OLAP : Intégration des informations

N. Spyratos
Université Paris-Sud

Le besoin : collecter des informations à partir des plusieurs sources dans le but d'exploiter leur synthèse

Le problème : les sources peuvent être autonomes et hétérogènes
autorisations nécessaires pour l'extraction
utilisation de plusieurs langages pour la définition de l'information à extraire

La solution : utiliser un *schéma fédérateur*, avec ou sans données, que les utilisateurs manipulent comme s'il s'agissait d'une base de données habituelle
- avec données courantes provenant d'une seule source (*vues matérialisées*) ou de plusieurs sources autonomes et éventuellement hétérogènes (*base de données intégrée*)
  →besoin d'outils pour la conception du schéma, transformation/chargement de données et propagation de changements
- avec données historiques (*entrepôt de données*, données en avance), **c'est notre cas**
  →besoin d'outils pour la conception du schéma, transformation/chargement de données, propagation de changements et rafraîchissement périodique
- sans données, au sein d'une seule source (*vues virtuelles*) ou lié à plusieurs sources autonomes et éventuellement hétérogènes (*médiateurs*, données à la demande)
  →besoin d'outils pour la récriture/évaluation des requêtes et la fusion des résultats

dans tous les cas, l'accès aux données se fait presque exclusivement en lecture

Constitution d'un entrepôt de données

Architecture à trois niveaux : sources – entrepôt – data mart (voir figure 1)
→ data mart : 'petit' entrepôt orienté sujet, dont les données sont dérivées de l'entrepôt

Extraction, Transformation, Chargement de données → outils ETL
Extracteur :
- traduction vers le langage source, évaluation, traduction vers le langage de l'entrepôt
- détection de changements aux sources

Intégrateur :
- réconciliation /correction d'erreurs/filtrage/estampillage pour conformer au schéma de l'entrepôt
- chargement de données, rafraîchissement

Caractéristiques principales des entrepôts de données

Un entrepôt de données est l'endroit où les données intégrées sont stockées et exploitées à l'aide d'un système de gestion de bases de données. Par conséquent, un entrepôt de données est avant tout une base de données, même si les caractéristiques suivantes le distinguent clairement des bases de données transactionnelles habituelles :

Utilisation :
les utilisateurs principaux sont les décideurs de l'entreprise
→ besoin des schémas faciles a lire (pas de schémas normalisés) ⇒ *schéma dimensionnel*

(voir Figure 2)

Mode d'accès :
souvent à travers un data mart, en lecture uniquement
→ des index qui ne sont pas efficaces pour le transactionnel le deviennent pour les entrepôts

Volume :
de l'ordre de tera octets
→ besoin d'algorithmes efficaces pour le chargement de données et l'évaluation des requêtes

Maintenance :
les mises à jour sont propagées des sources vers l'entrepôt
- immédiatement ou périodiquement (suivant la nature de l'application)
- par reconstruction ou de manière incrémentale

Rafraîchissement :
les données qui deviennent 'obsolètes' sont supprimées

Metadonnées :
Les métadonnées d'un entrepôt sont plus complexes et plus volumineuses que celles d'une base de données transactionnelle, et sont souvent gérées en dehors de l'entrepôt

Utilisation d'un entrepôt de données

- OLAP : Online Analytic Processing
  → résumés/agrégations selon plusieurs critères et sur plusieurs dimensions
- Génération de rapports
- Extraction de connaissances à partir de données (Fouille de données, 'Data Mining')

Plan du cours :

algèbre fonctionnel
→ définition, opérations, propriétés

le modèle fonctionnel pour l'analyse dimensionnelle de données
→ schéma dimensionnel et base de données dimensionnelle
langage de chemins et langage OLAP, schémas arborescents

analyse dimensionnelle en relationnel (ROLAP) :
→ représentation du modèle fonctionnel en relationnel, extensions du SQL, génération de rapports, mécanisme de vues et algorithmes incrémentaux de maintenance

aspects physiques :
→ index bitmap, index de jointure

exemples d'extraction de connaissances à partir de données :
→ règles d'association et algorithme A-priori, classification supervisée/non supervisée

Supports : Polycopié + Article (en anglais)

Contrôle de connaissances : Examen écrit (documents autorisés) + projet

TD-1 : Algèbre fonctionnelle

Rappelons d'abord qu'une fonction est désignée par f : X→Y, où f est le nom de la fonction, X son ensemble de départ et Y son ensemble d'arrivée. Nous noterons def(f) l'ensemble des éléments de X pour lesquels f est définie et range(f) l'ensemble des valeurs prises par f. Une fonction f est appelée :

fonction totale (ou application) si def(f)= X (sinon, f est appelée *fonction partielle*)
fonction injective (ou injection) si x≠x' implique f(x)≠f(x'), pour tous x,x' dans def(f)
fonction surjective (ou surjection) si range(f)= Y
fonction bijective si elle est à la fois injective et surjective.

Dans la suite, nous nous intéresserons uniquement aux fonctions totales, et l'algèbre fonctionnelle dont nous aurons besoin comportera quatre opérations :

Composition : prend en argument deux fonctions, f et g, telles que range(f)⊆def(g), et renvoie une fonction $g_o f$: def(f)→range(g) définie par $g_o f$ (x)= g(f(x)), x∈def(f)
Couplage : prend en argument deux fonctions, f et g, telles que def(f)=def(g), et renvoie une fonction f∧g : def(f)→range(f)×range(g) définie par f∧g(x)= <f(x), g(x)>, x∈def(f)
Projection : l'opération habituelle sur le produit de deux ou plusieurs ensembles

Restriction : prend en argument une fonction f : X→Y et un ensemble E⊆def(f), et renvoie une fonction f/E : E→Y définie par f/E(x)= f(x) pour tout x dans E

Proposition (cf figure 2)
Soit deux fonctions f : X→Y et g : X→Z, alors f= $\pi_{Y\,o}(f\wedge g)$ et g= $\pi_{Z\,o}(f\wedge g)$
(π_Y et π_Z sont des projections sur Y×Z)

Propriétés des inverses
Rappelons d'abord que l'inverse (ou réciproque) d'une fonction f : X→Y, noté f^{-1}, est définie comme suit : $f^{-1}(y)= \{x / f(x)= y\}$; il associe chaque élément y de range(f) à l'ensemble des éléments x de def(f) ayant y comme image (et donc chaque élément y n'appartenant pas a range(f) à l'ensemble vide). Voici quelques propriétés des inverses :
composition : $(g_o f)^{-1}(z)= \cup\{f^{-1}(y) / y\in g^{-1}(z)\}$, pour tout z∈range($g_o f$)
couplage : $(f\wedge g)^{-1}((y, z))= f^{-1}(y)\cap g^{-1}(z)$
restriction : $(f/E)^{-1}(y) = E\cap f^{-1}(y)$, pour tout y∈range(f/E)

Exercice : Pour les fonctions données en table 1, donner les résultats de calculs suivants :
1/ $g_2 o g_1$, $g_1 o g$, $g_2 o (g_1 o g)$, $(g_2 o g_1) o g$
2/ f∧g, g∧h, (f∧g)∧(g∧h), f∧(g∧h), (f∧g)∧h, $(g_1 o g)\wedge(h_1 o h)$, $f\wedge(g_2 o g_1 o g)\wedge(h_2 o h)$, g∧g
3/ vérifier si g= $\pi_{Store\,o}(g\wedge h)$ et h= $\pi_{Product\,o}(g\wedge h)$
4/ g/E, h/F, (g/E)∧(h/F), où E= {1, 2, 4, 5, 7, 8} et F= {2, 3, 4, 6, 7, 9}
5/ g^{-1}, $(g_1 o g)^{-1}$, $((g_2 o g_1) o g)^{-1}$, $(g\wedge h)^{-1}$, $((g_1 o g)\wedge(h_1 o h))^{-1}$, $(g/E)^{-1}$, $(h/F)^{-1}$
Quelles conclusions en tirez-vous concernant les propriétés des opérations de l'algèbre?

TD-2 : Expressions de chemin et langage OLAP

Schéma dimensionnel (voir figure 2)

Un schéma dimensionnel est un graphe connexe, orienté, acyclique, étiqueté tel que :
une seule racine, appelé *l'origine* et notée O
un sommet distingué, appelé *l'unité* et noté ⊥
chaque sommet A est associé à un domaine, noté dom(A) ; dom(⊥) est singleton
il n'y a aucune flèche avec ⊥ comme source
les étiquettes de toutes les flèches sont distinctes
les flèches de source O sont d'un de deux types, *dimension* ou *mesure*
il y a une dimension de source O et de cible ⊥, appelée *dimension unité* et notée ⊥ : O → ⊥
il n'y a pas de flèche entre les cibles de deux flèches de dimension
Terminologie : la notation f : X→ Y indiquera une *flèche* avec *étiquette* f, *source* X et *cible* Y dans le schéma; de même, on parlera de la *source* et de la *cible* d'un chemin dans le schéma

chaque sommet autre que l'origine est appelé *attribut*
les valeurs du domaine de O sont appelées des *objets*
la distinction entre dimensions et mesures est faite par le concepteur
tout schéma dimensionnel doit contenir au moins une mesure
chemin dimensionnel : tout chemin de source O commençant par une dimension
→ tout attribut d'un tel chemin est appelé un *niveau d'agrégation*
chemin de mesure : tout chemin de source O commençant par une mesure
→ tout attribut d'un tel chemin est appelé un *niveau de mesure*
la figure 2 montre un schéma dimensionnel où f, g, h sont les dimensions et m est la mesure

Base de données dimensionnelle (bdd)

Etant donné un schéma dimensionnel S, une bdd sur S est une fonction δ qui associe chaque sommet A de S à un sous-ensemble fini δ(A) du domaine de A, la flèche ⊥ à une fonction constante et chaque autre flèche f : X→Y de S à une fonction totale δ(f): δ(X)→δ(Y)

- les fonctions d'une bdd peuvent être données de manière explicite ou implicite
- dans la suite, nous omettrons le symbole δ, le contexte indiquant s'il s'agit d'une flèche ou d'une fonction
- le fait que les fonctions d'une bdd soient totales entraîne la contrainte suivante :

contrainte référentielle : range(f)⊆def(g) pour toutes fonctions f : X→Y et g : Y→Z

- le passage à un niveau supérieur induit un regroupement ou «agrégation» au niveau inférieur

Expression de chemin

Etant donné un schéma dimensionnel S, une expression de chemin sur S est une expression bien formée dont les opérandes sont des flèches de S et dont les opérateurs sont ceux de l'algèbre fonctionnelle. De manière plus formelle, une expression de chemin e sur S est définie par la grammaire suivante, où le symbole '::=' signifie 'peut être' et où p, q, e_1,… , e_j désignent des expressions de chemin :

e ::= f, où f : X→Y est une flèche de S ; source(e)= X et target(e)= Y
$q_{\circ}p$, où target(p)= source(q) ; source(e)= source(p) et target(e)= target(q)

$p \wedge q$, où source(p)=source(q) ; source(e)= source(p) et target(e)= target(p)×target(q)

p/E, où E⊆source(p) ; source(e)= E et target(e)= target(p)

$\pi_X(e_1 \wedge \ldots \wedge e_j)$, où $X= \{X_1, \ldots, X_r\} \subseteq \{target(e_1), \ldots, target(e_j)\}$;
source(e)= $target(e_1) \times \ldots \times target(e_j)$ et $target(e)= X_1 \times \ldots \times X_r$

expression dimensionnelle sur S : toute expression de chemin constituée uniquement de flèches figurant sur des chemins dimensionnels

expression de mesure sur S : toute expression de chemin constituée uniquement de flèches figurant sur un chemin de mesure

évaluation d'une expression de chemin e par rapport a une base δ sur S : se fait en remplaçant chaque flèche f figurant dans e par la fonction δ(f) qui lui est associée par δ, et en effectuant les opérations de l'algèbre fonctionnelle (le résultat étant toujours une fonction, donc propriété de fermeture, comme en relationnel); de manière plus formelle cette évaluation, notée eval(e, δ), est définie comme suit :

eval(e, δ)= f, si f est une flèche de S ;
$\delta(q) \circ \delta(p)$, si $e= q \circ p$
$\delta(p) \wedge \delta(q)$, si $e= p \wedge q$
δ(p)/E, si e= p/E
$\pi_X(\delta(e_1) \wedge \ldots \wedge (\delta(e_j))$, si $e= \pi_X(e_1 \wedge \ldots \wedge e_j)$

vue sur S : schéma dimensionnel S'dont chaque sommet est un sommet de S et donc chaque flèche est une expression de chemin sur S (un *data mart* est définie comme une vue sur S, pouvant être virtuelle ou matérialisée suivant les besoins de l'application)

Requête OLAP

Etant donné un schéma dimensionnel S, un 'OLAP Pattern' sur S est un couple P= (u, v), où u est une expression dimensionnelle, v une expression de mesure et source(u)= source(v)= O. La cible de u est appelée le *niveau d'agrégation* de P et la cible de v le *niveau de mesure* de P. Notons que, si u comporte des couplages, alors A peut être le produit de plusieurs attributs, c'est-à-dire, le niveau d'agrégation peut être composé de plusieurs autres niveaux 'simples' (et de même pour M).

Une *requête OLAP* sur S est alors un couple Q= <P, *op*>, où P est un OLAP pattern et *op* est une opération applicable sur le domaine du niveau de mesure de P. Si A est le niveau d'agrégation de P et M son niveau de mesure, alors la *réponse* à Q par rapport à une base δ sur S est définie comme une fonction $ans_{Q,\delta} : \pi_u \rightarrow A \times M$, où π_u est la partition induite par u sur O, c'est-à-dire $\pi_u= \{u^{-1}(y)/\ y \in range(u)\}$. Cette fonction est calculée comme suit :

1/ évaluer les expressions u et v, par rapport à la base δ ; nous obtenons alors deux fonctions, que nous notons également u et v ;
2/ calculer l'inverse $u^{-1}(y)$; supposons que $u^{-1}(y) = \{o_1, \ldots, o_r\}$ et notons que $u^{-1}(y) \in \pi_u$;
3/ calculer les images $v(o_i)$, i=1,..,r, obtenant ainsi le n-uplet $t(y)= < v(o_1), \ldots, v(o_r)>$;
4/ calculer le résultat de l'application de *op* sur t(y) soit RES(y)= *op*(t(y)) ;
5/ définir $ans_{Q,\delta}(u^{-1}(y))= (y,\ RES(y))$;
6/ exécuter les étapes 2 à 5 pour tous les y dans range(u).

Notons que RES est un 'attribut auxiliaire', qui ne figure pas dans le schéma (il s'agit d'un nom de variable, donné par l'utilisateur afin d'y placer le résultat des calculs ci-dessus).

Un exemple de requête OLAP et de son évaluation est donné par la table 1bis, où la réponse $ans_{Q,\delta}$ est présentée sous forme d'un ensemble de triplets $<u^{-1}(y), y, RES(y)>$.

Notons qu'en pratique, les utilisateurs s'intéressent souvent uniquement aux couples <y, RES(y)>, et non pas aux blocs d'objets $u^{-1}(y)$, qui a servit à leurs calcul. Cependant, du point

de vue du système, la connaissance de $u^{-1}(y)$ est utile, car elle peut servir à des optimisations de calcul lorsque deux ou plusieurs requêtes partagent la même expression dimensionnelle u. En effet, dans ce cas, les requêtes partagent également les $u^{-1}(y)$, par conséquent, leurs évaluations peuvent partager les résultats des étapes 1 et 2. Si, en plus, deux ou plusieurs requêtes partagent également la même expression de mesure v, alors nous pouvons utiliser un 'raccourci de notation', à savoir, écrire $Q= <P, op_1, op_2, ..,op_n>$ pour designer l'ensemble suivant de requêtes : $Q_1= <P, op_1>$, $Q_2= <P, op_2>$, …, $Q_n= <P, op_n>$.
Revenant aux couples <y, RES(y)>, qui intéressent l'utilisateur, nous constatons que chaque y dans range(u) est associé à un résultat RES(y) – et un seul. Il s'ensuit que cette association est une fonction RES : range(u)→M. Notons que, si A est composé de n niveaux d'agrégation simples, et M de k niveaux de mesure simples, alors cette fonction RES peut être visualisée sous forme d'un cube à n dimensions, où chaque n-uplet y de range(u) est visualisé comme une *cellule*, et le k-uplet RES(y) associé à y est visualisé comme le contenu de cette cellule. Cette représentation de la fonction RES est connue sous le nom de *data cube* associé à Q et δ, ou encore de *rapport* associé à Q et δ. Cependant, comme la fonction RES est définie sur range(u), et non pas sur A, il se peut que plusieurs cellules de ce cube n'aient pas de contenu. D'autres représentations de la fonction RES sont également possibles, et plusieurs outils de 'visualisation' sont actuellement disponibles pour la réalisation de telles représentations.

→ notion de sous-cube calculable à partir d'un cube déjà calculé

Le cas u=⊥
Il s'agit d'une forme particulière de requête OLAP, à savoir $Q= <(\perp, v), op>$. Ce cas présente un intérêt certain en pratique, car l'étape 1 ci haut renvoie une fonction constante (soit y sa seule valeur), l'étape 2 renvoie $u^{-1}(y)= O$, l'étape 3 renvoie les images par v de tous les objets de O, l'étape 4 applique l'opération op sur toutes ces images pour obtenir un seul résultat RES(y), l'étape 5 associe O au seul couple (y, RES(y)) et, par conséquent, la réponse $ans_{Q,\delta}$ est, elle aussi, une fonction constante (les calculs 1 à 5 étant effectués une seule fois). Par exemple, l'évaluation de la requête $Q= <(\perp, m), Sum>$ sur le schéma dimensionnel S de la figure 2, par rapport à la base de la table 1bis, renvoie $ans_{Q,\delta}(O)= (y, 3200)$, ce qui représente la somme de toutes les quantités de produits figurant dans la base.

OLAP sur des schémas arborescents
Dans les applications pratiques le schéma dimensionnel est souvent un arbre, et dans ce cas, afin de spécifier une requête OLAP, il suffit de donner les attributs constituant le niveau d'agrégation, ceux constituant le niveau de mesure, et les restrictions souhaitées sur les attributs. On peut même imaginer la mise en place d'une interface conviviale, où la spécification d'une requête OLAP se fait comme suit :
- spécification du niveau d'agrégation : 'cliquer' sur les attributs souhaités;
- spécification du niveau de mesure : 'cliquer' sur les attributs souhaités;
- spécification de restrictions : sous forme d'expressions <attribut> = <ensemble-de-valeurs>;
- spécification des opérations : 'cliquer' sur les opérations souhaitées dans un menu;
- specification de sortie : donner la variable qui contiendra le résultat (RES dans notre exemple).

Par exemple, sur le schéma de la figure 2 (qui est un arbre), la requête OLAP demandant les totaux par ville et produit, uniquement pour les villes de Paris et de Nice, et pour les produits P1 et P2 serait spécifiée comme suit:
- niveau d'agregation : cliquer sur City et Product

- niveau de mesure : cliquer sur Sales
- restrictions : City={Paris, Nice}, Product={P1, P2}
- operations: Sum
- variable de sortie: RES

Exercice

Un distributeur (grossiste) approvisionne plusieurs magasins en produits, en effectuant au plus une livraison par jour et par magasin. Les informations qui figurent sur chaque bon de livraison sont les suivantes : le numéro du bon de livraison, la date de livraison, la référence du magasin, et pour chaque type de produit livré sa référence et la quantité livrée (le nombre d'articles). Ces informations sont stockées chez le distributeur, et accumulées pendant des longues périodes afin de les analyser pour améliorer le service de distribution.
Les analyses se font suivant plusieurs axes, et à plusieurs niveaux, en analysant les mouvements des produits par jour et par mois, par ville et par région, par fournisseur et par catégorie de produit.
On supposera qu'un fournisseur peut fournir au distributeur des produits dans plusieurs catégories et qu'une catégorie de produit peut être fournie par plusieurs fournisseurs.
Nota : Souvent, la référence du magasin contient l'information sur la ville et la région où se trouve le magasin, et de même, la référence du produit contient l'information sur le fournisseur et la catégorie du produit.

1/ Se convaincre que le schéma dimensionnel S vu en cours (voir figure 2) convient comme interface pour les analystes.

2/ Pour chacune des expressions de chemin suivantes sur S, dire si elle est bien formée (on dit aussi *bien typée*), et si oui, donner la source, la cible, et le résultat de son évaluation sur la base dimensionnelle donnée en table 1 :
$e_1 = f \wedge g \wedge h$
$e_2 = g_1 \circ (g \wedge f)$
$e_3 = (g_1 \circ g) \wedge f$
$e_4 = (g_1 \circ g) \wedge (f_1 \circ f) \wedge (h \circ (h_1 \wedge h_2))$

3/ Définir et évaluer les requêtes OLAP suivantes (pour le schéma de la figure 2 et la base de la table 1) :
Q_1 : Les totaux de quantités livrées par type de produit
Q_2 : Les totaux de quantités livrées par catégorie
Q_3 : Les minimaux de quantités livrées par ville
Q_4 : Les moyennes de quantités livrées par ville et type de produit
Q_5 : Les totaux de quantités livrées par catégorie et ville
Q_6 : Les maximaux de quantités livrées par date et catégorie
Q_7 : Les totaux de quantités livrées par région et par mois
Q_8 : Les totaux de quantités livrées par fournisseur, magasin et mois
Q_9 : La somme de toutes les quantités livrées

4/ Définir le schéma d'un data mart permettant des analyses mensuelles par ville et catégorie de produits, et concernant uniquement les villes de Paris et de Lyon. Dans le contexte de ce data mart, définir la requête suivante et donner sa traduction en une requête sur l'entrepôt.
Q : Les totaux de quantités livrés par catégorie de produits, pendant le premier trimestre

OLAP en relationnel (ROLAP)

Dans ce paragraphe nous ferons les hypothèses simplificatrices suivantes :
- le schéma dimensionnel est un arbre ;
- les niveaux de mesure sont tous des niveaux de base
- les restrictions sont autorisées uniquement au niveau de l'agrégation de la requête OLAP

Nous pouvons représenter une base dimensionnelle sous forme d'une base relationnelle, sans perte d'information, si la condition suivante est remplie :
contrainte dimensionnelle : si f_1, f_2, .., f_n sont les flèches de dimension alors la fonction $f_1 \wedge f_2 \wedge .. \wedge f_n$ est injective
Si cette contrainte est satisfaite par la base dimensionnelle, alors chaque identificateur d'objet o est représenté par le n-uplet d'images $< f_1(o), f_2(o), .., f_n(o) >$, qui jouent le rôle de 'coordonnées' pour o, en ce sens que si $o \neq o'$ alors il existe f_i tel que $f_i(o) \neq f_i(o')$.

Nota : Si la contrainte référentielle n'est pas satisfaite alors nous pouvons toujours la satisfaire en ajoutant un ou plusieurs flèches de dimension supplémentaires.

Etant donnée une base δ sur un schéma dimensionnel S, et en supposant que la contrainte référentielle est satisfaite par δ, les tables de sa représentation relationnelle sont définies comme suit:

Table de faits FT(S)
- Les attributs de FT(S) sont tous les attributs de dimension, soit D_1, D_2, .., D_n, et tous les attributs de mesure, soit M_1, M_2, .., M_k, chacun avec le domaine qu'il a dans S ;
- L'unique clé de FT(S) est $\{D_1, D_2, .., D_n\}$
- Pour chaque $o \in O$ le n-puplet $< f_1(o), f_2(o), .., f_n(o), m_1(o), m_2(o), .., m_n(o) >$ est dans FT(S)

Tables de dimension $DT_i(S)$, i=1, 2, .., n
- Pour chaque dimension D_i, définir une table $DT_i(S)$ avec comme attributs tous les niveaux de D_i, chacun avec le domaine qu'il a dans S ;
- Chaque flèche $D \rightarrow D'$entre niveaux de la dimension D_i devient une dépendance fonctionnelle de $DT_i(S)$
- Effectuer une jointure relationnelle de toutes les tables binaires contenant les fonctions associées aux flèches de la dimension D_i, et placer le résultat dans $DT_i(S)$

++++++++++++++++
Dans la suite, nous noterons FT au lieu de FT(S), et DT_i au lieu de $DT_i(S)$. Les tables FT, DT_1, .., DT_n définies plus haut constituent la représentation en relationnel de la base dimensionnelle δ, et l'ensemble de ces tables est connu sous le nom *schéma en étoile*. Notons que la table de faits FT est en FNBC alors que, en général, les tables de dimension ne le sont pas; la décomposition de ces dernières afin d'obtenir un schéma en FNBC, conduit a ce qu'on appelle un 'schéma en flocons'. La table 2 montre le schéma en étoile représentant la base dimensionnelle de la table 1bis.

La question maintenant est de savoir comment une requête OLAP Q= (<u, v>), *op*), exprimée sur le schéma dimensionnel S, peut être représentée par une expression relationnelle équivalente sur le schéma en étoile (c'est-à-dire renvoyant toujours la même réponse que Q). Pour cela il faudra d'abord savoir comment une expression de chemin e sur S, de source O,

peut être représentée par une expression relationnelle équivalente rep(e) sur le schéma en étoile. Si A désigne la cible de e alors rep(e) est définie comme suit:
si A est constitué uniquement de niveaux de base alors rep(e)= π_A(FT)
sinon si A contient des niveaux non de base appartenant aux dimensions D_{i1}, .., DT_{ik}
alors rep(e)= π_A(FT⋈DT_{i1}⋈..⋈DT_{ik})

Pour évaluer la requête OLAP Q= (<u, v>), *op*), il suffit maintenant de savoir comment évaluer sa représentation, rep(Q)= (<rep(u), rep(v)>, *op*), dans le schéma étoile. En supposant que A= {A_1,.., A_n}, et que la cible de v est M, ceci se fait à l'aide de l'instruction group-by suivant du SQL :

Select A_1,.., A_n ***op***(M) ***as*** RES
From ***join*** FT, DT_{i1},.., DT_{ik}
Group By (A_1,.., A_n)

Par exemple, la requête OLAP de la table 1bis sera exprimée comme suit sur le schéma en étoile de la table 2:

Select Store, Supplier ***sum***(Sales) ***as*** RES
From ***join*** FT, ProdT
Group By (Store, Supplier)

Dans les applications pratiques, les décideurs ont souvent besoin des plusieurs analyses différentes dans un 'contexte', c'est-à-dire dans un ensemble de dimensions. Par exemple, dans le contexte de deux dimensions Store et Supplier, un décideur peut avoir besoin des ventes totaux par Store, ensuite par Supplier, et ensuite par Store et Supplier; de plus, il peut avoir besoin des moyennes des ventes par Store, et des minimaux des ventes par Supplier. Etant donné les gros volumes de données manipulées lors de l'évaluation de ces requêtes, il serait intéressant de pouvoir les spécifier toutes ensembles, pour que le système puisse réutiliser certains résultats. Ces considérations ont conduit à l'introduction de l'extension suivante de la requête group-by :

Select A_1,.., A_n ***op***(M) ***as*** RES
From ***join*** T_1,.., T_k
Group By ***grouping sets*** ($liste_1$ .. $liste_k$)

Chacune des listes $liste_1$ .. $liste_k$ contient zero, un ou plusieurs attributs parmi ceux qui figure dans la clause Select. Cette instruction étendue est équivalente à l'ensemble de k instructions group-by ordinaires formées en utilisant les listes $liste_1$ .. $liste_k$. Par exemple, soit l'instruction étendue suivante :

Select Store, Supplier ***sum***(Sales) ***as*** RES
From ***join*** FT, ProdT
Group By ***grouping sets*** ((Store), (Supplier), (Store, Supplier))

Cette instruction est équivalente à l'ensemble de trois instructions group-by suivantes:

Select Store ***sum***(Sales) ***as*** RES

From FT
Group By (Store)

Select Supplier ***sum***(Sales) ***as*** RES
From ***join*** FT, ProdT
Group By (Supplier)

Select Store, Supplier ***sum***(Sales) ***as*** RES
From ***join*** FT, ProdT
Group By (Store, Supplier)

Le résultat de l'évaluation de ces trois instructions est représenté dans une seule table avec valeurs nulles, dont les attributs sont Store, Supplier et RES. Notons que l'ordre des attributs dans chaque liste, ainsi que l'ordre des listes n'est pas important.

Il existe également deux variantes de l'extension grouping-sets, l'instruction Rollup et l'instruction Cube, que nous allons expliquer par deux exemples :

Select Store, Supplier ***sum***(Sales) ***as*** RES
From ***join*** FT, ProdT
Group By ***rollup*** (Store, Supplier)

Cette instruction est équivalente à l'instruction grouping-sets suivante:

Select Store, Supplier ***sum***(Sales) ***as*** RES
From ***join*** FT, ProdT
Group By ***grouping sets*** ((Store, Supplier), (Store), ( ))

Notons que les listes de grouping sets sont engendrées en dépilant la liste rollup de droite à gauche, un attribut à la fois. Par conséquent, l'ordre des attributs à l'intérieur de la liste rollup est important, à savoir un changement d'ordre conduit à une instruction rollup différente. Quant à l'instruction Cube, en voici un exemple :

Select Store, Supplier ***sum***(Sales) ***as*** RES
From ***join*** FT, ProdT
Group By ***cube*** (Store, Supplier)

Cette instruction est équivalente à l'instruction grouping-sets suivante:

Select Store, Supplier ***sum***(Sales) ***as*** RES
From ***join*** FT, ProdT
Group By ***grouping sets*** ((Store, Supplier), (Store), (Supplier), ( ))

Notons que les listes de 'grouping sets' sont engendrées par l'ensemble des parties de l'ensemble de tous les attributs de la clause Select figurant avant l'opération (avant *sum*, dans notre exemple). Par conséquent, l'ordre des attributs dans la liste 'cube' n'est pas important.

Indexing

The fact that data warehouses are read-only databases, allows for the use of new forms of indexing that would probably not be efficient for transactional databases. In this section we shall see briefly two such indexing schemes, the bitmap index and the join index.

The bitmap index

This index is useful when there is little or no updating of the database, and it is created over attributes whose active domain is of small size. Here, by "active domain" we mean the set of values actually present in the attribute column. We shall explain how a bitmap index is defined using the table T below, where we assume that the domain of the attribute Rating contains five different values.

T

CustId	Name	Sex	Rating
112	Joe	M	3
125	Ram	M	5
139	Sue	F	5
167	Woo	M	4

SexIndex

M	F
1	0
1	0
0	1
1	0

RatingIndex

1	2	3	4	5
0	0	1	0	0
0	0	0	0	1
0	0	0	0	1
0	0	0	1	0

Of the attributes in this table, Sex and Rating have a small active domain; therefore we shall create bitmap indexes over these two attributes. The bitmap index of Sex, is a binary table SexIndex(M, F), whose attributes are the values in the domain of Sex. This table has as many rows as there are rows in T, and each row is defined as follows: for each row of T if the row contains M then the corresponding row of SExIndex contains 10, and if the row of T contains F then the corresponding row of SexIndex contains 01. The table RatingIndex is defined similarly.

To see how these indexes are used in query processing, consider the following query: how many male customers have a rating of 5. To answer this query, we AND the M-column of SexIndex and the 5-column of the RatingIndex, and then count the number of 1s in the result. Clearly, we can use any combination of Boolean operations allowed on bit vectors (not just AND), depending on the query.

We note that the active domain of an attribute can be of small size either because of the nature of the attribute (e.g. Sex), or because it has been "summarized". For example, the domain of the attribute Salary might be "summarized" to contain only three values: the value LowSal meaning salaries less than 2000, the value AvgSal meaning between 2000 and 4000, and the value HighSal meaning more than 4000.

The join index

The join index allows to exploit the result of a join without actually performing the join. We shall explain this indexing scheme using the tables below, where we have added tuple identifiers.

R S JoinIndex JoinIndexSet

	A	B
1	a1	b1
2	a1	b2
3	a2	b1
4	a3	b2
5	a3	b4
6	a2	b5

	B	C
7	b1	c1
8	b1	c2
9	b2	c3
10	b3	c1
11	b3	c2
12	b1	c3

R	S
1	7
1	8
1	12
2	9
3	7
3	8
3	12
4	9

R-set	S-set
{1, 3}	{7, 8, 12}
{2, 4}	{9}

The join index for the tables R and S is actually a new table JoinIndex(R, S), whose attributes are the tables R and S. Both attributes R and S have the same domain, which is the set of tuple identifiers (here denoted as integers). As for the contents of JoinIndex, they are defined as follows : a tuple <i, j> of identifiers is in the JoinIndex if and only if the tuple i is in R, the tuple j is in S and their B-values are equal. For example, the tuple <1, 7> is in JoinIndex, because the tuple number 1 of R and the tuple number 7 of S have the same B-value (i.e., b1).

To see how the JoinIndex is used in query processing, consider the following query, expressed in the relational algebra: join R and S then project over A and C. We can find the result *without* performing the join, as follows: for each tuple <i, j> of JoinIndex, put the tuple <i.A, j.C> in the result. Here, i.A stands for the A-value of tuple number i from R, and j.C stands for the C-value of tuple number j from S.

We note that the join index can be implemented in more compact forms, such as the JoinIndexSet depicted above. In that representation, the attributes R-set and S-set have sets of tuple identifiers as values, and a row <I, J> of JoinIndexSet means that any pair of identifiers from I and J denote “joinable” tuples from R and S. Note that if we take the Cartesian product of each tuple <I, J> in JoinIndexSet and then the union of the results, we find exactly the set of tuples of the JoinIndex.

Knowledge Discovery in Databases (KDD)

Query languages help the user extract, or "discover" information that is not explicitly stored in a database. For example, in a transactional database containing the tables E(Emp, Dep) and M(Dep, Mgr), one can join the two tables and then project the result over Emp and Mgr to extract a new table: (Emp, Mgr), not explicitly stored in the database.

In a data warehouse, as we have seen, one needs to extract more sophisticated information for decision support purposes. For example, one can ask for the average monthly sales of products, per region and product category. The processing of such a query involves groupings and aggregate operations over large volumes of data, in order to provide the results necessary for report generation.

In recent years, decision support needs even more sophisticated data processing, whose result comes under various forms:
- *classification* of data items into pre-defined classes;
- *clustering* of data items, according to some similarity measure;
- *rules* describing properties of the data, such as co-occurrences of data values; and so on.

We also note that information extraction in the form of classification, clustering or rules has been studied since several years in statistics (exploratory analysis) and in artificial intelligence (knowledge discovery and machine learning). However, the data volumes of interest in those areas are relatively small, and the algorithms used do not scale up easily to satisfy the needs of decision support systems.

In what follows, we shall explain briefly how classification and clustering work, and then we shall see (in some detail) only one kind of rules, the so called "association rules".

Classification

Classification is the process of partitioning a set of data items into a number of disjoint subsets. These subsets are defined intentionally, in advance, usually by specifying the properties that each subset should satisfy. For example, consider the table T(ClId, Age, Sal) contained in the database of a bank. One can partition the set of tuples contained in this table into four disjoint subsets, defined by the following properties (or conditions):
(Age≤ 25) and (Sal≤ 2000)
(Age≤ 25) and (Sal> 2000)
(Age> 25) and (Sal≤ 2000)
(Age> 25) and (Sal> 2000)
These four subsets can be represented by a binary tree in which
- the root is labeled by the table T;
- each other node is labeled by a subset of tuples;
- each edge is labeled by the condition defining the subset represented by its end node.
In this representation, each subset is defined by the conjunction of the edge labels in the path leading to the leaf. We note that one or more subsets might be empty, depending on the current state of the table T.

Clustering

Clustering is the process of partitioning a set of data items into a number of subsets based on a "distance" (i.e. a function that associates a number between any two items): items that are "close" to each other are put in the same set, where closeness of two items is defined by the analyst. Thus, in the table T of our previous example, one might define a distance by:
dist(s, t)= abs(s.Sal-t.Sal), for all tuples s, t in T
Then one could cluster together all tuples s, t such that dist(s, t)≤ 1000. We note that the number of clusters is not known in advance and that no cluster is empty.

We note that the group-by instruction used in databases, and its extensions for data warehouses *do* perform classification and clustering, although of a limited form (i.e., using a specific set of criteria).

Association rules

Consider a (relational) table R whose schema contains n attributes A1, A2, .., An, such that dom(Ai)= {0, 1}, for all i; with a slight abuse of notation, we shall assume R= {A1, A2,…, An}. Consider also an instance of R, i.e., a set r of tuples over R (with tuple identifiers). An example of such a table is shown below, where the attributes are the items sold in a supermarket and one can think of each row as representing a transaction (a cashier slip): a 1 in the row, under item Ai means Ai was bought, and a 0 means the item was not bought, during transaction t.

Tid	**A1**	**A2**	**…**	**An**
.	.	.		.
.	.	.		.
t	1	0		1
.	.	.		.
.	.	.		.

An *association rule* over R is an expression of the form X⇒B, where X∈R and B∈(X\R). The intuition behind this definition is as follows: if the rule holds in r then the tuples having 1 under X tend to have 1 under B. As a consequence, such rules are useful in describing trends, thus useful in decision making. In this context, it is customary to use the term "item" instead of "attribute", and "itemset" instead of "attribute set". There are several application areas where association rules prove to be useful:

- In supermarket data collected from bar-code readers: columns represent to products for sale and each row represents to products purchased at one time.
- In databases concerning university students: the columns represent the courses, and each row represents the set of courses taken by one student.
- In citation index data: the columns represent the published articles, and each row represents the articles referenced by one article.
- In telephone network data collected during measurements: the columns represent the conditions, and each row represents the conditions that were present during one measurement.

We now introduce some concepts that are related to association rule extraction:

- Given a subset W of R, the *frequency* or *support* of W in r, denoted s(W, r), is the percentage of tuples in r having 1 under W, i.e.,

$$s(W, r)= |\{t\in r: t(A)=1 \text{ for all } A\in W\}|/|r|$$

We say that W is frequent in r if $s(W, r)\geq\alpha$, where α is a user-defined threshold. Clearly, if $W\subseteq Z\subseteq R$ then $s(W, r)\geq s(Z, r)$.

- Given an association rule $X\Rightarrow B$, the *frequency* or *support* of $X\Rightarrow B$ in r, and the confidence of $X\Rightarrow B$ in r, denoted $s(X\Rightarrow B, r)$ and $c(X\Rightarrow B, r)$, respectively, are defined as follows:

$$s(X\Rightarrow B, r)= s(X\cup\{B\}, r) \text{ and } c(X\Rightarrow B, r)= s(X\cup\{B\}, r)/s(X, r)$$

It follows that

$$0\leq c(X\Rightarrow B, r)\leq 1$$

The main problem here is to find *all* rules $X\Rightarrow B$ (i.e. for all X and B) whose support and confidence exceed given thresholds, minsup and minconf, given by the analyst. Such rules are called *interesting rules*. We stress the fact that, usually, no constraint is imposed on X or B, thus some of the extracted interesting rules might be unexpected.

Clearly, the search space for interesting rules is exponential in size, hence the need for efficient algorithms. In what follows, we describe such an algorithm and its implementation in SQL. The approach to design this algorithm uses two steps:

Step 1

find all frequent subsets X of R

Step 2

```
for each frequent set X
   for each B in X
      check whether (X\{B})B has sufficient high confidence
```

The approach also uses a basic fact which is easy to show:

- every subset of a frequent set is frequent

An immediate corollary is the following:

- if X has a non-frequent subset then X is not frequent

Based on this last result, we can now describe the basic computing steps of an algorithm for finding all frequent sets:

1. find all frequent sets of size i (we start with i=1);
2. use the frequent sets of size i to form candidates of size i+1
   (a candidate of size I is a set of cardinality i+1 whose each subset of size i is frequent);
3. among all candidates of size i+1 find all those that are frequent;
4. continue until no more candidates are found.

More formally, the algorithm can be stated as follows:

Algorithm FFS
(Find all Frequent Sets)

1. C:= {{A}: A∈R};
2. F:= ∅;
3. i:=1;
4. *while* C≠ ∅ *do begin*
5. F':= {X: X∈C and X is frequent};
6. F:=F∪F';
7. C:= {Y: |Y|=i+1 and for all W⊆Y with W=i, W is frequent}
8. i:=i+1; F':= ∅
9. *end*;

Clearly, the algorithm requires only one pass for step 5, thus the algorithm reads the relation r at most k+1 times, where k is the size of the largest frequent set. It follows that the worst-case complexity of the algorithm is k+1 (in terms of passes over the data).

As for the implementation of the algorithm over a relational DBMS, we note that the algorithm needs to compute supports of the form s({A1, A2, …, Am}). Each such support can be computed by a simple SQL query of the following form:

select count(*) *from* R *where* A1= 1 and A2= 1 and … and Am=1

Moreover, all these queries have the same form, thus there is no need to compile each one of them individually.

The above algorithm is a slightly modified version of the well known *A-priori Algorithm*, whose principles are found in one form or another in most software tools for the extraction of association rules from large tables.

Several remarks are in order here concerning association rules and their extraction. First, the table used for the extraction is usually *not* a relational table but can be derived from such a table. Moreover, the relational table itself can be either a database table or the result of a query to the database (in either case, it can be converted to a table where the extraction will take place). In this respect, we note that the process of information extraction from databases, better known today as Knowledge Discovery in Databases (KDD) is inherently interactive and iterative, containing several steps:
- understanding the domain;
- preparing the data set;
- discovering patterns (Data Mining);
- post-processing of discovered patterns;
- putting the results into use.

As for its application areas, they include the following:
- health care data (diagnostics);
- financial applications (trends);
- scientific data (data patterns);
- industrial applications (decision support).

Second, in typical applications of association rule extraction, the number of extracted rules might be quite large, from thousands to tens of thousands. In such cases the extracted association rules might be impossible to exploit. Several approaches have been proposed recently with the objective to lower the number of extracted rules :
- partial specification of X and/or B (i.e. constraints on X and/or B);
- reduced sets of rules (similar to reduced sets of dependencies);
- rule generalization (children clothing⇒dairy product instead of children pants⇒milk).

These approaches together with convenient and user-friendly visualizations of rules tend to render their use easier.

As a final remark, we would like to comment on the way association rules are likely to be used in practice. To this effect, let's consider an example. Suppose that the association rule *Pen⇒Ink* has been extracted as an interesting rule (i.e. one whose support and confidence exceed the pre-defined thresholds). Such a rule implies that customers buying pens have the tendency to also buy ink. One scenario in which this rule might be used is the following: assuming a large stock of unsold ink, the shop manager might decide to reduce the price of pens, in the hope that more customers would then buy pens (and therefore ink, following the tendency revealed by the extracted rule). However, such a use of the rule may lead to poor results, unless there is a *causal* link between pen purchases and ink purchases that somehow guarantees that the tendency will continue in the future as well. Indeed, association rules describe what the tendency was in the past, not what will be in the future: they are descriptive and *not* predictive rules. In other words, it may happen that, in the past, all customers bought ink not because they bought pens but simply because they usually buy all stationery together! If such is the case then the extracted rule *Pen⇒Ink* has been found to be interesting because of simple co-occurrence of pen purchase and pencil purchase and *not* because of the existence of a causal link!

In conclusion, although association rules do not necessarily identify causal links, they nevertheless provide useful starting points for identifying such links (based on further analysis and with the assistance of domain experts).

A JETER

Nous pouvons également procéder directement à la définition d'un schéma en étoile, à partir de l'ensemble U d'attributs de l'application, et de l'ensemble F de dépendances fonctionnelles sur U.

OLAP sur des schémas arborescents
Dans les applications pratiques les conditions suivantes sont souvent remplies, facilitant ainsi les notations concernant les requêtes OLAP :
- le schéma dimensionnel est un arbre
- les niveaux de mesure sont tous des niveaux de base
- les restrictions sont autorisées uniquement au niveau de l'agrégation de la requête

Dans ce cas, il suffit de donner les attributs constituant le niveau d'agrégation pour définir l'expression dimensionnelle u, et il n'est plus nécessaire d'indiquer l'expression de mesure v. Il s'ensuit que la spécification d'une requête OLAP peut se faire maintenant comme suit :

$$Q=<A_1/E_1, .., A_n/E_n, op>$$

où $A_1, .., A_n$ sont les attributs constituant le niveau d'agrégation, et $E_1, ..,E_n$ sont des sous-ensembles de $A_1, .., A_n$, respectivement, exprimant les éventuelles restrictions souhaitées. Par exemple, sur le schéma de la figure 2 (qui est un arbre), l'expression suivante est une requête OLAP bien définie, demandant les totaux par ville et produit, uniquement pour les villes de Paris et de Nice, et pour les produits P1 et P2 :

Q1= <City/{Paris, Nice}, Product/{P1, P2}, *Sum*>

Une écriture plus intuitive de cette même requête est la suivante:

Select City, Product, *Sum*(Sales) *as* RES
Having City={Paris, Nice} and Product={P1, P2}

Si aucune restriction n'est souhaitée alors la clause 'Having' n'est pas nécessaire :

Select City, Product, *Sum*(Sales) *as* RES

En conclusion, dans le cas d'un schéma arborescent, la spécification de requêtes est simplifiée, et on peut imaginer la mise en place d'une interface conviviale, où la spécification d'une requête OLAP se fait en deux étapes simples :
- 'cliquer' sur les attributs souhaités, en spécifiant pour chaque attribut les valeurs souhaitées;
- choisir une opération dans un menu.