Des fonctions d'oubli intelligentes dans les entrepôts de données

Aliou Boly*, Sabine Goutier**, Georges Hébrail*·**
*46, Rue Barrault, 75634 PARIS Cedex 13 - FRANCE
boly@enst.fr, hebrail@enst.fr
**1, Av. du Général de Gaulle, 92141 CLAMART Cedex - FRANCE
georges.hebrail@edf.fr, sabine.goutier@edf.fr

Résumé. Les entrepôts de données stockent des quantités de données de plus en plus massives et arrivent vite à saturation. Un langage de spécifications de fonctions d'oubli est défini pour résoudre ce problème. Dans le but d'offrir la possibilité d'effectuer des analyses sur l'historique des données, les spécifications définissent des résumés par agrégation et par échantillonnage à conserver parmi les données à 'oublier'. Cette communication présente le langage de spécifications ainsi que les principes et les algorithmes pour assurer de façon mécanique la gestion des fonctions d'oubli.

1 Introduction

De nos jours, bien que les moyens de stockage soient de plus en plus performants et de moins en moins chers, les entrepôts de données arrivent vite à saturation et la question des données à conserver sous forme d'historique va se poser rapidement. Il faut donc choisir quelles données doivent être archivées, et quelles données doivent être conservées actives dans les entrepôts de données. La solution qui est appliquée en général est d'assurer un archivage périodique des données les plus anciennes. Cette solution n'est pas satisfaisante car l'archivage et la remise en ligne des données sont des opérations coûteuses au point que l'on peut considérer que des données archivées sont des données perdues (en pratique inutilisables dans le futur) du point de vue de leur utilisation dans le cadre d'une analyse des données.

Dans cette communication, nous proposons une solution pour éviter la saturation des entrepôts de données. Un langage de spécifications de fonctions d'oubli des données anciennes est défini pour déterminer les données qui doivent être présentes dans l'entrepôt de données à chaque instant. Ces spécifications de fonctions d'oubli conduisent à supprimer de façon mécanique les données à 'oublier', tout en conservant un résumé de celles-ci par agrégation et par échantillonnage. L'agrégation et l'échantillonnage constituent deux techniques standard et complémentaires pour résumer des données. Considérons un entrepôt de données d'analyse des click-stream sur les sites web. Avec le temps, les données détaillées anciennes deviennent de moins en moins 'utiles' et peuvent donc être agrégées par jour ou par mois par exemple. En plus d'agréger des données, on peut conserver certaines données jugées intéressantes ou choisies de façon aléatoire dans le but de pouvoir effectuer des analyses sur les données de l'entrepôt.

Le langage de spécifications est défini dans le cadre du modèle relationnel : sur chaque table, est défini au moyen de spécifications un ensemble de n-uplets à archiver. Pour des raisons applicatives, parmi les n-uplets à archiver, des échantillons peuvent être conservés dans le cadre de l'utilisation de l'entrepôt. De plus, des algorithmes pour mettre à jour le

contenu de l'entrepôt conformément aux spécifications de fonctions d'oubli sont présentés. A noter que ces algorithmes gérés par l'administrateur de bases de données ont été étudiés et programmés dans un prototype sous Oracle. Cette communication a pour but d'apporter une réponse au problème de saturation des entrepôts mais l'approche décrite ici s'applique de façon générale aux bases de données relationnelles.

La suite de ce papier est structurée comme suit. La section 2 expose l'état de l'art des travaux en rapport avec les fonctions d'oubli. Dans la section 3, sont présentées les spécifications de fonctions d'oubli, après avoir défini formellement l'âge d'une donnée et présenté un exemple de motivation. La section 4 propose des structures de données adaptées au stockage des données agrégées et présente les algorithmes pour mettre à jour le contenu de l'entrepôt après application des fonctions d'oubli. La section 5 traite de la conservation des échantillons. Enfin, dans la section 6, une conclusion et des perspectives associées à ce travail sont présentées.

2 Etat de l'art

Ce travail est à relier aux travaux sur le « vacuuming », une approche permettant de supprimer physiquement dans une base de données temporelle, les données plus anciennes qu'une certaine date seuil. Ce concept a été développé par Jensen (Jensen, 1995): lorsqu'une date particulière est spécifiée, les données antérieures à cette date sont considérées comme étant inaccessibles et doivent donc être supprimées physiquement de la base de données.

Dans (Skyt et al., 2001), une technique de réduction des données est décrite dans le cadre des bases de données multidimensionnelles. Un langage de spécifications est proposé pour réduire la granularité des données les plus anciennes en agrégeant les données anciennes à des niveaux supérieurs (plus grossiers). Ces travaux sont similaires aux nôtres. Cependant, les approches que nous proposons dans cette communication s'appliquent aux entrepôts de données et plus largement aux bases de données relationnelles, où des contraintes référentielles peuvent exister entre les tables, à la différence de ces travaux, qui ne s'appliquent que sur des données multidimensionnelles stockées dans un cube de données. De plus, les auteurs ne proposent pas de conserver le détail de certaines données dans une perspective d'analyse des données. Plus récemment, beaucoup de travaux sont effectués sur les flux de données, où il existe un fort besoin de résumer les données par rapport au temps. Par exemple, dans (Chen et al., 2002), le problème du calcul des agrégats temporels pour les flux de données est étudié, et il est suggéré de maintenir les agrégats à différents niveaux de granularité par rapport au temps : les données les plus anciennes sont agrégées à des niveaux grossiers, alors que les données récentes sont agrégées avec un niveau plus fin. Dans cette communication, nous reprenons des concepts de ces travaux mais la façon dont sont agrégées les données est spécifiée par l'administrateur de l'entrepôt au lieu que cela soit contrôlé par le flux.

Ce travail est à relier aussi à l'expiration des données (Garcia-Molina et al., 1998) dans le cadre des vues matérialisées (voir (Gupta et Mumick, 2005)) : l'approche consiste à détecter les données dont la suppression n'affecte pas la maintenance des vues matérialisées de l'entrepôt.

On peut citer également les travaux de (Toman, 2001) sur l'expiration des données : le problème est d'étudier l'expiration des données dans le contexte des bases de données historisées qui stockent l'historique des différents états de la base de données. Cependant, dans nos travaux, nous ne nous intéressons pas à l'historique des différents états de la base de

données mais à l'historique des données stockées explicitement dans la base de données ou dans l'entrepôt de données.

On peut évoquer le principe du ramasse-miettes (Boehm, 2002) dans les programmes objet. Le ramasse-miettes constitue une forme de gestion automatique de la mémoire. Le principe est de déterminer les objets qui ne sont plus référencés et de récupérer le stockage utilisé par ces objets. Le ramasse-miettes doit analyser les liens entre les objets avant de les supprimer. Dans nos travaux aussi, lorsque les fonctions sont appliquées, des n-uplets peuvent être à archiver alors qu'ils sont référencés par d'autres n-uplets qui ne sont pas encore archivés. La section 4.3 traite un problème similaire, lorsque des contraintes référentielles (voir (Gardarin, 1999)) existent entre les tables.

Notre contribution dans cette communication est l'extension et la réalisation d'un prototype de mise en œuvre du langage de spécifications de fonctions d'oubli présenté dans (Boly et al., 2004). Dans ce travail, les spécifications – définies par l'administrateur de la base de données – permettent d'appliquer de façon mécanique les fonctions d'oubli, qui gèrent à la fois la suppression de données à archiver (avec une prise en compte des contraintes référentielles) et la mise à jour des résumés conservés qui sont des agrégats et des échantillons de données détaillées.

Dans (Chaudhuri et al., 2001), les auteurs montrent qu'on peut répondre à des requêtes en utilisant seulement des échantillons et des données agrégées au lieu de la base totale. D'une part, lorsque l'on ne dispose que de données agrégées sans les données détaillées correspondantes, il est toujours possible de répondre à des requêtes d'agrégation en fournissant des réponses approximatives. D'autre part, dans la théorie de l'échantillonnage (voir (Ardilly, 1994)), il est montré qu'on peut inférer des propriétés sur un ensemble de données à partir des échantillons. De plus, dans la fouille de données comme les arbres de décisions par exemple, on peut seulement considérer des agrégats sur des données détaillées. Cela justifie notre approche de conservation de résumés de données à archiver.

3 Spécification de fonctions d'oubli

3.1 Notion d'âge d'une donnée

Dans le cadre de ce travail, nous considérons que chaque donnée (n-uplet d'une table dans notre étude) est associée à une date notée t_s qui correspondra soit à la valeur d'un attribut de type Date explicitement associé à la donnée ou soit au timestamp du système représentant la date de dernière mise à jour de la donnée. L'âge d'une donnée calculé à la date courante notée t_c est défini comme étant la différence entre les dates t_c et t_s. Les dates t_c et t_s peuvent être exprimées dans les unités de temps suivantes : la seconde (SECOND), la minute (MINUTE), l'heure (HOUR), le jour (DAY), le mois (MONTH), le trimestre (QUARTER) et l'année (YEAR). Nous adoptons les conventions suivantes :

1 MINUTE = 60 s[secondes] 1 HOUR = 60*60 s

1 DAY = 24*3600 s 1 MONTH = 30 DAYS = 30*24*3600 s

1 QUARTER = 3*30 DAYS = 3*30*24*3600 s

1 YEAR = 365 DAYS = 365*24*3600 s

Pour calculer l'âge « $t_c - t_s$ », nous utiliserons les dates t_c et t_s à l'unité SECOND, quelles que soient les unités de temps de t_c et de t_s.

L'âge d'une donnée est calculé de la manière suivante :

- Si t_s est exprimée au niveau SECOND : age = *a_la_seconde*(t_c) – t_s
- Si t_s n'est pas exprimée au niveau SECOND : age = *a_la_seconde*(t_c) – *a_la_seconde*(t_s+1)

où *a_la_seconde* est une fonction qui transforme une date (qui n'est pas à la seconde) à une date exprimée à la seconde en considérant le premier instant de la période de temps couverte par cette date. Par exemple : *a_la_seconde*('17/02/06 12:05')='17/02/06 12:05:00', *a_la_seconde*('Fevrier 2006')='01/02/06 00:00:00'.

Quelques exemples d'illustration sont présentés ci-dessous:
t_c = '20/01/06 12:34:45', t_s = '20/01/06 10:50:54', t_c – t_s = 6231 SECOND = 1 HOUR t_c = '20/02/06', t_s = '20/01/06 11h', *a_la_seconde*(t_c) = '20/02/06 00:00:00', *a_la_seconde*(t_s+1) = '20/01/06 12:00:00', t_c – t_s = 2635200 SECOND = 30 DAY = 1 MONTH.

Enfin, une relation d'ordre peut être définie entre les âges : considérant deux âges age1 et age2, on dit que age1 < age2 si et seulement si : *en_secondes*(age1) < *en_secondes*(age2), où *en_secondes* est une fonction qui transforme en secondes une durée (un âge) qui est exprimée dans une autre unité de temps, en utilisant les conventions présentées ci-dessus.
Par exemple: 30 DAY < 3 MONTH < 1 YEAR.

3.2 Exemple de motivation

Nous considérons ici un entrepôt de données contenant les tables suivantes :
CLIENT (cliID, Ville, Sexe, Salaire),
COMMANDE (cmdID, Date_cmd, montant, cliID)

Pour chaque table, l'attribut souligné représente la clé primaire (identifiant). La table CLIENT a pour attributs *cliID* l'identifiant, la ville, le sexe et le salaire du client. Pour la table COMMANDE, *cmdID* désigne l'identifiant, *Date_cmd* la date associée à une commande, *montant* le montant de la commande et *cliID* est une clé étrangère référençant la table CLIENT. Considérons la fonction suivante sur la table COMMANDE.

```
SUMMARY TABLE COMMANDE {
    USE CLIENT JOIN BY cliID;
    TIMESTAMP = Date_cmd ;
    HIERARCHY (Geographie) : Ville→ Département→ Région ;
    LESS THAN 1 MONTH : DETAIL ;
    LESS THAN 3 MONTH : SUM (montant) BY CLIENT.Ville, CLIENT.Sexe, DAY ;
    LESS THAN 1 YEAR : SUM (montant) BY Département, MONTH ;
    LESS THAN 5 YEAR : SUM (montant) BY  Région, YEAR ;
    LESS THAN 10 YEAR : SUM (montant) BY YEAR ;
    KEEP SAMPLE (1000) WHERE montant>4000 ;
} END SUMMARY ;
```

La fonction d'oubli indique d'abord qu'elle s'applique à la table COMMANDE et que l'âge associé à chaque n-uplet de COMMANDE est le timestamp fourni par la colonne Date_cmd (TIMESTAMP = Date_cmd).

Les spécifications LESS THAN définissent quand les données détaillées (les n-uplets de la table COMMANDE) doivent être archivées et le résumé à conserver après application de la fonction d'oubli. Pour les données de moins de 30 jours (c'est-à-dire les données qui ont un âge inférieur à 1 mois), on garde le détail dans la table COMMANDE (c'est-à-dire les n-

uplets de la table COMMANDE). Les données qui ont un âge supérieur à 1 mois peuvent être archivées mais des versions agrégées de ces données doivent être conservées :

- Les données qui ont un âge compris entre 1 mois et 3 mois doivent être agrégées par Ville, Sexe et par jour. A noter que Ville et Sexe sont des attributs de la table CLIENT qui est référencée par COMMANDE. Cela est rendu possible grâce à la spécification « USE CLIENT JOIN BY cliID ».
- Les données qui ont un âge compris entre 3 mois et 1 an doivent être décrites par des agrégats au niveau Département et par mois. Le passage d'une agrégation par ville à une agrégation par département est possible en utilisant la hiérarchie Géographie. On suppose ici qu'il existe une hiérarchie nommée Géographie entre les attributs Ville, Département et Région.
- Les données qui ont un âge compris entre 1 an et 5 ans doivent être agrégées par Région et par Année.
- Les données qui ont un âge compris entre 5 ans et 10 ans doivent être agrégées par Année.
- Aucun agrégat n'est conservé pour les données qui ont un âge supérieur à 10 ans.

Enfin, la spécification « KEEP SAMPLE (1000) WHERE montant>4000 » signifie la conservation d'un échantillon aléatoire simple composé de 1000 n-uplets parmi les n-uplets archivés et qui ont un montant supérieur à 4000.

3.3 Langage de spécifications de fonctions d'oubli

Un langage a été défini pour spécifier la fonction d'oubli associée à chaque table de la base de données. La grammaire du langage est présentée en annexe. Il comprend des caractéristiques supplémentaires telles que la discrétisation d'attributs numériques dans le but de les utiliser comme attributs sur lesquels peut s'effectuer une agrégation[1].

Lorsque les spécifications de fonctions d'oubli sont définies, l'application des fonctions d'oubli est automatique : des algorithmes et des structures de stockage sont nécessaires pour assurer la gestion des fonctions d'oubli.

4 Gestion des fonctions d'oubli par agrégation

4.1 Structure de stockage pour les données agrégées

Une fonction d'oubli peut contenir plusieurs spécifications d'agrégation 'LESS THAN'. Chacune de ces spécifications indique un niveau d'agrégation dépendant de l'âge des données. A partir de ces spécifications 'LESS THAN', on associe une famille de cubes qui seront utilisés pour stocker les données agrégées.

Une famille de cubes $\boldsymbol{C}$ est une liste $(D_1,\ldots,D_n, TL, M)$, où $D_1,\ldots, D_n$ sont des noms de dimensions, TL est la dimension Temps et M un espace vectoriel de mesures. A chaque dimension D_i est associée une liste ordonnée de niveaux $L_i=Niveaux(D_i)$[2], comprenant la valeur ALL qui agrège complètement la dimension. Les niveaux d'une dimension représentent une hiérarchie. A noter que dans le cadre de ce travail, on considère qu'il existe seulement une

[1] Cette possibilité n'est pas présentée dans cette communication.

[2] On écrit $D_i=(L_i, \leq)$

hiérarchie par dimension. On note que la dimension Temps est organisée de la manière suivante : *Niveaux*(TL)=(SECOND, MINUTE, HOUR, DAY, MONTH, QUARTER, YEAR, ALL). Considérant notre exemple, nous avons :
C_COMMANDE = (EMPLACEMENT, SEXE, TL, TOTAL_MONTANT)
levels(EMPLACEMENT)=(VILLE, DEPARTEMENT, REGION, ALL), *levels*(SEXE) = (SEXE, ALL), *levels*(TL) = (DAY, MONTH, YEAR, ALL)

En utilisant la famille de cubes, on peut associer un cube de cette famille (appelé aussi cuboïde selon certaines références) à chaque spécification d'agrégation.

Un cube C de la famille de cubes $\boldsymbol{C}$ est défini par $(l_1,\ldots, l_n, tl, M)$ tel que $l_i \in L_i = Niveaux(D_i)$, $i = 1,\ldots, n$, $tl \in TL$. A chaque niveau l est associé un ensemble non vide de valeurs $dom(l)$, représentant le domaine de l et à chaque niveau tl est également associé un ensemble non vide de valeurs noté $dom(tl)$, représentant le domaine de tl.

Une cellule du cube C est un tuple $c = (x_1,\ldots, x_n, t, m)$, tel que $\forall\, i = 1,\ldots, n$, $x_i \in dom(l_i)$, $t \in dom(tl)$ et $m \in M$.

Reprenant l'exemple présenté au paragraphe 3.2, nous avons:
S_1: LESS THAN 3 MONTH : SUM (montant) BY Ville, Sexe, DAY ;
→ C_1=(VILLE, SEXE, DAY, SUM_MONTANT)
S_2: LESS THAN 1 YEAR : SUM (montant) BY Département, MONTH ;
→ C_2=(DEPARTEMENT, ALL, MONTH, SUM_MONTANT)
S_3: LESS THAN 5 YEAR : SUM (montant) BY Région, YEAR ;
→ C_3=(REGION, ALL, YEAR, SUM_MONTANT)
S_4: LESS THAN 10 YEAR : SUM (montant) BY YEAR ;
→ C_4=(ALL, ALL, YEAR, SUM_MONTANT)
Les différentes spécifications 'LESS THAN' peuvent être ordonnées par rapport à leur âge[3] :
$age(S_1) \leq age(S_2) \leq age(S_3) \leq age(S_4)$.
Les cubes peuvent également être ordonnés du moins agrégé au plus agrégé (on utilise la même notation $\leq$): $C_1 \leq C_2 \leq C_3 \leq C_4$. A noter que $C_1 \leq C_2$ signifie que la partition des données détaillées induite par C_1 est plus fine que celle induite par C_2[4].

4.2 Mise à jour des données agrégées

Les algorithmes que nous proposons pour mettre à jour les données agrégées sont basés sur trois propriétés : (1) les mesures d'agrégation sont additives, (2) les différents cubes sont disjoints, (3) la mise à jour des données agrégées et celle des données détaillées commutent.

Additivité des mesures d'agrégation : les mesures d'agrégation que l'on considère sont additives ou des expressions calculées à partir de mesures qui sont additives. Dans notre exemple, SUM(montant) représente une mesure additive. Cette propriété assure que si deux cubes C et C' sont tels que $C \leq C'$, les données appartenant au cube C' peuvent être calculées à partir des données du cube C. A noter que l'on ne considère pas la fonction AVG séparément dans cette communication (elle est remplacée par les fonctions SUM et COUNT).

[3] L'âge d'une spécification 'LESS THAN' est défini par la valeur d'âge spécifiée juste après le mot clé THAN.

[4] L'ordre est total car il existe seulement une hiérarchie par dimension.

Les différents cubes sont disjoints: nous avons vu que les spécifications d'agrégation sont ordonnées par rapport à leur âge. Considérant deux spécifications ordonnées S_{i-1} et S_i associées respectivement aux cubes C_{i-1} et C_i. Nous avons $age(S_{i-1}) \leq age(S_i)$. Un tuple d d'une table (COMMANDE dans notre exemple) est pris en compte dans un seul cube, de la manière suivante : $d \in C_i$ si et seulement si $age(S_{i-1}) \leq age(d) < age(S_i)$.
En conséquence, tous les cubes d'une fonction d'oubli sont disjoints : aucune cellule d'un cube est incluse (fonctionnellement) dans une cellule d'un cube plus agrégé. Les différents cubes ont des contenus exclusifs ; ils stockent des données de périodes différentes par rapport au temps.

Commutativité des mises à jour : les mises à jour à effectuer sur les données agrégées du fait des fonctions d'oubli sont de deux types : (1) un n-uplet de détail à archiver d'une table doit être stocké dans un cube (un seul), (2) des données déjà agrégées d'un cube peuvent être transférées dans un autre cube plus agrégé.

Puisque dans la répercussion des n-uplets de détail vers les cubes, chaque n-uplet de détail n'est pris en compte que dans un seul cube, celui où il satisfait le critère[5] de la spécification correspondante, il est alors équivalent de commencer par répercuter le détail vers les cubes puis d'effectuer le transfert des données agrégées entre les cubes, ou de commencer par le transfert entre les cubes pour ensuite répercuter les n-uplets de détail. A noter que les mises à jour à effectuer du fait des fonctions d'oubli peuvent être appliquées à tout instant, soit de façon régulière (par exemple chaque jour), ou soit de façon irrégulière ou soit sporadiquement. A chaque application des fonctions d'oubli, de nouveaux n-uplets de la table peuvent satisfaire les critères des spécifications d'agrégation et doivent donc être agrégés et stockés dans les cubes correspondants. Egalement, lorsque des données agrégées d'un cube C ne vérifient plus le critère de la spécification correspondante mais vérifient le critère correspondant à un cube plus agrégé C', ces données doivent alors être transférées dans ce cube C'. A noter que l'on fait l'hypothèse que l'espace de stockage nécessaire pour contenir les données des différents cubes est limité. Cela est rendu possible par le fait que les différents cubes sont disjoints (propriété présentée ci-dessus), et par un système d'activation/désactivation pour les cellules des cubes. Lorsqu'une donnée doit passer d'un cube à un autre, la cellule qu'elle occupait ne va pas être supprimée mais désactivée. Et donc, pour répercuter une donnée vers un cube, on commencera par vérifier si elle peut occuper la place d'une cellule désactivée auquel cas, cette cellule est activée.

En conséquence de la propriété de commutativité des mises à jour, nous distinguons deux procédures qui peuvent être exécutées dans un ordre quelconque : (1) la procédure qui transfère les données agrégées entre les cubes, (2) la procédure qui répercute les n-uplets de détail archivés vers les cubes. Dans les algorithmes présentés ci-dessous, chaque cellule d'un cube est caractérisée par deux champs : la position et la mesure. La position d'une cellule est l'ensemble des valeurs des niveaux de dimensions qui déterminent cette cellule. Par exemple, la position de la cellule ('Paris', 'F', '15/05/06', 2000) est ('Paris', 'F', '15/05/06') et la mesure correspondante est égale à 2000.

Transfert des données agrégées entre les cubes
Pour chaque cube C_j ($j = n, n-1, n-2, \ldots, 2$)

[5] Le critère d'une spécification détermine les données qui doivent être agrégées.

Pour chaque cube C_i ($i = j - 1, j - 2, \ldots, 1$)
 Pour chaque cellule $c = (x_1, \ldots, x_n, t, m)$ dans C_i avec $age(S_j) \geq age(t) > age(S_i)$
  /* S_j et S_i sont les spécifications associées respectivement aux cubes C_j et C_i */
  /* le parcours est accéléré par un index sur la dimension Temps */
  soit $\overline{c}$ la cellule dans le cube C_j couvrant c
  si ($\overline{c}$ trouvé) /* incrémentation */
   pour chaque mesure d'agrégation M_i dans $\overline{c}$
    si M_i est COUNT ou SUM $\overline{c}.M_i = \overline{c}.M_i + c.M_i$
    sinon si M_i est MIN $\overline{c}.M_i = \text{MIN}(\overline{c}.M_i, c.M_i)$
    sinon si M_i est MAX $\overline{c}.M_i = \text{MAX}(\overline{c}.M_i, c.M_i)$
  `sinon` /*vérifier s'il existe une cellule désactivée $c_{trouvé}$ dans C_j qui peut remplacer $\overline{c}$ */
   si (trouvé)
    $c_{trouvé}$.position := $\overline{c}$.position
    $c_{trouvé}$.mesure := $\overline{c}$.mesure
    activer $C_{trouvé}$
   sinon créer la cellule $\overline{c}$ dans C_j
 désactiver la cellule c dans le cube C_i

Il est à noter que dans la mise à jour des cubes, il est possible que des cellules d'un cube C_i ne puissent pas satisfaire le critère de la spécification correspondant au cube suivant C_{i+1} mais qu'elles vérifient le critère d'une spécification correspondant à un cube C_p tel que $p>i+1$. Cette situation survient en général lorsque les fonctions d'oubli n'ont pas été appliquées pendant un certain temps : par exemple, considérant notre exemple présenté au paragraphe 3.2, lorsque les fonctions d'oubli n'ont pas été appliquées pendant cinq ans, les données du cube correspondant à la deuxième spécification (LESS THAN 3 MONTH : SUM (montant) BY Ville, Sexe, DAY) doivent être directement transférées au cube le plus agrégé. Ceci explique la présence dans l'algorithme des deux premières « boucles Pour » imbriquées.

Répercussion des n-uplets de détail vers les cubes

On note que l'ordre de traitement des cubes n'est pas important car chaque n-uplet de détail ne va être stocké que dans un seul cube.

Pour chaque cube C_j ($j = 1, 2, \ldots, n$)
 soit C_j^{detail} = {d de T (table de base) tels que $age(S_j) \geq \text{age}(d) > age(S_{j-1})$ et agrégés selon les dimensions du cube C_j} /* S_j et S_{j-1} sont les spécifications associées respectivement aux cubes C_j et C_{j-1}; S_0 la spécification de DETAIL[6]*/
  /* on suppose qu'il existe un index sur la dimension Temps */
 Pour chaque cellule c de C_j^{detail}
  soit c' = la cellule du cube C_j ayant les mêmes valeurs pour les dimensions du cube C_j
  si (c' est trouvé) /* incrémentation */
   pour chaque mesure d'agrégation M_i dans c'
    si M_i est COUNT or SUM $c'.M_i = c'.M_i + c.M_i$

[6] Dans notre exemple LESS THAN 1 MONTH : DETAIL ;

sinon si M_i est MIN $c'.M_i = \text{MIN}(c'.M_i, c.M_i)$
sinon si M_i est MAX $c'.M_i = \text{MAX}(c'.M_i, c.M_i)$
sinon /*vérifier s'il existe une cellule désactivée $c_{trouvé}$ dans C_j qui peut remplacer c'*/
si (trouvé)
$c_{trouvé}$.position := c'.position
$c_{trouvé}$.mesure := c'.mesure
activer $c_{trouvé}$
sinon créer la cellule c' dans C_j

4.3 Gestion des contraintes référentielles

Dans cette section, nous considérons la définition de plusieurs fonctions d'oubli, chacune associée à une table. Le problème qui se pose alors est qu'il existe des contraintes référentielles entre les tables sur lesquelles sont définies les fonctions d'oubli. Considérons dans notre exemple, une table *FACTURE (FactID, Date_fact, montant, cmdID)*, où *FactID* est l'identifiant, *Date_fact* représente la date d'une facture, *montant* le montant d'une facture et *cmdID* est une clé étrangère référençant l'attribut *cmdID* de la table COMMANDE. On suppose que le timestamp dans la fonction d'oubli définie sur FACTURE est fourni par la colonne Date_fact. Les spécifications suivantes sont défnies :

LESS THAN 4 MONTH: DETAIL;
LESS THAN 6 MONTH: SUM(montant) BY MONTH.

Le problème apparaît lorsqu'un n-uplet t_1 à archiver d'une table (COMMANDE dans notre exemple) est référencé par un n-uplet t_2 d'une (autre) table (FACTURE dans notre exemple) qui n'est pas encore archivé. Considérant notre exemple, si on applique les fonctions d'oubli tous les mois (par exemple le 1er de chaque mois), il peut arriver que des n-uplets de la table COMMANDE soient spécifiés à archiver alors que les n-uplets de FACTURE correspondants ne sont pas encore archivés (puisque pour les factures, on les garde pendant 4 mois et que pour les commandes au delà d'1 mois, elles peuvent être archivées). Nous proposons la solution suivante : suspendre l'archivage de t_1 jusqu'à ce que t_2 soit archivé et le marquer à archiver. A chaque application des fonctions d'oubli, il faut alors vérifier si les n-uplets qui le référencent sont archivés, auquel cas le n-uplet devra être archivé. Et donc, dans notre exemple, on va marquer les n-uplets de la table COMMANDE spécifiés à archiver. Ils ne pourront être archivés que lorsque les n-uplets de la table FACTURE correspondants (qui les référencent) auront été archivés. A noter qu'aucune mise à jour n'est plus autorisée sur les n-uplets marqués à archiver.

5 Gestion de la conservation d'échantillons

Comme présenté au paragraphe 3.2, le langage de spécifications permet de conserver des échantillons de données archivées. Il s'agit de la spécification KEEP SAMPLE qui indique que l'on doit assurer la maintenance d'un échantillon aléatoire simple de taille fixe à chaque fois que des données détaillées sont archivées. Les n-uplets échantillonnés sont stockés dans une table séparée ayant la même structure que la table sur laquelle est définie la fonction d'oubli.

A chaque application des fonctions d'oubli, de nouveaux n-uplets peuvent être à archiver et doivent être pris en compte dans la mise à jour de l'échantillon. On suppose par exemple

qu'un échantillon aléatoire simple de 1000 individus est conservé parmi les n-uplets à archiver, et que 100 nouveaux n-uplets sont à archiver depuis la dernière mise à jour d'oubli.

La maintenance de l'échantillon est assurée de manière incrémentale : on utilise l'algorithme réservoir de Vitter (Vitter, 1985). La propriété fondamentale de ce type d'algorithme est qu'après le traitement de chaque individu, le réservoir[7] constitue un échantillon aléatoire des individus visités. A noter que la taille des échantillons est fixée dans la définition de la spécification KEEP SAMPLE.

Comme indiqué à la fin de la section 2, des échantillons aléatoires sur une population peuvent être utilisés pour inférer de l'information sur cette population totale à partir des données observées dans les échantillons (voir (Ardilly, 1994)). Reprenant notre exemple, on peut estimer la somme des montants de commandes à partir de l'échantillon tiré sur les n-uplets à archiver de la table COMMANDE. Pour effectuer de telles estimations, l'effectif de la population de n-uplets à échantillonner est conservé.

6 Conclusion et perspectives

Nous avons proposé une solution pour résoudre le problème de saturation des entrepôts de données. Un langage a été défini pour spécifier les données à archiver et les résumés (des agrégats et des échantillons) à conserver. Lorsque les fonctions d'oubli sont définies, les algorithmes présentés mettent à jour le contenu de l'entrepôt de façon automatique conformément aux spécifications de fonctions d'oubli. Un prototype a été développé sous Oracle pour l'implémentation du langage ainsi que des algorithmes présentés ici.

La perspective associée à ce travail est l'exploitation des résumés conservés pour effectuer des analyses et répondre de manière approximative aux requêtes sur les données après application des fonctions d'oubli. La solution présentée ici s'applique aux bases de données relationnelles, et donc au schéma en étoile. Il pourrait être intéressant de définir des fonctions d'oubli spécifiques aux cubes de données.

7 Références

Ardilly P., Les Techniques de sondage, Technip, 1994.

Boehm H., Bounding Space Usage of Conservative Garbage Collectors, Proceedings of the 2002 ACM SIGPLAN-SIGACT Symposium on Principles of Programming languages, pages 93-100, January 2002.

Boly A., Hébrail G., Picard M.L, EGC « Fonctions d'oubli dans les entrepôts de données », Clermont Ferrand, Janvier 2004.

Chaudhuri S., Das G., Motwani R., Narasayya V., A Robust Optimisation-Based Approach for Approximate Answering of Aggregate Queries. Proceedings of the 2002 ACM SIGMOD, pages 295-306, Santa Barbara, California, USA, 2001.

[7] Un réservoir est considéré comme un vecteur de longueur *n*, *n* étant la taille de l'échantillon.

Chen Y., Dong G., Han J., Wah B. W., and Wang J., Multi-dimensional regression analysis of time-series data streams. In Proc. 2002 Int. Conf. (VLDB'02), pages 323-334, Hong Kong, China, Aug. 2002.

Garcia-Molina H., Labio W. J., Yang J., Expiring data in a warehouse, Proceedings of the 24th VLDB Conference, pages 500-511, New York, USA, 1998.

Gardarin G., Bases de Données objet et relationnel, Eyrolles 1999.

Gupta H., Mumick I., Selection of Views to Materialize in a Data Warehouse. In IEEE Transactions on Knowledge and Data Engineering, TKDE, volume 17, pages 24-43, 2005.

Jensen C. S., "Vacuuming", in the TSQL2 Temporal Query Language, R. T. Snodgrass, editor, Chapter 23, pp. 451-462, Kluwer Academic Publishers, 1995.

Skyt J., Jensen C.S., Pedersen T. B., Specification-Based Data Reduction in Dimensional Data Warehouses, TimeCenter TechReport TR-61, 2001.

Toman D., Expiration of Historical Databases. Proceedings of the 8th International Symposium on Temporal Representation and Reasoning (TIME'01): 128-135, IEEE Press, 2001.

Vitter J. S., Random Sampling with a Reservoir. ACM Transactions on Mathematical Software, 11(1): pages 37-57, March 1985.

Annexe : Grammaire du langage

Les mots clés sont en majuscules, les constructions entourées de '[' et ']' sont facultatives, celles suivies de '+' doivent apparaître au moins une fois. Le symbole '|' signifie autre forme possible. Les éléments dont le nom commence par *id* représentent une séquence alphanumérique quelconque.

<fonction_oubli> ::= SUMMARY TABLE <nom_table> {
    <liste_use_join_table>;
    TIMESTAMP = SYSTEM| <nom_colonne> ;
    <specif_discretise>;
    <specif_hierarchie>;
    <specif_detail>;
    <liste_specif_agregation>;
    <specif_KEEP>;
}
END SUMMARY ;
<nom_table> ::= *id*Att
<liste_use_join_table>::=<use_join_table>|<liste_use_join_table>';'<use_join_table>
<use_join_table> ::= /* empty */ | USE <nom_table > JOIN BY <colonnes>
<colonnes> ::= <nom colonne>| <colonnes>',' <nom colonne>
<nom colonne> ::= *id*Att
<specif_discretise> ::= /* empty */ | <discretisation>| <specif_discretise>';' <discretisation>
<discretisation> ::= DISCRETISE '('<nom colonne>')'= *id*Att:'('(<intervalle> *id*Att;)$^{+}$')'
<intervalle> ::= ([|])<valeur numerique>, <valeur numerique>([|])
<valeur numerique> ::= ([+] | -)<nombre>

<nombre> ::= <chiffre>| <nombre><chiffre>
<chiffre> ::= 0| 1| 2| 3| 4| 5| 6| 7| 8| 9
<specif_hierarchie> ::= /* empty */ | <hierarchie>| <specif_hierarchie>';'<hierarchie>
<hierarchie> ::= HIERARCHY '('<nom dimension>')' ':' '('<niveaux hierarchie>')'
<nom dimension> ::= *id*Att
<niveaux hierarchie> ::= <niveau hierarchie>| <niveaux hierarchie> '→' <niveau hierarchie>
<niveau hierarchie> ::= *id*Att
<specif_detail> ::= /* empty */ | <critere>':'DETAIL
<liste_specif_agregation>::= <specif_agregation>| <liste_specif_agregation>';' <specif_agregation>
<specif_agregation> ::= <critere>':'<agregation donnees>
<critere> ::= LESS THAN <valeur> <niveau temps>
<valeur> ::= <chiffre>|<valeur>< chiffre>
<niveau temps > ::= SECOND| MINUTE| HOUR| DAY| MONTH| QUARTER| YEAR
<agregation donnees> ::= <mesures> <liste BY>
<mesures> ::= <agregation>| <mesures>','<agregation>
<agregation> ::= COUNT'(' '*' ')'| <AGG>'('<nom colonne>')'
<AGG> ::= COUNT| SUM| MIN| MAX| AVG
<liste BY> ::= /* empty */| BY <liste niveaux>
<liste niveaux> ::= <niveau>| <liste niveaux>','<niveau>
<niveau> ::= [<nom table>'.']<nom colonne>| <nom discretise>| <niveau temps>
<specif_KEEP> ::= /* empty */ | KEEP SAMPLE '('<valeur>')' [WHERE <condition>]
<condition> ::= <expression>| <condition> <op logique>
<expression> ::= [<nom table>'.']<nom colonne> <op> <expression simple>
<op> ::= <| >| ≥ | ≤ | =| <>
<expression simple> ::= *id*Att| <valeur numerique>
<op logique> ::= OR| AND

Summary

The amount of data stored in data warehouses grows very quickly so that they can get saturated. To overcome this problem, we propose a language for specifying forgetting functions on stored data. In order to preserve the possibility of performing interesting analyses of historical data, the specifications include the definition of some summaries of deleted data. These summaries are aggregates and samples of deleted data and are kept in the data warehouse. Once forgetting functions have been specified, the data warehouse is automatically updated in order to follow the specifications. This paper presents both the language for specifications, the structure of the summaries and the algorithms to update the data warehouse.