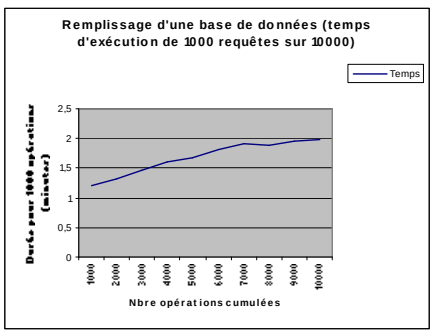


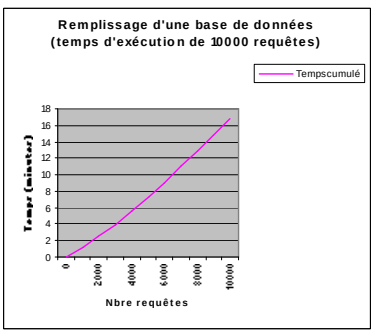
Index : ide,Vs - ide,Ve - fk,Vs,Ve





Index : ide,Vs - ide,Ve - fk,Ve

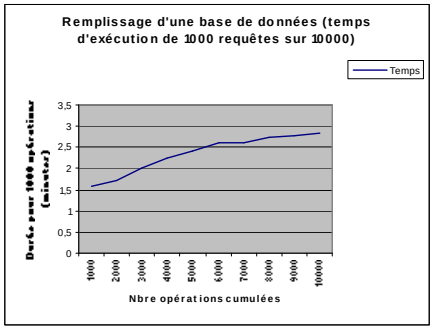


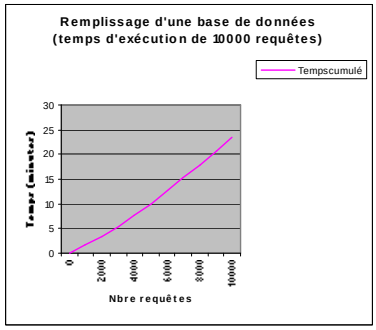


6.2.3.2. Base de données bitemporelles

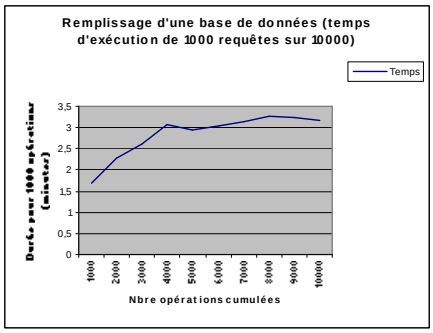
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
H_NV_EMPLOYE	10413	437	23,8
C_EMPLOYE	366	366	1
H_NV_PROJET	7372	316	23,3
C_PROJET	314	314	1

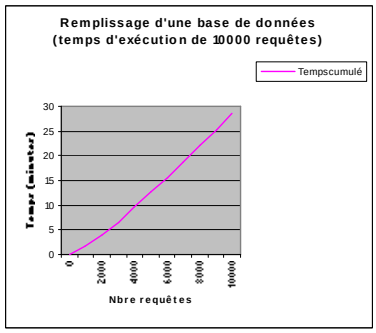
Index : ide,Vs,Ts



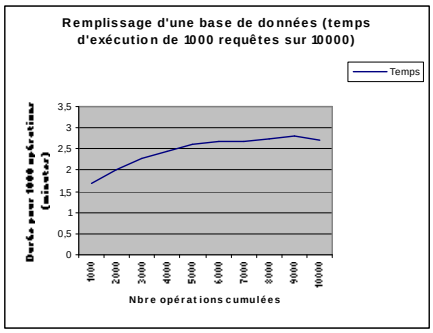


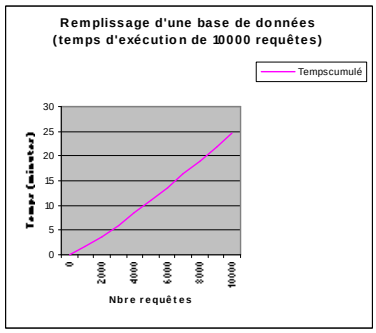
Index : ide,Vs,Ts - ide,Ve,Te - fk,Vs,Ve,Ts,Te





Index : ide,Vs,Ts - ide,Ve,Te - fk,Ve,Te



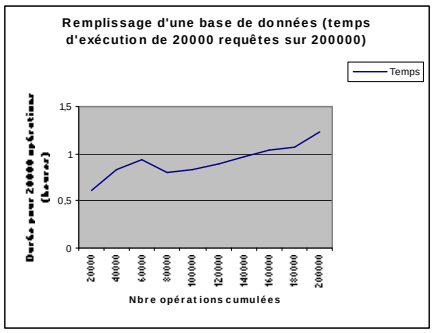


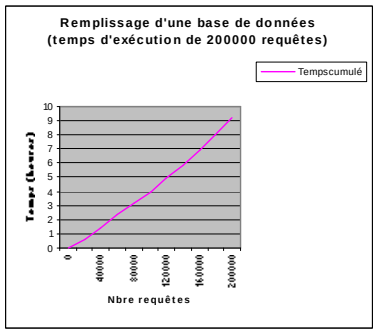
6.2.4. 200000 opérations

6.2.4.1. Base de données monotemporelles

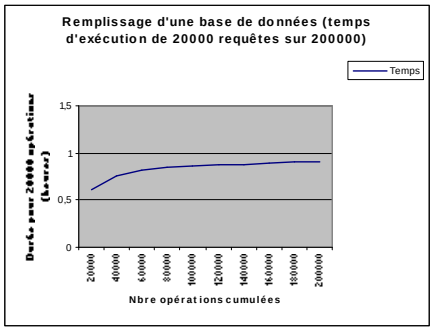
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
H_EMPLOYE	106822	3471	30,8
C_EMPLOYE	1045	1045	1
H_PROJET	64340	1978	32,5
C_PROJET	1688	1688	1

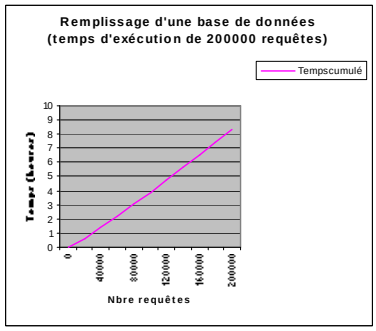
Index : ide,Vs



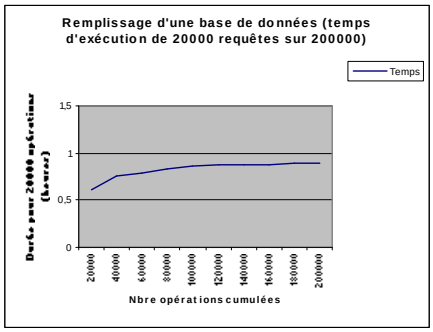


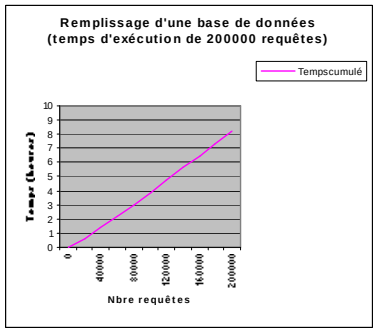
Index : ide,Vs - ide,Ve - fk,Vs,Ve





Index : ide,Vs - ide,Ve - fk,Ve

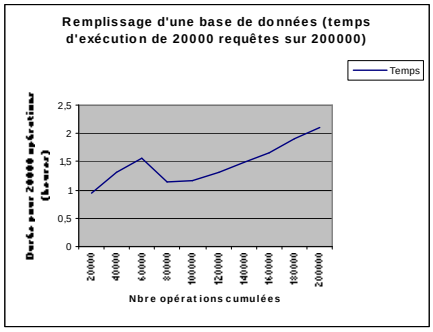


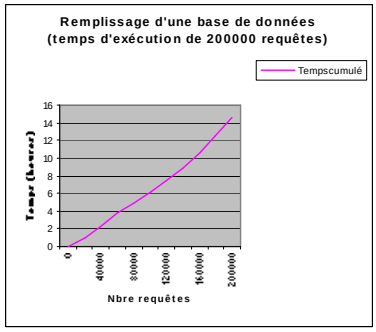


6.2.4.2. Base de données bitemporelles

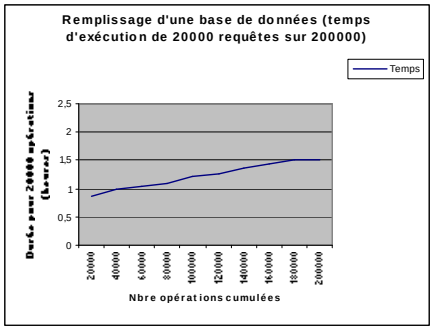
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
H_NV_EMPLOYE	217519	3466	62,8
C_EMPLOYE	1130	1130	1
H_NV_PROJET	156454	1990	78,6
C_PROJET	1687	1687	1

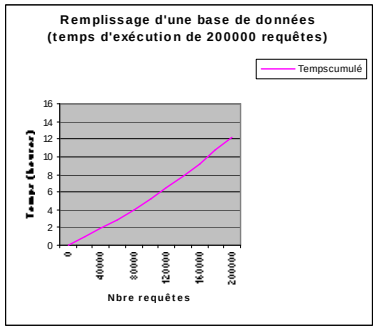
Index : ide,Vs,Ts



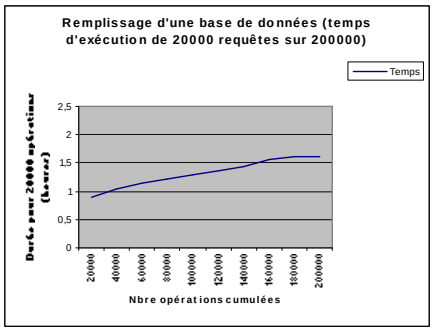


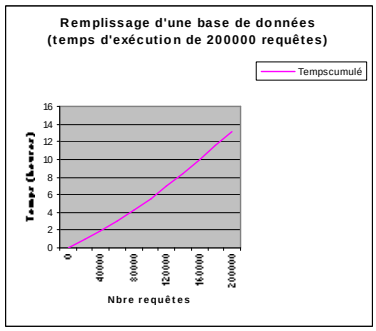
Index : ide,Vs,Ts - ide,Ve,Te - fk,Vs,Ve,Ts,Te





Index : ide,Vs,Ts - ide,Ve,Te - fk,Ve,Te





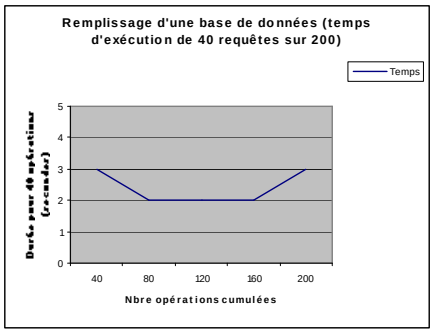
6.3. Configuration de données 3

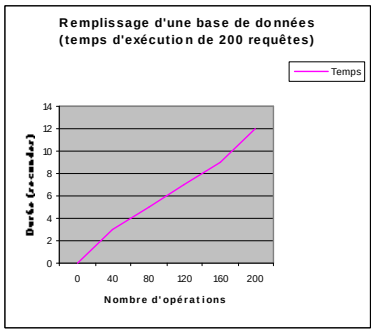
6.3.1. 200 opérations

6.3.1.1. Base de données monotemporelles

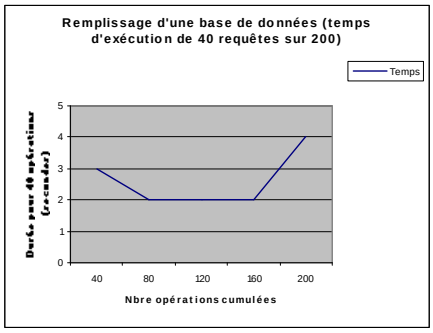
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_H_EMPLOYE	103	12	8,6
C_EMPLOYE	11	11	1
C_H_PROJET	68	10	6,8
C_PROJET	9	9	1

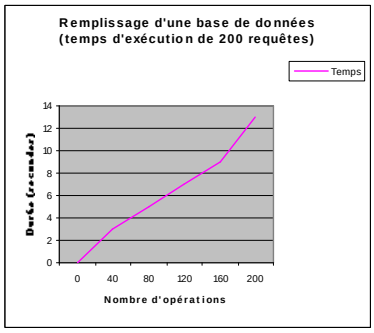
Index : ide,Vs



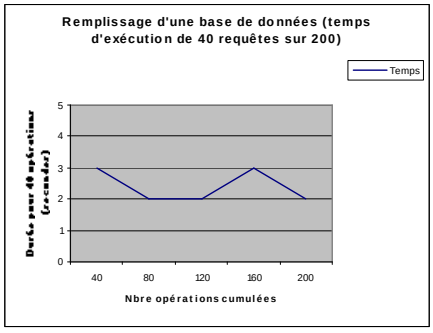


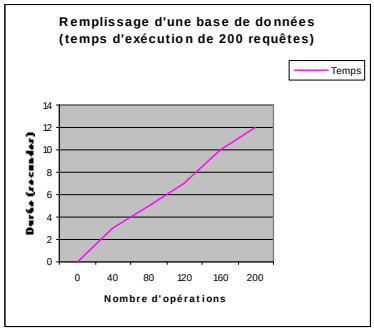
Index : ide,Vs - ide,Ve - fk,Vs,Ve





Index : ide,Vs - ide,Ve - fk,Ve

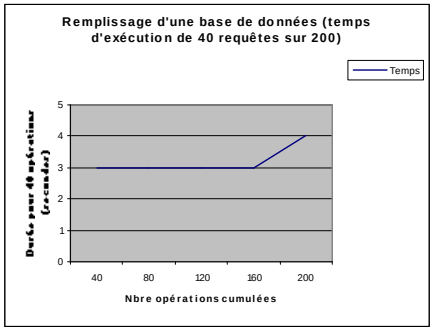


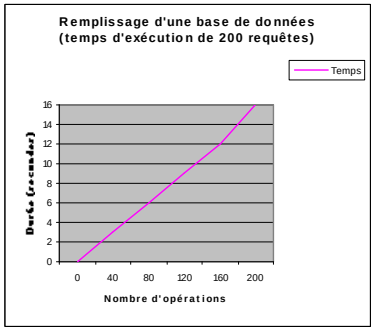


6.3.1.2. Base de données bitemporelles

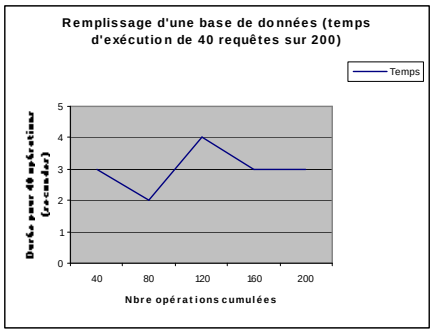
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_H_NV_EMPLOYE	206	12	17,2
C_EMPLOYE	11	11	1
C_H_NV_PROJET	163	10	16,3
C_PROJET	9	9	1

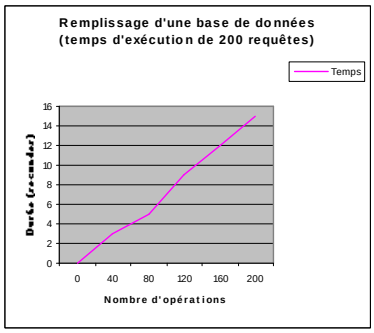
Index : ide,Vs,Ts



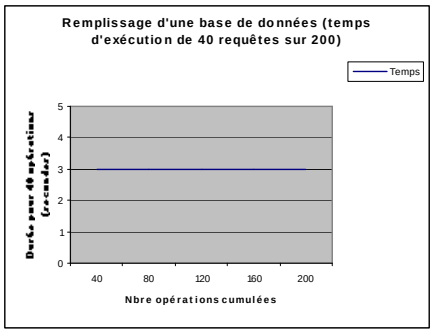


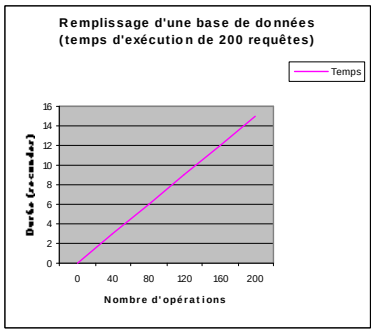
Index : ide,Vs,Ts - ide,Ve,Te - fk,Vs,Ve,Ts,Te





Index : ide,Vs,Ts - ide,Ve,Te - fk,Ve,Te



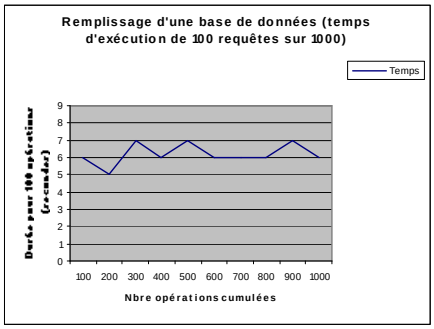


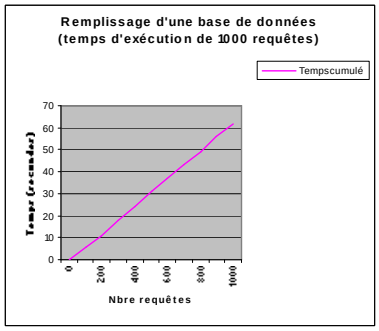
6.3.2. 1000 opérations

6.3.2.1. Base de données monotemporelles

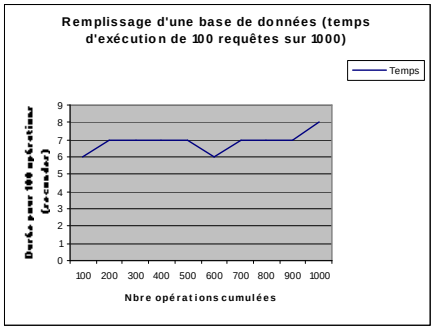
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_H_EMPLOYE	527	45	11,7
C_EMPLOYE	33	33	1
C_H_PROJET	334	41	8,1
C_PROJET	36	36	1

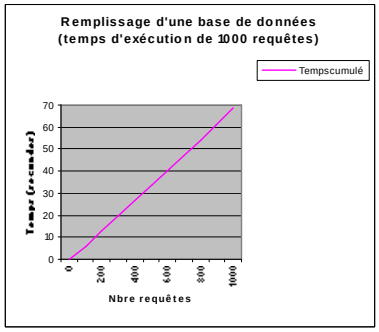
Index : ide,Vs



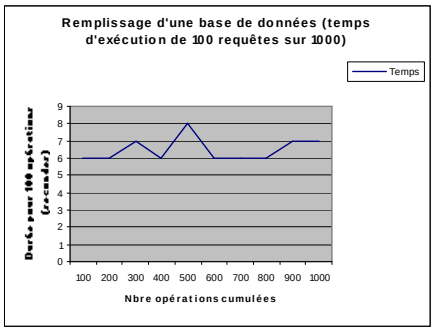


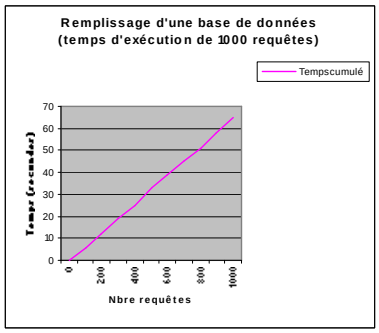
Index : ide,Vs - ide,Ve - fk,Vs,Ve





Index : ide,Vs - ide,Ve - fk,Ve

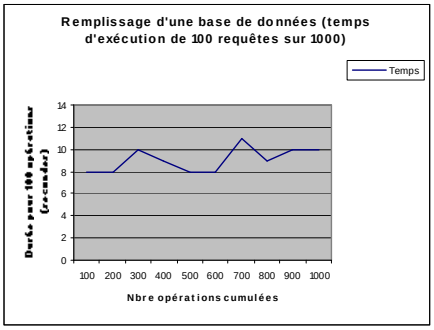


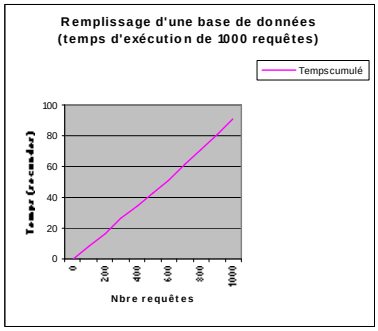


6.3.2.2. Base de données bitemporelles

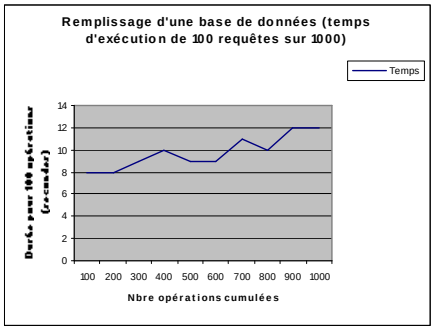
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_H_NV_EMPLOYE	1093	45	24,3
C_EMPLOYE	33	33	1
C_H_NV_PROJET	776	41	18,9
C_PROJET	36	36	1

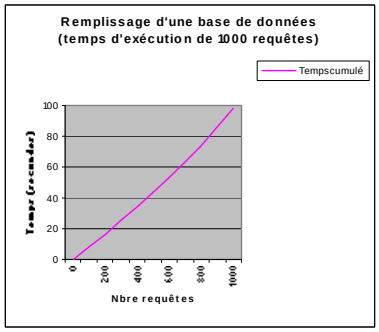
Index : ide,Vs,Ts



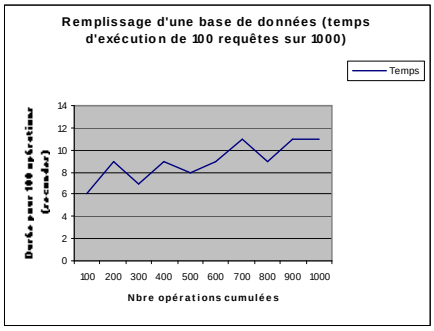


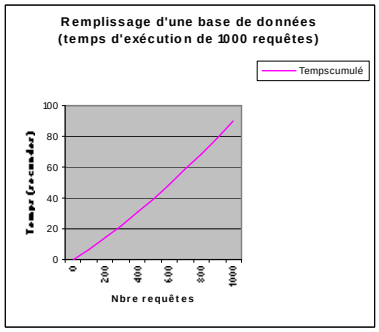
Index : ide,Vs,Ts - ide,Ve,Te - fk,Vs,Ve,Ts,Te





Index : ide,Vs,Ts - ide,Ve,Te - fk,Ve,Te



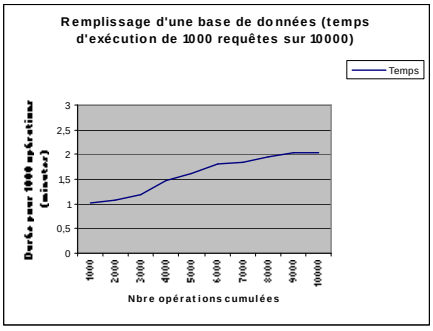


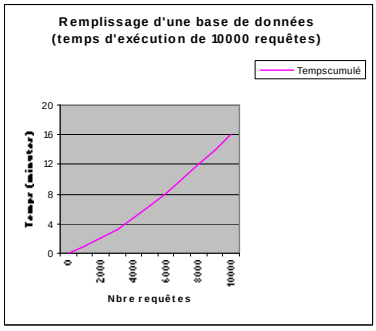
6.3.3. 10000 opérations

6.3.3.1. Base de données monotemporelles

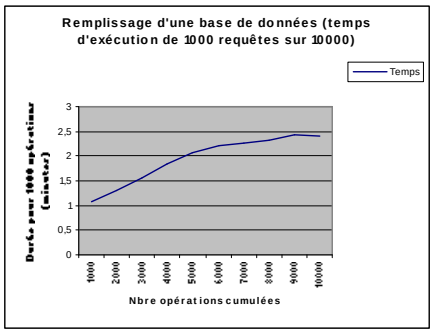
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_H_EMPLOYE	5433	482	11,3
C_EMPLOYE	359	359	1
C_H_PROJET	3384	339	10
C_PROJET	314	314	1

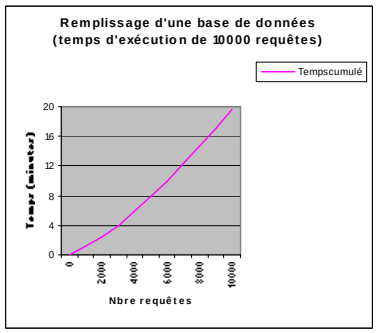
Index : ide,Vs



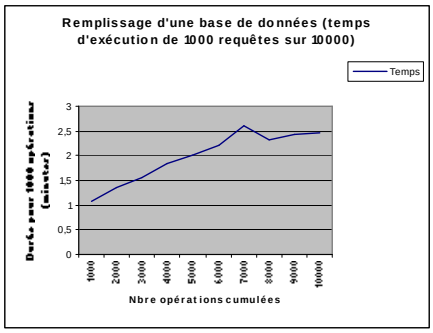


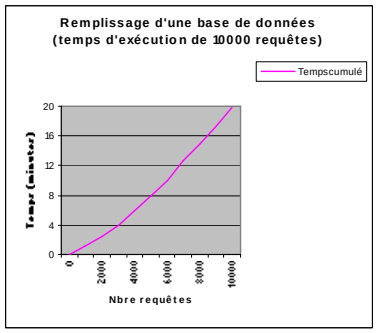
Index : ide,Vs - ide,Ve - fk,Vs,Ve





Index : ide,Vs - ide,Ve - fk,Ve

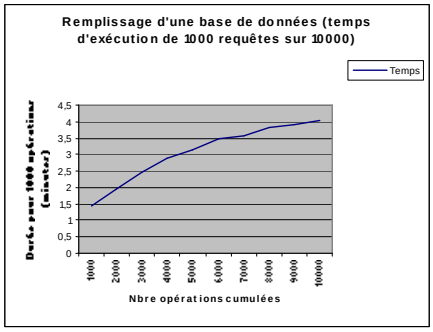


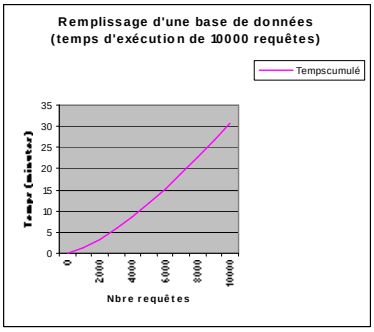


6.3.3.2. Base de données bitemporelles

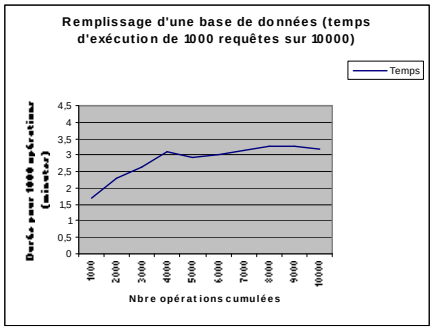
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_H_EMPLOYE	10779	482	22,4
C_EMPLOYE	366	366	1
C_H_PROJET	7686	339	22,7
C_PROJET	314	314	1

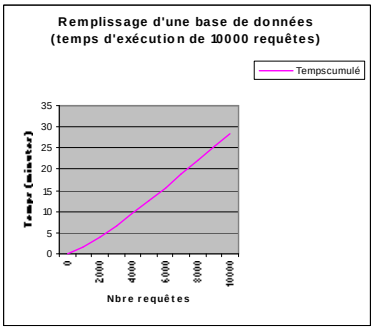
Index : ide,Vs,Ts



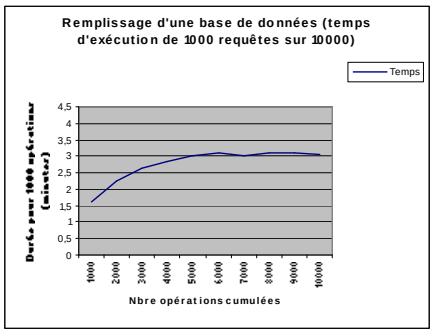


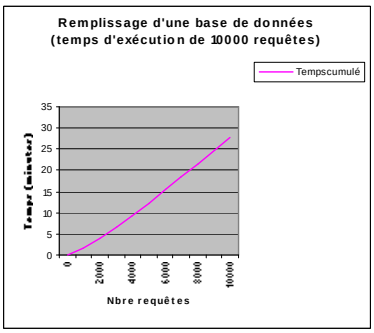
Index : ide,Vs,Ts - ide,Ve,Te - fk,Vs,Ve,Ts,Te





Index : ide,Vs,Ts - ide,Ve,Te - fk,Ve,Te



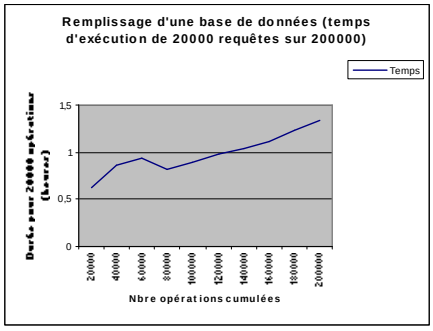


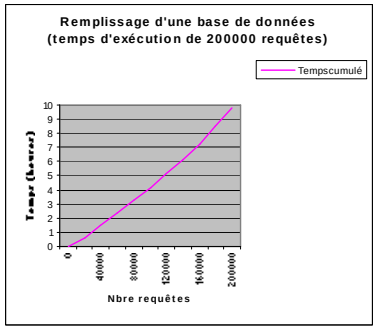
6.3.4. 200000 opérations

6.3.4.1. Base de données monotemporelles

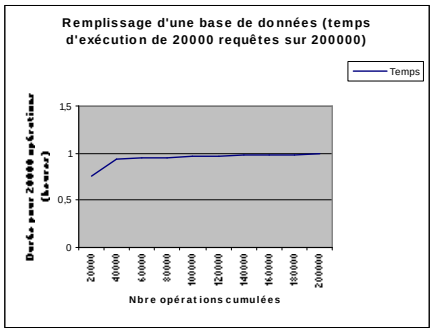
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_H_EMPLOYE	107867	3471	31,1
C_EMPLOYE	1045	1045	1
C_H_PROJET	66028	2000	33
C_PROJET	1688	1688	1

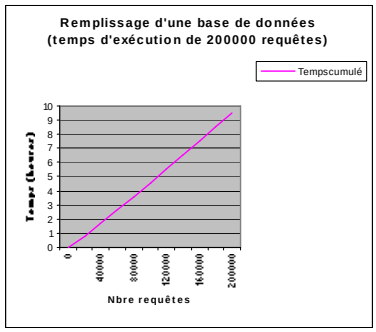
Index : ide,Vs



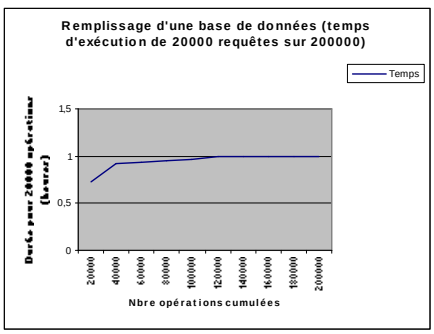


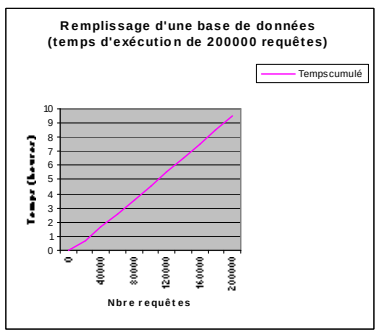
Index : ide,Vs - ide,Ve - fk,Vs,Ve





Index : ide,Vs - ide,Ve - fk,Ve

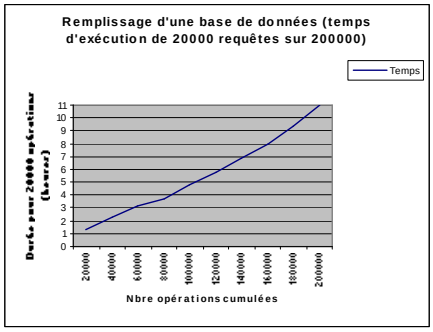


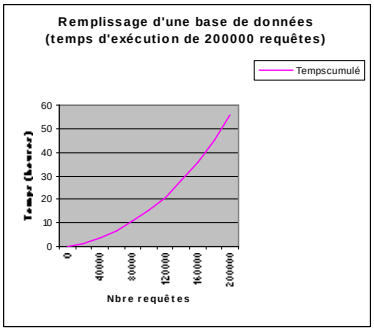


6.3.4.2. Base de données bitemporelles

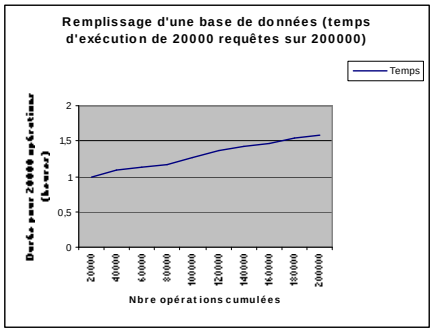
Table	Nbre lignes	Nbre objets	Nbre lignes/obj
C_H_EMPLOYE	218647	3471	63
C_EMPLOYE	1130	1130	1
C_H_PROJET	158141	2000	79,1
C_PROJET	1687	1687	1

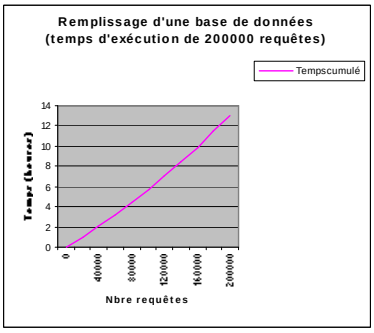
Index : ide,Vs,Ts



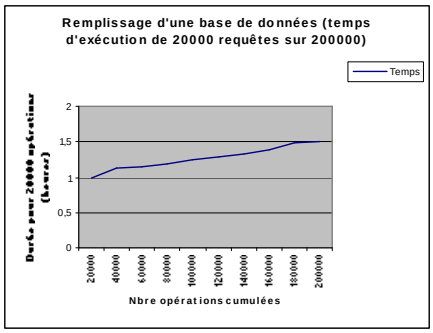


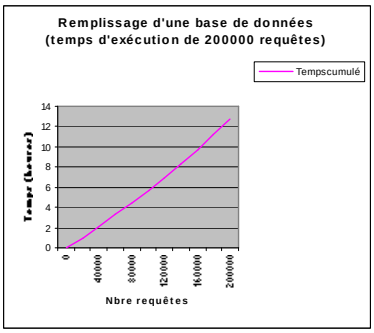
Index : ide,Vs,Ts - ide,Ve,Te - fk,Vs,Ve,Ts,Te





Index : ide,Vs,Ts - ide,Ve,Te - fk,Ve,Te





TimeStamp

Volume 3 - Documents techniques

Deuxième partie

Documentations techniques des outils

Les outils de génération de bases de données temporelles

DB-Main Manual Series

Projet TimeStamp

PATTERNS DE GÉNÉRATION DE TRIGGERS DESTINÉS À LA GESTION DE L'ASPECT TEMPOREL DES BASES DE DONNÉES

VIRGINIE DETIENNE

MARS 2001

The University of Namur - LIBD
Institut d'Informatique
Rue Grandgagnage, 21 B-5000 Namur
http://www.info.fundp.ac.be/libd

1 Introduction

La méthodologie de création de bases de données temporelles est analogue à la méthodologie classique (données non temporelles) et comporte les phases suivantes : analyse conceptuelle, conception logique, conception physique et génération de code.

Lors de la phase de génération du code – SQL dans notre cas - des triggers destinés à gérer automatiquement l'aspect temporel des données doivent être créés.

Nous allons décrire ici les patterns de génération de deux triggers complexes, à savoir un trigger gérant la mise-à-jour, et un trigger vérifiant l'intégrité référentielle d'une table référencée.

2 Description des notations

Les commentaires entre parenthèses décrivent dans quels cas les différents patterns doivent être générés.

La signification des différentes notations est la suivante :

Configuration 1 : Les colonnes temporelles et non temporelles sont groupées dans la même table.
Les états courants, passés, non valides et futurs sont groupés dans la même table.

Configuration 3 : Les colonnes temporelles et non temporelles sont groupées dans la même table.
Les états courants sont groupés dans une table et les états passés et non valides sont enregistrés dans une autre.

Configuration 7 : Les colonnes temporelles et non temporelles sont groupées dans la même table.
Les états courants, passés et non valides sont groupés dans une même table. Les états courants sont également recopiés dans une autre table.

TE hist : nom de la table contenant les historiques.
TE cour : nom de la table contenant les états courants.

Les préfixes **att** et **var** désignent respectivement le nom des colonnes et le nom de variables correspondant aux colonnes.
Si aucun suffixe, n'est indiqué, il s'agit de toutes les colonnes.

Les suffixes suivants signifient :
_ide: colonnes de l'identifiant d'entité.
_nontemp : colonnes non temporelles.
_temp : colonnes temporelles.
_tempnonnull : colonnes temporelles obligatoires.
_tempnull : colonnes temporelles facultatives.

3 Trigger de mise-à-jour

La mise-à-jour d'une entité concerne trois comportements différents : l'évolution, la correction et la mort.

3.1 Evolution

Lors de l'évolution d'une entité, il convient d'enregistrer la nouvelle valeur de la colonne temporelle ainsi que les dates du monde réel (table valid time ou bitemporelle) et d'enregistrement (table transaction time ou bitemporelle) auxquelles se sont produites la modification.

Le programmeur mentionne dans sa requête la nouvelle valeur à insérer ainsi que le moment auquel cette évolution a eu lieu dans le monde réel (Vstart).

La création de nouveaux états et la mise-à-jour du Transaction Time sont réalisés automatiquement par le trigger.

L'exemple suivant montre pour différentes requêtes la manière dont l'évolution est gérée automatiquement par le trigger. Les cases grisées en caractères gras indiquent les nouvelles données insérées dans la table.

Cet exemple porte sur la première configuration de données, mais les trois configurations décrites sont gérées par le trigger. Lors de l'exécution de ce trigger, d'autres triggers de vérification des contraintes référentielles sont déclenchés.

Table Transaction Time

```
UPDATE EMPLOYE
SET Adresse='Mons'
WHERE Matricule ='BRA' ;
```

C_H_EMPLOYE						
Matricule	**Nom**	**Adresse/t**	**Salaire/t**	**Service**	**Tstart**	**Tend**
BRA	Brassens	Liège	90000	INFO	30	**35**
BRA	**Brassens**	**Mons**	**90000**	**INFO**	**35**	**999**

Table Valid Time

```
UPDATE EMPLOYE
SET Adresse='Mons',
    Vstart=15
WHERE Matricule ='BRA'
AND Vend=999;
```

C_F_H_EMPLOYE						
Matricule	**Nom**	**Adresse/v**	**Salaire/v**	**Service**	**Vstart**	**Vend**
BRA	Brassens	Liège	90000	INFO	1	**15**
BRA	**Brassens**	**Mons**	**90000**	**INFO**	**15**	**999**

Table bitemporelle

```
UPDATE EMPLOYE
SET Adresse='Mons',
    Vstart=15
WHERE Matricule ='BRA'
AND Vend=999;
```

C_F_H_NV_EMPLOYE								
Matricule	**Nom**	**Adresse/b**	**Salaire/b**	**Service**	**Vstart**	**Vend**	**Tstart**	**Tend**
BRA	Brassens	Liège	90000	INFO	1	999	30	**35**
BRA	**Brassens**	**Liège**	**90000**	**INFO**	**1**	**15**	**35**	**999**
BRA	**Brassens**	**Mons**	**90000**	**INFO**	**15**	**999**	**35**	**999**

3.2 Correction

Dans les tables Valid Time et bitemporelles, des corrections portant sur les colonnes non temporelles, temporelles ou les bornes des états peuvent être réalisées.

Il convient de les gérer de manière à ce que la table reste normalisée après la correction (pas de recouvrements et pas de trous).

Correction d'une colonne non temporelle

Ce type de requête s'applique à l'état courant.

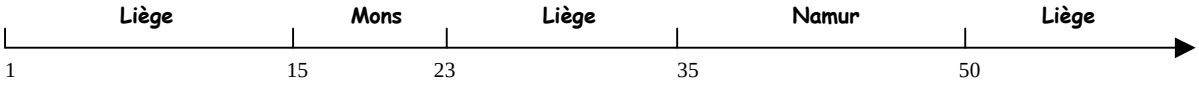
```
UPDATE EMPLOYE
SET Service='SECR'
WHERE Matricule ='BRA'
AND Vend=999;
```

Cette requête implique une modification de Service dans l'état courant.

Correction d'une colonne temporelle

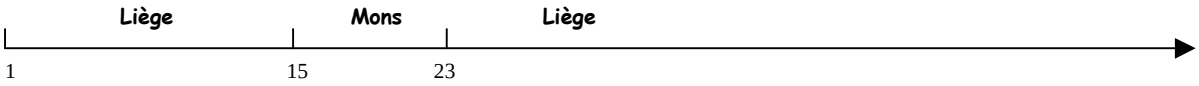
Pour corriger une colonne temporelle, il convient d'indiquer la nouvelle valeur à enregistrer ainsi que l'état sur lequel on travaille. Lors des corrections, on est parfois amené à faire des regroupements d'états (coalescing).

Supposons que pour l'employé BRA, on ait les adresses suivantes :



Si on corrige l'adresse de l'intervall [35 ,50[et qui'on indique que la valeur correcte est Liège, il convient de regrouper les trois états adjacents et de même valeur.

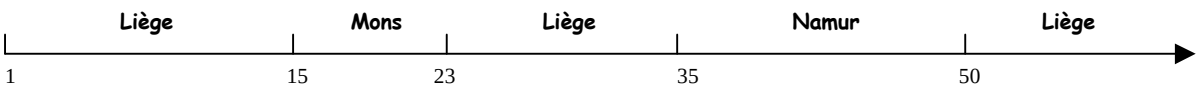
```
UPDATE EMPLOYE
SET Adresse='Liège'
WHERE Matricule ='BRA'
AND Vstart=35;
```



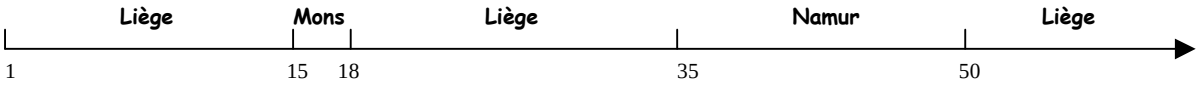
Correction d'une borne d'état

Les bornes de Valid Time peuvent être corrigées. Vstart et Vend peuvent donc augmenter ou diminuer, cependant un intervalle doit avoir un Vstart plus petit que le Vend.

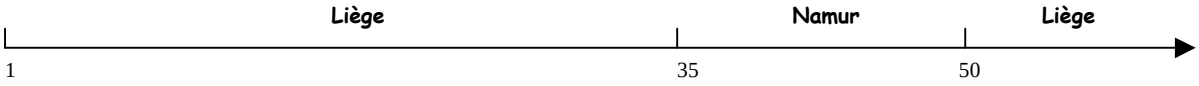
Les bornes de l'intervalle adjacent à l'intervalle modifié doivent être adaptées afin qu'il n'y ait pas de recouvrements ou de trous dans la table. Lorsqu'un état est complètement recouvert par un nouvel état, il est supprimé. Lorsque deux état sont nouvellement adjacents et qu'ils sont égaux, ils sont synthétisés.



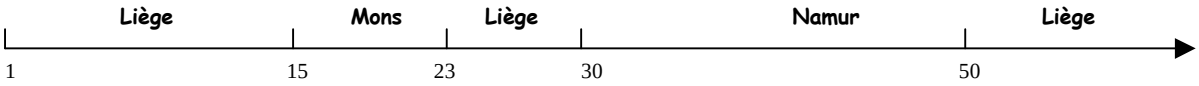
```
UPDATE EMPLOYE
SET Vstart=18
WHERE Matricule ='BRA'
AND Vstart=23;
```




```
UPDATE EMPLOYE
SET Vstart=10
WHERE Matricule ='BRA'
AND Vstart=23;
```




```
UPDATE EMPLOYE
SET Vend=30
WHERE Matricule ='BRA'
AND Vstart=23;
```



3.3 Mort

Pour les tables Valid Time et bitemporelles, la mort d'une entité correspond à la correction du Vend de l'état courant.

Dans les tables Transaction time, la correction des enregistrements n'est pas autorisée. La mort d'une entité doit donc être déclarée par un DELETE qui est géré par le trigger de mise-à-jour.

3.4 Patterns

Trigger de mise à jour

procedure GestionMaj()

{
(Appliquer trigger pour tables temporelles et historiques)
T_HistoryType = valid, transaction ou bitemporal
Type historique

/*Liste des noms (string) des attributs du type d'entités*/
/*Liste des variables correspondants aux attr du TE (3 prem. lettres)*/
/*Liste des noms (string) des attributs du type d'entités sans les dates*/
/*Liste des variables correspondants aux attr du TE (3 prem. lettres) sans les dates*/
/*Liste des noms (string) des attributs composant l'idt d'entité*/
/*Liste des variables correspondants aux attributs composant l'idt d'entité*/
/*Liste des noms des attributs non temporels*/
/*Liste des variables correspondants aux attributs non temporels*/
/*Liste des noms des attributs temporels*/
/*Liste des variables correspondants aux attributs temporels*/

/*Entête*/

/*Déclaration du trigger*/

/*Déclaration des variables*/

/*Début du trigger*/

/*Stockage de l'état sur lequel on travaille*/
(configurations 1, 7)
SELECT att
INTO var
FROM TE hist
WHERE att_ide=:o.att_ide
AND att_ide=:o.att_ide
AND Tend=999 *(transaction - bitemporal)*
AND Vend=:o.Vend *(valid - bitemporal)*

(configuration 3)
SELECT count(*) INTO ct1
FROM TE cour
WHERE att_ide=:o.att_ide
AND att_ide=:o.att_ide
AND Vstart= :o.Vstart *(valid – bitemporal)*
AND Tstart= :o.Tstart *(transaction)*

IF ct1<>0 (a)
THEN
SELECT att, Vstart, 999 (valid – bitemporal)
Tstart (transaction)
INTO var, Vs,Ve (valid – bitemporal)
Ts (transaction)
FROM TE cour
WHERE att_ide=:o.att_ide
AND att_ide=:o.att_ide

ELSE (a)

****

RAISE interv_non_conforme (transaction)

****

(valid – bitemporal)

SELECT att, Vstart, Vend (valid – bitemporal)

INTO var, Vs,Ve (valid – bitemporal)

FROM TE hist

WHERE att_ide=:o.att_ide

AND att_ide=:o.att_ide

AND Vstart= :o.Vstart ;

****

END IF ; (a)

/*Cas d'une modification d'un attribut non temporel*/

IF (var_ide<>:n.att_ide) OR (var_nontemp<>:n.att_nontemp) (b)

THEN (b)

(configuration 1 et7)

UPDATE TE hist

SET att_ide=:n.att_ide, (déclenche trigger modification attribut stable)

att_nontemp=:n.att_nontemp

WHERE att_ide=:o.att_ide

AND Vend=999 (valid - bitemporal)

AND Tend=999; (transaction - bitemporal)

(configuration 3 et 7)

UPDATE TE cour

SET att_ide=:n.att_ide, (déclenche trigger modification attribut stable)

att_nontemp=:n.att_nontemp

WHERE att_ide=:o.att_ide;

END IF ; (b)

/*Cas d'une modification d'un attribut temporel ou d'une date*/

(quand att temporels OU TE valid-bitemporal
car si TE transaction sans att, ni évolution, ni correction ne sont possibles)

IF (var_nontemp=:n.att_nontemp) AND (var_nontemp=:n.att_nontemp) (c)

(quand att non temp)

AND ((var_temp<>:n.att_temp) OR (var_temp<>:n.att_temp)

OR (:o.att_tempnull is null AND :n.att_tempnull is not null)

(quand att temp)

OR (Vs<>:n.Vstart) OR (Ve<>:n.Vend))

(quand valid – bitemporal)

THEN (c)

/*Critère montrant si c'est une évolution ou une correction*/

(quand att temp ET TE valid – bitemporal
car quand att temp et transaction, pas de correction, seulement des évolutions donc pas besoin de faire le test
car quand valid-bitemporal et pas att temp, pas d'évolution, seulement des corrections donc pas besoin de faire le test)

(configuration 1, 3, 7)

SELECT count(*) INTO ct2

FROM TE hist

WHERE att_ide=:o.att_ide

AND att_ide=:o.att_ide

AND Vstart=:n.Vstart (valid - bitemporal)

AND Tend=999 (bitemporal)

(configuration 3)

SELECT count(*) INTO ct3
FROM TE cour
WHERE att_ide=:o.att_ide
AND att_ide=:o.att_ide
AND Vstart=:n.Vstart (valid - bitemporal)
AND Tend=999 (bitemporal)

/*Evolution*/

****

(quand att temp ET valid-bitemporal
car quand pas d'att temp, pas d'évolution possible
car quand transaction, seulement évolution, donc pas besoin de tester si c'est une évolution ou une correction)

(configuration 1 et 7)

IF (ct2=0) AND ((var_temp<>:n.att_temp) OR (var_temp<>:n.att_temp) (d)
OR (:o.att_tempnull is null AND :n.att_tempnull is not null)) (valid - bitemporal)
THEN

****

(quand att temp ET transaction
car évolution possible mais pas de test Evol-correct préalable)

(configuration 1 et 7)

IF (var_temp<>:n.att_temp) OR (var_temp<>:n.att_temp) (d)
OR (:o.att_tempnull is null AND :n.att_tempnull is not null) (transaction)
THEN

****

IF Vs>=:n.Vstart (valid - bitemporal) (e)
THEN (e)
RAISE interv_non_conforme;
ELSE (e)

****

INSERT INTO TE hist (bitemporal)
VALUES(var,var,...,Vs,:n.Vstart,curr_date.NEXTVAL,999);

UPDATE TE hist (bitemporal)
SET Tend=curr_date.CURRVAL
WHERE att_ide=:n.att_ide
AND Vend=999
AND Tend=999;

INSERT INTO TE hist (bitemporal)
VALUES(:n.att,:n.att,...,:n.Vstart,Ve,curr_date.NEXTVAL,999);

****

UPDATE TE hist (valid)
SET Vend=:n.Vstart
WHERE att_ide=:n.att_ide
AND Vend=999 ;

INSERT INTO TE hist (valid)
VALUES(:n.att,:n.att,...,:n.Vstart,Ve);

****

(configuration7)

UPDATE TE cour (valid - bitemporal)
SET att_temp=:n.att_temp
, att_temp=:n.att_temp
WHERE att_ide=:n.att_ide
AND att_ide=:n.att_ide;

****

END IF; (valid - bitemporal) (e)

****

UPDATE TE hist (transaction)
SET Tend=:curr_date.NEXTVAL
WHERE att_ide=:n.att_ide
AND Tend=999,

INSERT INTO TE hist (transaction)
VALUES(:n.att,:n.att,...,curr_date.NEXTVAL,999);

****

(configuration7)
UPDATE TE cour (transaction)
SET att_temp=:n.att_temp
, att_temp=:n.att_temp
WHERE att_ide=:n.att_ide
AND att_ide=:n.att_ide;

****

(quand att temp ET valid-bitemporal
car quand pas d'att temp, pas d'évolution possible
car quand transaction, seulement évolution, donc pas besoin de tester si c'est une évolution ou une correction)

(configuration 3)
IF (ct2=0) AND (ct3=0) AND ((var_temp<>:n.att_temp) OR (var_temp<>:n.att_temp) (d)
OR (:o.att_tempnull is null AND :n.att_tempnull is not null)) (valid - bitemporal)
THEN

****

(quand att temp ET transaction
car évolution possible mais pas de test Evol-correct préalable)

(configuration 3)
IF (var_temp<>:n.att_temp) OR (var_temp<>:n.att_temp) (d)
OR (:o.att_tempnull is null AND :n.att_tempnull is not null) (transaction)
THEN

****

IF Vs>=:n.Vstart (valid - bitemporal) (e)
THEN (e)
RAISE interv_non_conforme;
ELSE (e)

****

INSERT INTO TE hist (bitemporal)
VALUES (var,var,...,Vs,Ve,Ts,curr_date.NEXTVAL);

INSERT INTO TE hist (bitemporal)
VALUES (var,var,...,Vs,:n.Vstart,curr_date.CURRVAL,999);

UPDATE TE cour (bitemporal)
SET att_temp=:n.att_temp
, att_temp=:n.att_temp
, att_temp=:n.att_temp
, Vstart=:n.Vstart
, Tstart=curr_date.CURRVAL
WHERE att_ide:=:n.att_ide;

****

INSERT INTO TE hist (valid)
VALUES (var,var,...,Vs,:n.Vstart);

UPDATE TE cour (valid)
SET att_temp=:n.att_temp
, att_temp=:n.att_temp
, att_temp=:n.att_temp

```
        , Vstart=:n.Vstart
      WHERE att_ide:=:n.att_ide;
*****************************

  END IF;                (valid - bitemporal)                    (e)
*****************************
  INSERT INTO TE hist          (transaction)
  VALUES (var,var,...,Ts,curr_date.NEXTVAL);

  UPDATE TE cour           (transaction)
  SET att_temp=:n.att_temp
      , att_temp=:n.att_temp
      , att_temp=:n.att_temp
      , Tstart=curr_date.CURRVAL
  WHERE att_ide:=:n.att_ide;
*****************************

 ELSE                                                          (d)

/*Correction*/
  *****************************
  (configurations 1, 7)
  IF (ct2<>0) OR                                               (f)
                  ((ct2=0) AND (Vs<> :n.Vstart)
                  AND (var_tempnonnull= :n.att_tempnonnull)
                  AND (var_tempnonnull= :n.att_tempnonnull)
                  AND ((var_tempnull=:n.att_tempnull)
                       OR (var_tempnull is null AND :n.att_tempnull is  null))  )
                                                                        (valid – bitemporal)
     (quand att temp ET TE valid – bitemporal
          car quand att temp et non valid – bitemporal (transaction), pas de correction, seulement des évolutions
          car quand valid-bitemporal et pas att temp, pas de correction d'attribut temporel)

  THEN                                                         (f)
  *****************************

  (configurations 1, 7)
  IF (Vs<> :n.Vstart) OR (Ve<> :n.Vend)                        (valid – bitemporal)
                                                               (f)
     (quand pas att temp ET TE valid – bitemporal
          car comme pas d'att temp, seule correction possible)
  THEN                                                         (f)
  *****************************
     (quand attributs temporels)
     /*Correction attribut temporel*/
     IF (var_temp<> :n.att_temp) OR (var_temp<> :n.att_temp)          (g)
     OR (:o.att_tempnull is null AND :n.att_tempnull is not null)
     THEN                                                      (g)

        /*Coalescing*/
        SELECT count(*) INTO ct4
        FROM TE hist
        WHERE att= :n.att
        AND att= :n.att
        AND Tend=999            (bitemporal)
        AND (Vend= :n.Vstart        (valid – bitemporal)
            OR Vstart= :n.Vend) ;

        IF ct4=0                                               (h)
        THEN                                                   (h)
```

```
**************************
/*Maj de ligne à corriger – Tend=now*/                    (bitemporal)
UPDATE TE hist
SET Tend=curr_date.NEXTVAL
WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND (Vstart= :n.Vstart
    OR Vend= :n.Vend)
AND Tend=999 ;

/*Création de la ligne reprenant les informations corrigées*/   (bitemporal)
INSERT INTO TE hist
        VALUES ( :n.att, :n.att, :n.Vstart, :n.Vend,curr_date.CURRVAL,999) ;
        **************************
/*Màj de la ligne à corriger*/                            (valid)
UPDATE TE hist
SET att_temp= :n.att_temp
    , att_temp= :n.att_temp
WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Vstart= :n.Vstart ;

**************************


  ELSE                                                    (h)
        ***************************
/*Réalisation du coalescing*/                       (valid – bitemporal)
SELECT min(Vstart), max(Vend)
INTO MinVs, MaxVe
FROM TE hist
WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Tend=999                                    (bitemporal)
AND ((att_temp= :n.att_temp
        AND att_temp= :n.att_temp
        AND att_nontemp= :n.att_nontemp
        AND att_nontemp= :n.att_nontemp
        AND (Vend= :n.Vstart
            OR Vstart= :n.Vend))
        OR (   Vstart= :n.Vstart
            AND Vend= :n.Vend)  ) ;
**************************

UPDATE TE hist                                          (bitemporal)
SET Tend=curr_date.NEXTVAL
WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Tend=999
AND Vstart= :n.Vstart
AND Vend= :n.Vend ;

/*Maj des lignes à corriger – Tend=now*/                (bitemporal)
UPDATE TE hist
SET Tend=curr_date.CURRVAL
WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Tend=999
AND att_temp= :n.att_temp
AND att_temp= :n.att_temp
```

```
        AND att_nontemp= :n.att_nontemp
        AND att_nontemp= :n.att_nontemp
        AND (Vend= :n.Vstart
            OR     Vstart= :n.Vend)
        ;

        INSERT INTO TE hist                                    (bitemporal)
        VALUES( :n.att, :n.att,MinVs,MaxVe,curr.date.CURRVAL,999) ;
        ****************************

        DELETE FROM TE hist                                    (valid)
        WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vstart= :n.Vstart
        AND Vend= :n.Vend ;


              /*Suppression des lignes à corriger*/                    (valid)
        DELETE FROM TE hist
        WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND att_temp= :n.att_temp
        AND att_temp= :n.att_temp
        AND att_nontemp= :n.att_nontemp
        AND att_nontemp= :n.att_nontemp
        AND (Vend= :n.Vstart
            OR Vstart= :n.Vend) ;


        INSERT INTO TE hist                                    (valid)
        VALUES ( :n.att, :n.att,…,MinVs,MaxVe) ;

        ****************************

    END IF ;                                   (h)
```

(configuration 3)

```
IF (ct2<>0) OR (ct3<>0) OR                                          (f)
                ((ct2=0) AND (ct3=0) AND (Vs<> :n.Vstart)
                AND (var_tempnonnull= :n.att_tempnonnull)
                AND (var_tempnonnull= :n.att_tempnonnull)
                AND ((var_tempnull=:n.att_tempnull)
                     OR (var_tempnull is null AND :n.att_tempnull is  null))    )
                                                                    (valid – bitemporal)
```

(quand att temp ET TE valid – bitemporal
car quand att temp et non valid – bitemporal (transaction), pas de correction, seulement des évolutions
car quand valid-bitemporal et pas att temp, pas de correction d'attribut temporel)

```
THEN                                                                (f)
   ****************************
```

(configuration 3)

```
IF (Vs<> :n.Vstart) OR (Ve<> :n.Vend)                    (valid – bitemporal)
                                                                    (f)
```

(quand pas att temp ET TE valid – bitemporal
car comme pas d'att temp, seule correction possible)

```
THEN                                                                (f)
   ****************************
```

/*Correction attribut temporel d'un état passé*/

```
 IF (ct2<>0) AND (  (var_temp<> :n.att_temp) OR (var_temp<> :n.att_temp)
          OR (:o.att_tempnull is null AND :n.att_tempnull is not null) )       (g)
(quand att temp ET TE valid – bitemporal
   car quand att temp et transaction, pas de correction, seulement des évolutions
   car quand valid-bitemporal et pas att temp, pas de correction d'attribut temporel)
 THEN                                                                    (g)
/*Test de coalescing*/
      /*Regarde s'il existe des états égaux adjacents antérieurs ou postérieurs dans
        H_NV*/
      SELECT count(*) INTO ct4     (valid – bitemporal)
FROM TE hist
WHERE att= :n.att
AND att= :n.att
AND Tend=999              (bitemporal)
AND (Vend= :n.Vstart          (valid – bitemporal)
   OR Vstart= :n.Vend) ;

      /*Regarde s'il existe un état égal adjacent postérieur dans C_*/
      SELECT count(*) INTO ct5                    (valid – bitemporal)
FROM TE cour
WHERE att= :n.att
AND att= :n.att
AND Vstart= :n.Vend;

IF ct4=0 AND ct5=0                                    (h)
THEN                                          (h)
   ****************************
   /*Pas de coalescing*/
   /*Maj de ligne à corriger – Tend=now*/                  (bitemporal)
   UPDATE TE hist
   SET Tend=curr_date.NEXTVAL
   WHERE att_ide= :n.att_ide
   AND att_ide= :n.att_ide
   AND (Vstart= :n.Vstart
      OR Vend= :n.Vend)
   AND Tend=999 ;

   /*Création de la ligne reprenant les informations corrigées*/   (bitemporal)
   INSERT INTO TE hist
         VALUES ( :n.att, :n.att, :n.Vstart, :n.Vend,curr_date.CURRVAL,999) ;
         ****************************

   /*Màj de la ligne à corriger*/                        (valid)
   UPDATE TE hist
   SET att_temp= :n.att_temp
      , att_temp= :n.att_temp
   WHERE att_ide= :n.att_ide
   AND att_ide= :n.att_ide
   AND Vstart= Vs;

   ****************************

ELSE                                          (h)
   /*Réalisation du coalescing*/

   /*Enregistrement des dates extrêmes*/
   SELECT min(Vstart), max(Vend)                       (valid - bitemporal)
   INTO MinVs, MaxVe
   FROM TE hist
   WHERE att_ide= :n.att_ide
```

```
AND att_ide= :n.att_ide
AND Tend=999                                    (bitemporal)
AND ((att_temp= :n.att_temp
        AND att_temp= :n.att_temp
        AND att_nontemp= :n.att_nontemp
        AND att_nontemp= :n.att_nontemp
        AND (Vend= :n.Vstart
            OR Vstart= :n.Vend))
        OR (   Vstart= :n.Vstart
            AND Vend= :n.Vend)  ) ;
****************************


/*Clôture de l'état à corriger*/
UPDATE TE hist                                  (bitemporal)
SET Tend=curr_date.NEXTVAL
WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Tend=999
AND Vstart= :n.Vstart
AND Vend= :n.Vend ;

/*Maj des lignes à corriger – Tend=now*/          (bitemporal)
UPDATE TE hist
SET Tend=curr_date.CURRVAL
WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Tend=999
AND att_temp= :n.att_temp
AND att_temp= :n.att_temp
AND att_nontemp= :n.att_nontemp
AND att_nontemp= :n.att_nontemp
AND (Vend= :n.Vstart
    OR     Vstart= :n.Vend)
;

/*Cas où l 'état courant est impliqué dans le coalescing*/
IF ct5<>0                                       (i)
THEN                                            (i)
    SELECT att, att, Vstart, Tstart                     (bitemporal)
    INTO var2, var2, Vs2, Ts2
    FROM TE cour
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide ;

    INSERT INTO TE hist                         (bitemporal)
    VALUES( var2, var2, Vs2, 999, Ts2, curr_date.CURRVAL) ;

    UPDATE TE cour                              (bitemporal)
    SET att_temp= :n.att_temp
        , att_temp= :n.att_temp
        , Vstart=MinVs
        , Tstart=curr_date.CURRVAL
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide ;



/*Cas où l'état courant n'est pas impliqué dans le coalescing*/
ELSE                                            (i)
```

```
        INSERT INTO TE hist                                  (bitemporal)
        VALUES( :n.att, :n.att,MinVs,MaxVe,curr.date.CURRVAL,999) ;

    END IF ;                                                 (i)
    ****************************
    /*Suppression de l'état à corriger*/
    DELETE FROM TE hist                                      (valid)
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart= :n.Vstart
    AND Vend= :n.Vend ;


        /*Suppression des lignes coalescées à corriger*/                    (valid)
    DELETE FROM TE hist
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND att_temp= :n.att_temp
    AND att_temp= :n.att_temp
    AND att_nontemp= :n.att_nontemp
    AND att_nontemp= :n.att_nontemp
    AND (Vend= :n.Vstart
        OR Vstart= :n.Vend) ;

    /*Cas où l'état courant est impliqué dans le coalescing*/
    IF ct5<>0                                                (j)
    THEN                                                     (j)
        UPDATE TE cour
        SET Vstart=MinVs
        WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide ;

    /*Cas où l'état courant n'est pas impliqué dans le coalescing*/
    ELSE                                                     (j)
        INSERT INTO TE hist                                  (valid)
        VALUES ( :n.att, :n.att,…,MinVs,MaxVe) ;
    END IF ;                                                 (j)

    ****************************

END IF ;                                                     (h)
```

/*Correction attribut temporel d'un état courant*/

```
ELSIF (ct3<>0) AND (  (var_temp<> :n.att_temp) OR (var_temp<> :n.att_temp)
OR (:o.att_tempnull is null AND :n.att_tempnull is not null) )   (g)
(quand att temp ET TE valid – bitemporal
    car quand att temp et transaction, pas de correction, seulement des évolutions
    car quand valid-bitemporal et pas att temp, pas de correction d'attribut temporel)

 THEN                                                        (g)
/*Test de coalescing*/
      /*Regarde s'il existe des états égaux adjacents antérieurs ou postérieurs dans
        H_NV*/
        SELECT count(*) INTO ct4   (valid – bitemporal)
FROM TE hist
WHERE att= :n.att
AND att= :n.att
AND Tend=999            (bitemporal)
```

```
AND Vend= :n.Vstart  ;        (valid – bitemporal)

IF ct4=0                                                    (h)
THEN                                                      (h)
    ****************************
    /*Pas de coalescing*/
    /*Maj de ligne à corriger – Nouvelle information*/                    (bitemporal)
    UPDATE TE cour
    SET att_temp= :n.att_temp
        , att_temp= :n.att_temp
        , Vstart=Vs
                , Tstart=curr_date.CURRVAL
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide;

    /*Création de la ligne reprenant les informations qui ne sont plus valides*/
    INSERT INTO TE hist                                          (bitemporal)
            VALUES ( :n.att, :n.att, Vs, 999, Ts, curr_date.CURRVAL) ;

            ****************************
    /*Màj de la ligne à corriger*/                               (valid)
    UPDATE TE cour
    SET att_temp= :n.att_temp
        , att_temp= :n.att_temp
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide;

****************************

ELSE                                                  (h)
    /*Réalisation du coalescing*/

    /*Enregistrement des dates extrêmes*/
    SELECT Vstart                                  (valid - bitemporal)
    INTO MinVs
    FROM TE hist
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Tend=999                                   (bitemporal)
    AND att_temp= :n.att_temp
    AND att_temp= :n.att_temp
    AND att_nontemp= :n.att_nontemp
    AND att_nontemp= :n.att_nontemp
    AND Vend= Vs
    ;
    ****************************

            /*Création de la ligne reprenant les anciennes informations courantes qui ne
            sont plus valides*/
    INSERT INTO TE hist                                  (bitemporal)
    VALUES( var, var, Vs, 999, Ts, curr_date.CURRVAL) ;

    /*Maj des lignes à corriger – Tend=now*/             (bitemporal)
    UPDATE TE hist
    SET Tend=curr_date.CURRVAL
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Tend=999
    AND att_temp= :n.att_temp
    AND att_temp= :n.att_temp
```

```
        AND att_nontemp= :n.att_nontemp
        AND att_nontemp= :n.att_nontemp
        AND Vend= Vs ;



        /*Màj de l'état courant*/
        UPDATE TE cour                                  (bitemporal)
        SET att_temp= :n.att_temp
            , att_temp= :n.att_temp
            , Vstart=MinVs
            , Tstart=curr_date.CURRVAL
        WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        ;

        ****************************
        /*Suppression de l'état coalescé à corriger*/
        DELETE FROM TE hist                             (valid)
        WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vend=Vs ;

        /*Mise à jour de l'état courant*/                   (valid)
        UPDATE TE cour
        SET att_temp= :n.att_temp
            , att_temp= :n.att_temp
                    , Vstart=MinVs
        WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide ;
        ****************************

    END IF ;                                            (h)

ELSE                                                    (g)
(quand att temp)

    /*Correction Vstart ou Vend*/
        (quand TE valid – bitemporal
                car quand non valid – bitemporal (transaction), pas de correction, seulement des évolutions)
    /*Vérification de validité de l'intervalle*/
    IF :n.Vstart>= :n.Vend                                  (k)
    THEN                                                (k)
        RAISE interv_non_conforme ;
    ELSE                                                (k)
        (Configurations 1,7)
        /*Cas d'une correction de Vstart*/                  (valid – bitemporal)
        IF :n.Vstart<> :o.Vstart                            (l)
        THEN                                            (l)
        ****************************
            /*Annulation de l'état*/                    (bitemporal)
            UPDATE TE hist
            SET Tend=curr_date.NEXTVAL
            WHERE att_ide= :n.att_ide
            AND att_ide= :n.att_ide
            AND Vstart= :o.Vstart
            AND Tend=999 ;

            ****************************
            /*Vérification de l'existence d'états antérieurs*/
```

```
SELECT COUNT(*) INTO ct6                    (valid – bitemporal)
FROM TE hist
WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Vend= :o.Vstart
AND Tend=999                          (bitemporal)
;

/*Existence d'états antérieurs*/              (valid – bitemporal)
IF ct6<>0                                   (m)
        THEN                                        (m)
   /*Vstart diminue*/                       (valid – bitemporal)
   IF :n.Vstart< :o.Vstart                        (n)
   THEN                                     (n)
      /*Regarde s'il faut appliquer le coalescing*/
      SELECT COUNT INTO ct4                 (valid – bitemporal)
      FROM TE hist
      WHERE att_ide= :n.att_ide
      AND att_ide= :n.att_ide
      AND att= :n.att
      AND att= :n.att
      AND Vstart< :n.Vstart
      AND Vend>=:n.Vstart
      AND Tend=999 ;                        (bitemporal)

      IF ct4<>0                             (o)
      THEN                                  (o)
         ****************************
         /*Application du coalescing*/
         SELECT Vstart INTO Vs2             (bitemporal)
         FROM TE hist
         WHERE att_ide= :n.att_ide
         AND att_ide= :n.att_ide
         AND att= :n.att
         AND att= :n.att
         AND Vstart< :n.Vstart
         AND Vend>=:n.Vstart
         AND Tend=999 ;                     (bitemporal)

         /*Création de la ligne reprenant l'information corrigée*/
         INSERT INTO TE hist                (bitemporal)
         VALUES(var, var, var, Vs2,Ve,curr_date.CURRVAL,999) ;

         /*Annulation de l'état coalescé*/
         UPDATE TE hist                     (bitemporal)
                  SET Tend=curr_date.CURRVAL
                  WHERE att_ide= :n.att_ide
                    AND att_ide= :n.att_ide
                  AND att= :n.att
         AND att= :n.att
         AND Vend<>Ve /*ne pas annuler la ligne qui vient d'être créée*/
                  AND Vstart< :n.Vstart
         AND Vend>=:n.Vstart
                    AND Tend=999 ;

                  /*Annulation des états entièrement recouverts*/
                  UPDATE TE hist                (bitemporal)
                  SET Tend=curr_date.CURRVAL
                  WHERE att_ide= :n.att_ide
                    AND att_ide= :n.att_ide
```

```
                AND Vstart>= :n.Vstart
                AND Vend<= :o.Vstart
                AND Tend=999 ;

    ***************************

    /*Mise-à-jour de l'état coalescé*/
    UPDATE TE hist                          (valid)
    SET Vend= :o.Vend
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
                AND Vstart< :n.Vstart
    AND Vend>=:n.Vstart
                ;

    /*Suppression de l'état modifié*/
    DELETE FROM TE hist                     (valid)
                WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide
                AND Vstart= :o.Vstart
    AND Vend= :o.Vend
                  ;

                /* Suppression des états entièrement recouverts*/
                DELETE FROM TE hist             (valid)
                WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide
                AND Vstart>= :n.Vstart
                AND Vend<= :o.Vstart
                ;

    ***************************

ELSE                                    (o)
/*Pas de coalescing*/

    ***************************
    /*Création de la ligne reprenant l'information corrigée*/
    INSERT INTO TE hist                     (bitemporal)
    VALUES(var, var, :n.Vstart,Ve,curr_date.CURRVAL,999) ;

    /*Regarde s'il existe un état partiellement recouvert*/
    SELECT COUNT(*) INTO ct7                (bitemporal)
    FROM TE hist
                WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart< :n.Vstart
    AND Vend> :n.Vstart
    AND Tend 999 ;

    IF ct7<>0                               (p)
    THEN                                (p)

                /*Enregistre les données de l'état partiellement
                recouvert*/
        SELECT att, att, Vstart             (bitemporal)
        INTO var2, var2, Vs2
        FROM TE hist
                WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
```

```
        AND Vstart< :n.Vstart
        AND Vend> :n.Vstart
        AND Tend 999 ;

                /*Création du nouvel état remplaçant l'état partiellement
                recouvert*/
                INSERT INTO TE hist                (bitemporal)
                VALUES(var2, var2, Vs2, :n.Vstart, curr_date.CURRVAL,
                999) ;

        /*Annulation de l'état partiellement recouvert*/
        UPDATE Te hist                    (bitemporal)
        SET Tend=curr_date.CURRVAL
                WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vstart< :n.Vstart
        AND Vend> :n.Vstart
        AND Tend 999 ;


    END IF ;                              (p)

    /*Annulation des états entièrement recouverts*/
    UPDATE Te hist                        (bitemporal)
    SET Tend=curr_date.CURRVAL
            WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart>= :n.Vstart
    AND Vend<= :o.Vstart
    AND Tend 999 ;


    ****************************
    /*Mise-à-jour de l'état modifié*/
    UPDATE TE hist                        (valid)
    SET Vstart= :n.Vstart
    WHERE att_ide= :n.att_ide
    AND Vstart=o.Vstart ;

    /*Mise-à-jour de l'état partiellement recouvert*/
    UPDATE TE hist                        (valid)
            SET Vend= :n.Vstart
           WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart< :n.Vstart
    AND Vend> :n.Vstart
     ;

    /*Suppression des états entièrement recouverts*/
    DELETE FROM TE hist                   (valid)
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart>= :n.Vstart
    AND Vend<= :o.Vstart
    AND Tend 999 ;
    ****************************

  END IF ;                                (o)

ELSE                                      (n)
```

/*Vstart augmente*/

```
    ****************************
    /*Création de la ligne reprenant l'information corrigée*/
    INSERT INTO TE hist                    (bitemporal)
    VALUES(var, var, :n.Vstart,Ve,curr_date.CURRVAL,999) ;

    /*Enregistre les données de l'état qui doit être étendu*/
    SELECT att, att, Vstart                (bitemporal)
    INTO var2, var2, Vs2
    FROM TE hist
            WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vend= :o.Vstart
    AND Tend 999 ;

    /*Annulation de l'état qui doit être étendu*/
    UPDATE TE hist                         (bitemporal)
    SET Tend=curr_date.CURRVAL
            WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vend= :o.Vstart
    AND Tend 999 ;

    /*Création du nouvel état remplaçant l'état qui doit être étendu*/
    INSERT INTO TE hist                    (bitemporal)
    VALUES(var2, var2, Vs2, :n.Vstart, curr_date.CURRVAL, 999) ;

    ****************************
    /*Mise-à-jour de l'état modifié*/
    UPDATE TE hist                         (valid)
    SET Vstart= :n.Vstart
    WHERE att_ide= :n.att_ide
    AND Vstart=o.Vstart ;

    /*Mise-à-jour de l'état qui doit être étendu*/
    UPDATE TE hist                         (valid)
    SET Vend= :n.Vstart
            WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vend= :o.Vstart
     ;

    ****************************

END IF ;                                   (n)

ELSE                                       (m)
/*Pas d'état antérieur*/
    ****************************
    /*Création de la ligne reprenant l'information corrigée*/
    INSERT INTO TE hist                    (bitemporal)
    VALUES(var, var, :n.Vstart,Ve,curr_date.CURRVAL,999) ;
    ****************************
    /* Mise-à-jour de la ligne reprenant l'information corrigée*/
    UPDATE TE hist                         (valid)
    SET Vstart= :n.Vstart
    WHERE att_ide= :n.att_ide
    AND Vstart=o.Vstart ;

    ****************************
```

```
    END IF ;                                        (m)


ELSE                                                (l)
/*Cas d'une correction de Vend*/

    ****************************
    /*Annulation de l'état*/                        (bitemporal)
    UPDATE TE hist
    SET Tend=curr_date.NEXTVAL
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart= :o.Vstart
    AND Tend=999 ;

    ****************************
    /*Vérification de l'existence d'états postérieurs*/
    SELECT COUNT(*) INTO ct6                        (valid – bitemporal)
    FROM TE hist
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart= :o.Vend
    AND Tend=999                                (bitemporal)
    ;

    /*Existence d'états postérieurs*/               (valid – bitemporal)
    IF ct6<>0                                       (q)
            THEN                                        (q)
       /*Vend augmente*/                            (valid – bitemporal)
       IF :n.Vend> :o.Vend                              (r)
       THEN                                         (r)
          /*Regarde s'il faut appliquer le coalescing*/
          SELECT COUNT INTO ct4                     (valid – bitemporal)
          FROM TE hist
          WHERE att_ide= :n.att_ide
          AND att_ide= :n.att_ide
          AND att= :n.att
          AND att= :n.att
          AND Vstart< =:n.Vend
          AND Vend>:n.Vend
          AND Tend=999 ;                            (bitemporal)


          IF ct4<>0                                 (s)
          THEN                                      (s)
          ****************************

             /*Application du coalescing*/
             SELECT Vend INTO Ve2                   (bitemporal)
             FROM TE hist
             WHERE att_ide= :n.att_ide
             AND att_ide= :n.att_ide
             AND att= :n.att
             AND att= :n.att
             AND Vstart< =:n.Vend
             AND Vend>:n.Vend
             AND Tend=999 ;                             (bitemporal)
```

```
    /*Création de la ligne reprenant l'information corrigée*/
    INSERT INTO TE hist                    (bitemporal)
    VALUES(var, var, var, Vs,Ve2,curr_date.CURRVAL,999) ;

    /*Annulation de l'état coalescé*/
    UPDATE TE hist                         (bitemporal)
                SET Tend=curr_date.CURRVAL
                WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide
                AND att= :n.att
    AND att= :n.att
    AND Vstart<>Vs /*ne pas annuler la ligne qui vient d'être créée*/
                AND Vstart< =:n.Vend
    AND Vend>:n.Vend
                  AND Tend=999 ;

                /*Annulation des états entièrement recouverts*/
                UPDATE TE hist                 (bitemporal)
                SET Tend=curr_date.CURRVAL
                WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide
                AND Vstart< :n.Vend
                AND Vend<= :n.Vend
                AND Vstart>= :o.Vend
                AND Tend=999 ;

    ****************************
    /*Mise-à-jour de l'état coalescé*/
    UPDATE TE hist                         (valid)
    SET Vstart= :o.Vstart
                  WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide
                AND att= :n.att
    AND att= :n.att
                  AND Vstart< =:n.Vend
    AND Vend>:n.Vend
                  ;

    /*Suppression de l'état modifié*/
    DELETE FROM TE hist                    (valid)
                  WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide
                AND Vstart= :o.Vstart
                AND Vend= :o.Vend ;

                /*Suppression des états entièrement recouverts*/
                DELETE FROM TE hist            (valid)
                WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide
                AND Vstart< :n.Vend
                AND Vend<= :n.Vend
                AND Vstart>= :o.Vend
                ;

    ****************************

ELSE                                   (s)
/*Pas de coalescing*/

    ****************************
```

```
/*Création de la ligne reprenant l'information corrigée*/
INSERT INTO TE hist                    (bitemporal)
VALUES(var, var, Vs, :n.Vend,curr_date.CURRVAL,999) ;

/*Regarde s'il existe un état partiellement recouvert*/
SELECT COUNT(*) INTO ct7               (bitemporal)
FROM TE hist
            WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Vstart< :n.Vend
AND Vend>:n.Vend
AND Tend 999 ;

IF ct7<>0                              (t)
THEN                                   (t)

            /*Enregistre les données de l'état partiellement
            recouvert*/
    SELECT att, att, Vend              (bitemporal)
    INTO var2, var2, Ve2
    FROM TE hist
            WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart< :n.Vend
    AND Vend>:n.Vend
    AND Tend 999 ;

            /*Création du nouvel état remplaçant l'état partiellement
            recouvert*/
            INSERT INTO TE hist                (bitemporal)
            VALUES(var2, var2, :n.Vend, Ve2, curr_date.CURRVAL,
            999) ;

    /*Annulation de l'état partiellement recouvert*/
    UPDATE Te hist                     (bitemporal)
    SET Tend=curr_date.CURRVAL
            WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart< :n.Vend
    AND Vend>:n.Vend
    AND Tend 999 ;


END IF ;                               (t)

/*Annulation des états entièrement recouverts*/
UPDATE TE hist                         (bitemporal)
SET Tend=curr_date.CURRVAL
            WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
             AND Vstart< :n.Vend
           AND Vend<= :n.Vend
           AND Vstart>= :o.Vend
AND Tend 999 ;


**************************
/*Mise-à-jour de l'état modifié*/
UPDATE TE hist
SET Vend= :n.Vend
```

```
                    WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vstart= :o.Vstart ;

        /*Regarde s'il existe un état partiellement recouvert*/
        SELECT COUNT(*) INTO ct7                (valid)
        FROM TE hist
                    WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vstart< :n.Vend
        AND Vend>:n.Vend
         ;

        IF ct7<>0                               (t)
        THEN                                (t)

                    /*Mise-à-jour de l'état partiellement recouvert*/
            UPDATE TE hist                  (valid)
            SET Vstart= :n.Vend
                    WHERE att_ide= :n.att_ide
            AND att_ide= :n.att_ide
            AND Vstart< :n.Vend
            AND Vend> :n.Vend
            ;
        END IF ;                                (t)

        /*Suppression des états entièrement recouverts*/
        DELETE FROM TE hist                     (valid)
        WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vstart< :n.Vend
        AND Vend<= :n.Vend
        AND Vstart>= :o.Vend
        AND Tend 999 ;
        ***************************

    END IF ;                                    (s)

ELSE                                            (r)
/*Vend diminue*/
    ***************************
    /*Création de la ligne reprenant l'information corrigée*/
    INSERT INTO TE hist                         (bitemporal)
    VALUES(var, var, Vs, :n.Vend, curr_date.CURRVAL,999) ;

    /*Enregistre les données de l'état qui doit être étendu*/
    SELECT att, att, Vend                       (bitemporal)
    INTO var2, var2, Ve2
    FROM TE hist
                WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart= :o.Vend
    AND Tend 999 ;

    /*Annulation de l'état qui doit être étendu*/
    UPDATE TE hist                              (bitemporal)
    SET Tend=curr_date.CURRVAL
                WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart= :o.Vend
```

```
            AND Tend 999 ;

            /*Création du nouvel état remplaçant l'état qui doit être étendu*/
            INSERT INTO TE hist                    (bitemporal)
            VALUES(var2, var2, :n.Vend, Ve2, curr_date.CURRVAL, 999) ;

            ****************************

            /*Mise-à-jour de l'état modifié*/
            UPDATE TE hist                         (valid)
                        SET Vend= :n.Vend
                        WHERE att_ide= :n.att_ide
                           AND att_ide= :n.att_ide
                        AND Vstart= :o.Vstart ;

                        /*Mise-à-jour de l'état étendu*/
                        UPDATE TE hist                         (valid)
                        SET Vstart= :n.vend
                        WHERE att_ide= :n.att_ide
                           AND att_ide= :n.att_ide
                        AND Vstart= :o.Vend ;

            ****************************

         END IF ;                                  (r)

      ELSE                                         (q)
      /*Pas d'état postérieur*/
         ****************************
         /*Création de la ligne reprenant l'information corrigée*/
         INSERT INTO TE hist                    (bitemporal)
         VALUES(var, var, Vs, :n.Vend,curr_date.CURRVAL,999) ;
         ****************************
         /*Mise-à-jour de l'état modifié*/
         UPDATE TE hist                         (valid)
                  SET Vend= :n.Vend
                  WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide
                  AND Vstart= :o.Vstart ;
         ****************************

         (Configuration 7)
         /*Suppression de l'état courant dans la table C*/
         IF :o.vend=999                    (valid - bitemporal)
         THEN
            DELETE FROM TE cour
                        WHERE att_ide= :n.att_ide
            AND att_ide= :n.att_ide ;
         END IF ;

      END IF ;                                     (q)


END IF ;                                           (l)
```

(Configuration 3)

```
    /*Cas d'une correction de Vstart d'un état passé*/      (valid – bitemporal)
IF (:n.Vstart<> :o.Vstart) AND (ct1=0)                          (l)
THEN                                                    (l)
*****************************
   /*Annulation de l'état*/                             (bitemporal)
   UPDATE TE hist
   SET Tend=curr_date.NEXTVAL
   WHERE att_ide= :n.att_ide
   AND att_ide= :n.att_ide
   AND Vstart= :o.Vstart
   AND Tend=999 ;

         ****************************
   /*Vérification de l'existence d'états antérieurs*/
   SELECT COUNT(*) INTO ct6                             (valid – bitemporal)
   FROM TE hist
   WHERE att_ide= :n.att_ide
   AND att_ide= :n.att_ide
   AND Vend= :o.Vstart
   AND Tend=999                                 (bitemporal)
   ;

   /*Existence d'états antérieurs*/                     (valid – bitemporal)
   IF ct6<>0                                            (m)
        THEN                                                (m)
      /*Vstart diminue*/                              (valid – bitemporal)
      IF :n.Vstart< :o.Vstart                               (n)
      THEN                                              (n)
         /*Regarde s'il faut appliquer le coalescing*/
         SELECT COUNT INTO ct4                          (valid – bitemporal)
         FROM TE hist
         WHERE att_ide= :n.att_ide
         AND att_ide= :n.att_ide
         AND att= :n.att
         AND att= :n.att
         AND Vstart< :n.Vstart
         AND Vend>=:n.Vstart
         AND Tend=999 ;                                 (bitemporal)

         IF ct4<>0                                      (o)
         THEN                                           (o)
            /*Application du coalescing*/
            SELECT Vstart INTO Vs2                      (bitemporal)
            FROM TE hist
            WHERE att_ide= :n.att_ide
            AND att_ide= :n.att_ide
            AND att= :n.att
            AND att= :n.att
            AND Vstart< :n.Vstart
            AND Vend>=:n.Vstart
            AND Tend=999 ;                              (bitemporal)

            ****************************
            /*Création de la ligne reprenant l'information corrigée*/
            INSERT INTO TE hist                         (bitemporal)
            VALUES(var, var, var, Vs2,Ve,curr_date.CURRVAL,999) ;
```

```
/*Annulation de l'état coalescé*/
UPDATE TE hist                          (bitemporal)
            SET Tend=curr_date.CURRVAL
            WHERE att_ide= :n.att_ide
              AND att_ide= :n.att_ide
            AND att= :n.att
AND att= :n.att
AND Vend<>Ve /*ne pas annuler la ligne qui vient d'être créée*/
            AND Vstart< :n.Vstart
AND Vend>=:n.Vstart
              AND Tend=999 ;

            /*Annulation des états entièrement recouverts*/
            UPDATE TE hist                  (bitemporal)
            SET Tend=curr_date.CURRVAL
            WHERE att_ide= :n.att_ide
              AND att_ide= :n.att_ide
            AND Vstart>= :n.Vstart
            AND Vend<= :o.Vstart
            AND Tend=999 ;

**************************
/*Mise-à-jour de l'état colasescé*/
UPDATE TE hist                          (valid)
SET Vend= :o.Vend
              WHERE att_ide= :n.att_ide
              AND att_ide= :n.att_ide
            AND att= :n.att
AND att= :n.att
            AND Vstart< :n.Vstart
AND Vend>=:n.Vstart
              ;

            /*Suppression de l'état modifié*/
            DELETE FROM TE hist             (valid)
              WHERE att_ide= :n.att_ide
              AND att_ide= :n.att_ide
            AND Vstart= :o.Vstart ;


            /*Annulation des états entièrement recouverts*/
            DELETE FROM TE hist             (valid)
            WHERE att_ide= :n.att_ide
              AND att_ide= :n.att_ide
            AND Vstart>= :n.Vstart
            AND Vend<= :o.Vstart
            ;

**************************

ELSE                                    (o)
/*Pas de coalescing*/

**************************
/*Création de la ligne reprenant l'information corrigée*/
INSERT INTO TE hist                     (bitemporal)
VALUES(var, var, :n.Vstart,Ve,curr_date.CURRVAL,999) ;

/*Regarde s'il existe un état partiellement recouvert*/
```

```
SELECT COUNT(*) INTO ct7                (bitemporal)
FROM TE hist
            WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Vstart< :n.Vstart
AND Vend> :n.Vstart
AND Tend 999 ;

IF ct7<>0                               (p)
THEN                                    (p)

            /*Enregistre les données de l'état partiellement
            recouvert*/
    SELECT att, att, Vstart             (bitemporal)
    INTO var2, var2, Vs2
    FROM TE hist
            WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart< :n.Vstart
    AND Vend> :n.Vstart
    AND Tend 999 ;

            /*Création du nouvel état remplaçant l'état partiellement
            recouvert*/
            INSERT INTO TE hist             (bitemporal)
            VALUES(var2, var2, Vs2, :n.Vstart, curr_date.CURRVAL,
            999) ;

    /*Annulation de l'état partiellement recouvert*/
    UPDATE TE hist                      (bitemporal)
    SET Tend=curr_date.CURRVAL
            WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart< :n.Vstart
    AND Vend> :n.Vstart
    AND Tend 999 ;


END IF ;                                (p)

/*Annulation des états entièrement recouverts*/
UPDATE TE hist                          (bitemporal)
SET Tend=curr_date.CURRVAL
            WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Vstart>= :n.Vstart
AND Vend<= :o.Vstart
AND Tend 999 ;


**************************
/*Mise-à-jour de l'état*/               (valid)
UPDATE TE hist
SET Vstart= :n.Vstart
WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Vstart= :o.Vstart ;
```

```
                    /*Mise-à-jour de l'état partiellement recouvert*/
                      UPDATE TE hist
                      SET Vend= :n.Vstart
        WHERE att_ide= :n.att_ide
                      AND att_ide= :n.att_ide
        AND Vstart< :n.Vstart
        AND Vend> :n.Vstart
                      ;

        /*Suppression des états entièrement recouverts*/
        DELETE FROM TE hist                    (valid)
        WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vstart>= :n.Vstart
        AND Vend<= :o.Vstart
        AND Tend 999 ;
        ***************************

    END IF ;                                   (o)

ELSE                                           (n)
/*Vstart augmente*/
    ****************************
    /*Création de la ligne reprenant l'information corrigée*/
    INSERT INTO TE hist                        (bitemporal)
    VALUES(var, var, :n.Vstart,Ve,curr_date.CURRVAL,999) ;

    /*Enregistre les données de l'état qui doit être étendu*/
    SELECT att, att, Vstart                    (bitemporal)
    INTO var2, var2, Vs2
    FROM TE hist
              WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vend= :o.Vstart
    AND Tend 999 ;

    /*Annulation de l'état qui doit être étendu*/
    UPDATE TE hist                             (bitemporal)
    SET Tend=curr_date.CURRVAL
              WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vend= :o.Vstart
    AND Tend 999 ;

    /*Création du nouvel état remplaçant l'état qui doit être étendu*/
    INSERT INTO TE hist                        (bitemporal)
    VALUES(var2, var2, Vs2, :n.Vstart, curr_date.CURRVAL, 999) ;

    ****************************
    /*Mise-à-jour de l'état*/                        (valid)
    UPDATE TE hist
    SET Vstart= :n.Vstart
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart= :o.Vstart ;

    /*Mise-à-jour de l'état qui doit être étendu*/
    UPDATE TE hist                             (valid)
    SET Vend= :n.Vstart
              WHERE att_ide= :n.att_ide
```

```
            AND att_ide= :n.att_ide
            AND Vend= :o.Vstart
             ;

            ****************************

        END IF ;                                        (n)

                ****************************
    ELSE                                            (m)
    /*Pas d'état antérieur*/

        /*Création de la ligne reprenant l'information corrigée*/
        INSERT INTO TE hist                         (bitemporal)
        VALUES(var, var, :n.Vstart,Ve,curr_date.CURRVAL,999) ;
        ****************************
        /*Mise-à-jour de l'état*/                   (valid)
        UPDATE TE hist
        SET Vstart= :n.Vstart
        WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vstart= :o.Vstart ;
        ****************************

    END IF ;                                        (m)


        /*Cas d'une correction de Vstart de l'état courant*/ (valid – bitemporal)
ELSIF (:n.Vstart<> :o.Vstart) AND (ct1<>0)                  (l)
THEN                                                (l)
****************************
    /*Enregistrement des anciennes valeurs courantes dans H_NV_*/
    INSERT INTO TE hist                             (bitemporal)
    VALUES(var,var,Vs,999,Ts,curr_date.CURRVAL,999) ;
            ****************************

            /*Mise-à-jour de l'état*/                       (valid)
    UPDATE TE cour
    SET Vstart= :n.Vstart
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart= :o.Vstart ;
    ****************************
    /*Vérification de l'existence d'états antérieurs*/
    SELECT COUNT(*) INTO ct6                        (valid – bitemporal)
    FROM TE hist
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vend= :o.Vstart
    AND Tend=999                                (bitemporal)
    ;

    /*Existence d'états antérieurs*/                (valid – bitemporal)
    IF ct6<>0                                       (m)
            THEN                                            (m)
        /*Vstart diminue*/                          (valid – bitemporal)
        IF :n.Vstart< :o.Vstart                             (n)
        THEN                                        (n)
            /*Regarde s'il faut appliquer le coalescing*/
            SELECT COUNT INTO ct4                   (valid – bitemporal)
```

```
FROM TE hist
WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND att= :n.att
AND att= :n.att
AND Vstart< :n.Vstart
AND Vend>=:n.Vstart
AND Tend=999 ;                          (bitemporal)

IF ct4<>0                               (o)
THEN                                    (o)
    /*Application du coalescing*/
    SELECT Vstart INTO Vs2                  (valid – bitemporal)
    FROM TE hist
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND att= :n.att
    AND att= :n.att
    AND Vstart< :n.Vstart
    AND Vend>=:n.Vstart
    AND Tend=999 ;                      (bitemporal)

    ****************************
    /*Mise-à-jour de l'état courant avec les nouvelles données*/
    UPDATE TE cour                      (bitemporal)
    SET Vstart=Vs2,
                    Tstart=curr_date.CURRVAL
                WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide ;

    /*Annulation de l'état coalescé*/
    UPDATE TE hist                      (bitemporal)
                SET Tend=curr_date.CURRVAL
                WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide
                AND att= :n.att
    AND att= :n.att
    AND Vend<>Ve /*ne pas annuler la ligne qui vient d'être créée*/
                AND Vstart< :n.Vstart
    AND Vend>=:n.Vstart
                  AND Tend=999 ;

                /*Annulation des états entièrement recouverts*/
                UPDATE TE hist                  (bitemporal)
                SET Tend=curr_date.CURRVAL
                WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide
                AND Vstart>= :n.Vstart
                AND Vend<= :o.Vstart
                AND Tend=999 ;

    ****************************
                  /*Mise à jour de l'état*/
                UPDATE TE cour                  (valid)
                SET Vstart=Vs2
                WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide;


    /*Annulation de l'état coalescé*/
```

```
    DELETE FROM TE hist                    (valid)
                WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide
                AND att= :n.att
    AND att= :n.att
    AND Vend<>Ve /*ne pas annuler la ligne qui vient d'être créée*/
                AND Vstart< :n.Vstart
    AND Vend>=:n.Vstart
                  ;

                /*Annulation des états entièrement recouverts*/
                DELETE FROM TE hist                (valid)
                WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide
                AND Vstart>= :n.Vstart
                AND Vend<= :o.Vstart
                ;
    ****************************

ELSE                                    (o)
/*Pas de coalescing*/

    ****************************
    /*Mise-à-jour de l'état courant avec les nouvelles données*/
    UPDATE TE cour                      (bitemporal)
    SET Vstart=Vs2,
                    Tstart=curr_date.CURRVAL
                WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide ;


    /*Regarde s'il existe un état partiellement recouvert*/
    SELECT COUNT(*) INTO ct7            (bitemporal)
    FROM TE hist
                  WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart< :n.Vstart
    AND Vend> :n.Vstart
    AND Tend 999 ;

    IF ct7<>0                           (p)
    THEN                            (p)

                    /*Enregistre les données de l'état partiellement
                    recouvert*/
        SELECT att, att, Vstart             (bitemporal)
        INTO var2, var2, Vs2
        FROM TE hist
                    WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vstart< :n.Vstart
        AND Vend> :n.Vstart
        AND Tend 999 ;

                    /*Création du nouvel état remplaçant l'état partiellement
                    recouvert*/
                    INSERT INTO TE hist             (bitemporal)
                    VALUES(var2, var2, Vs2, :n.Vstart, curr_date.CURRVAL,
                    999) ;
```

```
        /*Annulation de l'état partiellement recouvert*/
        UPDATE TE hist                          (bitemporal)
        SET Tend=curr_date.CURRVAL
                        WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vstart< :n.Vstart
        AND Vend> :n.Vstart
        AND Tend 999 ;


    END IF ;                                    (p)

    /*Annulation des états entièrement recouverts*/
    UPDATE TE hist                              (bitemporal)
    SET Tend=curr_date.CURRVAL
                    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart>= :n.Vstart
    AND Vend<= :o.Vstart
    AND Tend 999 ;


    ****************************


                /*Mise-à-jour de l'état partiellement recouvert*/
                  UPDATE TE hist
                  SET Vend= :n.Vstart
    WHERE att_ide= :n.att_ide
                  AND att_ide= :n.att_ide
    AND Vstart< :n.Vstart
    AND Vend> :n.Vstart
                  ;

    /*Suppression des états entièrement recouverts*/
    DELETE FROM TE hist                         (valid)
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart>= :n.Vstart
    AND Vend<= :o.Vstart
    AND Tend 999 ;
    ****************************

  END IF ;                                      (o)

ELSE                                            (n)
/*Vstart augmente*/
  ****************************
  /*Mise-à-jour de l'état courant avec les nouvelles données*/
  UPDATE TE cour                                (bitemporal)
  SET Vstart=Vs2,
                  Tstart=curr_date.CURRVAL
            WHERE att_ide= :n.att_ide
  AND att_ide= :n.att_ide ;

  /*Enregistre les données de l'état qui doit être étendu*/
  SELECT att, att, Vstart                       (bitemporal)
  INTO var2, var2, Vs2
  FROM TE hist
            WHERE att_ide= :n.att_ide
```

```
        AND att_ide= :n.att_ide
        AND Vend= :o.Vstart
        AND Tend 999 ;

        /*Annulation de l'état qui doit être étendu*/
        UPDATE TE hist                          (bitemporal)
        SET Tend=curr_date.CURRVAL
                WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vend= :o.Vstart
        AND Tend 999 ;

        /*Création du nouvel état remplaçant l'état qui doit être étendu*/
        INSERT INTO TE hist                     (bitemporal)
        VALUES(var2, var2, Vs2, :n.Vstart, curr_date.CURRVAL, 999) ;

        ***************************

        /*Mise-à-jour de l'état qui doit être étendu*/
        UPDATE TE hist                              (valid)
        SET Vend= :n.Vstart
                WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vend= :o.Vstart
         ;

        ***************************

    END IF ;                                    (n)

            ****************************
ELSE                                            (m)
/*Pas d'état antérieur*/

    /*Mise-à-jour de l'état courant avec les nouvelles données*/
    UPDATE C_EMPLOYE                            (bitemporal)
    SET Vstart= :n.Vstart,
                Tstart= curr_date.CURRVAL
            WHERE Matricule= :n.Matricule ;;
    ****************************


END IF ;                                        (m)




ELSIF ( :n.Vend<> :o.Vend) AND (ct1=0)                  (l)
THEN                                            (l)
/*Cas d'une correction de Vend d'un état passé*/


    ****************************
    /*Annulation de l'état*/                    (bitemporal)
    UPDATE TE hist
    SET Tend=curr_date.NEXTVAL
    WHERE att_ide= :n.att_ide
```

```
AND att_ide= :n.att_ide
AND Vstart= :o.Vstart
AND Tend=999 ;

        ****************************

/*Vend augmente*/                    (valid – bitemporal)
IF :n.Vend> :o.Vend                          (r)
THEN                                  (r)
   /*Vend augmente*/

   /*Regarde s'il faut appliquer un coalescing avec la table H_NV_*/
   SELECT COUNT(*) INTO ct6            (valid – bitemporal)
   FROM TE hist
   WHERE att_ide= :n.att_ide
   AND att_ide= :n.att_ide
   AND att= :n.att
   AND att= :n.att
   AND Vstart<= :n.Vend
   AND Vend>=:n.Vend
   AND Tend=999                        (bitemporal)
   ;

   /*Regarde s'il faut appliquer un coalescing avec la table C_*/
   SELECT COUNT(*) INTO ct4            (valid – bitemporal)
   FROM TE cour
   WHERE att_ide= :n.att_ide
   AND att_ide= :n.att_ide
   AND att= :n.att
   AND att= :n.att
   AND Vstart<= :n.Vend
   AND 999>=:n.Vend
   ;

   IF ct6<>0                                 (s)
   THEN                                      (s)
      /*Application du coalescing dans la table H_NV_*/

      ****************************
      SELECT Vend INTO Ve2          (bitemporal)
      FROM TE hist
      WHERE att_ide= :n.att_ide
      AND att_ide= :n.att_ide
      AND att= :n.att
      AND att= :n.att
      AND Vstart< =:n.Vend
      AND Vend>:n.Vend
      AND Tend=999 ;                     (bitemporal)

      /*Création de la ligne reprenant l'information corrigée*/
      INSERT INTO TE hist                (bitemporal)
      VALUES(var, var, var, Vs,Ve2,curr_date.CURRVAL,999) ;

      /*Annulation de l'état coalescé*/
      UPDATE TE hist                     (bitemporal)
            SET Tend=curr_date.CURRVAL
            WHERE att_ide= :n.att_ide
            AND att_ide= :n.att_ide
            AND att= :n.att
      AND att= :n.att
```

```
        AND Vstart<>Vs /*ne pas annuler la ligne qui vient d'être créée*/
                AND Vstart< =:n.Vend
        AND Vend>:n.Vend
                AND Tend=999 ;

                /*Annulation des états entièrement recouverts*/
                UPDATE TE hist                          (bitemporal)
                SET Tend=curr_date.CURRVAL
                WHERE att_ide= :n.att_ide
                AND att_ide= :n.att_ide
                AND Vstart< :n.Vend
                AND Vend<= :n.Vend
                AND Vstart>= :o.Vend
                AND Tend=999 ;

        ****************************
        /*Suppression de l'état modifié*/
        DELETE FROM TE hist                     (valid)
                WHERE att_ide= :n.att_ide
                AND att_ide= :n.att_ide
                AND Vend= :o.Vend
                ;

                /*Mise-à-jour de l'état coalescé*/
        UPDATE TE hist                          (valid)
        SET Vstart= :o.Vstart
                   WHERE att_ide= :n.att_ide
                AND att_ide= :n.att_ide
                AND att= :n.att
        AND att= :n.att
                   AND Vstart< =:n.Vend
        AND Vend>:n.Vend
                ;

                /*Suppression des états entièrement recouverts*/
                DELETE FROM TE hist                     (valid)
                WHERE att_ide= :n.att_ide
                AND att_ide= :n.att_ide
                AND Vstart< :n.Vend
                AND Vend<= :n.Vend
                AND Vstart>= :o.Vend
                ;

        ****************************
    ELSIF ct4<>0                                    (s)
    THEN                                            (s)
    /*Application du coalescing dans la table C_*/

        ****************************
        /*Selection des dates*/
        SELECT Vstart, Tstart INTO Vs2, Ts2     (bitemporal)
        FROM TE cour
                   WHERE att_ide= :n.att_ide
                AND att_ide= :n.att_ide ;

        /*Mise-à-jour de la ligne reprenant l'information corrigée dans C*/
UPDATE TE cour                          (bitemporal)
        SET Vstart=Vs,
            Tstart=curr_date.CURRVAL
                   WHERE att_ide= :n.att_ide
```

```
                AND att_ide= :n.att_ide ;

        /*Création de l'ancien état courant*/
                INSERT INTO TE hist                     (bitemporal)
        VALUES(var, var, var, Vs2,999,Ts2,curr_date.CURRVAL) ;

                /*Annulation des états entièrement recouverts*/
                UPDATE TE hist                  (bitemporal)
                SET Tend=curr_date.CURRVAL
                WHERE att_ide= :n.att_ide
                AND att_ide= :n.att_ide
                AND Vstart< :n.Vend
                AND Vend<= :n.Vend
                AND Vstart>= :o.Vend
                AND Tend=999 ;

        ****************************
        /*Mise-à-jour de l'état C_*/
        UPDATE TE cour                  (valid)
        SET Vstart= :o.Vstart
                   WHERE att_ide= :n.att_ide
                AND att_ide= :n.att_ide
                ;

        /*Suppression de l'état*/
                   DELETE FROM TE hist                  (valid)
                WHERE att_ide= :n.att_ide
                AND att_ide= :n.att_ide
                AND Vstart= :o.Vstart ;

                /*Suppression des états entièrement recouverts*/
                DELETE FROM TE hist             (valid)
                WHERE att_ide= :n.att_ide
                AND att_ide= :n.att_ide
                AND Vstart< :n.Vend
                AND Vend<= :n.Vend
                AND Vstart>= :o.Vend
                ;

        ****************************

    ELSE                                        (s)
    /*Pas de coalescing*/

        ****************************
        /*Création de la ligne reprenant l'information corrigée*/
        INSERT INTO TE hist                     (bitemporal)
        VALUES(var, var, Vs, :n.Vend,curr_date.CURRVAL,999) ;

        /*Cas dans lequel l'état étendu est dans H_NV_*/
SELECT count(*) INTO ct8                (bitemporal)
        FROM TE hist
                   WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vstart< :n.Vend
                AND Vend> :n.Vend
        AND Tend=999 ;

        IF ct8<>0                                   (t)
        THEN                                    (t)
```

```
    /*Enregistre les données de l'état partiellement recouvert*/
    SELECT att, att, Vend                    (bitemporal)
    INTO var2, var2, Ve2
    FROM TE hist
                    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart< :n.Vend
                    AND Vend> :n.Vend
    AND Tend=999 ;

                    /*Création du nouvel état remplaçant l'état partiellement
                   recouvert*/
    INSERT INTO TE hist                      (bitemporal)
    VALUES(var2,var2,:n.Vend,Ve2,curr_date.CURRVAL,999) ;

    /*Annulation de l'état partiellement recouvert*/
    UPDATE TE hist                           (bitemporal)
    SET Tend=curr_date.CURRVAL
                    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart< :n.Vend
                    AND Vend> :n.Vend
    AND Tend=999 ;


END IF ;                                          (t)


              /*Regarde s'il existe un état partiellement recouvert dans C_*/
SELECT count(*) INTO ct9                 (bitemporal)
FROM TE cour
                WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Vstart<= :n.Vend
;

IF ct9<>0                                        (u)
THEN                                             (u)
    /*Enregistre les données de l'état qui doit être étendu*/
    SELECT att, att, Vstart, Tstart              (bitemporal)
    INTO var, var, Vs2, Ts2
    FROM TE cour
                    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide;

                    /*Mise-à-jour de l'état courant*/
    UPDATE TE cour                           (bitemporal)
    SET Vstart= :n.Vend,
        Tstart=curr_date.CURRVAL
                    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide ;

    /*Création de l'ancien état courant*/
    INSERT INTO TE hist                      (bitemporal)
    VALUES(var,var,Vs2,999,Ts2,curr_date.CURRVAL) ;

END IF ;                                         (u)

/*Annulation des états entièrement recouverts*/
UPDATE TE hist                           (bitemporal)
```

```
    SET Tend=curr_date.CURRVAL
                WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart< :n.Vend
    AND Vend<= :n.Vend
    AND Vstart>= :o.Vend
    AND Tend=999 ;

END IF ;                                        (s)
*****************************
    /*Mise-à-jour de l'état*/                   (valid)
    UPDATE TE hist
    SET Vend= :n.Vend
    WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart= :o.Vstart ;

            /*Regarde s'il existe un état partiellement recouvert dans H_*/
    SELECT count(*) INTO ct8                    (valid)
    FROM TE hist
                WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart< :n.Vend
    AND Vend> :n.Vend ;

    IF ct8<>0                                   (t)
    THEN                                        (t)
        /*Mise à jour de l'état partiellement recouvert*/
        UPDATE TE hist                          (valid)
        SET Vstart= :n.Vend
                    WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vstart< :n.Vend
        AND Vend> :n.Vend ;
    END IF ;                                        (t)

    /*Regarde s'il existe un état partiellement recouvert dans C_*/
    SELECT count(*) INTO ct9                    (valid)
    FROM TE cour
                WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart< :n.Vend
    ;

    IF ct9<>0                                   (u)
    THEN                                        (u)
        /*Mise à jour de l'état partiellement recouvert*/
        UPDATE TE cour                          (valid)
        SET Vstart= :n.Vend
                    WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        ;

    END IF ;                                        (u)

    /*Suppression des états entièrement recouverts*/
    DELETE FROM H_PROJET                        (valid)
                WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
            AND Vstart< :n.Vend
```

```
                    AND Vend<= :n.Vend
                AND Vstart>= :o.Vend
                ;
    END IF ;                                        (s)
ELSE                                                (r)
/*Vend diminue*/
    ****************************
    /*Création de la ligne reprenant l'information corrigée*/
    INSERT INTO TE hist                             (bitemporal)
    VALUES(var, var, Vs, :n.Vend, curr_date.CURRVAL,999) ;

    /*Cas dans lequel l'état étendu est dans H_NV*/
    SELECT count(*) INTO ct8                        (bitemporal)
    FROM TE hist
            WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart= :o.Vend
    AND Tend=999 ;

    IF ct8<>0                                       (t)
    THEN                                            (t)

        /*Enregistre les données de l'état qui doit être étendu*/
        SELECT att, att, Vend                       (bitemporal)
        INTO var2, var2, Ve2
        FROM TE hist
                WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vstart= :o.Vend
        AND Tend 999 ;

        /*Annulation de l'état qui doit être étendu*/
        UPDATE TE hist                              (bitemporal)
        SET Tend=curr_date.CURRVAL
                WHERE att_ide= :n.att_ide
        AND att_ide= :n.att_ide
        AND Vstart= :o.Vend
        AND Tend 999 ;

                /*Création du nouvel état remplaçant l'état qui doit être étendu*/
        INSERT INTO TE hist                         (bitemporal)
        VALUES(var2,var2,:n.Vend,Ve2,curr_date.CURRVAL,999) ;
    END IF ;                                            (t)

    /*Cas dans lequel l'état étendu est dans C_*/
    SELECT count(*) INTO ct9                        (bitemporal)
    FROM TE cour
            WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart= :o.Vend
     ;

    IF ct9<>0                                       (u)
    THEN                                            (u)

        /*Enregistre les données de l'état qui doit être étendu*/
        SELECT att, att, Vend                       (bitemporal)
        INTO var2, var2, Ve2
        FROM TE cour
                WHERE att_ide= :n.att_ide
```

```
    AND att_ide= :n.att_ide
    ;

    /*Annulation de l'état qui doit être étendu*/
    UPDATE TE cour                          (bitemporal)
    SET Tstart=curr_date.CURRVAL,
            Vstart= :n.Vend
                WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    ;

                /*Création du nouvel état remplaçant l'état qui doit être étendu*/
    INSERT INTO TE hist                     (bitemporal)
    VALUES(var2,var2,Vs2,999,Ts2,curr_date.CURRVAL) ;
END IF ;                                        (u)

****************************
/*Mise-à-jour de l'état*/                   (valid)
UPDATE TE hist
SET Vend= :n.Vend
WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Vstart= :o.Vstart ;

/*Cas dans lequel l'état étendu est dans H_*/
SELECT count(*) INTO ct8                    (valid)
FROM TE hist
            WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Vstart= :o.Vend
;

IF ct8<>0                                       (t)
THEN                                            (t)

    /*Mise-à-jour de l'état qui doit être étendu*/
    UPDATE TE hist                          (valid)
    SET Vstart= :n.Vend
                WHERE att_ide= :n.att_ide
    AND att_ide= :n.att_ide
    AND Vstart= :o.Vend
    ;

END IF ;                                        (t)

/*Cas dans lequel l'état étendu est dans C_*/
SELECT count(*) INTO ct9                    (valid)
FROM TE cour
            WHERE att_ide= :n.att_ide
AND att_ide= :n.att_ide
AND Vstart= :o.Vend
 ;

IF ct9<>0                                       (u)
THEN                                            (u)

    /*Mise-à-jour de l'état qui doit être étendu*/
    UPDATE TE cour                          (valid)
    SET Vstart= :n.Vend
                WHERE att_ide= :n.att_ide
```

```
                    AND att_ide= :n.att_ide
                    ;

                END IF ;                                        (u)

                ****************************

            END IF ;                                            (r)

        ELSIF (:n.Vend<>:o.Vend) AND (ct1<>0)                   (l)
        /*Cas d'une correction de Vend de l'état courant*/
            ****************************
            /*Suppression de l'objet dans C_*/
            DELETE FROM C_EMPLOYE               (bitemporal)
                    WHERE att_ide= :n.att_ide
            AND att_ide= :n.att_ide ;

                    /*Insertion des anciennes valeurs courantes dans H_NV_*/
            INSERT INTO TE hist                 (bitemporal)
            VALUES(var, var, Vs, 999,TS,curr_date.CURRVAL) ;

                    /*Insertion des dernières informations sur l'entité morte dans H_NV_*/
            INSERT INTO TE hist                 (bitemporal)
            VALUES(var, var, Vs, :n.Vend,curr_date.CURRVAL,999) ;

            ****************************
            /*Création de la ligne reprenant l'information corrigée*/
            INSERT INTO TE hist                 (valid)
            VALUES(var, var, Vs, :n.Vend) ;

            /*Suppression de la ligne C*/
            DELETE FROM TE cour                 (valid)
                    WHERE att_ide= :n.att_ide
            AND att_ide= :n.att_ide ;

            ****************************

        END IF ;                        (l)

    END IF ;                            (k)
  END IF ;                              (g)
 END IF ;                               (f)
END IF ;                                (d)
END IF ;                                (c)


}
```

4 Trigger vérifiant l'intégrité d'une table référencée

4.1 Vérifications de l'intégrité référentielle

Ce trigger a pour but de vérifier que l'intégrité référentielle est respectée lors d'une insertion ou d'une modification dans une table référencée.

Les règles d'intégrité référentielles à respecter sont mentionnées dans le document « Normalisation – Troisième Version – Juillet 2000 ».

4.2 Patterns

Trigger de gestion des entités référencées par une clé étrangère temporelle

(Configurations 1, 7)

(Appliquer trigger pour tables temporelles et historiques pour config 1 et 7)

/*Trigger de gestion des entités référencées par une clé étrangère temporelle*/

```
/*******************************/
/*Mise-à-jour*/
/*Entité temporelle*/
(uniquement pour valid time car dans les autres cas de temporalisation (bitemporal), le valid time n'est pas modifié. De nouveaux états sont créés.
Pas de vérification pour modifications du transaction time. C'est censé être géré par les autres triggers ! ! !)
IF UPDATING THEN                                    (valid)

   /*Entité temporelle*/
     /*Modification de Vstart*/
     /*******************************/
     (uniquement si source v.v ou b.b car si v.o ou b.o, il faut juste s'assurer que la cible est courante et comme il s'agit ici de Vstart, le problème ne se pose pas)
   IF (:new.Vstart > :old.Vstart)                   (source valid.valid – bitemporal.bitemporal)
   THEN
      (pour chaque ref)
      /*Regarde si c'est un intervalle d'extrémité*/
      SELECT COUNT(*) INTO ct1
      FROM TE hist
      WHERE att_ide= :new.att_ide
      AND att_ide= :new.att_ide
      AND Vend= :old.Vstart ;

      /*Cas intervalle extrémité*/
      IF ct1=0
      THEN
         SELECT COUNT(*) INTO ct2
         FROM TE source hist
         WHERE cle= new.att_ide
         AND cle= new.att_ide
         /*******************************/
         AND Vstart< :new.Vstart                    (source valid.valid – bitemporal.bitemporal)
         /*******************************/
         AND Tend=999                               (source bitemporal.bitemporal)
         /*******************************/
      END IF ;                                      (source valid.valid – bitemporal.bitemporal)
         /*******************************/
```

```
    IF (ct2<>0)                          (source valid.valid – bitemporal.bitemporal)
    THEN
        RAISE Obj_ref ;
    END IF ;

  END IF ;

  /*Modification de Vend*/
  /*******************************/
  (pour sources o.o, v.o, b.o, il faut vérifier qu'une entité non courante n'est pas référencée par une entité courante)
  IF (:new.Vend<> :old.Vend)
  AND ( :new.Vend< :old.Vend)                (source v.o – v.v – b.o – b.b)
  THEN
    (pour chaque ref)
    /*Regarde si c'est un intervalle d'extrémité*/
    SELECT COUNT(*) INTO ct1
    FROM TE hist
    WHERE att_ide= :new.att_ide
    AND att_ide= :new.att_ide
    AND Vstart= :old.Vend ;

    /*Cas intervalle extrémité*/
    IF ct1=0
    THEN
        SELECT COUNT(*) INTO ct2
        FROM TE source hist
        WHERE cle= new.att_ide
        AND cle= new.att_ide
        /*******************************/
        AND new.Vend<999                   (source o.o)
        /*******************************/
        AND Vend=999                       (source v.o – b.o)
        /*******************************/
        AND Vend> :new.Vend                (source v.o – v.v – b.o – b.b)
        /*******************************/
        AND Tend=999                       (source b.o – b.b)
        /*******************************/
    END IF ;                               (source v.o – v.v – b.o – b.b)
        /*******************************/

    IF (ct2<>0)                            (source v.o – v.v – b.o – b.b)
    THEN
        RAISE Obj_ref ;
    END IF ;
  END IF ;

END IF ;
/*******************************/

/*Insertion*/
IF INSERTING THEN
  (Pour chaque référence)
  /*Vérification début*/
  /*Regarde si c'est un intervalle d'extrémité*/
  SELECT COUNT(*) INTO ct1
  FROM TE hist
  WHERE att_ide= :new.att_ide
  AND att_ide= :new.att_ide
  /*******************************/
  AND Vstart< :new.Vstart                  (valid référencé par v.v ou b.b)
  /*******************************/
```

```
AND Tstart< :new.Tstart                    (transaction référencé par t.t ou b.b)
/*******************************/
AND Vstart< :new.Vstart                    (bitemporal référencé par v.v)
  AND Tend=999
/*******************************/
AND Vend=999                               (bitemporal référencé par t.t)
  AND Tstart< :new.Tstart
/*******************************/
AND Vstart< :new.Vstart                    (bitemporal référencé par b.b)
  AND Tstart< :new.Tstart
/*******************************/
;

/*Cas intervalle extrémité*/
IF ct1=0
THEN
    SELECT COUNT(*) INTO ct2
     FROM TE source hist
    WHERE cle= new.att_ide
    AND cle= new.att_ide
    /*******************************/
    AND Vstart< :new.Vstart                (cible valid
                                                source v.v – b.b)
    /*******************************/
    AND Tend=999                       (cible valid
                                                source b.b)
    /*******************************/
    AND Tstart< :new.Tstart                (cible transaction
                                                source t.t –b.b)
    /*******************************/
    AND Vend=999                       (cible transaction
                                                source b.b)
    /*******************************/
    AND Vstart< :new.Vstart                (cible bitemporal
                                           source v.v)
       /*******************************/
    AND Tstart< :new.Tstart                (cible bitemporal
                                           source t.t)
       /*******************************/
    AND Vstart< :new.Vstart                (cible bitemporal
       AND Tstart< :new.Tstart                 source b.b)
       /*******************************/
       ;

       IF ct2<>0
       THEN
            RAISE Obj_ref ;
       END IF ;

END IF ;
/*******************************/

/*Vérification fin*/
(Pour chaque référence)
/*Regarde si c'est un intervalle d'extrémité*/
SELECT COUNT(*) INTO ct1
FROM TE hist
WHERE att_ide= :new.att_ide
AND att_ide= :new.att_ide
/*******************************/
```

```
AND Vend > :new.Vend                    (valid référencé par o.o - v.o - v.v – b.o - b.b)
/*******************************/
AND Tend > :new.Tend                    (transac référencé par o.o - t.o - t.t – b.o - b.b)
/*******************************/
AND Vend > :new.Vend                    (bitemporal référencé par v.o - v.v)
  AND Tend=999
/*******************************/
AND Vend=999                            (bitemporal référencé par t.o - t.t)
  AND Tend > :new.Tend
/*******************************/
AND Vend > :new.Vend                        (bitemporal référencé par o.o - b.o - b.b)
  AND Tend > :new.Tend
/*******************************/
;

/*Cas intervalle extrémité*/
IF ct1=0
THEN
    SELECT COUNT(*) INTO ct2
    FROM TE source hist
    WHERE cle= new.att_ide
    AND cle= new.att_ide
    /*******************************/
    AND :new.Vend<999                       (cible valid
                                        source o.o)
    /*******************************/
    AND Vend=999                        (cible valid
                                                source v.o – b.o)
    /*******************************/
    AND Vend > :new.Vend                    (cible valid
                                                source v.o - v.v – b.o - b.b)
    /*******************************/
    AND Tend=999                        (cible valid
                                                source b.o - b.b)
    /*******************************/
    AND :new.Tend<999                       (cible transaction
                                        source o.o)
    /*******************************/
    AND Tend=999                        (cible transaction
                                                source t.o – b.o)
    /*******************************/
    AND Tend > :new.Tend                    (cible transaction
                                                source t.t – b.b)
    /*******************************/
    AND Vend=999                        (cible transaction
                                                source b.o - b.b)
    /*******************************/
    AND :new.Vend<999                       (cible bitemporal
    AND :new.Tend<999                       source o.o)
    /*******************************/
    AND Vend=999                        (cible bitemporal
                                                source v.o)
    /*******************************/
    AND Vend > :new.Vend                    (cible bitemporal
                                        source v.o - v.v)
        /*******************************/
        AND Tend=999                        (cible bitemporal
                                                source t.o)
        /*******************************/
    AND Tend > :new.Tend                    (cible bitemporal
```

```
                                        source t.o - t.t)
        /*******************************/
        AND Vend=999                    (cible bitemporal
        AND Tend=999                    source b.o)
        /*******************************/
      AND Vend > :new.Vend              (cible bitemporal
        AND Tend > :new.Tend                source b.o - b.b)
        /*******************************/
        ;

        IF ct2<>0
        THEN
            RAISE Obj_ref ;
        END IF ;

   END IF ;
   /*******************************/
END IF ;

EXCEPTION
    WHEN Obj_ref THEN
      raise_application_error(-20304,'Cet objet est référencé et la modification le rend non valide') ;
END ;
```

(Configuration 3)

(Appliquer trigger pour tables temporelles pour config 3)

/*Trigger de gestion des entités référencées par une clé étrangère C*/

/*Déclaration trigger*/

/*Déclaration variables*/

/*Début trigger*/

*/*******************************/*
/*Mise-à-jour*/
/*Entité temporelle*/
(uniquement pour valid time car dans les autres cas de temporalisation (bitemporal), le valid time n'est pas modifié. De nouveaux états sont créés.
Pas de vérification pour modifications du transaction time. C'est sensé être géré ar les autres triggers ! ! !)
IF UPDATING THEN (valid)

/*Entité temporelle*/
/*Modification de Vstart*/
(uniquement si source v.v ou b.b car si v.o ou b.o, il faut juste s'assurer que la cible est courante et comme il s'agit ici de Vstart, le problème ne se pose pas)
*/*******************************/*
IF (:new.Vstart <> :old.Vstart) (source valid.valid – bitemporal.bitemporal)
AND (:new.Vstart > :old.Vstart)
THEN
*/*Regarde s'il existe des états dans H_*/*
SELECT COUNT(*) INTO ct1
FROM TE hist
WHERE att_ide= :new.att_ide
AND att_ide= :new.att_ide
;

*/*N'existe pas d'état dans H_*/*
IF ct1=0
THEN
(Pour chaque référence)
*/*Regarde s'il existe etat qui référence objet avec*
Vstart< :new.Vstart dans TE source H/*
SELECT COUNT(*) INTO ct2
FROM TE source hist
WHERE cle= new.att_ide
AND cle= new.att_ide
*/*******************************/*
AND Vstart< :new.Vstart (source valid.valid – bitemporal.bitemporal)
*/*******************************/*
AND Tend=999 (source bitemporal.bitemporal)
*/*******************************/*
;
*/*******************************/*

*/*Regarde s'il existe état qui référence objet avec*
Vstart< :new.Vstart dans TE source C/*
SELECT COUNT(*) INTO ct…
FROM TE source cour
WHERE cle= new.att_ide
AND cle= new.att_ide
*/*******************************/*
AND Vstart< :new.Vstart ; (source valid.valid – bitemporal.bitemporal)

```
        /*******************************/

        IF (ct2<>0)                    (source valid.valid – bitemporal.bitemporal)
        THEN
            RAISE Obj_ref ;
        END IF ;

    END IF ;
  END IF ;
END IF ;


/*Insertion*/
IF INSERTING THEN                      (valid – transaction – bitemporal)
    (Pour chaque référence)
    /*Vérification début*/
    /*Regarde s'il existe des états valides dans H_NV_*/
    SELECT COUNT(*) INTO ct1
    FROM TE hist
    WHERE att_ide= :new.att_ide
    AND att_ide= :new.att_ide
    /*******************************/
      AND Tend=999 ;                   (bitemporal)
    /*******************************/
    ;

    /*N'existe pas d'états valides dans H_NV_*/
    IF ct1=0
    THEN
        (Pour chaque référence)
        /*Regarde s'il existe état qui référence objet avec
        Vstart< :new.Vstart dans TE source H*/
        SELECT COUNT(*) INTO ct2
        FROM TE source hist
        WHERE cle= new.att_ide
        AND cle= new.att_ide
        /*******************************/
        AND Vstart< :new.Vstart            (cible valid
                                                source v.v – b.b)
        /*******************************/
        AND Tend=999                   (cible valid
                                                source b.b)
        /*******************************/
        AND Tstart< :new.Tstart            (cible transaction
                                                source t.t – b.b)
        /*******************************/
        AND Vend=999                   (cible transaction
                                                source b.b)
        /*******************************/
        AND Vstart< :new.Vstart            (cible bitemporal
                                       source v.v)
          /*******************************/
        AND Tstart< :new.Tstart            (cible bitemporal
                                       source t.t)
          /*******************************/
        AND Vstart< :new.Vstart            (cible bitemporal
          AND Tstart< :new.Tstart              source b.b)
          /*******************************/
          ;

          IF ct2<>0
```

```
        THEN
            RAISE Obj_ref ;
        END IF ;


    /*Regarde s'il existe etat qui référence objet avec
    Vstart< :new.Vstart dans TE source C*/
    SELECT COUNT(*) INTO ct3
    FROM TE source cour
    WHERE cle= new.att_ide
    AND cle= new.att_ide
    /*******************************/
    AND Vstart< :new.Vstart                 (cible valid
                                                source v.v – b.b)
    /*******************************/
    AND Tstart< :new.Tstart                 (cible transaction
                                                source t.t – b.b)
    /*******************************/
    AND Vstart< :new.Vstart                 (cible bitemporal
                                            source v.v)
       /*******************************/
    AND Tstart< :new.Tstart                 (cible bitemporal
                                            source t.t)
       /*******************************/
    AND Vstart< :new.Vstart                 (cible bitemporal
       AND Tstart< :new.Tstart                  source b.b)
       /*******************************/
       ;

       IF (ct3<>0)                          (source v.v – t.t – b.b)
    THEN
        RAISE Obj_ref ;
       END IF ;

  END IF ;
  /*******************************/



EXCEPTION
    WHEN Obj_ref THEN
      Raise_application_error(-20304,'Cet objet est référencé et la modification le rend non valide') ;
END ;
```

(Configurations 3)

(Appliquer trigger pour tables temporelles pour config 3)

/*Trigger de gestion des entités référencées par une clé étrangère H_NV_*/

/*Déclaration trigger*/

/*Déclaration variables*/

/*Début trigger*/

/*Mise-à-jour*/
/*Entité temporelle*/
IF UPDATING THEN (valid)
(uniquement pour valid time car dans les autres cas de temporalisation (bitemporal), le valid time n'est pas modifié. De nouveaux états sont créés.
Pas de vérification pour modifications du transaction time. C'est sensé être géré ar les autres triggers ! ! !)

/*Entité temporelle*/
/*Modification de Vstart*/
*/*********************************/*
IF (:new.Vstart <> :old.Vstart) (source valid.valid – bitemporal.bitemporal)
AND (:new.Vstart > :old.Vstart)
THEN
*/*Regarde si c'est un intervalle d'extrémité*/*
SELECT COUNT(*) INTO ct1
FROM TE hist
WHERE att_ide= :new.att_ide
AND att_ide= :new.att_ide
AND Vend= :old.Vstart ;

*/*Cas intervalle extrémité*/*
IF ct1=0
THEN
(Pour chaque référence)
SELECT COUNT(*) INTO ct2
FROM TE source hist
WHERE cle= new.att_ide
AND cle= new.att_ide
*/*********************************/*
AND Vstart< :new.Vstart (source valid.valid – bitemporal.bitemporal)
*/*********************************/*
AND Tend=999 (source bitemporal.bitemporal)
*/*********************************/*
;
*/*********************************/*

SELECT COUNT(*) INTO ct3
FROM TE source cour
WHERE cle= new.att_ide
AND cle= new.att_ide
*/*********************************/*
AND Vstart< :new.Vstart (source valid.valid – bitemporal.bitemporal)
*/*********************************/*
;
*/*********************************/*
IF (ct2<>0) OR (c3<>0) (source valid.valid – bitemporal.bitemporal)
THEN
RAISE Obj_ref ;
END IF ;

```
    END IF ;                                    (source valid.valid – bitemporal.bitemporal)
        /*******************************/
  END IF ;


   /*Modification de Vend*/
   /*******************************/
   IF (:new.Vend<> :old.Vend)
   AND ( :new.Vend< :old.Vend)                  (source valid.valid – bitemporal.bitemporal)
   THEN
     /*Regarde si c'est un intervalle d'extrémité*/
     SELECT COUNT(*) INTO ct1
     FROM TE hist
     WHERE att_ide= :new.att_ide
     AND att_ide= :new.att_ide
     AND Vstart= :old.Vend ;

     SELECT COUNT(*) INTO ct2
     FROM TE cour
     WHERE att_ide= :new.att_ide
     AND att_ide= :new.att_ide
     ;

     /*Cas intervalle extrémité*/
     IF (ct1=0) AND (ct2=0)
     THEN
         (Pour chaque référence)
         SELECT COUNT(*) INTO ct3
         FROM TE source hist
         WHERE cle= new.att_ide
         AND cle= new.att_ide
         /*******************************/
         AND Vend > :new.Vend                   (source valid.valid – bitemporal.bitemporal)
         /*******************************/
         AND Tend=999                           (source bitemporal.bitemporal)
         /*******************************/
         ;
         /*******************************/

         SELECT COUNT(*) INTO ct4
         FROM TE source cour
         WHERE cle= new.att_ide
         AND cle= new.att_ide
         /*******************************/
         AND Vend > :new.Vend                   (source valid.valid – bitemporal.bitemporal)
         /*******************************/
         ;
         /*******************************/
         IF (ct3<>0) OR (ct4<>0)                (source valid.valid – bitemporal.bitemporal)
         THEN
             RAISE Obj_ref ;
         END IF ;

     END IF ;                                   (source valid.valid – bitemporal.bitemporal)
         /*******************************/
  END IF ;


END IF ;
```

```
/*Insertion*/
IF INSERTING THEN
    (Pour chaque référence)
    /*Vérification début*/
    /*Regarde si c'est un intervalle d'extrémité*/
    SELECT COUNT(*) INTO ct1
    FROM TE hist
    WHERE att_ide= :new.att_ide
    AND att_ide= :new.att_ide
    /*******************************/
    AND Vstart< :new.Vstart                 (valid référencé par v.v ou b.b)
    /*******************************/
    AND Tstart< :new.Tstart                 (transaction référencé par t.t ou b.b)
    /*******************************/
    AND Vstart< :new.Vstart                 (bitemporal référencé par v.v)
      AND Tend=999
    /*******************************/
    AND Vstart=999                          (bitemporal référencé par t.t)
      AND Tstart< :new.Tstart
    /*******************************/
    AND Vstart< :new.Vstart                 (bitemporal référencé par b.b)
      AND Tstart< :new.Tstart
    /*******************************/
    ;

    /*Cas intervalle extrémité*/
    IF ct1=0
    THEN
        (Pour chaque référence)
        SELECT COUNT(*) INTO ct2
        FROM TE source hist
        WHERE cle= new.att_ide
        AND cle= new.att_ide
        /*******************************/
        AND Vstart< :new.Vstart             (cible valid
                                                source v.v – b.b)
        /*******************************/
        AND Tend=999                    (cible valid
                                                source b.b)
        /*******************************/
        AND Tstart< :new.Tstart             (cible transaction
                                                source t.t – b.b)
        /*******************************/
        AND Vend=999                    (cible transaction
                                                source b.b)
        /*******************************/
        AND Vstart< :new.Vstart             (cible bitemporal
                                        source v.v)
           /*******************************/
        AND Tstart< :new.Tstart             (cible bitemporal
                                        source t.t)
           /*******************************/
        AND Vstart< :new.Vstart             (cible bitemporal
           AND Tstart< :new.Tstart              source b.b)
           /*******************************/
           ;

        SELECT COUNT(*) INTO ct3
         FROM TE source cour
        WHERE cle= new.att_ide
        AND cle= new.att_ide
```

```
    /*******************************/
    AND Vstart< :new.Vstart                 (cible valid
                                                source v.v – b.b)
    /*******************************/
    AND Tstart< :new.Tstart                 (cible transaction
                                                source t.t – b.b)
    /*******************************/
    AND Vstart< :new.Vstart                 (cible bitemporal
                                            source v.v)
       /*******************************/
    AND Tstart< :new.Tstart                 (cible bitemporal
                                            source t.t)
       /*******************************/
    AND Vstart< :new.Vstart                 (cible bitemporal
       AND Tstart< :new.Tstart                  source b.b)
       /*******************************/
       ;

       IF (ct2<>0) OR (ct3<>0)
       THEN
            RAISE Obj_ref ;
       END IF ;

END IF ;
/*******************************/

/*Vérification fin*/
/*Regarde si c'est un intervalle d'extrémité*/
SELECT COUNT(*) INTO ct1
FROM TE hist
WHERE att_ide= :new.att_ide
AND att_ide= :new.att_ide
/*******************************/
AND Vend > :new.Vend                    (valid référencé par v.o - v.v – b.o - b.b)
/*******************************/
AND Tend > :new.Tend                    (transac référencé par t.o - t.t – b.o - b.b)
/*******************************/
AND Vend > :new.Vend                    (bitemporal référencé par v.o -  v.v)
  AND Tend=999
/*******************************/
AND Vstart=999                          (bitemporal référencé par t.o - t.t)
  AND Tend > :new.Tend
/*******************************/
AND Vend > :new.Vend                    (bitemporal référencé par b.o - b.b)
  AND Tend > :new.Tend
/*******************************/
;

SELECT COUNT(*) INTO ct2
FROM TE cour
WHERE att_ide= :new.att_ide
AND att_ide= :new.att_ide ;


/*Cas intervalle extrémité*/
IF (ct1=0) AND (ct2=0)
THEN
    SELECT COUNT(*) INTO ct3
    FROM TE source hist
    WHERE cle= new.att_ide
```

```
        AND cle= new.att_ide
        /*******************************/
        AND Vend=999                        (cible valid
                                                source v.o – b.o)
        /*******************************/
        AND Vend > :new.Vend                    (cible valid
                                                source v.v – b.b)
        /*******************************/
        AND Tend=999                        (cible valid
                                                source b.o - b.b)
        /*******************************/
        AND Tend=999                        (cible transaction
                                                source t.o – b.o)
        /*******************************/
        AND Tend > :new.Tend                    (cible transaction
                                                source t.t – b.b)
        /*******************************/
        AND Vend=999                        (cible transaction
                                                source b.o - b.b)
        /*******************************/
        AND Vend=999                        (cible bitemporal
                                                source v.o)
        /*******************************/
        AND Vend > :new.Vend                    (cible bitemporal
                                            source v.o - v.v)
           /*******************************/
           AND Tend=999                         (cible bitemporal
                                                source t.o)
           /*******************************/
        AND Tend > :new.Tend                    (cible bitemporal
                                            source t.o - t.t)
           /*******************************/
           AND Vend=999                         (cible bitemporal
           AND Tend=999                         source b.o)
           /*******************************/
        AND Vend > :new.Vend                    (cible bitemporal
           AND Tend > :new.Tend                     source b.o - b.b)
           /*******************************/
           ;

        (courant référence du non courant, problème)
        SELECT COUNT(*) INTO ct4
        FROM TE source cour
        WHERE cle= new.att_ide
        AND cle= new.att_ide
        /*******************************/
           ;

           IF (ct3<>0) OR (ct4<>0)
           THEN
                RAISE Obj_ref ;
           END IF ;

    END IF ;
    /*******************************/
END IF ;
EXCEPTION
     WHEN Obj_ref THEN
       Raise_application_error(-20304,'Cet objet est référencé et la modification le rend non valide') ;
END ;
```

5 Conclusion

Les patterns de génération des triggers de mise-à-jour et de vérification de l'intégrité référentielle des tables référencée ont été définis pour trois configurations de données et pour les différents types de temporalisation disponibles (non temporel, valid Time, Transaction Time et bitemporel) pour chacun des objets.

TimeStamp

Volume 3 - Documents techniques

Deuxième partie

Documentations techniques des outils

Les outils d'exploitation des bases de données temporelles

DB-Main Manual Series

Projet TimeStamp

ELABORATION D'UN GÉNÉRATEUR DE SCRIPTS SQL DE POPULATION D'UNE BASE DE DONNÉES TEMPORELLES

THOMAS LIEUTENANT

FEVRIER 2001

The University of Namur - LIBD
Institut d'Informatique
Rue Grandgagnage, 21 B-5000 Namur
http://www.info.fundp.ac.be/libd

Projet TimeStamp

Elaboration d'un générateur de scripts SQL de population d'une BD

Version 1

FÉVRIER 2001

Thomas LIEUTENANT

Introduction

Dans le cadre du volet d'études fonctionnelles du projet TimeStamp, il apparaissait utile de disposer d'un générateur automatique de scripts SQL permettant de garnir rapidement une base de données dont les tables offrent un nombre de lignes suffisant que pour être représentatives des tailles susceptibles d'être rencontrées dans le milieu d'exploitation, et ce pour obtenir des données de performances des outils de gestion développés dans le cadre du projet qui soient significatives.

L'objectif étant donc de disposer d'un matériau pour nos tests, le générateur développé ne travaille que sur deux tables, lesquelles permettent de travailler avec les requêtes courantes (jointures, agrégation, projection). Les tables sur lequel le générateur travaille sont celles décrites en annexe.

1. Description

Il s'agit d'un programme Win32 tournant en mode console et portant le nom de *generator.exe* .

Le programme nécessite la présence dans son répertoire des fichiers suivants:

- noms.txt, le fichier des noms qui sera utilisé pour générer des nouveaux employés
- loc.txt, le fichier des localités qui sera utilisé pour générer des nouveaux employés
- projets.txt, le fichier utilisé pour générer des nouveaux projets
- params.cfg, le fichier des paramètres de configuration.

La sortie du programme est constituée du fichier script.sql destiné à être fourni à la console ISQL Interbase.

2. Format des fichiers

Les fichiers noms.txt et loc.txt présentent, respectivement, un nom ou une localité par ligne. Un nom ne peut excéder 10 caractères et une localité 10 caractères également.

Le fichier projet doit respecter le format de ligne suivant:

```
<projet>:<theme1>:<theme2>:<theme3>
```

et la dernière ligne se terminer par `;`.

Un projet ne peut excéder 12 caractères et un thème 22. Il est obligatoire de fournir 3 thèmes par projet, mais ils peuvent être identiques.

Le fichier params.cfg doit respecter le format suivant:

```
<ligne>[6]
end:

<ligne> := <mot-clé>:<valeur>;

<mot-clé> := employe | projet | update | delete | insert
             | op_max

<valeur> := 1..9[0..9][*]
```

A l'exception de la valeur associée à op_max, les valeurs correspondent aux pourcentages de travail sur la table ou aux pourcentages d'opérations, et par conséquent le total des valeurs assignées respectivement à (employe,projet) et (update,delete,insert) doit être égal à 100. Il s'agit d'une moyenne sur un grand nombre de tirages (i.e. l'exécution du programme ne garantit pas le respect absolu de ces pourcentages).

La valeur associée à op_max représente le nombre maximum d'opérations (i.e. de lignes dans le script) attendu. Le programme ne garantit pas le respect de cette valeur et peut se terminer avant, selon les autres paramètres.

Le nombre maximum d'insertions est fixe, pour tenir compte des possibilités fournies par les fichiers noms.txt (3515 max.) et projets.txt.

ex.: demander 50000 opérations avec 100% d'inserts ne fournira que (3515 + *nbre_projets*) lignes qui est le nombre maximum d'insertions permises, selon la taille des fichiers.

3. Remarques générales

Le code C utilisé pour programmer le générateur est totalement portable sous UNIX et peut donc être compilé sous ce système.

Les mots-clés du fichier params.cfg peuvent apparaître dans n'importe quel ordre, et seule la première lettre est réellement significative.

Le script généré, script.sql, provoque des déclenchements de triggers pour l'update: travaillant sur un domaine de valeurs restreint (localités, thèmes, budget), des "update" peuvent être générés qui reprennent les valeurs existantes.

4. Programme annexe : gen_prjt.exe

Un petit programme « compagnon » a été développé en annexe au générateur de script, afin de disposer d'un fichier *projets.txt* de grande taille et conforme au format requis par le générateur. Il s'agit également d'un programme Win32 s'exécutant en mode console, générant sa sortie dans le répertoire courant. Il demande un argument en ligne de commande qui est le nombre de projets désirés.

Les projets sont générés selon le canevas PR-*aaaa*-*lnn*, où *aaaa* représente une année (de 1999 à …), *a* une lettre et *nn* un numéro de série (incrémental). Un thème reprend la « nom » du projet avec un numéro attribué aléatoirement entre 1 et 3, pour respecter le fait que les thèmes d'un projet peuvent être identiques. Le canevas général pour un thème est THM-PR*aaaalnn-n*.

Annexe

Les tables utilisées pour les tests de performance TimeStamp et garnies par le générateur développé sont des vues générées à partir des tables suivantes:

```
create table H_PROJET(
 INTITULE char(12) not null,
 debut    integer  not null,
 fin      integer  default 999999 not null,
 THEME    char(22) not null,
 BUDGET   decimal(8) not null,
 primary key (INTITULE,debut));

create table H_EMPLOYE(
 CODE     char(5)  not null,
 debut    integer  not null,
 fin      integer  default 999999 not null,
 NOM      char(10) not null,
 STATUT   char(1)  not null,
 ADRESSE  char(10) not null,
 PROJET   char(12) not null,
 primary key (CODE,debut));
```

vues de travail pour le générateur :

```
create view PROJET(INTITULE,debut,fin,THEME,BUDGET)
 as select INTITULE,debut,fin,THEME,BUDGET
    from   H_PROJET
    where  fin = 999999;

create view EMPLOYE(CODE,debut,fin,NOM,STATUT,ADRESSE,PROJET)
 as select CODE,debut,fin,NOM,STATUT,ADRESSE,PROJET
    from   H_EMPLOYE
    where  fin = 999999;
```

TimeStamp

Volume 3 - Documents techniques

Deuxième partie

Documentations techniques des outils

Les outils d’exploitation des bases de données temporelles

DB-Main Manual Series

Projet TimeStamp

T-ODBC: DOCUMENTATION TECHNIQUE

VERSION 1.0

THOMAS LIEUTENANT, DIDIER ROLAND, VIRGINIE DETIENNE

OCTOBRE 2002

The University of Namur - LIBD
Institut d'Informatique
Rue Grandgagnage, 21 B-5000 Namur
http://www.info.fundp.ac.be/libd

T-ODBC: DOCUMENTATION TECHNIQUE

VERSION 1.0 - OCTOBRE 2002

Thomas LIEUTENANT, Didier ROLAND, Virginie DETIENNE

Chapitre 1

Présentation

La présente documentation technique sera organisée selon les divers modules de l'API T-ODBC.

La première section s'intéressera à la fonction TSQLExecDirect qui constitue le noyau de l'interface utilisateur. Les sections suivantes traiteront de chacun des modules T-ODBC, à savoir le parseur, le traitement du coalescing, le traitement de l'agrégation, le traitement de la jointure et enfin les fonctions additionnelles que sont le couple `register`/`unregister` et l'opérateur de normalisation.

Faisons d'emblée une remarque générale : la grande majorité des fonctions de l'API ont un paramètre représentant la connexion ODBC ouverte au préalable sur la base de données, paramètre de type SQLHDBC. Celui-ci est nécessaire pour chaque traitement effectué au moyen d'appels ODBC, et permet de ne pas avoir à rouvrir une nouvelle connexion au sein de chaque fonction, ce qui serait non seulement pénalisant pour les performances, mais également peu rationnel, la connexion représentant en quelque sorte une unité de traitement. Nous n'y reviendrons dès lors plus dans la suite de ce document.

L'API T-ODBC est développée en se basant exclusivement sur les fonctionnalités ODBC 3.0. Le programmeur utilisant T-ODBC veillera donc à déclarer sa propre application comme répondant aux spécifications de la version 3.

Par ailleurs, il convient d’attirer l’attention du lecteur sur des abus ou raccourcis de langage qui seront utilisés dans ce document. Par exemple, l’API fait usage d’un type composé d’une structure C nommée elemtree. Ce type est utilisé comme type de cellules de listes chaînées génériques. Lorsque nous parlerons d’une «cellule de type X, de valeur Y», nous entendrons en fait «la cellule dont le champ type contient le string X, et le champ value le string Y».

Chapitre 2

L'interface utilisateur

Ce chapitre décrit le fonctionnement interne des fonctions TSQLFreeStmt, TSQLAllocStmt, TSQLExecDirect et TSQLGetError. Outre la présentation de ces fonctions, nous nous attacherons donc à décrire les fonctions auxquelles elles font appel et que l'utilisateur ne connaît pas.

2.1 Gestion du descripteur de requête T-ODBC

Prototypes des fonctions:

```
SQLRETURN TSQLAllocStmt (SQLHDBC hdbc, TSQLHSTMT* thstmt)
SQLRETURN TSQLFreeStmt (TSQLHSTMT* thstmt)
```

Avant de pouvoir utiliser la fonction `TSQLExecDirect` pour exécuter du code SQL ou mini-TSQL, il convient d'allouer un *statement handle*. Ceci est une exigence du fonctionnement normal d'ODBC. Cependant, l'API T-ODBC utilise dans bon nombre de ses fonctions des informations récurrentes, liées à une exécution particulière. Nous avons donc choisi de créer un type propre à T-ODBC, ce qui nous permet en outre d'alléger l'interface des fonctions constitutives de l'API.

Ce type, nommé `TSQLHSTMT`, par analogie au type standard ODBC nommé `SQLHSTMT`, est décrit comme étant une structure C :

TYPE DU CHAMP	NOM DU CHAMP	DESCRIPTION
SQLHSTMT	hstmt	descripteur standard ODBC
short	nbcol	nombre de colonnes de la requête
int	type	type de temporalité
bool	coal	indicateur de demande coalescing
bool	agreg	indicateur de requête d'agrégation temporelle

Table 2.1 - Description du type TSQLHSTMT

Les fonctions TSQLAllocStmt et TSQLFreeStmt ont pour but, respectivement, d'allouer et de libérer le *statement handle* de type TSQLHSTMT. Il s'agit donc d'encapsuler le traitement normal ODBC, en sus de l'allocation mémoire nécessaire à la variable de type TSQLHSTMT*. Il est évident que TSQLAllocStmt procède à l'allocation mémoire, tandis que TSQLFreeStmt procède à la libération de la mémoire alloué par son pendant.

2.2 Exécuter du code SQL ou mini-TSQL

Prototype de la fonction:

```
SQLRETURN TSQLExecDirect (SQLHDBC hdbc, TSQLHSTMT thstmt, char* pass, char* user,
                          char* DataName, SQLCHAR* query)
```

Par rapport à son homologue ODBC, on remarquera que la fonction demande de redonner le couple login/mot de passe, ainsi que le nom DSN de la base de données sur laquelle l'application va travailler. Cette exigence est due à la structure interne de la fonction. En effet, il s'avère nécessaire d'isoler le fonctionnement interne du reste de l'application (i.e. l'application faisant appel à la fonction T-ODBC) en ce qui concerne la gestion des transactions sur la base données. Pour ce faire, la fonction ouvre une nouvelle connexion ce qui a pour effet de réaliser cet isolement, le «COMMIT» ou «ROLLBACK» en ODBC s'effectuant au niveau de la connexion.

Le but de la fonction TSQLExecDirect est de déterminer si la requête est exprimée en SQL ou mini-TSQL, après avoir interrogé le dictionnaire temporel lié à une base de données supportant l'API afin de déterminer la valeur attribuée à l'infini. La valeur ainsi récupérée est placée dans la variable globale `max_time`, de type `int`.

Il s'agit donc dans un premier temps de faire appel au parseur (fonction `BuildTree`, à la section suivante). Ensuite, sur base d'un code de retour de la fonction du parseur, la fonction `TSQLExecDirect` fait directement appel à `SQLExecDirect` ou à `TSQLExec`. Le choix de l'appel est dicté par la détection ou non d'une requête mini-TSQL (en fonction de la syntaxe de celle-ci).

S'il s'agit de faire appel à `TSQLExec`, et donc à une interprétation d'une requête mini-TSQL, la fonction `TSQLExecDirect` marquera dans le dictionnaire temporel qu'une connexion T-ODBC est active.

2.3 Exécuter du code mini-TSQL

Prototype de la fonction :

```
SQLRETURN TSQLExec (SQLHDBC hdbc, TSQLHSTMT* thstmt, elemtree* arbre,
                    char* user, char* pass, char* DataName)
```

L'exécution de code mini-TSQL est confiée à la fonction `TSQLExec`, dont le but est de lancer le traitement adéquat selon le type de requête identifié par le parseur : coalescing, agrégation, jointure ou opérateur spécialisé.

La fonction `TSQLExec` ouvre également la connexion interne mentionnée ci-dessus, et l'on retrouve dès lors les paramètres nécessaires dans son prototype.

Après exécution des modules ad hoc, il appartient à `TSQLExec` de supprimer de la base de données les diverses tables temporaires nécessaires pour le traitement effectué (`RDB$TEMP_COAL` pour le coalescing, `RDB$AGREG_INTERV`, `RDB$AGREG_INTERV_VAL`, `RDB$AGREG_COAL`, `RDB$AGREG_FINAL` pour l'agrégation, `RDB$JOIN_FR` et `RDB$JOIN_TO` pour la jointure).

2.4 Gestion des erreurs

Prototype de la fonction:

```
RETCODE TSQLGetError(int num_record, SQLCHAR SQLState[6], SQLINTEGER NativeError,
                     SQLCHAR* Msg,  int size1, void* Info, int size2,
                      SQLCHAR* Function, int size3)
```

En ce qui concerne la gestion des erreurs, l'interface utilisateur est constituée de la fonction `TSQLGetError`, dont la tâche est d'interroger un descripteur d'erreur. Ce dernier est en fait constitué d'une liste chaînée classique, dans laquelle sont placées les diverses erreurs rencontrées lors de l'exécution des fonctions de l'API. Le code de retour de `TSQLGetError` est soit `SQL_SUCCESS` soit `SQL_NO_DATA`, ce dernier signifiant qu'il n'existe pas de données pour le numéro d'enregistrement demandé, ou que le descripteur d'erreur (i.e. la liste chaînée) est vide (i.e. pointeur `NULL`).

Le nombre d'enregistrements d'erreur pouvant être variable, il est nécessaire d'en connaître le nombre avant de tenter de les récupérer. Ceci se fait en demandant à la fonction `TSQLGetError` de fournir les informations du champ 0, le nombre de champs de descriptions d'erreur étant alors retourné sous forme d'entier dans le paramètre `Info`, qu'il est nécessaire de *caster* vers *int*. Le paramètre `Msg` peut également contenir une information. Si c'est le cas, il s'agira d'un message signalant qu'une erreur est survenue lors du traitement de l'erreur au sein de l'API: ce message correspond donc à une erreur provoquée par les fonctions ODBC de traitement d'erreur. Les autres paramètres, pour le champ numéro 0, sont non significatifs (`NULL` ou indéterminé).

Les contraintes de gestion de la mémoire C nous obligent en outre à demander à l'utilisateur de la fonction de fournir la longueur des paramètres, en bytes. Si les informations disponibles dans le descripteur sont plus grandes que le tampon fourni, alors elles sont tronquées. L'argument `size2` n'a de sens que si `Info` est *casté* vers *char**.

Une fois le nombre d'erreurs connues, les informations sont obtenues par appel successif, la fonction fournissant toute information disponible pour le numéro d'erreur demandé: le code `SQLState`, le

numéro d'erreur natif `NativeError`, le message associé `Msg` et éventuellement une information supplémentaire Info, à *caster* vers *char**, ce dernier paramètre pouvant être `NULL` si rien n'est disponible pour le champ du descripteur d'erreur demandé. Enfin, le nom de la fonction ayant causé l'erreur est également communiqué par le paramètre `Function`. Il peut s'agir d'un nom de fonction ODBC utilisée au sein de l'API, ou de manière plus générique de «`TSQLExecDirect`». Dans ce cas, le code `SQLState` donne plus d'information quant à la localisation de l'erreur au sein de l'API.

Pour garnir le descripteur d'erreur, l'API utilise deux fonctions internes, `InfoErreurODBC` et `InfoErreurTODBC`. Comme leur nom l'indique, ces fonctions sont destinées, respectivement, à obtenir les informations d'erreur retournées par les fonctions standard ODBC ou à créer une description d'erreur particulière à notre implémentation.

Les prototypes de ces fonctions sont les suivants:

```
RETCODE InfoErreurTODBC (char State[6], char* Message, char* Fonction)
RETCODE InfoErreurODBC(SQLSMALLINT HandleType, SQLHANDLE Handle, char* fonction)
```

Le chapitre 8 du présent document est consacré à la description des fonctions de traitement d'erreur au sein de l'API.

Chapitre 3

Le parseur

3.1 Présentation générale

Afin de traiter les requêtes mini-TSQL, il a été nécessaire de développer un parseur spécifique pour reconnaître la syntaxe du langage. Dans ce cadre relativement strict, le parseur crée un arbre représentant la requête mini-TSQL. Une requête qui ne comporte pas de table temporelle dans sa clause `FROM` arrêtera le travail du parseur, renvoyant le code `SQL_NOT_TSQL` qui est utilisé par `TSQLExecDirect` pour faire appel à la fonction ODBC standard qu'est `SQLExecDirect` : il n'y a dans ce cas aucun traitement de la part de l'API T-ODBC.

Si le parseur termine son exécution avec succès, il renvoie `SQL_SUCCESS`.

3.2 L’acquisition de mots

Les fonctions du parseur s'appuient sur l'analyseur lexical produit par le programme FLEX. Le fichier d'entrée fournit à FLEX est proposé en annexe.

L'analyseur FLEX a été voulu générique : il ne reconnaît donc pas les mots-clés du langage mini-TSQL, ni ceux de SQL, mais des chaînes quelconques de caractères devant débuter par une lettre. La reconnaissance des mots-clé du langage est confiée aux fonctions `readtokenfirst` et `readtoken` qui font partie de l'API. `ReadTokenFirst` se charge également de l'initialisation de l'analyseur, dont la réinitialisation est confiée à la fonction `ReadClose`(). L'analyseur est configuré pour prendre ses données d'entrée dans un string et non dans un fichier (ce qui est l'environnement normal), grâce à l'emploi de la fonction `yy_scan_string` dans `ReadTokenFirst`.

Les prototypes de ces 3 fonctions sont les suivants :

```
readtokenfirst (char* type, char* token, const char* requete)
readtoken(char* type, char * token)
int readclose()
```

Un enrichissement du langage, par l'ajout de mots-clés, se marquera donc par l'enrichissement correspondant de la fonction `readtoken` ou `readtokenfirst` (selon la position du mot-clé dans la syntaxe mini-TSQL).

La fonction `readtokenfirst` n'est appelée qu'une fois, au début du traitement de parsage. Les acquisitions de token suivantes se feront par appel à readtoken. Ces fonctions renvoient toutes deux un token et son type, lequel correspond à la nature du token acquis : entier quelconque, entier susceptible d'être une borne temporelle, mot-clé du langage, ou string quelconque.

Le tableau suivant reprend ces divers types :

Opérateur mathématique	`eq, ls, gr`
Délimiteur de période mini-TSQL	`brack_sq, inv_coma`
Elements de syntaxe	`pt,pc,brack_op,brack_cl,coma`
Mot-clé mini-TSQL	`select, from, where, equals, contains, before, meets, overlaps, starts, finishes, transaction, valid, period, timepoint, now, and, or, not, using, timetable, interval, group by, every, normalize, check, create, register, unregister`

Entier	`Integer`
Entier servant de borne temporelle	`int0_999`
Identificateur de fonction agrégative	`FCT_STAT`
String générique	`token`
Fin de string ou de requête	`eof, end`

Table 3.1 - Types retournés par readtokenfirst et readtoken

La fonction `readclose`() permet, à la fin du processus de parsage, de stopper le parseur `yylex`() et de le relancer, afin de permettre l'exécution de plusieurs requêtes sans avoir à relancer l'application (p.ex. constituée d'une interface graphique simple).

3.3 Construction de l'arbre de la requête

Le parser de requête mini-TSQL suit une stratégie top-down avec lecture en avance d'un élément. Donc toutes les fonctions du parser que nous allons rencontrer contiennent les mêmes paramètres *type* et *token* que la fonction readtoken de la section précédente, ces paramètres contenant toujours, aussi bien lors de l'appel que lors du retour, le prochain symbole lu à traiter.

Remarque: dans cette section, nous ferons constamment référence à l'*arbre de la requête*, désignant ainsi la structure chaînée dont le pointeur initial est l'argument `arbre` de la plupart des fonctions traitées ci-après. Par ailleurs, il est évident que la remarque faite au tout début de ce document est toujours d'application (concernant les raccourcis de langage traitant des éléments de type `elemtree`).

3.3.1 La fonction Principale: BuildTree

Le prototype de la principale fonction du module de parsage est le suivant :

```
RETCODE BuildTree (SQLHDBC hdbc, TSQLHSTMT* thstmt, char* query, elemtree* arbre)
```

La fonction a pour but d'établir l'arbre représentant la requête, arbre qui sera créé par les diverses composantes du parseur sous forme de liste chaînée dont le pointeur initial est le paramètre «arbre». Le paramètre «query» représente quant à lui la requête exprimée sous forme de string.

La fonction `BuildTree` crée un arbre dont les différentes branches principales représentent chacune une clause de la requête mini-TSQL, selon le principe présenté dans la conception originelle de l'API ([RAMLOT, 2000], cependant, l'enrichissement des fonctionnalités de l'API, et du langage, a conduit à une réorganisation de l'arbre, ainsi qu'à l'ajout de branches supplémentaires).

La figure 3.1 montre l'organisation générale, ainsi que le détail de chaque branche, en ce compris l'utilisation des listes chaînées de type `tlist` génériques.

Les sections suivantes concernent spécifiquement la lecture et l'analyse de parties spécifiques des requêtes et la création de chaque branche de l'arbre.

3.3.2 Initialisation du parseur: ParseInit

Prototype de la fonction:

```
RETCODE ParseInit(TSQLHSTMT* thstmt, elemtree* arbre, char* query)
```

Le parseur est initialisé par appel à la fonction `ParseInit`, laquelle consiste principalement en l'appel à `readtokenfirst`, laquelle est décrite à la section précédente.

La fonction `ParseInit` crée en outre le premier «plateau» de l'arbre de la requête, c'est-à-dire qu'il initialise l'arbre en créant les cellules-têtes de chaque branche (`select`, `from`, `whereT`, `whereNT`, `join`, `sup`). Enfin, les champs `agreg` et `coal` du descripteur de requête TSQL reçoivent leurs valeurs par défaut, à savoir, respectivement, `false` et `true`.

La fonction ParseInit renvoie toujours `SQL_SUCCESS` (code de retour inutile, mais conforme à la forme générale et disponible pour d'éventuelles évolutions).

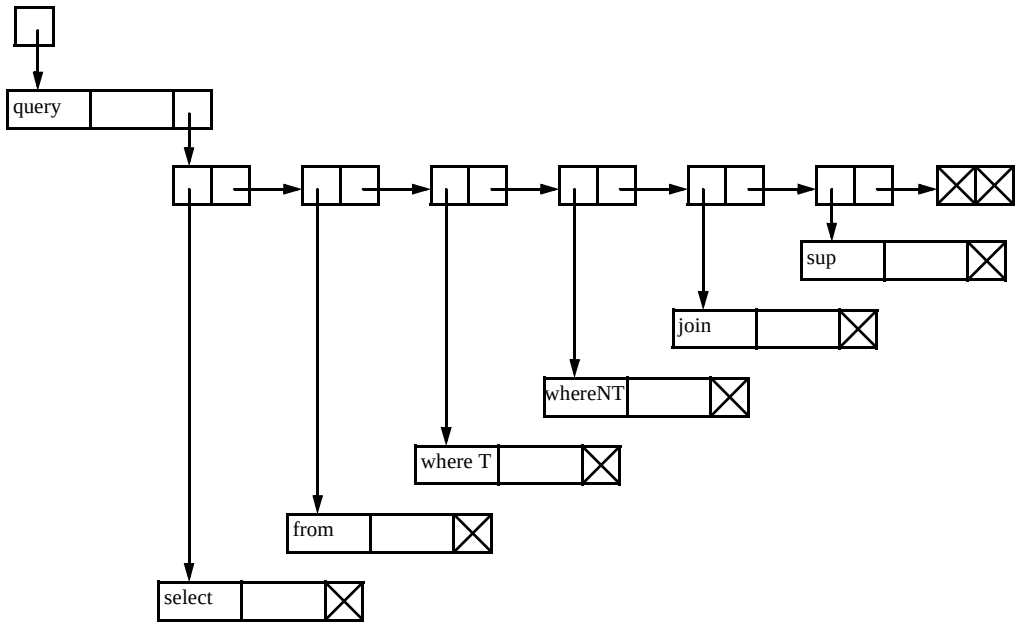


Figure 3.1 - Arbre syntaxique d'une requête SQL.

3.3.3 Le traitement des fonctions spécifiques mini-TSQL

Prototype de la fonction:

```
RETCODE ParseTSQLFunction (SQLHDBC hdbc, TSQLHSTMT* thstmt, elemtree* arbre)
```

La syntaxe du langage mini-TSQL introduit des mots-clés spécifiques qui ne se rapportent pas à la syntaxe générale du langage SQL. Ces mots-clés, à savoir **`REGISTER`**, **`UNREGISTER`** et **`NORMALIZE`** correspondent à des fonctions particulières de l'API T-ODBC. Les deux premiers sont en outre totalement spécifiques à notre implémentation qui fait appel à un dictionnaire temporel pour la gestion de la base de données.

Il nous est donc apparu utile de regrouper le traitement de ces mots-clés au sein d'une fonction particulière, appelée ParseTSQLFunction, laquelle analyse la syntaxe propre à ces mots-clés, créant les cellules appropriées dans l'arbre décrivant la requête. Nous utilisons pour ces mots-clés la branche «SUP», avec les cellules suivantes:

Champ Type	Champ Value	Fils de la cellule
reg	/	paramètres de la fonction
unreg	/	nom de la table
normalize	/	nom de la table, mode et type d'erreurs rechercheées

Table 3.2 - Cellules créées par ParseTSQLFunction

Pour chacune des cellules utilisées, le champ «sons» est utilisé pour enregistrer les paramètres de la fonction. UNREGISTER ne demande comme paramètre que le seul nom de table, nous aurons donc une tlist à cellule unique, dont le type est «token» et la valeur <nom_de_table>. En ce qui concerne la fonction REGISTER, outre le nom de table, nous devons enregistrer dans la liste des paramètres le nom de schéma et de catalogue utilisés. Si la fonction est utilisée sans ces paramètres optionnels, les cellules sont créées mais leur champ «value» est un string vide. Nous utilisons les types «schema» et «catalog» pour ces cellules. Pour la fonction NORMALIZE, il faut stocker le nom de la table, le mode (CHECK ou CREATE), le nom de la table à créer (comme fils du nœud CREATE) le cas échéant, et un nœud pour le type d'erreurs à recherché si cette information est spécifiée.

Le chapitre 7 traite des fonctions utilisées pour implémenter ces opérateurs.

3.3.4 Construction de la branche de la clause SELECT

Prototype de la fonction:

```
RETCODE ParseSelect (TSQLHSTMT* thstmt, elemtree* arbre)
```

La fonction `ParseSelect` a pour objet la création de la branche «SELECT» de l'arbre de la requête. Elle traite donc la clause SELECT d'une requête mini-TSQL. Pour chaque nom de colonne rencontré, une cellule de type elemtree est créée, dont le champ type contient «`token`» et le champ value contient le mot lu par le scanner.

Le traitement des fonctions agrégatives (`count`, `min`, `max` et `avg`) est particulier, afin d'en suivre la syntaxe. Pour celles-ci, il y a création de deux cellules de type elemtree, la première étant dédiée à la fonction agrégative elle-même, tandis que la seconde, unique membre de la tlist fille de la première, contient l'argument de la fonction agrégative, représentée comme un token. Le champ type de la première cellule est «`FCT_STAT`», le champ value comportant alors la nom de la fonction.

La fonction `ParseSelect` détecte également la présence du mot-clé «`every`» qui annule la demande de coalescing: le champ «coal» du descripteur `TSQLHSTMT` prend alors la valeur false.

La fonction retourne SQL_SUCCESS ou SQL_ERROR. Si SQL_ERROR est retourné, une information est disponible dans le descripteur d'erreur.

3.3.5 Construction de la branche de la clause FROM

Prototype de la fonction:

```
RETCODE ParseFrom (SQLHDBC hdbc, TSQLHSTMT* thstmt, elemtree* arbre)
```

La clause FROM en mini-TSQL contient, outre les noms de tables, les éventuels mot-clés `VALID` ou `TRANSACTION` qui précisent la dimension temporelle sur laquelle la requête opère. Le parseur doit donc tenir compte de l'éventuelle présence de l'un de ces mots-clés, dans le respect de la syntaxe mini-TSQL.

Outre le contrôle du respect de la syntaxe, la fonction ParseFrom vérifie également la cohérence du mot-clé `VALID` ou `TRANSACTION` avec le type temporel de la table spécifiée, en faisant appel à la fonction `VerifTabTemp` qui a pour but de donner le type temporel d'une table enregistrée dans le dictionnaire temporel T-ODBC.

Si les tables spécifiées dans la requête ne sont pas de type temporel, et en l'absence de mot-clé `VALID` ou `TRANSACTION`, la fonction retourne `SQL_NOT_TSQL`. Si une erreur de syntaxe ou de cohérence (`TRANSACTION` sur une table en valid time, p.ex.) est détectée, la fonction retourne `SQL_ERROR`.

Si aucun problème ne survient et que l'arbre de la branche est construit correctement, la fonction retourne `SQL_SUCCESS`.

3.3.6 Construction de la branche de la clause WHERE

Prototype des fonctions:

```
RETCODE ParseWhere (TSQLHSTMT* thstmt, elemtree* arbre)
RETCODE readCondition (elemtree* & p, char* firsttyp)
void TokenToTree(char* typ, char* tok, tlist* buf)
RETCODE BuildCondTemp(elemtree* & left, elemtree* & right, char* & op, tlist* buf)
```

La clause `WHERE`, séparée en conditions temporelles et conditions non temporelles selon la syntaxe mini-TSQL, est traitée par une fonction dédiée, `BuildCondTemp` en ce qui concerne les conditions temporelles. Les conditions non temporelles ne subissent pas de traitement particulier : les tokens de la requête sont acquis séquentiellement et placés dans une liste chaînée, ce qui permet de supporter la syntaxe complète de SQL pour cette clause. Cette tâche est supportée par la fonction `TokenToTree`, dont le but est de transformer le résultat de readtoken en cellule de l'arbre, branche `WhereNT`.

La fonction principale de traitement de la clause WHERE est ParseWhere. Celle-ci fait appel à BuildCondTemp pour construire l'arbre des conditions temporelles, tandis que le traitement des conditions non temporelles y est directement réalisé à l'aide de la fonction TokenToTree.

Si la clause SELECT contient une requête d'agrégation, il convient de ne pas traiter particulièrement la clause WHERE, puisque celle-ci fera l'objet d'un nouveau traitement lors du coalescing initial requis par l'agrégation temporelle. Dès lors, la fonction reconstruit, sur base des éléments renvoyés par le parseur, le string original, passé de manière «brute» au module d'agrégation.

Les codes de retour possibles sont SQL_SUCCESS ou SQL_ERROR, en cas d'erreur de syntaxe.

3.3.6.1 Construction de la branche des conditions temporelles

Le prototype de la fonction de construction des conditions est le suivant :

```
RETCODE BuildCond(SQLHDBC hdbc, elemtree* arbre, elemtree* &condT, tlist* condNT,
                  bool &join, char* typ, char* tok)
```

Cette fonction analyse l'entièreté de la clause WHERE et construit les branches WHERET et WHERENT de l'arbre. Si une expression de jointure temporelle est rencontrée, cette fonction appelle la fonction Parse_join() (qui va elle-même générer le sous-arbre du nœud Join de l'arbre général) et transmet (via le paramètre booléen "join") le fait qu'une jointure a été lue ou non.

Les paramètres "hdbc" et "arbre" sont seulement requis pour être transmis à Parse_join. Les paramètres condT, condNT et join servent au retour des résultats. Les paramètres "typ" et "tok" contiennent les informations du token lu en avance.

CondT est l'arbre syntaxique des conditions temporelles qui sera attaché à la branche WhereT de l'arbre par la fonction appelante. Chaque nœud de cet arbre correspond à un opérateur AND, OR ou NOT, les feuilles sont des termes temporels. Les priorités des opérateurs sont gérées grâce à deux piles (une pour les OR, une pour les AND). Le traitement d'une expression entre parenthèse est effectué par un appel récursif à BuildCond. Les termes temporels sont lus par la fonction ReadCondition dont le prototype est le suivant :

```
RETCODE readCondition (elemtree* &p, char* typ, char* tok)
```

Le résultat de ReadCondition est un mini-arbre pointé par p, composé d'un nœud correspondant à un opérateur temporel qui a deux enfants, ses deux arguments. Les arguments des opérateurs temporels sont lus grâce aux fonctions "read_time_operand()", "analyse_timepoint_string()" et "analyse_period_string()" dont la sémantique est évidente.

CondNT est une liste des tokens faisant partie d'une condition non-temporelle (c'est à dire ne faisant pas partie d'une condition temporelle, ni d'une jointure temporelle) qui ont été lus. C'est liste pourra être utilisée directement comme liste de fils pour le nœud WhereNT de l'arbre général.

3.3.7 Construction de la branche de jointure

Prototype de la fonction :

```
RETCODE Parse_join(SQLHDBC hdbc, elemtree* arbre, tlist* bufNT, char* typ, char*
tok)
```

La fonction `Parse_join` a pour objectif la création de la branche «JOIN». Il s'agit d'une fonction auxiliaire du module `BuildTree`.

La cellule de tête de cette branche est de type «JOIN». Sa valeur indique les états à prendre en considération lors de la jointure. Si la valeur est *f*, l'opérateur de jointure ne prend en compte que les états courants. Si c'est *t*, ce sont les historiques qui sont retenus. Cette décision dépend de l'aspect temporel des tables et des colonnes. Si au moins une des tables n'est pas temporelle, ce sont les états courants qui sont retenus. Si les deux tables sont temporelles et si au moins une des colonnes figurant dans les conditions de jointure est non temporelle et non identifiante d'entité, la jointure se fait sur les états courants. Dans le cas contraire, elle se fait sur les historiques.

Cette cellule a pour fille une `tlist` de 2 cellules de type `elemtree`, correspondant aux tables origine et destination de la requête de jointure. Ces cellules ont respectivement pour type «table_from» et «table_to». Le champs `value` est occupé par le nom logique de la table.

Chacune de ces cellules est parent d'une `tlist` de cellules de type `elemtree`. Ces éléments correspondent aux colonnes figurant dans la condition de jointure. Ils sont de type `token` et ont pour valeur le

nom de la colonne. Pour les deux tables, l'ordre des colonnes dans la `tlist` est celui d'apparition dans la requête.

3.3.8 Construction de la branche relative aux clauses additionnelles

Prototype de la fonction

```
RETCODE ParseSup (TSQLHSTMT* thstmt, elemtree* arbre)
```

La syntaxe mini-TSQL prévoit l'usage du mots-clé GROUP BY en conjonction avec une requête d'agrégation.

La fonction crée la branche «SUP» dans l'arbre de la requête. Cette branche est constituée (i.e. champ «sons» de la cellule elemtree dont le champ value est «SUP») d'une tlist d'une cellule de type GROUPBY qui reprend l'ensemble des noms de colonnes (tokens) sous forme de tlist pointée par le champ «sons», selon la structure habituelle de l'arbre de la requête. On profite par ailleurs du parseur de la clause pour en compter le nombre d'éléments (i.e. le nombre de colonnes qui y sont déclarées), ce qui sera utile dans le reste du code lors du traitement de la liste chaînée. On utilise le champ «value» de la cellule «GROUPBY» pour y stocker le nombre d'éléments (converti sous forme de littéral).

Chapitre 4

Le traitement du coalescing

4.1 Présentation

Le traitement du coalescing fait l'objet principal du mémoire d'O.Ramlot [RAMLOT 2000], et nous renvoyons donc le lecteur à celui-ci pour l'explication de l'algorithme utilisé. Nous nous attacherons ici à la description des fonctions que nous utilisons.

Le code du coalescing se trouve dans le fichier module_coal.cpp, tandis que la plupart des fonctions auxiliaires (section 4.3 ci-dessous) sont regroupées au sein du fichier odbcfc.cpp.

4.2 Fonctions principales

Prototype des fonctions:

```
RETCODE TSQLCoalesce (SQLHDBC hdbc, TSQLHSTMT* thstmt, elemtree* quer);
RETCODE TempCoalescing(SQLHDBC hdbc, char* select, int nbcol, int nbcoladd,
                       char* coaltable, char* restype, int* tabposkey,
                       int lengthpos);
```

La fonction `TSQLCoalesce` a pour tâche de préparer les structures nécessaires au coalescing (création de la table temporaire), de créer la requête de coalescing en ajoutant à la requête initiale les colonnes temporelles ad hoc (selon le type de table) ainsi que les colonnes faisant partie de l'identifiant d'entité et non nécessairement sélectionnées dans la requête initiale. En effet, ce sont ces colonnes qui servent de base au coalescing, et elles doivent donc être présentes dans la requête effective. La requête littérale est reconstituée à partir de l'arbre à l'aide des fonctions du module `BuildString`. Il est alors possible d'effectuer le traitement de coalescing, à l'aide de la fonction `TempCoalescing` qui implémente l'algorithme développé dans [RAMLOT 2000]. Enfin, la requête de l'utilisateur est adaptée afin de travailler sur la table contenant le résultat du coalescing, à savoir `RDB$TEMP_COAL`. La fonction renvoie la valeur `SQL_SUCCESS` si son déroulement n'a pas posé de problème, et `SQL_ERROR` sinon. Une information supplémentaire peut être disponible dans le descripteur d'erreur T-ODBC.

La fonction `TempCoalescing` a de multiples arguments nécessaires à son exécution. Outre le string de la requête, `select`, il faut lui fournir le nombre de colonnes initial nbcol ainsi que le nombre de colonnes ajoutées nbcoladd. Elle demande également à connaître le nom de la table temporaire (en fait, il s'agira de `RDB$TEMP_COAL`), le type temporel de résultat `restype` (i.e., BT,T ou V). Enfin, on lui communique les positions dans la requête des colonnes appartenant à l'identifiant d'entité via le tableau `tabposkey`, ainsi que la longueur de ce tableau, `lengthpos`. (pour rappel, il n'y a pas de moyen direct en C de connaître facilement la taille d'un tableau).

4.3 Fonctions auxiliaires

```
void AddPeriod(elemtree* quer, char* restype)
bool sameTab(sqlchar* list1, sqlchar* list2, int lengthtot, int ts, int te)
void copyall(sqlchar* list1, sqlchar* list2, int length)
bool idemEntity(sqlchar* curentity, sqlchar* liststring, int* tabposkey,
                int lenghtpos)
void inserttab(sqlchar* liststring, tlist* tabprobV, int length,
               List<colinfo*> &listInfo)
bool allegal(sqlchar* list1, sqlchar* list2, int length, inte vs, int ve,
             int ts, int te)
```

```
void insertprobV(tlist* &tabprobV, int nbcol, int nbcoladd, tlist* &tabfinal,
                 sqlchar* &elemref, int vs, int ve, int ts, int te,
                  List<colinfo*>&listInfo, int tou)
RETCODE TempCreateResult(SQLHDBC hdbc, SQLHSTMT hstmt, elemtree* quer,
                         char* nametab1, char*coaltable, elemtree* listaddselect,
                          List<colinfo*>&listInfo)
RETCODE TempInfoTable(SQLHDBC hdbc, char*table, e_when time, tabinfo & info)
RETCODE TempNowValue(SQLHDBC hdbc, int* now)
RETCODE TempEntitykey(SQLHDBC hdbc, char* phystab, elemtree *listorder,
                      int & nbrkey)
```

Les fonctions auxiliaires utilisées dans le cadre du module de coalescing se répartissent en deux groupes: l'un a pour objet le traitement de listes chaînées (soit un traitement simple, soit un traitement complexe appuyant le processus de coalescing), et l'autre se compose de fonctions reposant sur ODBC afin d'interagir avec la base de données. Ces dernières, devant travailler sur la BD, utilisent toutes le paramètre `hdbc` représentant la connexion ouverte en début de traitement.

4.3.1 Les fonctions de traitement

La fonction `sameTab` a pour but de déterminer si toutes les valeurs des deux tableaux (pointés par `list1` et `list2`) dont les indices sont compris entre «0» et «`lengthot`», sauf en "ts" et en "te", sont respectivement les mêmes, auquel cas la fonction retourne `true`. Dans le cas contraire, c'est la valeur `false` qui est retournée.

La fonction `copyall` a pour but de copier l'ensemble des éléments (null-terminated strings) d'un tableau `list2`, de longueur connue `length`, dans un tableau `list1` préexistant, de longueur minimum `length`. La fonction ne procède donc pas elle-même a l'initialisation du tableau de destination.

La fonction `idemEntity` a pour but de déterminer si tous les éléments (null-terminated strings) des deux tableaux (pointés par `curentity` et `listring`) dont les indices sont compris dans un tableau d'entier (pointé par `tabposkey`) de longueur connue `lengthpos` sont respectivement les mêmes. La fonction est utilisée dans le cadre du coalescing pour déterminer si deux lignes concernent la même entité. Si c'est le cas, la fonction retourne la valeur `true`, et dans le cas contraire elle retourne la valeur `false`.

La fonction `inserttab` a pour but d'ajouter à une liste chaînée de type `tlist` (pointée par `tabprobV`) l'ensemble des éléments (null-terminated strings) dont le type est connu (dans `listInfo`) d'un tableau (pointé par `liststring`) de longueur connue (`length`).

La fonction `allegal` a pour but de déterminer si tous les éléments (null-terminated strings) des tableaux pointés par `list1` et `list2`, dont les indices sont compris entre 0 et `length`, sauf en `vs`, `ve`, `ts` et `te`, sont respectivement les mêmes. Dans ce cas, la fonction retourne la valeur `true`, et dans le cas contraire elle retourne la valeur `false`.

La fonction `insertprobV` a pour but d'insérer dans une liste chaînée de type `tlist` dont les éléments sont eux-mêmes de pareilles listes (pointée par `tabprobV`) une nouvelle liste en tenant compte de la résolution des problèmes horizontaux, telle qu'expliquée dans [RAMLOT 2000]. Le paramètre `nbcol` représente le nombre de colonnes apparaissant dans la clause `SELECT` de la requête initiale, tandis que le paramètre `nbcoladd` représente le nombre de colonnes ajoutées par l'algorithme de traitement à cette requête (i.e. les colonnes de dimension temporelle) et que les paramètres `vs`, `ve`, `ts` et `te` représentent la position de ces paramètres. Le paramètre `tou` sert quant à lui de compteur, afin de savoir si la fonction a déjà été exécutée pour l'état courant: ceci nous permet de résoudre le problème dit «de la tenaille» lors du coalescing bitemporel. Le résultat de l'insertion de `elemref` dans `tabprobV` est la liste `tabfinal`. `listInfo` contient des informations utiles à propos des colonnes traitées.

4.3.2 Les fonctions reposant sur ODBC

La fonction `TempCreateResult` a pour but de créer la table intermédiaire `RDB$TEMP_COAL`, appelée «table de travail» dans [RAMLOT 2000]. Cette fonction compose la requête de création de table, sur base des informations disponibles dans le dictionnaire temporel T-ODBC concernant les types des

colonnes, et leur dimension, ces dernières informations étant stockées dans la liste `listInfo`. La fonction se base en effet sur la table physique correspondant à la table logique spécifiée dans la requête initiale afin de créer une table temporaire de même structure. La fonction renvoie `SQL_SUCCESS` si la création s'effectue sans problème, ou un des codes d'erreur standard ODBC en cas de problème (selon la fonction ODBC ayant provoqué l'erreur).

La fonction `TempInfoTable` a pour but de déterminer le nom de la table physique à laquelle accéder, sur base de la temporalité de la requête (accès aux états courant uniquement, ou passés également) et du nom de la table logique spécifiée dans la requête. La fonction donne son résultat dans la structure à 2 champs `info` de type `tabinfo`: outre le nom de table est également fourni son type temporel. La fonction renvoie `SQL_SUCCESS` si son déroulement se termine avec succès, et `SQL_ERROR` sinon.

La fonction `TempoNowValue` a pour but de récupérer dans le dictionnaire temporel T-ODBC la valeur correspond au mot-clé mini-TSQL `now`. Cette valeur est conservée dans la table `RDB$TEMP_TIME`. La fonction renvoie `SQL_SUCCESS` si elle se termine avec succès et `SQL_ERROR` sinon. Dans ce cas, une information est disponible dans le descripteur d'erreur T-ODBC.

La fonction `TempEntityKey` a pour but de déterminer l'identifiant d'entité d'une table temporelle phystab, par interrogation de la table `RDB$TEMP_KEY` du dictionnaire temporel T-ODBC. La liste des colonnes constituant l'identifiant est donnée par `listorder`, tandis que `nbrkey` donne le nombre de colonnes dans la liste. Si la fonction se déroule correctement, le code de retour prend la valeur SQL_SUCCESS. Dans le cas contraire, une information supplémentaire est disponible dans le descripteur d'erreur T-ODBC.

Chapitre 5

Le traitement de l'agrégation

5.1 Présentation

Le traitement de l'agrégation, repris dans le module module_agreg.cpp, se base sur le document TimeStamp *Elaboration d'une procédure d'agrégation temporelle* [HAINAUT 2002].

La fonction principale du module d'agrégation est la fonction `TSQLAggregate`. Celle-ci repose sur quatre phases implémentant chacune des étapes de l'agrégation.

Ainsi, il s'agit tout d'abord de réaliser un coalescing initial de la projection de la table initiale sur les colonnes de la requête. Remarquons que les colonnes temporelles qui ne seraient pas spécifiées dans la requête sont ajoutées.

La phase suivante est celle de la valuation des intervalles minimaux, réalisée grâce à une jointure SQL entre la table des intervalles minimaux et la table résultant de la projection de la table de départ sur les colonnes présentes dans la requête.

Une fois la valuation des intervalles minimaux réalisée, on peut procéder à l'agrégation proprement dite, par usage d'une requête SQL standard. Exécuter une requête temporelle revient donc au final à exécuter une requête classique, mais sur une table construite à cet effet.

Enfin, l'opérateur de coalescing est appliqué sur le résultat de la requête, pour donner un résultat normalisé. Cette dernière phase n'est pas exécutée si l'utilisateur a fait usage du mot-clé `EVERY` dans sa requête.

Notre opérateur d'agrégation temporelle ne traite qu'une dimension temporelle. Dans le cas de l'utilisation d'une table bitemporelle, le parseur exigera la présence du mot-clé `VALID` ou `TRANSACTION` afin de spécifier la dimension à traiter (la plus utile étant sans conteste le valid time). En outre, seules les valeurs valides dans la table sont prises en compte pour l'agrégation, c'est-à-dire les valeurs pour lesquelles Tend est égal à la valeur représentant le futur infini (spécifiée dans la table `RDB$TEMP_TIME`).

Les positions des colonnes temporelles et des fonctions d'agrégation sont gérées dynamiquement pour répondre à une syntaxe très souple de la requête d'agrégation.

5.2 Fonctions principales

5.2.1 TSQLAggregate

La fonction TSQLAggregate peut être perçue comme le moteur de l'exécution d'une requête temporelle agrégative. En effet, son rôle principal est d'enchaîner les quatre phases décrites ci-dessus en se servant des fonctions décrites ci-après.

Le coalescing initial fait appel à la fonction TSQLAggregate. Le résultat est stocké dans la table temporaire RDB$AGREG_COAL.

La création des intervalles minimaux se fait en deux phases, la première consistant à extraire de la table toutes les bornes temporelles, ce que nous réalisons à l'aide d'une requête faisant intervenir la clause UNION. Une clause ORDER BY permet de trier le résultat par identifiant et par valeur de borne temporelle de départ. Le résultat de cette fonction est une table nommée RDB$AGREG_INTERV, contenant, pour chaque valeur d'identifiant d'entité, les intervalles minimaux présentés sous forme de colonnes temporelles «classiques».

La valuation des intervalles minimaux se réalise par jointure entre `RDB$AGREG_COAL` et `RDB$AGREG_INTERV`. Le résultat est placé dans `RDB$AGREG_INTERV_VAL`. Il s'agit ici de la simple

exécution d'une requête SQL, construite dynamiquement à l'aide de l'arbre de la requête initiale (qui donne les colonnes présentes dans `RDB$AGREG_COAL`), par utilisation des fonctions ODBC ad hoc.

L'exécution de l'agrégation proprement dite se fait sur la table `RDB$AGREG_INTERV_VAL`. Le résultat est placé dans `RDB$AGREG_FINAL`. Cette fonction repose sur les opérateurs d'agrégation SQL, et ne présente donc pas de difficulté particulière. L'essentiel du traitement effectué par cette fonction est la construction dynamique de la requête SQL.

5.3 Fonctions auxiliaires

Le module d'agrégation fait appel à des fonctions spécialisées pour gérer les tables temporaires. Les créations de tables sont assurées par des fonctions de structure identique, adaptées au cas particulier de la table à créer. Garnir ces tables intermédiaires est également réalisé par des fonctions *ad hoc*. Le fonctionnement d'une requête de création de table est présenté au chapitre 9. Nous signalerons simplement ici que chaque fonction peut présenter des variations par rapport à la fonction générique, par exemple pour récupérer les valeurs de type, longueur, longueur décimale, des colonnes si celles-ci sont également nécessaires ailleurs dans le code de l'API (par exemple, lors de l'utilisation de la fonction SQLBindParameter qui demande à connaître le type SQL exact des colonnes).

Le choix d'utiliser des fonctions auxiliaires dédiées pour garnir les tables avec une liste de valeurs (tlist*) a pour but principal la lisibilité du code. On peut en effet se demander si ceci n'a pas un impact négatif sur les performances, puisque ces fonctions réalisent une seule insertion par appel, utilisant chaque fois la relativement lourde structure ODBC: allocation de *statement*, création de la requête, exécution et enfin libération du *statement*. Il serait sans doute plus efficace d'utiliser le schéma ODBC reposant sur le couple `SQLPrepare`/`SQLExecute`.

Chapitre 6

Le traitement de la jointure temporelle

6.1 Présentation

Le traitement de la jointure temporelle requérant une série de fonctions ad hoc, nous avons choisi de regrouper celles-ci au sein d'un module dédié, dans le fichier nommé join.cpp. Nous trouvons donc dans celui-ci les fonctions du parseur traitant de la requête de jointure, ainsi que les fonctions de préparation et d'adaptation de la requête initiale à la jointure temporelle, dont le traitement et la complexité sont décrites dans le document [RAMLOT 2000].

Comme pour les autres modules, notre approche est ici relative aux choix d'implémentations et à la structuration du code et non à la réflexion qui a conduit aux algorithmes de traitement.

La question de savoir comment combiner coalescing et jointure a fait l'objet d'une discussion présentée dans [RAMLOT 2000]. Nous avons choisi l'option de réaliser le coalescing avant la jointure, et de restreindre les jointures temporelles aux conditions de jointure, les autres conditions, temporelles ou non, devant être réalisées avant ou après le jointure. L'opérateur de jointure est également limité à deux tables.

Le principe de l'opérateur de jointure consiste donc à réaliser le coalescing sur les deux tables, puis à effectuer la jointure entre ces tables, en y ajoutant les conditions temporelles nécessaires afin de sélectionner les états concernés et d'effectuer la comparaison des intervalles. Pour que cette requête puisse s'exécuter convenablement, des fonctions externes imin et imax, donnant respectivement le minimum et le maximum de 2 entiers, doivent avoir été déclarées.

6.2 Fonctions principales

```
RETCODE Parse_join(SQLHDBC hdbc, elemtree* arbre, tlist* bufNT, char* typ,
                   char* tok)
RETCODE VerifArbre(elemtree* arbre);
RETCODE CreateTableFr(SQLHDBC hdbc, elemtree* arbre, char* user, char* pass,
                      char* DataName, tlist* liste_select);
RETCODE CreateTableTo(SQLHDBC hdbc, elemtree* arbre, char* user, char* pass,
                      char* DataName, tlist* liste_select);
char* CreationRequeteJoin(elemtree* arbre, char* add_select, char* tradjoin);
```

La fonction `Parse_join` est une fonction auxiliaire du module `BuildTree`. Elle a pour objectif de créer la branche «JOIN» (voir chapitre 3.3.7 Construction de la branche de jointure).

La fonction `VerifArbre` vérifie des conditions nécessaires au bon déroulement d'une jointure. Il ne peut en effet y avoir de conditions temporelles et non temporelles, et les colonnes timestamp (*Vstart*, *Vend*, *Tstart*, *Tend*) ne peuvent figurer dans la clause SELECT.

Les fonctions `CreateTableFr` et `CreateTableTo` permettent respectivement la création des tables RDB$JOIN_FR et RDB$JOIN_TO. Ces fonctions assurent la création des tables et leur remplissage avec des données coalescées.

La fonction `CreationRequeteJoin` crée la chaîne de caractères représentant la requête de jointure finale.

6.3 Fonctions auxiliaires

```
char* GetInfoColonne (SQLHDBC hdbc, char* colonne, char* tabfrom, char* tabdest);
tlist* CreationListeSelect(SQLHDBC hdbc, elemtree* arbre);
tlist* GetInfoKey (SQLHDBC hdbc, char* colonnedr,char* colonnegau,
                   elemtree* arbre);
int IdentifyJoinCase(SQLHDBC hdbc, char* tab_fr, char *tab_to, char ktype[2]);
char* SerializeJoin(elemtree* join);
char* TraductionJoin(SQLHDBC hdbc, elemtree* arbre, char* &add_select);
RETCODE SupprimerTables(SQLHDBC hdbc);
```

La fonction GetInfoColonne interroge la table `RDB$TEMP_COLUMN` du dictionnaire temporel T-ODBC afin de donner le nom de la table de la colonne `colonne`.

La fonction CreationListeSelect a pour but de renvoyer une `tlist` d'éléments `coltab`.

```
struct coltab {char* colonne;
               char* table;
              };
```

Ces éléments donnent les noms des colonnes, et de leur table, figurant dans les conditions de jointure et dans la clause SELECT.

La fonction `GetInfoKey` interroge les tables `RDB$TEMP_KEY` et `RDB$TEMP_COLUMN` du dictionnaire temporel T-ODBC afin d'y récupérer des informations relatives aux colonnes `colonnedr` et `colonnegau`. Elle fournit son résultat sous forme de liste chaînée du type générique `tlist`, dont les éléments sont de type `KeyInfo`. Celui-ci est défini comme suit:

```
struct keyInfo {char table[129];
                char Key_type[2];
                char colonne[129];
                char targ_tab[129];
                char targ_col[129];
                char targ_type[2];
               };
```

Les informations répertoriées sont les noms des colonne, les noms des tables et les types des colonnes (temporelle, non temporelle ou identifiante d'entité). Un test est effectué afin de contrôler que les colonnes appartiennent aux tables concernées par la jointure.

La fonction `IdentifyJoinCase` permet de connaître le «type» de requête de jointure qui est demandé. En effet, selon l'aspect temporel des tables et des colonnes intervenant dans la requête de jointure, les conditions temporelles de jointure varient. La fonction renvoie un code identifiant le type de requête (valeur entière attribuée à un littéral par #define).

La fonction `SerializeJoin` est utilisée par `TraductionJoin` et a pour but de recréer une partie de la chaîne de caractères de la jointure finale, sur base des tokens (i.e. noms de colonnes) enregistrés dans l'arbre de la requête, dans la branche *JOIN*. La fonction suppose que les tokens ont été enregistrés dans leur ordre d'apparition dans la requête initiale, et ne vérifie dès lors plus dans les tables `RDB$TEMP_KEY` et `RDB$TEMP_COLUMN` les correspondances de colonnes. Les tokens des sous-branches *table_from* et *table_to* sont placés deux à deux, autour d'un signe «=», dans leur ordre d'apparition dans la liste. Le pointeur retourné est NULL si une erreur est survenue.

La fonction `TraductionJoin` a pour objectif de créer une partie de la requête de jointure finale. En fonction de l'aspect temporel des tables et des colonnes, elle renvoie la chaîne de caractères de la clause SELECT qui mentionne les minima et les maxima des intervalles, ainsi que la chaîne de caractères de la clause WHERE qui mentionne les états pris en considération.

`SupprimerTables` permet de supprimer les tables `RDB$JOIN_FR` et `RDB$JOIN_TO`.

Chapitre 7

Fonctions additionnelles

7.1 Enregistrement de table dans le dictionnaire temporel

7.1.1 Fonction Register

```
bool IsPrimKey(char* col, tlist* prim_key_list);
char* getType(short int SQLType);
SQLRETURN Register (SQLHDBC hdbc, elemtree* arbre);
```

La fonction C `register` implémente la commande mini-TSQL du même nom, et réalise donc l'enregistrement d'une table dans le dictionnaire temporel T-ODBC.

Pour ce faire, plusieurs opérations sont nécessaires. La fonction est dès lors divisée en deux étapes: la première est celle de la collecte d'informations, et la seconde celle de l'enregistrement de données dans le dictionnaire temporel.

En ce qui concerne la collecte d'information, il s'agit tout d'abord d'obtenir la description de la table visée à partir des métadonnées du SGBD, à l'aide de la fonction ODBC SQLGetInfo, dont le prototype est le suivant:

```
SQLRETURN SQLGetInfo(SQLHDBC ConnectionHandle, SQLUSMALLINT InfoType,
                     SQLPOINTER InfoValuePtr, SQLSMALLINT BufferLength,
                      SQLSMALLINT * StringLengthPtr);
```

Nous devons également connaître l'identifiant d'entité de la table à enregistrer, afin de mettre ces informations dans la table `RDB$TEMP_KEY`. Ceci peut s'effectuer à l'aide de la fonction `SQLPrimaryKeys`, dont le prototype est le suivant:

```
SQLRETURN SQLPrimaryKeys(SQLHSTMT StatementHandle, SQLCHAR * CatalogName,
                         SQLSMALLINT NameLength1, SQLCHAR * SchemaName,
                         SQLSMALLINT NameLength2, SQLCHAR * TableName,
                          SQLSMALLINT NameLength3);
```

La fonction Register requiert des informations sur la base de données qui ne sont pas accessibles autrement que par interrogation de l'utilisateur, d'autant que ces paramètres sont optionnels: leur présence dépend en fait du SGDB utilisé. Il s'agit du nom de schéma, catalogue ou utilisateur. Là où Access utilise le nom de catalogue comme référence pour la base de données, Oracle utilise le nom de schéma, tandis qu'Interbase ne demande aucun de ces paramètres.

Une autre fonction ODBC nous permet de connaître les colonnes de la table à enregistrer, ainsi que leur type, longueur, longueur décimale, etc: ceci est nécessaire pour garnir la table `RDB$TEMP_COLUMN`, et peut se faire à l'aide de la fonction ODBC `SQLColumns`, dont le prototype est le suivant:

```
SQLRETURN SQLColumns(SQLHSTMT StatementHandle, SQLCHAR * CatalogName,
                     SQLSMALLINT NameLength1, SQLCHAR * SchemaName,
                     SQLSMALLINT NameLength2, SQLCHAR * TableName,
                     SQLSMALLINT NameLength3, SQLCHAR * ColumnName,
                      SQLSMALLINT NameLength4);
```

Le type temporel de la table est déterminé par la présence des colonnes temporelles dont le nom doit suivre la nomenclature standard utilisée par l'API. La présence de la seule colonne Vstart fait reconnaître la table comme monotemporelle en valid time, de même que la seule colonne Tstart fait de la table une table monotemporelle en transaction time. La présence de ces deux colonnes mènera à l'enregistrement de la table comme étant bitemporelle.

Les fonctions auxiliaires `IsPrimKey` et `getType` sont utilisées pour, respectivement, savoir si une colonne fait partie de l'identifiant d'entité de la table à enregistrer, et pour obtenir le littéral correspondant à un type SQL connu sous forme d'entier (selon les définitions de sql.h)

7.1.2 Fonction Unregister

`SQLRETURN Unregister(SQLHDBC hdbc, elemtree *arbre)`

L'implémentation de la fonction unregister est plus aisée que celle de son contraire. Il s'agit ici en effet d'une simple suppression de toute référence à la table dans le dictionnaire temporel. La fonction est donc constituée de l'enchaînement de trois requêtes «`DELETE`», travaillant respectivement sur `RDB$TEMP_TABLE`, `RDB$TEMP_COLUMN` et `RDB$TEMP_KEY`.

Remarque importante, la fonction ne garantit pas la cohérence de son exécution: un échec sur le troisième `DELETE` par exemple entraînera certes le retour du code `SQL_ERROR`, mais nullement l'annulation des deux premières requêtes. Nous avons choisi cette approche afin de laisser l'entier contrôle des transactions au programmeur final. A charge pour lui de contrôler le code de retour de la fonction, et de prendre les mesures adéquates.

7.2 Opérateur de normalisation

L'opérateur de normalisation a pour but de vérifier le caractère normalisé d'une table temporelle (*cfr. document TimeStamp Normalisation*). Cette vérification porte sur quatre aspects de la normalisation, à savoir l'absence d'états incohérents dans l'historique d'une table (erreurs de type 1), l'absence d'états non-disjoints identiques (erreurs de type 2), l'absence d'états jointifs identiques (erreurs de type 3) et l'absence d'états manquants (erreurs de type 4).

Normaliser une table suppose dès lors deux actions: supprimer les incohérences et compléter les ruptures de séquence historique. Cette dernière action peut se faire à l'aide de la valeur NULL ou de valeurs par défaut.

Deux fonctions de l'opérateur de normalisation sont implémentées. La première se contente de signaler les erreurs (mode «CHECK»), tandis que la deuxième crée une table corrigée, normalisée (mode «CREATE»). Dans les deux cas, si des erreurs de normalisation sont détectées, la fonction crée une table de même nom que la table d'origine, mais suffixée par `_NORMAL_ERRORS`, dans laquelle apparaîtront le type d'erreurs et les lignes en conflit ou les trous.

Le code de retour de la fonction est `SQL_SUCCESS_WITH_INFO`. Si aucune erreur de normalisation n'est détectée, la fonction retourne `SQL_SUCCESS`. Le code `SQL_ERROR` correspond quant à lui à une erreur d'exécution, et `TSQLGetError` permet d'obtenir de plus amples informations.

La détection d'états incohérents se fait à l'aide de requêtes SQL standard, par auto-jointure, avec un tri sur les identifiants d'entité. Suivant le type de temporalité :

```
select T1.<id d'entité>,
       T1.vstart,T1.vend,T1.tstart,T1.tend,T2.vstart,T2.vend,T2.tstart,T2.tend
from <table> T1, <table> T2
where T1.<id d'entité> = T2.<id d'entité>
  and not(T1.<autres colonnes non temporelles> =
          T2.<autres colonnes non temporelles>)
  and T1.tstart<T2.tend and T2.tstart<T1.tend
  and T1.vstart<T2.vend and T2.vstart<T1.vend
order by T1.<id d'entité>
```

ou

```
select T1.<id d'entité>,
       T1.vstart,T1.vend,T2.vstart,T2.vend
from <table> T1, <table> T2
where T1.<id d'entité> = T2.<id d'entité>
  and not (T1.<autres colonnes non temporelles> =
           T2.<autres colonnes non temporelles>)
  and T1.vstart<T2.vend and T2.vstart<T1.vend
order by T1.<id d'entité>
```

ou

```
select T1.<id d'entité>,
       T1.tstart,T1.tend,T2.tstart,T2.tend
from <table> T1, <table> T2
where T1.<id d'entité> = T2.<id d'entité>
  and not (T1.<autres colonnes non temporelles> =
           T2.<autres colonnes non temporelles>)
  and T1.tstart<T2.tend and T2.tstart<T1.tend
order by T1.<id d'entité>
```

La requête SQL produit son résultat «doublé» (si l'état X de l'entité A entre en conflit avec l'état Y de la même entité, alors l'état Y entre en conflit avec l'état X). Pour corriger ce problème, les erreurs détectées sont temporairement gardées en mémoire centrale. Au moment de la détection d'une erreur, on vérifie si sa réflexive n'est pas déjà en mémoire centrale. Si oui, on ne fait rien, sinon, on l'enregistre dans la table d'erreurs et on la rajoute à la liste en mémoire centrale. Puisque l'auto-jointure est triée sur l'identifiant d'entité, les erreurs le seront également. Donc, toutes les erreurs d'une même entité apparaîtront successivement. La liste en mémoire centrale peut donc être vidée à chaque changement d'entité.

Ces requêtes sont prises en charges par la procédure `TSNormalize_overlaps()`.

La détection d'états non-disjoints identiques se fait avec la même procédure `TSNormalize_overlaps`() (un paramètre booléen `same_value` indique quelle forme choisir) mais avec une variante dans la forme des requêtes :

```
select T1.<id d'entité>,
       T1.vstart,T1.vend,T1.tstart,T1.tend,T2.vstart,T2.vend,T2.tstart,T2.tend
from <table> T1, <table> T2
where T1.<id d'entité> = T2.<id d'entité>
  and T1.<autres colonnes non temporelles> = T2.<autres colonnes non temporelles>
  and not(T1.vstart=T2.vstart and T1.vend=T2.vend
          and T1.tstart=T2.tstart and T1.tend=T2.tend)
  and T1.tstart<T2.tend and T2.tstart<T1.tend
  and T1.vstart<T2.vend and T2.vstart<T1.vend
order by T1.<id d'entité>
```

ou

```
select T1.<id d'entité>,T1.vstart,T1.vend,T2.vstart,T2.vend
from <table> T1, <table> T2
where T1.<id d'entité> = T2.<id d'entité>
  and T1.<autres colonnes non temporelles> = T2.<autres colonnes non temporelles>
  and not(T1.vstart=T2.vstart and T1.vend=T2.vend)
  and T1.vstart<T2.vend and T2.vstart<T1.vend
order by T1.<id d'entité>
```

ou

```
select T1.<id d'entité>,T1.tstart,T1.tend,T2.tstart,T2.tend
from <table> T1, <table> T2
where T1.<id d'entité> = T2.<id d'entité>
  and T1.<autres colonnes non temporelles> = T2.<autres colonnes non temporelles>
  and not(T1.tstart=T2.tstart and T1.tend=T2.tend)
  and T1.tstart<T2.tend and T2.tstart<T1.tend
order by T1.<id d'entité>
```

La détection d'états jointifs est aussi réalisée à l'aide d'une requête SQL, tel qu'implémenté dans la procédure `TSNormalize_neighbour`(...) :

```
Insert into <table>_NORM_ERRORS
select 3,T1.<id d'entité>,
       T1.vstart,T1.vend,T1.tstart,T1.tend,T2.vstart,T2.vend,T2.tstart,T2.tend
from <table> T1, <table> T2
where T1.<id d'entité> = T2.<id d'entité>
  and T1.<autres colonnes non temporelles> = T2.<autres colonnes non temporelles>
  and ((T1.tstart<T2.tend and T2.tstart<T1.tend and T1.vend=T2.vstart)
       or(T1.tend=T2.tstart and T1.vstart=T2.vstart and T1.vend=T2.vend))
```

ou

```
Insert into <table>_NORM_ERRORS
select 3,T1.<id d'entité>,T1.vstart,T1.vend,T2.vstart,T2.vend
from <table> T1, <table> T2
where T1.<id d'entité> = T2.<id d'entité>
  and T1.<autres colonnes non temporelles> = T2.<autres colonnes non temporelles>
  and T1.vend=T2.vstart
```

ou

```
Insert into <table>_NORM_ERRORS
select 3,T1.<id d'entité>,T1.tstart,T1.tend,T2.tstart,T2.tend
from <table> T1, <table> T2
where T1.<id d'entité> = T2.<id d'entité>
  and T1.<autres colonnes non temporelles> = T2.<autres colonnes non temporelles>
  and T1.tend=T2.tstart
```

où "3", dans la clause `select`, est le type d'erreur.

Remarquons que ces requêtes ne sont pas symétriques au niveau de la clause `where` contrairement aux requêtes de détection d'états non-disjoints. Par conséquent, les résultats n'apparaissent plus en double et l'insertion dans la table résultat peut être opérée directement grâce au `Insert into`.

Le traitement des interruptions dans un historique à «trous» est procédural. En monotemporel, repérer une telle rupture de séquence temporelle se fait avec l'algorithme suivant, pour une table monotemporelle en valid time (la transformation pour l'équivalent en transaction time est immédiate) :

```
EXEC SQL select <id d'entité>,vstart,vend
         from <table>
         order by <id d'entité>,vstart,vend
idref = {}
EXEC SQL fetch into :r
while not end-of-file do
  if  not idref=r.id
    idref = r.id
    veref = r.vend
  end-if
  if veref>=r.vstart
    if veref<r.vend
      veref = r.vend
    end-if
  else
    trous = trous ∪ {(idref,veref,r.vstart)}
    veref = r.vend
  end-if
  EXEC SQL fetch into :r
end-while
```

Le cas bitemporel est plus complexe, devant traiter ensemble les deux dimensions. Il y a rupture de séquence bitemporelle lorsque le transaction time est interrompu, ou lorsque, à l'intérieur d'un même intervalle transaction time, il n'y a pas de continuité valid time. En voici l'algorithme :

```
EXEC SQL select <id d'entité>,vstart,vend,tstart,tend
         from <table>
         order by <id d'entité>,tstart
EXEC SQL fetch into :r
while not end-of-file do
  idref = r.id
  ligne = {}
  tref = r.tstart
  while not end-of-file and r.id=idref do
    tsref = teref
    while not end-of-file and r.tstart=tsref do
      ligne = ligne ∪ {(r.vstart,r.vend,r.tstart,r.tend)}
      EXEC SQL fetch into :r
    end-while
```

```
      if not end-of-file
        teref = min ({l.tend|l∈ligne}∪{r.tstart})
      else
        teref = min {l.tend|l∈ligne}
      end-if
      do normalisation-mono-valid(idref,ligne,tsref,teref,trous)
      ligne = ligne \ {l∈ligne|l.tend=teref}
      if (not end-fo-file and r.tstart>teref and ligne={})
        trous = trous ∪ {(idref,0,999,teref,r.tstart)}
      end-if
    end-while
    while not ligne={}
      tsref = teref
      teref = min{l.tend|l∈ligne}
      do normalisation-mono-valid(idref,ligne,tsref,teref,trous)
      ligne = ligne \ {l∈ligne|l.tend=teref}
    end-while
  end-while
  do coalescing(trous)
```

La procédure `coalescing`() est celle du chapitre 4. La procédure `normalisation-mono-valid`() est la suivante :

```
do tri(ligne)
r = premier(ligne)
veref = r.tend
while not end-of-list do
  if veref>=r.vstart
    if veref<r.vend
      veref = r.vend
    end-if
  else
    trous = trous ∪ {(idref,veref,r.vstart,tsref,teref)}
    veref = r.vend
  end-if
  r = suivant(ligne)
end-while
```

Dans les fichiers sources de T-ODBC, ces deux algorithmes sont intégrés dans la procédure `TSNormalize_holes`(...). La fonction de tri est `sort_ligne`() qui ordonne les cellules de la liste suivant l'ordre du champs vstart de chacune des cellules.

Globalement, la normalisation est prise en charge par la fonction `TSNormalize`(...) qui fait appel à une première fonction `TSNormalize_init`(...) pour initialiser toute une série de structures de données, puis à chaque des trois procédures précitées selon les besoins.

Chapitre 8

Le traitement d'erreur

Afin de permettre à l'utilisateur de l'API de récupérer les informations relatives à une erreur survenue lors de l'utilisation de l'API T-ODBC, il est apparu nécessaire de créer une fonction `TSQLGetError`. Le but de cette fonction n'est pas de mimer le comportement des fonctions ODBC de contrôle d'erreur que sont `SQLGetDiagRec` et `SQLGetDiagField`, mais de fournir à l'utilisateur une méthode simple pour obtenir les informations d'erreur. Celles-ci pourront soit être spécifiques à T-ODBC (relatives par exemple à la syntaxe mini-TSQL ou à une erreur de traitement dans l'un des modules), ou provenir d'informations d'erreur de fonctions ODBC standard utilisées au sein des divers modules de l'API.

La fonction `TSQLGetError` utilise un descripteur d'erreur, implémenté sous forme d'une liste de structures C, (tlist de struct) afin de fournir les informations voulues.

Cette liste aura au préalable été garnie par l'API lors de la survenance d'une erreur, fatale ou non.

Les erreurs internes à l'API sont traitées à l'aide des fonctions standard ODBC que sont `SQLGetDiagField` et `SQLGetDiagRec`. Dès lors, l'usage de ces dernières devra être évité après un appel à une fonction T-ODBC, l'appel aux fonctions de contrôle d'erreur ODBC étant destructeur (i.e., les informations une fois récupérées sont détruites).

Le descripteur d'erreurs T-ODBC enregistrera également toute erreur survenue au sein de l'API et considérée comme non fatale (code de retour `SQL_SUCCESS_WITH_INFO`). Dès lors, il faut prévoir un mécanisme permettant de savoir quel est le nombre d'éléments disponibles dans ce descripteur, puisque de nombreuses erreurs peuvent y être enregistrées. Il faut également savoir que ODBC peut générer plusieurs informations relatives à une même cause d'erreur, ce qui augmente également le nombre de champs à récupérer pour obtenir la totalité de l'information disponible. Une méthode simple est d'utiliser l'identificateur de champ 0 pour récupérer cette information. Nous restons ainsi cohérents avec ODBC qui numérote les informations à partir de 1.

Un élément du descripteur d'erreur comporte plusieurs champs, utiles pour l'analyse de l'erreur :

- le code d'erreur `SQLSTATE`

- le code d'erreur natif

- un message d'erreur explicite

- le nom de la fonction ayant provoqué l'erreur

Tirant parti de la flexibilité des codes d'erreur SQLSTATE, nous introduisons une gamme de code de classe «TODBC», dont la sous-classe donne le module où l'erreur est survenue. Cette information sera essentiellement utile pour la maintenance du code de l'API.

Les classes d'erreur SQLSTATE définies par la documentation ODBC sont les suivantes :

01, 07, 08, 21, 22, 23, 24, 25, 28, 34, 3C, 3D, 3F, 40, 42, 44,

IM : erreur du Driver Manager ODBC

HY : erreur du driver ou de la source de données, ODBC 3.x

SI : erreur du driver ou de la source de données, ODBC 2.0 et antérieur

Nous utilisons le code **TD** comme code de classe pour une erreur spécifique à l'API T-ODBC. Tous les codes de T-ODBC sont listés dans le manuel du programmeur.

Pour traiter ses erreurs, l'API T-ODBC dispose de trois fonctions :

- `InfoErreurODBC` dont le rôle est de faire les appels nécessaires à `SQLGetDiagField` et `SQLGetDiagRec`. Il est à noter que ces deux dernières fonctions de l'API ODBC ne sont présentes qu'à partir de la version 3.0.

- `InfoErreurTODBC` dont le rôle est de garnir le descripteur d'erreurs d'un SQLSTATE et d'un message explicite propres à l'API T-ODBC.

- `TSQLClearErrorList`() dont le rôle est de vider la liste des erreurs. Cette fonction est appelée lors de chaque appel à `TSQLExecDirect`(), `TSQLAllocStmt`() et `TSQLFreeStmt`().

`InfoErreurODBC` garnit le descripteur d'erreur TODBC sur base des informations de `SQLGetDiagField` et `SQLGetDiagRec`. Ces fonctions peuvent elles-mêmes se terminer sur une condition d'erreur, ce qui est considéré comme erreur fatale au sein de l'API T-ODBC : `InfoErreurODBC` retourne alors `SQL_ERROR` et entraîne la fin des traitements de l'API. Dans ce cas, `TSQLGetError` peut être utilisée pour savoir quelle était la condition de terminaison de `InfoErreurODBC` : cette information est disponible sous forme de message explicite dans l'élément 0, champ Info. Il est à noter que, comme InfoErreurODBC enregistre également des erreurs non fatales, le champ ElemNb de l'élément 0 peut être non nul.

Les prototypes des fonctions internes sont les suivants:

```
RETCODE InfoErreurODBC(SQLSMALLINT HandleType, SQLHANDLE Handle, char* fonction)
RETCODE InfoErreurTODBC (char State[6], char* Message,char* Fonction)
void TSQLClearErrorList()
```

L'interface utilisateur est constituée par la fonction `TSQLGetError`, décrite au chapitre 2, les quatre fonctions (trois internes et l'interface) sont regroupées au sein du module `gestion_erreurs.cpp`.

Les types utilisés dans le module de gestion d'erreur sont les suivants:

```
typedef struct err_struct_tag {SQLCHAR SQLState[6];
                               SQLINTEGER NativeError;
                               SQLCHAR* ErrorMessage;
                               SQLCHAR* Function;
                               SQLCHAR* InfoSup;
                               } err_struct;


typedef struct err_struct_init_tag {SQLINTEGER ElemNb;
                                    SQLCHAR* Info;
                                    } err_struct_init;
tlist* ErrDesc=NULL;
```

Le descripteur d'erreur est constitué de la tlist pointée par `ErrDesc`, dont le premier élément est un pointeur vers une structure de type `err_struct_init`, les éléments suivants étant des pointeurs vers un structure de type `err_struct`.

Chapitre 9

Fonctions auxiliaires

9.1 Gestion des listes

La programmation de l'API T-ODBC fait un usage important des listes chaînées génériques et dispose dans sa bibliothèque de fonctions de gestion de listes.

Les listes utilisées par l'API sont de structure classique, simplement chaînée. Les éléments de ces listes sont constitués de structures à deux champs, le premier étant un pointeur vers `void` destiné à accueillir l'élément à stocker, le second champ étant un pointeur vers l'élément suivant. Particularité de ces listes, elles se terminent par une structure dont les deux pointeurs sont `NULL`.

Les fonctions de cette bibliothèque permettent de créer et détruire la liste, ainsi que d'y ajouter un élément. L'élément ajouté vient se placer en fin de liste. Par ailleurs, nous disposons également d'une fonction créant la structure utilisée dans l'arbre représentant la requête mini-TSQL (type elemtree*, structure à trois champs), ainsi que d'une fonction de recherche de cellule dans l'arbre.

Types utilisés:

```
struct tlist{
             void* elem;
             tlist* next;
            }
struct elemtree {
                 char type[30];
                 char value[30];
                 tlist* sons;
                }
```

Prototypes des fonctions de gestion de liste:

```
tlist* tlist_new (); //création d'une liste vide
void tlist_set (tlist *L, void *v); //ajout d'un élément à la fin de la liste
void tlist_delete(tlist *L); //destruction de la liste avec tous ses éléments
void tlist_eltr_delete(tlist* L); //destruction d'une liste d'elemtree
void tlist_non_destructive_delete(tlist* L); //destruction d'une liste en
                                              //conservant ses éléments
void tlist_empty(tlist* L); //destruction des éléments d'une liste
void tlist_one_del(tlist* &L, tlist *prev, tlist* &del); //destruction d'un
                                                          //élément d'une liste
void tlist_delete_tab(tlist *L, int n); //destruction d'une liste de tableaux de
                                        // n strings
elemtree* elemtree_new(char* typ, char* val); //création d'un elemtree
void elemtree_update_val(elemtree* e, char* val); //modification de la valeur d'un
                                                  //elemtree
void delete_tree (elemtree* e); //destruction d'un elemtree
void elemtree_swap(elemtree* e1, elemtree* e2); //inversion des valeurs et type
                                                //de deux elemtree
elemtree* findpoint(elemtree* e, char* search, int number); //recherche du
          //number-ième elemtree dont le type est "search" dans un arbre dont
          //la racine est l'elemtree e et dont les branches sont des listes
          // d'elemtree.
```

9.2 Mise en minuscules d'un string:

```
char* lower(char* input)
```

La fonction retourne le string passé en argument (input) mis en minuscules. Cette fonction est utile afin de rendre les requêtes envoyées à TSQLExecDirect non sensibles à la casse en ce qui concerne notamment des noms de table ou de colonne. Un problème se pose en effet en ce qui concerne les requêtes internes à l'API, notamment lors de l'accès au dictionnaire temporel. Les requêtes SQL sont alors sensibles à la casse, ce qui n'est pas désirable puisque ne correspondant pas au comportement normal: un nom de colonne ou de table doit pouvoir être reconnu quelle qu'en soit la casse. Nous choisissons dès lors d'utiliser systématiquement des minuscules. Notons que cette fonction alloue de la mémoire pour le string résultat avec un `malloc`(); ce string devra donc être libéré avec un `free`().

9.3 Vérification du caractère temporel d'une table:

```
RETCODE VerifTabTemp (SQLHDBC hdbc, char* table, char* type)
```

Par interrogation du dictionnaire temporel (table `RDB$TEMP_TABLE`), la fonction vérifie qu'une table (paramètre `table`) est temporelle et retourne (paramètre `type`) sa dimension temporelle: monotemporelle en valid time ou transaction time, ou bitemporelle, telle qu'enregistrée dans le dictionnaire. Si la table n'y est pas enregistrée, alors le type retourné est NT.

La valeur de la fonction est `SQL_SUCCESS` si l'exécution s'est déroulée correctement, et `SQL_ERROR` sinon.

9.4 Obtenir les informations relatives aux colonnes d'une table

```
SQLRETURN info_col (SQLHDBC hdbc, char* tab, Tinfo_tab* info)
```

La fonction info_col permet d'obtenir le type et la longueur des colonnes d'une table donnée. On peut ainsi allouer plus précisément les tampons utilisés par la fonction `SQLBindCol`.

Le principe de info_col est de faire appel à la fonction standard ODBC `SQLColumns` qui donne toute une série d'informations sur les colonnes d'une table. Nous nous intéressons ici au nom, au type, et à la longueur de ce type, des colonnes. Dans l'ensemble de résultats de `SQLColumns`, nous ne récupérons donc que les colonnes numérotées 4, 7 et 9.

Le code de retour de `info_col` est `SQL_SUCCESS` si tout s'est bien passé, `SQL_ERROR` sinon.

La fonction fait usage d'un type global afin de permettre l'utilisation des informations qu'elle obtient dans l'ensemble du code de l'API. Ce type est défini comme suit:

```
typedef struct {char* table;
                char* col;
                int size;
                int dec_size;
                } info_tab;
typedef tlist Tinfo_tab;
```

9.5 Convertir un littéral représentant un type SQL en valeur numérique

```
short Sql_CharToShort (char* sqltype)
```

La fonction Sql_CharToShort a pour but de convertir en valeur de type short un littéral (char* sqltype) de même sémantique. Ceci est utile pour les fonctions ODBC dont le paramètre représentant le type SQL est de type short, alors que le type enregistré dans le dictionnaire temporel T-ODBC est un littéral. La compréhension du code de cette fonction est immédiate, puisque il ne s'agit en fait que d'un unique switch réalisant le mapping string <-> valeur entière. La fonction utilise les définitions littérales des valeur entières.

Chapitre 10

Remarques générales

10.1 Liste des fichiers composant la *dll* T-ODBC

Voici une liste des fichiers de T-ODBC:

- normal.cpp : les fonctions du traitement de la normalisation
- buildtree.cpp: le parseur, construction de l'arbre de la requête, sauf traitement d'une condition de jointure
- buildstring.cpp: écriture de l'arbre de la requête sous forme de string en SQL
- fct_aux.cpp: diverses fonctions auxiliaires (lower, VerifTabTemp, Infocol, SQLCharToShort)
- tsql.cpp: les fonctions «TSQL», donc le fichier «coeur» de la dll
- module_coal.cpp : les fonctions du coalescing, en conjonction avec
- tsqlex.cpp et odbcfc.cpp
- module_agreg.cpp: traitement de l'agrégation temporelle
- gestion_erreurs.cpp: les fonctions InfoErreurODBC, InfoErreurTODBC et TSQLGetError
- reg_unreg.cpp : les fonctions d'enregistrement/suppression d'une table dans le dict. temporel
- join.cpp: fonctions spécifiques au traitement de la jointure:
- main.c: les fonctions readtoken, readtokenfirst et readclose
- tree.cpp: bibliothèque des fonctions de gestion de l'arbre et des listes chaînées (tlist_new, tlist_delete, elemtree_new etc)
- lex.yy.x : le fichier généré par FLEX
- myodbc32.lib

10.2 Les #define globaux

- SQL_NOT_TSQL=-5;
- TSQL_GROUPBY=-6;
- lgr_maxi=51; --> attention, la largeur maxi d'une colonne en nombres de caractères !!!
- SQL_TSQL_JOIN=-6;
- DEBUG; --> déclenche la création du fichier trace.txt !!!
- TSQL_UNREG=-7;
- TSQL_REG=-8;
- TSQL_NORMAL=-9;
- SQL_TSQL=-10;
- SQL_NOT_TEMP=-11;

Chapitre 11

Annexes

1.[RAMLOT 2000] Oliver RAMLOT, Contribution à la mise au point d'un langage d'accès aux bases de données temporelles - Mémoire de fin d'études, FUNDP, 2000, TimeStamp (Volume 3 : Documents techniques - Première partie : Rapports techniques de recherche - L'exploitation des données).

2.[HAINAUT 2002] Jean-Luc HAINAUT, Introduction pratique aux bases de données temporelles, 2002, TimeStamp (Volume 1 : Introduction aux bases de données temporelles - Tutoriels).

TimeStamp

Volume 3 - Documents techniques

Deuxième partie

Documentations techniques des outils

Les outils d'exploitation des bases de données temporelles

DB-Main Manual Series

Projet TimeStamp

PETIT PLAN DE TEST DE T-ODBC

DIDIER ROLAND, VIRGINIE DETIENNE

SEPTEMBRE 2002

The University of Namur - LIBD
Institut d'Informatique
Rue Grandgagnage, 21 B-5000 Namur
http://www.info.fundp.ac.be/libd

DB-Main Manual Series

Petit plan de test de TODBC

Didier ROLAND, Virginie DETIENNE

Septembre 2002

The University of Namur - LIBD

Introduction

Ce document reprend de manière brutte toute une série de requêtes mini-TSQL2 qui ont été soumises à TODBC, et qui sont livrées avec leur résultat. Cette liste se veut assez large, sans être tout à fait exhaustive, quand à la couverture des possibilités de TODBC et devraient permettre de tester la presque totalité de ce module. Ainsi, toute évolution future de TODBC ne devrait en rien altérer ces résultats.

Cette liste peut également être vue comme un catalogue d'exmples en complément au manuel de l'utilisateur.

Tables de test n°1

Tableau 1: Employe

MATRICULE	NOM	ADRESSE /b	SALAIRE /b	SERVICE	VSTART	VEND	TSTART	TEND
BRA	Brassens	Liege	90000	INFO	1	9999999	50	105
SOU	Souchon	Bruxelles	80000	PERS	25	9999999	55	9999999
BRE	Brel	Bruxelles	90000	COMP	10	9999999	60	9999999
CAB	Cabrel	Namur	60000	REMU	25	9999999	65	9999999
SHE	Sheller	Mons	50000	COMP	30	9999999	70	9999999
BRA	Brassens	Liege	90000	INFO	1	15	105	125
BRA	Brassens	Mons	90000	INFO	15	9999999	105	110
BRA	Brassens	Mons	90000	INFO	15	23	110	125
BRA	Brassens	Liege	90000	INFO	23	9999999	110	115
BRA	Brassens	Liege	90000	INFO	23	35	115	125
BRA	Brassens	Namur	90000	INFO	35	9999999	115	120
BRA	Brassens	Namur	90000	INFO	35	50	120	130
BRA	Brassens	Liege	90000	INFO	50	9999999	120	9999999
BRA	Brassens	Liege	90000	INFO	1	35	125	130
BRA	Brassens	Liege	90000	INFO	1	22	130	9999999
BRA	Brassens	Namur	90000	INFO	22	50	130	9999999

Tableau 2: Service

NUMERO	NOM /b	DIRECTEUR /b	BUDGET /b	SECRETAIRE /b	VSTART	VEND	TSTART	TEND
INFO	Informatique		1000000	Simon	1	9999999	30	75
PERS	Personnel		800000	Muller	20	9999999	35	85
COMP	Comptabilite		1000000	Duchemin	30	9999999	40	100
REMU	Remunerations		200000	Lepage	20	9999999	45	95
INFO	Informatique		1000000	Simon	1	5	75	9999999
INFO	Informatique	BRA	1000000	Simon	5	9999999	75	9999999
PERS	Personnel		800000	Muller	20	30	85	9999999
PERS	Personnel	BRE	800000	Muller	30	9999999	85	9999999
REMU	Remunerations		200000	Lepage	20	25	95	9999999
REMU	Remunerations	CAB	200000	Lepage	25	9999999	95	9999999
COMP	Comptabilite		1000000	Duchemin	30	35	100	9999999
COMP	Comptabilite	SHE	1000000	Duchemin	35	9999999	100	9999999

Tableau 3: Projet

NUMERO	NBREJOURS /v	BUDGET /v	DATEDEBUT	RESPONSABLE /v	CLIENT	VSTART	VEND
P5	30	80000	20	BRA	C012	30	9999999
P1	50	100000	40	BRE	C011	10	80
P1	50	100000	40	SHE	C011	80	90
P1	50	100000	40	BRE	C011	90	9999999

Tableau 4: Client

CODE
C011
C012
C013
C014
C015

Tableau 5: Employe2

MATRICULE	NOM	ADRESSE /b	SALAIRE /b	SERVICE	VSTART	VEND	TSTART	TEND
BRA	Brassens	Liege	90000	INFO	1	9999999	50	***100***
SOU	Souchon	Bruxelles	80000	PERS	25	9999999	55	9999999
BRE	Brel	Bruxelles		COMP	10	9999999	60	9999999
CAB	Cabrel	Namur	60000	REMU	25	9999999	65	9999999
SHE	Sheller	Mons	50000	COMP	30	9999999	70	9999999
BRA	Brassens	Liege	90000	INFO	1	15	105	125
BRA	Brassens	Mons	90000	INFO	15	9999999	105	110
BRA	Brassens	***Liege***	90000	INFO	15	23	***114***	125
BRA	Brassens	Liege	90000	INFO	23	9999999	***108***	115
BRA	Brassens	Liege	90000	INFO	23	35	115	***127***
BRA	Brassens	Namur	90000	INFO	35	9999999	115	***118***
BRA	Brassens	Namur	90000	INFO	***38***	50	120	130
BRA	Brassens	Liege	90000	INFO	50	9999999	120	9999999
BRA	Brassens	Liege	90000	INFO	1	35	125	130
BRA	Brassens	Liege	90000	INFO	1	***20***	130	9999999
BRA	Brassens	Namur	90000	INFO	22	***53***	130	9999999

Requêtes test n°1

select numero from projet

```
P1        10        9999999
P5        30        9999999
```

select nom from employe

```
Brassens              1       9999999   50        9999999
Brel                  10      9999999   60        9999999
Cabrel                25      9999999   65        9999999
Sheller               30      9999999   70        9999999
Souchon               25      9999999   55        9999999
```

select code from client

```
C011
C012
C013
C014
C015
```

select nom from employe where matricule>’C’

```
Cabrel              25      9999999  65       9999999
Sheller             30      9999999  70       9999999
Souchon             25      9999999  55       9999999
```

select nom from employe where matricule>%C

```
SQLExecDirect: HY000 - 911
[Oracle][ODBC][Ora]ORA-00911: invalid character
```

select toto from employe

```
TSQLExecDirect: TDC12
Unknown column name
```

select nom from toto

```
TSQLExecDirect: TDP07
Unknown table name
```

select nom,salaire from employe

```
Brassens            90000   1        9999999  50       9999999
Brel                90000   10       9999999  60       9999999
Cabrel              60000   25       9999999  65       9999999
Sheller             50000   30       9999999  70       9999999
Souchon             80000   25       9999999  55       9999999
```

select nom salaire from employe

```
TSQLExecDirect: TDPA6
Syntax error, ',' or 'from' expected
```

select nom from employe where timepoint’20’ before valid(employe)

```
Cabrel              25      9999999  65       9999999
Sheller             30      9999999  70       9999999
Souchon             25      9999999  55       9999999
```

select nom from employe where period’[10,25)’ before valid(employe)

```
Cabrel              25      9999999  65       9999999
Sheller             30      9999999  70       9999999
Souchon             25      9999999  55       9999999
```

select nom from employe where period’10,25’ before valid(employe)

```
TSQLExecDirect: TDP05
Invalid time period
```

select vstart,nom from employe

```
1         Brassens           9999999 50       9999999
10        Brel               9999999 60       9999999
25        Cabrel             9999999 65       9999999
30        Sheller            9999999 70       9999999
25        Souchon            9999999 55       9999999
```

select employe.nom from employe

```
Brassens            1       9999999  50       9999999
Brel                10      9999999  60       9999999
Cabrel              25      9999999  65       9999999
Sheller             30      9999999  70       9999999
Souchon             25      9999999  55       9999999
```

select nom,avg(salaire) from valid(employe) group by nom

```
Brassens            90000   1        9999999
Brel                90000   10       9999999
Cabrel              60000   25       9999999
Sheller             50000   30       9999999
Souchon             80000   25       9999999
```

select nom,avg(salaire) from valid(employe) group by matricule,nom

```
Sheller             50000   SHE      30      9999999
Cabrel              60000   CAB      25      9999999
Souchon             80000   SOU      25      9999999
Brassens            90000   BRA      1       9999999
Brel                90000   BRE      10      9999999
```

select nom,vstart,avg(salaire) from valid(employe) group by nom,vstart

```
Sheller             30        50000     9999999
Cabrel              25        60000     9999999
Souchon             25        80000     9999999
Brassens            1         90000     9999999
Brel                10        90000     9999999
```

select nom,vstart,avg(salaire) from valid(employe) group by vstart,nom

```
Sheller             30        50000     9999999
Cabrel              25        60000     9999999
Souchon             25        80000     9999999
Brassens            1         90000     9999999
Brel                10        90000     9999999
```

select employe.nom,avg(salaire) from valid(employe) group by nom

```
Sheller             50000     30        9999999
Cabrel              60000     25        9999999
Souchon             80000     25        9999999
Brassens            90000     1         9999999
Brel                90000     10        9999999
```

select nom,avg(employe.salaire) from valid(employe) group by nom

```
TSQLExecDirect: TDPA9
Syntax error, ')' expected
```

select nom,matricule,avg(salaire) from valid(employe) group by nom

```
TSQLExecDirect: TDPA0
Column in "select" not in "group by"
```

select nom,avg(salaire) from employe group by nom

```
TSQLExecDirect: TDPF3
Temporal dimension not specified for bitemporal table
```

select nom,min(vstart) from valid(employe) group by nom

```
TSQLExecDirect: TDPA3
Temporal column not allowed in aggregation function
```

select numero,count(directeur) from valid(service) group by numero

```
COMP      0         30        35
INFO      0         1         5
PERS      0         20        30
REMU      0         20        25
COMP      1         35        9999999
INFO      1         5         9999999
PERS      1         30        9999999
REMU      1         25        9999999
```

select directeur,avg(budget) from valid(service) group by directeur

```
CAB       200000    25        9999999
          500000    20        25
BRE       800000    30        9999999
          800000    25        30
BRA       1000000   5         9999999
SHE       1000000   35        9999999
          1000000   1         5
          1000000   30        35
```

select nom,avg(salaire),count(service) from valid(employe) group by nom

```
Brassens            90000     1         1         9999999
Brel                90000     1         10        9999999
Cabrel              60000     1         25        9999999
Sheller             50000     1         30        9999999
Souchon             80000     1         25        9999999
```

select numero,max(nbrejours),avg(budget),min(datedebut),count(nbrejours) from projet group by numero,responsable

```
P1        50        100000    40        1         BRE       10        80
P1        50        100000    40        1         BRE       90        9999999
P1        50        100000    40        1         SHE       80        90
```

```
P5        30        80000     20        1         BRA         30       9999999
```

select every nom from employe

```
Brassens
Souchon
Brel
Cabrel
Sheller
Brassens
Brassens
Brassens
Brassens
Brassens
Brassens
Brassens
Brassens
Brassens
Brassens
Brassens
```

select nom from employe where nom like '%re%'

```
Brel                10        9999999   60        9999999
Cabrel              25        9999999   65        9999999
```

select nom from employe where timepoint'18' before valid(employe), nom like '%re%'

```
Cabrel              25        9999999   65        9999999
```

select nom from employe where nom like '%re%', timepoint'18' before valid(employe)

```
TSQLExecDirect: TDP10
Unexpected "," found; temporal conditions should precede non-temporal conditions
```

select nom,avg(salaire) from valid(employe)
where timepoint'18' before valid(employe), nom like '%re%'
group by nom

```
TSQLExecDirect: TDPA7
Temporal condition not allowed with an aggregation
```

select nom,avg(salaire) from valid(employe) where nom like '%re%' group by nom

```
Brel                90000     10        9999999
Cabrel              60000     25        9999999
```

select matricule from employe where period’[15,20)’ before valid(employe)

```
CAB         25        9999999   65        9999999
SHE         30        9999999   70        9999999
SOU         25        9999999   55        9999999
```

select matricule from employe where period’[15,25)’ before valid(employe)

```
SHE         30        9999999   70        9999999
```

select matricule from employe where period’[10,25)’ starts valid(employe)

```
BRE         10        9999999   60        9999999
```

select matricule from employe where period’[115,9999999)’ finishes valid(employe)

```
BRA         1         9999999   50        9999999
BRE         10        9999999   60        9999999
CAB         25        9999999   65        9999999
SHE         30        9999999   70        9999999
SOU         25        9999999   55        9999999
```

select matricule from employe where period’[15,20)’ during valid(employe)

```
BRA         1         9999999   50        9999999
BRE         10        9999999   60        9999999
```

select matricule from employe where period’[25,50)’ during valid(employe)

```
BRA         1         9999999   50        9999999
BRE         10        9999999   60        9999999
```

select matricule from employe where period’[15,20)’ meets valid(employe)

```
no result
```

select matricule from employe where period’[15,25)’ meets valid(employe)

```
CAB        25       9999999   65       9999999
SOU        25       9999999   55       9999999
```

select matricule from employe where period’[15,30)’ overlaps valid(employe)

```
CAB        25       9999999   65       9999999
SOU        25       9999999   55       9999999
```

select matricule from employe where period’[15,25)’ overlaps valid(employe)

```
no result
```

select matricule from employe where period’[25,9999999)’ equals valid(employe)

```
CAB        25       9999999   65       9999999
SOU        25       9999999   55       9999999
```

select matricule from employe where timepoint’20’ before valid(employe)

```
CAB        25       9999999   65       9999999
SHE        30       9999999   70       9999999
SOU        25       9999999   55       9999999
```

select matricule from employe where timepoint’25’ before valid(employe)

```
SHE        30       9999999   70       9999999
```

select matricule from employe where valid(employe) before timepoint’10000000’

```
BRA        1        9999999   50       9999999
BRE        10       9999999   60       9999999
CAB        25       9999999   65       9999999
SHE        30       9999999   70       9999999
SOU        25       9999999   55       9999999
```

select matricule from employe where timepoint’25’ starts valid(employe)

```
CAB        25       9999999   65       9999999
SOU        25       9999999   55       9999999
```

select matricule from employe where timepoint’9999999’ finishes valid(employe)

```
BRA        1        9999999   50       9999999
BRE        10       9999999   60       9999999
CAB        25       9999999   65       9999999
SHE        30       9999999   70       9999999
SOU        25       9999999   55       9999999
```

select matricule from employe where timepoint’20’ during valid(employe)

```
BRA        1        9999999   50       9999999
BRE        10       9999999   60       9999999
```

normalize employe check

```
no result (= normalized, complete)
```

normalize employe2 check

```
TSQLExecDirect: TDN03
Normalization: the table is corrupted (errors of type 1)
----------
TSQLExecDirect: TDN04
Normalization: the table contains similar covering states (errors of type 2)
----------
TSQLExecDirect: TDN05
Normalization: the table contains similar joint states (errors of type 3)
----------
TSQLExecDirect: TDN06
Normalization: the table is not complete (errors of type 4)

select type_err,matricule,vstart,vend,tstart,
       tend,vstart_conflict,vend_conflict,tstart_conflict,tend_conflict
from employe2_norm_errors
1         BRA       23        9999999   108       115       15        9999999   105       110
1         BRA       22        53        130       9999999   50        9999999   120       9999999
2         BRA       1         35        125       130       23        35        115       127
```

```
3         BRA       1         15        105       125       15        23        114       125
3         BRA       15        23        114       125       23        9999999   108       115
3         BRA       15        23        114       125       23        35        115       127
4         BRA       0         9999999   100       105
4         BRA       15        23        110       114
4         BRA       35        38        120       130
4         BRA       20        22        130       9999999
```

normalize employe2 check type 3

```
TSQLExecDirect: TDN05
Normalization: the table contains similar joint states (errors of type 3)

select type_err,matricule,vstart,vend,tstart,
       tend,vstart_conflict,vend_conflict,tstart_conflict,tend_conflict
from employe2_norm_errors
3         BRA       1         15        105       125       15        23        114       125
3         BRA       15        23        114       125       23        9999999   108       115
3         BRA       15        23        114       125       23        35        115       127
```

normalize employe2 create result type 1

```
TSQLExecDirect: TDN01
'create' not allowed with type 1 in normalize
```

normalize employe2 create result type 3

```
TSQLExecDirect: TDN05
Normalization: the table contains similar joint states (errors of type 3)

select type_err,matricule,vstart,vend,tstart,
       tend,vstart_conflict,vend_conflict,tstart_conflict,tend_conflict
from employe2_norm_errors
3         BRA       1         15        105       125       15        23        114       125
3         BRA       15        23        114       125       23        9999999   108       115
3         BRA       15        23        114       125       23        35        115       127

select * from result
BRA       Brassens  Liege     90000     INFO      1         9999999   50        100
SOU       Souchon   Bruxelles 80000     PERS      25        9999999   55        9999999
BRE       Brel      Bruxelles 90000     COMP      10        9999999   60        9999999
CAB       Cabrel    Namur     60000     REMU      25        9999999   65        9999999
SHE       Sheller   Mons      50000     COMP      30        9999999   70        9999999
BRA       Brassens  Liege     90000     INFO      1         15        105       114
BRA       Brassens  Liege     90000     INFO      1         9999999   114       115
BRA       Brassens  Liege     90000     INFO      1         35        115       130
BRA       Brassens  Namur     90000     INFO      38        50        120       130
BRA       Brassens  Mons      90000     INFO      15        9999999   105       110
BRA       Brassens  Liege     90000     INFO      23        9999999   108       114
BRA       Brassens  Namur     90000     INFO      35        9999999   115       118
BRA       Brassens  Liege     90000     INFO      50        9999999   120       9999999
BRA       Brassens  Liege     90000     INFO      1         20        130       9999999
BRA       Brassens  Namur     90000     INFO      22        53        130       9999999
```

normalize employe2 create result type 4

```
TSQLExecDirect: TDN06
Normalization: the table is not complete (errors of type 4)

select type_err,matricule,vstart,vend,tstart,
       tend,vstart_conflict,vend_conflict,tstart_conflict,tend_conflict
from employe2_norm_errors
4         BRA       0         9999999   100       105
4         BRA       15        23        110       114
4         BRA       35        38        120       130
4         BRA       20        22        130       9999999

Select * from result
BRA       Brassens  Liege     90000     INFO      1         9999999   50        100
SOU       Souchon   Bruxelles 80000     PERS      25        9999999   55        9999999
BRE       Brel      Bruxelles 90000     COMP      10        9999999   60        9999999
CAB       Cabrel    Namur     60000     REMU      25        9999999   65        9999999
SHE       Sheller   Mons      50000     COMP      30        9999999   70        9999999
BRA       Brassens  Liege     90000     INFO      1         15        105       125
```

```
BRA      Brassens  Mons     90000     INFO     15       9999999  105      110
BRA      Brassens  Liege    90000     INFO     15       23       114      125
BRA      Brassens  Liege    90000     INFO     23       9999999  108      115
BRA      Brassens  Liege    90000     INFO     23       35       115      127
BRA      Brassens  Namur    90000     INFO     35       9999999  115      118
BRA      Brassens  Namur    90000     INFO     38       50       120      130
BRA      Brassens  Liege    90000     INFO     50       9999999  120      9999999
BRA      Brassens  Liege    90000     INFO     1        35       125      130
BRA      Brassens  Liege    90000     INFO     1        20       130      9999999
BRA      Brassens  Namur    90000     INFO     22       53       130      9999999
BRA      -         -        0         -        0        9999999  100      105
BRA      -         -        0         -        15       23       110      114
BRA      -         -        0         -        35       38       120      130
BRA      -         -        0         -        20       22       130      9999999
```

normalize employe2 create result

```
TSQLExecDirect: TDN03
Normalization: the table is corrupted (errors of type 1)
----------
TSQLExecDirect: TDN04
Normalization: the table contains similar covering states (errors of type 2)
----------
TSQLExecDirect: TDN05
Normalization: the table contains similar joint states (errors of type 3)
----------
TSQLExecDirect: TDN06
Normalization: the table is not complete (errors of type 4)

No "result" table created.

select type_err,matricule,vstart,vend,tstart,
       tend,vstart_conflict,vend_conflict,tstart_conflict,tend_conflict
from employe2_norm_errors
1        BRA      23       9999999  108      115      15       9999999  105      110
1        BRA      22       53       130      9999999  50       9999999  120      9999999
2        BRA      1        35       125      130      23       35       115      127
3        BRA      1        15       105      125      15       23       114      125
3        BRA      15       23       114      125      23       9999999  108      115
3        BRA      15       23       114      125      23       35       115      127
4        BRA      0        9999999  100      105
4        BRA      15       23       110      114
4        BRA      35       38       120      130
4        BRA      20       22       130      9999999
```

normalize employe2 create result type 4

select nom,adresse from result

```
Brassens            Liege
Souchon             Bruxelles
Brel                Bruxelles
Cabrel              Namur
Sheller             Mons
Brassens            Liege
Brassens            Mons
Brassens            Liege
Brassens            Liege
Brassens            Liege
Brassens            Namur
Brassens            Namur
Brassens            Liege
Brassens            Liege
Brassens            Liege
Brassens            Namur
-                   -
-                   -
-                   -
-                   -
```

register result

select nom,adresse from result

```
Brassens            Liege               1     9999999   50       100
-                   -                   0     9999999   100      105
Brassens            Mons                15    9999999   105      110
-                   -                   15    23        110      114
Brassens            Liege               23    9999999   108      114
Brassens            Liege               1     15        105      114
Brassens            Liege               1     9999999   114      115
Brassens            Liege               1     35        115      130
-                   -                   35    38        120      130
Brassens            Namur               38    50        120      130
Brassens            Namur               35    9999999   115      118
Brassens            Liege               23    35        125      127
Brassens            Liege               50    9999999   120      9999999
Brassens            Liege               1     20        130      9999999
-                   -                   20    22        130      9999999
Brassens            Namur               22    53        130      9999999
Brel                Bruxelles           10    9999999   60       9999999
Cabrel              Namur               25    9999999   65       9999999
Sheller             Mons                30    9999999   70       9999999
Souchon             Bruxelles           25    9999999   55       9999999
```

select nom from employe2

```
Brassens            1       9999999   105       110
Brassens            23      9999999   110       114
Brassens            1       15        110       114
Brassens            1       9999999   114       118
Brassens            1       35        118       130
Brassens            38      9999999   120       130
Brassens            1       9999999   50        100
Brassens            22      9999999   130       9999999
Brassens            1       20        130       9999999
Brel                10      9999999   60        9999999
Cabrel              25      9999999   65        9999999
Sheller             30      9999999   70        9999999
Souchon             25      9999999   55        9999999
```

Tables de test n°2

Les requêtes suivantes sont des jointures temporelles. Elles ont été réalisées sur des tables différentes de celles utilisées par les requêtes précédentes.

Tableau 6: Employe

Matricule	Nom	Adresse /b	Salaire /b	Service	Vstart	Vend	Tstart	Tend
BRA	Brassens	Liege	90000	INFO	1	9999999	50	105
BRA	Brassens	Mons	90000	INFO	15	9999999	105	110
BRA	Brassens	Liege	90000	INFO	23	9999999	110	115
BRA	Brassens	Namur	90000	INFO	35	9999999	115	120
BRA	Brassens	Liege	90000	INFO	1	15	105	125
BRA	Brassens	Mons	90000	INFO	15	23	110	125
BRA	Brassens	Liege	90000	INFO	23	35	115	125
BRA	Brassens	Liege	90000	INFO	1	35	125	130
BRA	Brassens	Namur	90000	INFO	35	50	120	130
BRA	Brassens	Liege	90000	INFO	1	22	130	135
BRA	Brassens	Namur	90000	INFO	22	50	130	135
BRA	Brassens	Liege	90000	INFO	50	9999999	120	135
BRA	Brassens	Liege	90000	INFO	1	9999999	135	140
BRA	Brassens	Mons	90000	INFO	40	9999999	140	145
BRA	Brassens	Liege	90000	INFO	60	9999999	145	150
BRA	Brassens	Namur	90000	INFO	80	9999999	150	155
BRA	Brassens	Liege	90000	INFO	1	40	140	9999999
BRA	Brassens	Mons	90000	INFO	40	60	145	9999999
BRA	Brassens	Liege	90000	INFO	60	80	150	9999999
BRA	Brassens	Namur	90000	INFO	80	90	155	9999999
BRA	Brassens	Liege	90000	INFO	90	9999999	155	9999999
BRE	Brel	Bruxelles	90000	COMP	10	9999999	60	9999999
CAB	Cabrel	Namur	60000	REMU	25	9999999	65	9999999
SHE	Sheller	Mons	50000	COMP	30	9999999	70	9999999
SOU	Souchon	Bruxelles	80000	PERS	25	9999999	55	9999999

Tableau 7: Projet

Numéro	NbreJours /v	Budget/ /v	DateDe-but	Responsable /v	Client	Vstart	Vend
P1	50	100000	40	BRE	C011	10	80
P1	50	100000	40	SHE	C011	80	90
P1	50	100000	40	BRE	C011	90	95
P1	50	100000	40	SHE	C011	95	98
P1	50	120000	40	SHE	C011	98	105
P1	50	150000	40	SHE	C011	105	110
P1	50	100000	40	SHE	C011	110	120
P1	60	100000	40	SHE	C011	120	9999999
P5	30	80000	20	BRA	C012	30	110
P5	30	100000	20	BRA	C012	110	9999999

Tableau 8: Service

Numero	Nom /b	Directeur /b	Budget /b	Secretaire /b	Vstart	Vend	Tstart	Tend
COMP	Comptabilite		1000000	Duchemin	30	9999999	40	100
COMP	Comptabilite	SHE	1000000	Duchemin	35	9999999	100	160
COMP	Comptabilite	SOU	1000000	Duchemin	50	9999999	160	165
COMP	Comptabilite	SHE	1000000	Duchemin	70	9999999	165	170
COMP	Comptabilite	SHE	1500000	Duchemin	80	9999999	170	175
COMP	Comptabilite		1000000	Duchemin	30	35	100	9999999
COMP	Comptabilite	SHE	1000000	Duchemin	35	50	160	9999999
COMP	Comptabilite	SOU	1000000	Duchemin	50	70	165	9999999
COMP	Comptabilite	SHE	1000000	Duchemin	70	80	170	9999999
COMP	Comptabilite	SHE	1500000	Duchemin	80	94	175	9999999
COMP	Comptabilite	SHE	1300000	Duchemin	94	9999999	175	9999999
INFO	Informatique		1000000	Simon	1	9999999	30	75
INFO	Informatique	BRA	1000000	Simon	5	9999999	75	180
INFO	Informatique	BRA	1000000	Panisse	12	9999999	180	185
INFO	Informatique	BRA	1000000	Marius	34	9999999	185	190
INFO	Informatique		1000000	Simon	1	5	75	9999999
INFO	Informatique	BRA	1000000	Simon	5	12	180	9999999
INFO	Informatique	BRA	1000000	Panisse	12	34	185	9999999
INFO	Informatique	BRA	1000000	Marius	34	52	190	9999999
INFO	Informatique	BRA	1000000	Micha	52	9999999	190	9999999
PERS	Personnel		800000	Muller	20	9999999	35	85
PERS	Personnel		800000	Muller	20	30	85	9999999
PERS	Personnel	BRE	800000	Muller	30	9999999	85	9999999
REMU	Remunerations		200000	Lepage	20	9999999	45	95
REMU	Remunerations		200000	Lepage	20	25	95	9999999
REMU	Remunerations	CAB	200000	Lepage	25	9999999	95	9999999

Tableau 9: LOCAL

Numero	Etage /t	Responsable /t	NbreArmoires /t	Tstart	Tend
L1	1	BRA	5	195	220
L1	1	SOU	5	220	225
L1	1	SOU	10	225	230
L1	1	CAB	10	230	235
L1	1	BRA	10	235	9999999
L2	1	BRA	8	200	240
L2	1	BRA	20	240	9999999
L3	2	CAB	10	205	245
L3	2	CAB	8	245	250
L3	2	BRA	8	250	255
L3	2	BRA	14	255	9999999
L4	2	CAB	12	210	9999999
L5	2	SHE	14	215	260
L5	2	SHE	5	260	265
L5	2	CAB	5	265	270
L5	2	CAB	10	270	9999999

Tableau 10: Client

Code
C011
C012
C013
C014
C015

Requêtes test n°2

Pour chacune des requêtes, nous allons spécifier le type de temps des tables et des colonnes mentionnés dans la condition de jointure.

Nous écrirons : type table.type colonne - type table.type colonne

Les types de temps sont représentés par les valeurs suivantes :

0 : non temporel

ide : identifiant d'entité

v : Valid Time

t : Transaction Time

b : Bitemporel

b.o - b.ide

Select numero,budget,adresse,salaire from EMPLOYE,SERVICE where service==numero

```
COMP            1300000         Bruxelles       90000
COMP            1300000         Mons            50000
INFO            1000000         Liege           90000
PERS            800000          Bruxelles       80000
REMU            200000          Namur           60000
```

b.b - b.ide

Select numero,directeur,adressefrom EMPLOYE,SERVICE

whereservice.directeur==employe.matricule

```
INFO            BRA             Liege           5       9999999 75      105
INFO            BRA             Mons            15      9999999 105     110
INFO            BRA             Liege           23      9999999 110     115
INFO            BRA             Namur           35      9999999 115     120
INFO            BRA             Liege           5       15      105     125
INFO            BRA             Mons            15      23      110     125
INFO            BRA             Liege           23      35      115     125
INFO            BRA             Liege           5       35      125     130
INFO            BRA             Namur           35      50      120     130
INFO            BRA             Liege           50      9999999 120     135
INFO            BRA             Liege           5       22      130     135
INFO            BRA             Namur           22      50      130     135
INFO            BRA             Liege           5       9999999 135     140
INFO            BRA             Mons            40      9999999 140     145
INFO            BRA             Liege           60      9999999 145     150
INFO            BRA             Namur           80      9999999 150     155
INFO            BRA             Liege           5       40      140     9999999
INFO            BRA             Mons            40      60      145     9999999
```

```
INFO            BRA             Liege           60      80      150     9999999
INFO            BRA             Namur           80      90      155     9999999
INFO            BRA             Liege           90      9999999 155     9999999
PERS            BRE             Bruxelles       30      9999999 85      9999999
REMU            CAB             Namur           25      9999999 95      9999999
COMP            SHE             Mons            35      9999999 100     160
COMP            SHE             Mons            70      9999999 165     9999999
COMP            SHE             Mons            35      50      160     9999999
COMP            SOU             Bruxelles       50      9999999 160     165
COMP            SOU             Bruxelles       50      70      165     9999999
```

v.v - b.ide

Select numero,budget,responsable,servicefrom EMPLOYE,PROJET

whereresponsable==matricule

```
P5              80000   BRA             INFO            30      110
P5              100000  BRA             INFO            110     9999999
P1              100000  BRE             COMP            10      80
P1              100000  BRE             COMP            90      95
P1              100000  SHE             COMP            80      90
P1              100000  SHE             COMP            95      98
P1              120000  SHE             COMP            98      105
P1              150000  SHE             COMP            105     110
P1              100000  SHE             COMP            110     9999999
```

t.t - b.ide

Select numero,responsable,servicefrom EMPLOYE,LOCAL where responsable==matricule

```
L1              BRA             INFO            195     220
L1              BRA             INFO            235     9999999
L3              BRA             INFO            250     9999999
L2              BRA             INFO            200     9999999
L1              CAB             REMU            230     235
L5              CAB             REMU            265     9999999
L3              CAB             REMU            205     250
L4              CAB             REMU            210     9999999
L5              SHE             COMP            215     265
L1              SOU             PERS            220     230
```

b.o - b.ide

Select employe.salaire,employe.service,service.budgetfrom EMPLOYE,service

whereemploye.service==service.numero

```
90000   COMP            1300000
50000   COMP            1300000
90000   INFO            1000000
80000   PERS            800000
60000   REMU            200000
```

v.v - b.ide

Select numero,employe.adresse,servicefrom EMPLOYE,PROJET

whereprojet.responsable==matricule

```
P5              Liege           INFO            30      40
P5              Mons            INFO            40      60
P5              Liege           INFO            60      80
P5              Namur           INFO            80      90
P5              Liege           INFO            90      9999999
P1              Bruxelles       COMP            10      80
```

```
P1                Bruxelles         COMP              90        95
P1                Mons              COMP              80        90
P1                Mons              COMP              95        9999999
```

v.v - b.ide

Select numero,employe.adresse,servicefrom EMPLOYE,PROJET

whereresponsable==employe.matricule

```
P5                Liege             INFO              30        40
P5                Mons              INFO              40        60
P5                Liege             INFO              60        80
P5                Namur             INFO              80        90
P5                Liege             INFO              90        9999999
P1                Bruxelles         COMP              10        80
P1                Bruxelles         COMP              90        95
P1                Mons              COMP              80        90
P1                Mons              COMP              95        9999999
```

b.b - b.ide / b.o - b.ide

Selectmatricule,salaire,service,directeur,budget from EMPLOYE,SERVICE

wherematricule==directeurand service==numero

```
BRA               90000     INFO              BRA               1000000
CAB               60000     REMU              CAB               200000
SHE               50000     COMP              SHE               1300000
```

b.b - b.ide / b.o - b.ide

Selectmatricule,salaire,service,directeur,budgetfrom EMPLOYE,SERVICE

wherenumero==serviceand matricule==directeur

```
SHE               50000     COMP              SHE               1300000
BRA               90000     INFO              BRA               1000000
CAB               60000     REMU              CAB               200000
```

b.b - t.t

Select local.numero,responsable,service.numero, directeurfrom local,service

wheredirecteur==responsable

```
L1                BRA               INFO              BRA               195       220
L1                BRA               INFO              BRA               235       9999999
L3                BRA               INFO              BRA               250       9999999
L2                BRA               INFO              BRA               200       9999999
L1                CAB               REMU              CAB               230       235
L5                CAB               REMU              CAB               265       9999999
L3                CAB               REMU              CAB               205       250
L4                CAB               REMU              CAB               210       9999999
L5                SHE               COMP              SHE               215       265
```

v.v - b.ide / v.v - b.b

Select projet.numero,responsable,matricule,service from projet,employe

wherematricule==responsableandbudget==salaire

```
NO        DATA
```

v.v - b.ide

Select projet.numero,responsable,matricule,servicefrom projet,employe

wherematricule==responsable

```
P5                BRA               BRA               INFO              30        9999999
P1                BRE               BRE               COMP              10        80
P1                BRE               BRE               COMP              90        95
P1                SHE               SHE               COMP              80        90
P1                SHE               SHE               COMP              95        9999999
```

v.v - t.t

Select projet.numero,projet.responsable,local.numero,budget,local.responsable

from projet,local

whereprojet.responsable==local.responsable

```
TDPJ04    Invalid   case
```

v.v - t.t

Selectprojet.numero,projet.responsable,local.numero,budget,local.responsable

from projet,local

whereprojet.responsable==local.responsableandlocal.responsable==client

```
TDPJ04    Invalid   case
```