Interrogation de données structurellement hétérogènes dans les bases de données orientées documents

Hamdi Ben Hamadou, Faiza Ghozzi, André Péninou, Olivier Teste

▶ **To cite this version:**

Hamdi Ben Hamadou, Faiza Ghozzi, André Péninou, Olivier Teste. Interrogation de données structurellement hétérogènes dans les bases de données orientées documents. Journées Francophones Extraction et Gestion de Connaissances (EGC 2018), Jan 2018, Paris, France. pp.155-166. hal-02864404

HAL Id: hal-02864404
https://hal.science/hal-02864404

Submitted on 11 Jun 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Open Archive Toulouse Archive Ouverte

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible

This is an author's version published in:
http://oatao.univ-toulouse.fr/22255

Official URL
https://editions-rnti.fr/?inprocid=1002430

To cite this version: Ben Hamadou, Hamdi and Ghozzi, Faiza and Péninou, André and Teste, Olivier *Interrogation de données structurellement hétérogènes dans les bases de données orientées documents.* (2018) In: Journées Francophones Extraction et Gestion de Connaissances (EGC 2018), 22 January 2018 - 26 January 2018 (Paris, France).

Any correspondence concerning this service should be sent to the repository administrator: tech-oatao@listes-diff.inp-toulouse.fr

Interrogation de données structurellement hétérogènes dans les bases de données orientées documents

Hamdi Ben Hamadou*, Faiza Ghozzi**
André Péninou*, Olivier Teste*

*IRIT, Université de Toulouse, UT3, UT2J, CNRS
118 Route de Narbonne - 31062 Toulouse, France
{hamdi.ben-hamadou, peninou, teste}@irit.fr
**Université de Sfax, ISIMS, MIRACL
Sakiet Ezzit 3021, Tunisie
faiza.ghozzi@isims.usf.tn

Résumé. Les systèmes orientés documents permettent de stocker tout document, quel que soit leur schéma. Cette flexibilité génère une potentielle hétérogénéité des documents qui complexifie leur interrogation car une même entité peut être décrite selon des schémas différents. Cet article présente une approche d'interrogation transparente des systèmes orientés documents. Pour cela, nous proposons de générer un dictionnaire de façon automatique lors de l'insertion des documents, et qui associe à chaque attribut tous les chemins permettant d'y accéder. Ce dictionnaire permet de réécrire la requête utilisateur à partir de disjonctions de chemins afin de retrouver tous les documents quelles que soient leurs structures. Nos expérimentations montrent des coûts d'exécution de la requête réécrite largement acceptables comparés au coût d'une requête sur schémas homogènes.

1 Introduction

Les systèmes de stockage « not-only SQL » (NoSQL) ont connu un important développement ces dernières années en raison de leur capacité à gérer de manière flexible et efficace d'importantes masses de données hétérogènes, Floratou et al. (2012); Stonebraker (2012). Les approches orientées documents sont couramment utilisées comme par exemple les systèmes MongoDB (Chodorow et Dirolf, 2010) ou CouchDB (Anderson et al., 2010). Ces systèmes reposent sur le principe de *« schemaless »* consistant à ne plus considérer un schéma unique pour un ensemble de données , appelé collection de documents (Chevalier et al., 2015). Cette flexibilité dans la structuration des données complexifie l'interrogation pour les utilisateurs qui doivent connaître les différents schémas des données manipulées (Chouder et al., 2017). Cet article traite de la problématique d'interrogation de données hétérogènes dans les systèmes NoSQL orientés documents.

Il existe différents types d'hétérogénéités (Shvaiko et Euzenat, 2005) : *L'hétérogénéité structurelle* désigne le problème de structures variables entre les documents. *L'hétérogénéité*

syntaxique considère que différents attributs peuvent désigner le même concept tandis que *l'hétérogénéité sémantique* considère qu'un attribut peut correspondre à différents concepts. Les travaux développés dans cet article se focalisent sur l'hétérogénéité structurelle des documents.

Pour permettre l'interrogation des données structurellement hétérogènes dans les systèmes NoSQL, deux approches sont essentiellement suivies : soit les données sont transformées pour être rendues homogènes dans un schéma unique (Tahara et al., 2014), soit les données sont conservées de manière hétérogène, mais les différents schémas possibles sont inférés pour permettre l'interrogation (Wang et al., 2015). La première approche a pour avantage de faciliter l'interrogation pour l'utilisateur qui manipule ainsi un schéma unique. Cependant cette approche nécessite des pré-traitements pouvant s'avérer coûteux et difficilement compatible avec des environnements dynamiques. La seconde approche consiste à inférer les différents schémas pour permettre leur interrogation. La variabilité des schémas rend la construction des requêtes plus complexe car elle nécessite la manipulation des différents schémas. Nos travaux se placent dans cette deuxième approche, en rendant transparente pour l'utilisateur l'hétérogénéité structurelle des documents.

La section 2 expose en détail le problème d'hétérogénéité abordé dans cet article. La section 3 propose un état de l'art et dans la section 4 nous donnons une formalisation de nos proposition. Enfin, la section 5 présente les résultats de nos premières expérimentations.

2 Problème d'interrogation des documents structurellement hétérogènes

Dans les systèmes orientés documents les données sont représentées en utilisant les notations JavaScript Object Notation (JSON). Un document est considéré comme une paire (clé, valeur) où la clé est un identifiant unique et sa valeur est représentée au format JSON (Bourhis et al., 2017)

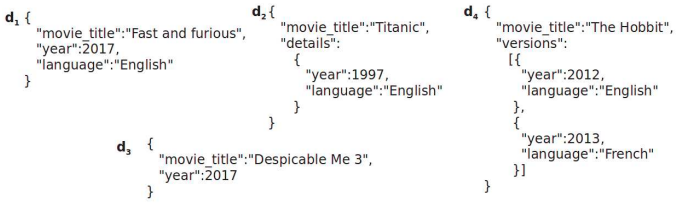

FIG. 1 – *Exemple de quatre documents au format JSON.*

Considérons une collection $C = \{d_1, d_2, d_3, d_4\}$ constituée de 4 documents structurellement hétérogènes présentés figure 1. Chaque document d_i correspond à une paire (k_i, v_i) où k_i est la clé et v_i sa valeur. Nous adoptons la notation $d[i]$ pour accéder à la valeur du document d_i. Nous accédons aux sous parties composant une valeur par une expression navigationnelle basée sur les attributs. Ainsi les expressions suivantes sont possibles :

— $d[1] = \{$ "$movie\_title$" : "$Fast\ and\ furious$", "$year$" : 2017, "$language$" : "$English$"$\}$
— $d[1.movie\_title] =$ "$Fast\ and\ furious$"
— $d[4.versions] = [\{$"$year$" : 2012, "$language$" : "$English$"$\}$,
$\{$"$year$" : 2013, "$language$" : "$French$"$\}]$

— $d[4.versions.2] = \{\text{“}year\text{”} : 2013, \text{“}language\text{”} : \text{“}French\text{”}\}$

Nous considérons uniquement des collections structurellement hétérogènes, en considérant qu'un attribut peut être situé à différentes positions dans les différents schémas (par exemple, “$year$” entre d_1 , d_2 et d_3), ou ne pas être présent (par exemple, “$details$” est présent dans d_2 mais pas dans d_1, d_3, d_4).

Cette hétérogénéité structurelle complexifie l'interrogation d'une collection de documents, en particulier lorsque cette collection comporte de très nombreux documents structurés avec une grande variabilité. Pour obtenir un résultat correct, l'utilisateur doit écrire autant de requêtes qu'il n'y a de versions de schémas concernées par la formulation de l'interrogation.

Considérons que l'on souhaite obtenir la liste des titres et des années de parution des films. En se basant sur la seule connaissance du schéma du document d_1, on peut formuler une requête avec les attributs “$movie\_title$”, “$year$” qui permet d'obtenir :

$[\{\text{“}movie\_title\text{”} : \text{“}Fast\ and\ furious\text{”}, \text{“}year\text{”} : 2017\}, \{\text{“}movie\_title\text{”} : \text{“}The\ Hoobit\text{”}\},$
$\{\text{“}movie\_title\text{”} : \text{“}Despicable\ Me\ 3\text{”}, \text{“}year\text{”} : 2017\}, \{\text{“}movie\_title\text{”} : \text{“}Titanic\text{”}\}]$

Le résultat est incorrect en raison de l'hétérogénéité structurelle de l'attribut “$year$” qui est imbriqué de différentes manières dans les documents d_2 et d_4.

Une autre formulation peut être faite avec les attributs : “$movie\_title$”, “$details.year$”. Cette requête produit un résultat comportant le même type d'incorrections. Pour obtenir un résultat complet, il est nécessaire de construire une requête très complexe tenant compte des différents schémas des documents.

Nous proposons dans cet article une approche permettant à un utilisateur d'exprimer simplement une requête à partir des attributs, sans avoir à tenir compte des différentes positions structurelles des attributs, tout en conservant les structures originelles des documents. La requête permet d'obtenir un résultat *« complet »*, de manière transparente par rapport à l'hétérogénéité structurelle de la collection de documents (sans avoir à connaître et à manipuler avec exhaustivité les différents schémas).

3 Etat de l'art

Dans cette section, nous nous intéressons aux langages de requêtes de données semi-structurées ainsi qu'aux systèmes de stockage existants pour lesquels nous analysons leur capacité à prendre en compte l'hétérogénéité structurelle des données.

Beaucoup de travaux ont étudié l'interrogation de documents semi-structurés et des BD de documents. XQuery (Li et al., 2004) est un langage de requête standardisé par le W3C [1] qui propose une syntaxe « SQL-like » pour interroger des documents XML . Il couvre la plupart des fonctionnalités offertes par SQL (interrogation, agrégations, fermeture, etc.). XQuery reprend le langage XPath (Clark et al., 1999) pour localiser et naviguer parmi les différents nœuds d'un document XML. Ces deux langages ne spécifient pas comment traiter de grandes collections de documents et très peu de mises en œuvre supportent des échelles de volumes importantes. Dans le cadre des bases de données NoSQL, différentes propositions de langages d'interrogation ont été faites, en particulier pour interroger des documents JSON. Citons JSONiq (Florescu et Fourny, 2013) et SQL ++ (Ong et al., 2014) qui proposent une couche prenant en charge les requêtes utilisateurs sur des données semi-structurées ou structurées. L'utilisateur doit construire ses requêtes avec ces nouveaux langages, proches de XQuery ou SQL, mais qui

1. w3c.org

nécessitent parfois des paramétrages fins des processeurs de requêtes, ex. : SQL ++ (Ong et al., 2014), pour répondre aux traitements particuliers de ces données. En bilan, tous ces langages ou systèmes offrent les outils pour interroger n'importe quelles données mais demandent à l'utilisateur de connaître les schémas de ces données et ne lui offrent pas de facilités pour interroger facilement des documents hétérogènes.

Une autre approche d'interrogation consiste à modifier la structure lors du stockage et à interroger les données sans langage de requête orienté document. Par exemple (Tahara et al., 2014) proposent le système Sinew qui aplatit les données et les charge dans un SGBD relationnel (tables). Jaql (Beyer et al., 2011) propose un nouveau langage de script pour interroger simultanément des documents stockés dans des magasins différents et les requêtes sont découpées pour être parallélisées en se basant sur le paradigme map-reduce (Thusoo et al., 2009). Au-delà des coûts d'évaluation des requêtes, l'utilisateur doit connaître la structure des documents pour les interroger correctement.

MongoDB est l'un des systèmes NoSQL orientés documents les plus utilisés apportant des solutions efficaces pour le passage à l'échelle (scalability) et la distribution. Dans sa version standard, le langage d'interrogation des données propre à MongoDB ne permet pas d'interroger des données structurellement hétérogènes de façon transparente. Pour y arriver, l'utilisateur doit construire des fonctions spécifiques prenant en charge les différentes structures possibles des données et les intégrer dans ses requêtes (parcours de tous les schémas possibles). Cela rend difficile et complexe l'interrogation et la rend très sensible à l'intégration de nouvelles données (nouveaux schémas).

(Wang et al., 2015) et (Herrero et al., 2016) traitent la problématique de la découverte et l'intégration de nouveaux schémas. (Wang et al., 2015) proposent de ramener tous les schémas de documents dans un même « schéma type » (skeleton) afin d'aider l'utilisateur dans la découverte d'attributs ou de sous-schémas dans la collection. (Herrero et al., 2016) proposent au contraire d'extraire séparément tous les schémas présents dans la collection afin d'aider l'utilisateur à connaître tous les schémas et tous les attributs présents dans la collection de documents. Si elles permettent de découvrir les schémas des données, ces approches laissent à l'utilisateur la responsabilité de prendre en charge l'hétérogénéité des données lors de l'interrogation.

4 Interrogation de documents hétérogènes

Notre approche permet l'interrogation de collections de documents qui sont structurellement hétérogènes dans les systèmes NoSQL. L'utilisateur exprime sa requête en se basant, d'une part, sur sa connaissance de la nature des données à traiter et, d'autre part, sur la connaissance d'au moins un schéma de donnée existant dans la collection, mais sans devoir connaître tous les schémas et/ou tous les différents chemins qui mènent aux différents attributs. Cette requête est réécrite de manière transparente pour l'utilisateur afin de prendre en compte l'hétérogénéité des documents (schémas multiples dans la collection).

4.1 Modélisation des données à schémas multiples

Définition 1 (Collection) Une collection, notée C, est un ensemble de documents

$$C = \{d_1,\ d_2,\ \ldots,\ d_c\}$$

Tout document est considéré comme une paire (clé, valeur) où sa valeur est de la forme $v_i = \{a_{i,1} : v_{i,1}, \ldots, a_{i,n} : v_{i,n}\}$

Définition 2 (Document) Un document, noté $\forall i \in [1, c],\ d_i$, est une paire clé-valeur.

$$d_i = (k_i, v_i)$$

— k_i est la clé identifiant le document ;

— $v_i = \{a_{i,1} : v_{i,1}, \ldots, a_{i,n} : v_{i,n_i}\}$ est la valeur du document. La valeur v_i d'un document, notée également $d[i]$, est une valeur *objet*, chaque $a_{i,j}$ est appelé *attribut* et chaque $v_{i,j}$ est elle-même une valeur *atomique* (numérique, chaîne, booléenne, nulle) ou *complexe* (objet, tableau) définis ci-après ;

Une valeur atomique est formée comme suit.

— $v_{i,j} = n$ si $n \in \mathbb{N}^*$ l'ensemble valeurs numériques (entiers ou réels) ;

— $v_{i,j} =$ "s" si s est une chaîne de caractères formée dans l'ensemble des caractères $Unicode\mathbb{A}^*$;

— $v_{i,j} = b$ si $b \in B$ l'ensemble des booléens $\{true, false\}$;

— $v_{i,j} = \perp$ est la valeur nulle ;

Une valeur complexe est formée comme suit.

— $v_{i,j} = \{a_{i,j,1} :\ v_{i,j,1},\ \ldots,\ a_{i,j,n_{i,j}} :\ v_{i,j,n_{i,j}}\}$ est une valeur objet où $v_{i,j,k}$ sont des *valeurs* et $a_{i,j,k}$ sont des chaînes de caractères dans $\mathbb{A}^*$ appelés *attributs* ;

— $v_{i,j} = [v_{i,j,1}, \ldots, v_{i,j,n_{i,j}}]$ est une valeur tableau où chaque $v_{i,j,k}$ sont des valeurs ;

Dans le cas des objets et des tableaux, les valeurs $v_{i,j,k}$ peuvent être également des valeurs complexes, permettant ainsi plusieurs niveaux d'imbrications. Lorsqu'une imbrication est présente, nous adoptons une notation par chemin de navigation. Pour les imbrications de tableaux, nous introduisons dans le chemin un entier correspondant à l'indice (Bourhis et al., 2017; Hidders et al., 2017) (cf. section 2).

Définition 3 (Schéma) Le schéma, noté s_i, extrait de la valeur v_i de d_i est défini par

$$s_i = \{p_1, \ldots, p_m\}$$

où p_i est un attribut apparaissant dans la valeur v_i du document d_i, ou n'importe quel chemin d'accès à un attribut non feuille ou feuille dans le cas d'imbrications de valeurs complexes dans d_i (Bourhis et al., 2017). Plus formellement, rappelant d_i a pour valeur $v_i = \{a_{i,1} : v_{i,1}, \ldots, a_{i,n} : v_{i,n}\}$, son schéma s_i est défini par : $\forall k \in [1..n]$

— si $v_{i,j}$ est atomique, $s_i = s_i \cup \{a_{i,j}\}$;

— si $v_{i,j}$ est objet, $s_i = s_i \cup \{a_{i,j}\} \cup \{\cup_{p \in s_{i,j}} a_{i,j}.p\}$ avec $s_{i,k}$ le schéma de la valeur $v_{i,j}$

— si $v_{i,j}$ est tableau, $s_i = s_i \cup \{a_{i,j}\} \cup_{j=1}^{||v_{i,j}||} (\{a_{i,j}.j\} \cup \{\cup_{p \in s_{i,j,k}} a_{i,j}.j.p\})$ avec $s_{i,j,k}$ le schéma de la j^{eme} valeur du tableau $v_{i,k}$;

Exemple. Considérons les documents d_1 et d_2. Leur schéma est constitué des ensembles d'attributs suivants.

$s_1 = \{movie\_title,\ year,\ language\}$

$s_2 = \{movie\_title,\ details,\ details.year,\ details.language\}$

Dans le cas du document d_2, l'attribut "$details$" contient une valeur complexe, formant deux chemins d'accès "$details.year$" et "$details.language$" dans le schéma.

Définition 4 (Schéma de collection). Le schéma S d'une collection C est défini par

$$S = \bigcup_{i=1}^{c} s_i$$

Définition 5 (Dictionnaire). Le dictionnaire d'une collection est défini par

$$dict = \{(p_i,\ \triangle_i)\} \forall p_i \in S$$

— $p_i \in S$ est un chemin appartenant au schéma d'au moins un document de la collection ;
— $\triangle_i = \{p_{p_i,1},\ \ldots,\ p_{p_i,k}\} \subseteq S$, est l'ensemble des chemins d'accès à p_i ;

Notons que pour la suite de l'article, et par abus de langage, nous nommerons les chemins p_i *"attributs"* ; nous parlerons donc de chemins du dictionnaire ou d'attributs dans le dictionnaire.

Exemple. Le dictionnaire de la collection décrite à la figure 1 est défini ci-dessous. Chaque entrée p_i permet d'accéder aux différents chemins possibles de positionnement ;

$(year,\ \{year,\ details.year,\ versions.1.year,\ versions.2.year\})$ indique que l'entrée *"year"* correspond à 4 positions possibles dans les documents.

$dict = \{$
$(movie\_title, \{movie\_title\})$,
$(year, \{year, details.year, versions.1.year, versions.2.year\})$,
$(language, \{language, details.language, versions.1.language, versions.2.language\})$,
$(details, \{details\}), (details.year, \{details.year\}), (details.language, \{details.language\})$,
$(versions, \{versions\}), (versions.1, \{version.1\})$,
$(versions.1.year, \{versions.1.year\}), (versions.1.language, \{versions.1.language\})$,
$(versions.2, \{versions.2\}), (versions.2.year, \{versions.2.year\})$,
$(versions.2.language, \{versions.2.language\})$
$\}$

4.2 Interrogation de données à schémas multiples

L'interrogation d'une collection de documents s'opère par une composition d'opérateurs unaires. Dans cet article, nous limitons l'interrogation aux opérations de projection et de sélection pouvant s'exprimer avec les commandes *"find"* et *"aggregate"* de MongoDB.

4.2.1 Noyau minimum fermé d'opérateurs élémentaires

Nous définissons un noyau minimum fermé d'opérateurs élémentaires. On note C_{in} la collection de documents interrogée, et C_{out}la collection de documents résultante.

Définition 6 (Projection) L'opérateur de projection réduit le schéma des documents à un sous-ensemble d'attributs ; on note

$$\pi_A(C_{in}) = C_{out}$$

où $A \subseteq S_{in}$ est un sous-ensemble d'attributs de S_{in} (schémas de la collection C_{in})

Définition 7 (Sélection). L'opérateur de sélection permet de restreindre une collection de documents aux seuls documents satisfaisant un prédicat de sélection ; on note

$$\sigma_p(C_{in}) = C_{out}$$

où p est un prédicat (ou condition) de sélection. Un prédicat simple est une expression $a_k\ \omega_k\ v_k$ avec $a_k \subseteq S_{in}$ est un *attribut*, $\omega_k \in \{=\ ;\ >\ ;\ <\ ;\ \neq\ ;\ \geq\ ;\ \leq\}$ est un opérateur de comparaison, et

v_i une valeur. Les prédicats peuvent se combiner avec les opérandes $\Omega = \{ \vee, \wedge, \neg\}$ formant un prédicat complexe.

On note $Norm_p$ la forme conjonctive normale du prédicat *p*, notée comme suit.

$$Norm_p = \wedge_i(\vee_j a_i,\ {}_j\varpi_i,\ {}_jv_{i,j})$$

Définition 8 (Requête). Une requête Q est construite par composition d'opérateurs

$$Q = q_1 \circ \ldots \circ q_r$$

avec $\forall i \in [1, r] q_i \in \{\pi, \sigma\}$

Exemple. Considérons la collection des documents de la figure 1.

$q_1 : \sigma_{language=\text{“}English\text{”}}(C) = \{d'_1\}$ avec
$d'[1] = \{\text{“}movie\_title\text{”} :\ \text{“}Fast\ and\ furious\text{”},\ \text{“}year\text{”} :\ 2017,\ \text{“}language\text{”} : \text{“}English\text{”}\}$

$q_2 : \pi_{movie\_title,year}(C) = \{d'_1,\ d'_2,\ d'_3,\ d'_4\}$ avec
$d'[1] = \{\text{“}movie\_title\text{”} :\ \text{“}Fast\ and\ furious\text{”},\ \text{“}year\text{”} :\ 2017\}$
$d'[2] = \{\text{“}movie\_title\text{”} :\ \text{“}Titanic\text{”}\}$
$d'[3] = \{\text{“}movie\_title\text{”} :\ \text{“}Despicable\ Me\ 3\text{”},\ \text{“}year\text{”} :\ 2017\}$
$d'[4] = \{\text{“}movie\_title\text{”} :\ \text{”}The\ Hobbit\text{”}\}$

$q_3 : \pi_{movie\_title,year}(\sigma_{langugage=\text{“}English\text{”}}(C)) = \{d'_1\}$ avec
$d'[1] = \{\text{“}movie\_title\text{”} :\ \text{“}Fast\ and\ furious\text{”},\ \text{“}year\text{”} :\ 2017\}$

La requête q_3 est construite par composition. On peut remarquer que les requêtes q_1 et q_3 ne retournent pas l'ensemble des documents possibles, car l'attribut “$language$” utilisé dans le prédicat de sélection est structurellement hétérogène entre les différents documents de la collection interrogée. De manière analogue la projection de l'attribut “$year$” dans la requête q_2 est également perturbée par l'hétérogénéité, ne permettant pas d'obtenir la valeur attendue pour les documents d'_2 et d'_4 où l'attribut projeté “$year$” est ignoré.

4.2.2 Extension de requêtes aux collections hétérogènes

L'hétérogénéité structurelle des documents complexifie l'interrogation car elle n'est pas gérée nativement par les opérateurs de la plupart des systèmes NoSQL ; par exemple MongoDB (*“find"*) ne reconnaît pas de manière automatique les différentes structures des documents d'une collection ; les attributs non positionnés de manière compatible à la requête sont ignorés.

Notre approche consiste à faciliter l'interrogation pour les utilisateurs, par reformulation automatique des requêtes. Ce processus exploite le dictionnaire des données afin de reformuler la requête en prenant en compte les multiples schémas des documents de la collection interrogée. L'algorithme 1 décrit ce processus d'extension automatique de la requête utilisateur.

— Lors d'une projection, la liste des attributs projetés A_i est étendue par l'union des chemins d'accès $\triangle_k$ à chaque attribut a_k de la liste projetée. Ces chemins sont obtenus à partir du dictionnaire des données.

— Lors d'une sélection, le prédicat de sélection p, en forme normale conjonctive, est étendu par l'ensemble des disjonctions formées à partir des chemins d'accès $\triangle_{i,j}$ de chaque attribut $a_{i,j}$.

Algorithme 1 : Extension automatique de la requête utilisateur

entrée : Q
sortie : Q_{ext}
$Q_{ext} \leftarrow id$ // identité
foreach $q_i \in Q$ **do**
  switch q_i **do**
    case π_{A_i} // projection
      $A_{ext} \leftarrow \bigcup_{\forall a_k \in A_i} \triangle_k$
      $Q_{ext} \leftarrow Q_{ext} \circ \pi_{A_{ext}}$
    end
    case σ_{Norm_p} // sélection
      $P_{ext} \leftarrow \bigwedge_k \left(\bigvee_l \bigvee_{a_j \in \triangle_{k,l}} a_j \; \varpi_{k,l} \; v_{k,l} \right)$
      $Q_{ext} \leftarrow Q_{ext} \circ \sigma_{P_{ext}}$
    end
  endsw
end

Exemple. Considérons la requête q_3 ($\pi_{movie\_title,\ year}\ (\sigma_{langugage=\text{“}English\text{”}}\ (C)$) de l'exemple précédent. Le moteur de réécriture des requêtes, à partir des entrées du dictionnaire suivantes :
$(movie\_title, \{movie\_title\}), (year, \{year, details.year, versions.1.year, versions.2.year\}),$
$(language, \{language, details.language, versions.1.language, versions.2.language\})$
et en appliquant l'algorithme 1, permet d'obtenir la requête étendue suivante :
$\pi_{movie\_title,\ year,\ details.year,\ versions.1.year,\ versions.2.year}(\sigma_{language=\text{“}English\text{”}} \vee$
$versions.1.language=\text{“}English\text{”} \vee details.language=\text{“}English\text{”} \vee versions.2.language=\text{“}English\text{”}$
$(C))$ dont le résultat est $\{d'_1,\ d'_2,\ d'_3,\ d'_4\}$ avec
$d'[1] = \{\text{“}movie\_title\text{”} : \text{“}Fast\ and\ furious\text{”},\ \text{“}year\text{”} : 2017\}$
$d'[2] = \{\text{“}movie\_title\text{”} : \text{“}Titanic\text{”},\ \text{”}details\text{”} : \{\text{“}year\text{”} : 2017\}\}$
$d'[3] = \{\text{“}movie\_title\text{”} : \text{“}Despicable\ Me\ 3\text{”},\ \text{“}year\text{”} :\ 2017\text{”}\}$
$d'[4] = \{\text{“}movie\_title\text{”} : \text{“}The\ Hobbit\text{”},\ \text{“}versions\text{”} : [\{\text{“}year\text{”} :\ 2017\}]\}$

5 Expérimentations

Nous avons implémenté un outil appelé *Easy-Q* afin de mettre en oeuvre l'algorithme de réécritures de requêtes proposé dans cet article ainsi que la construction du dictionnaire défini. *Easy-Q* procède à la création du dictionnaire d'une manière automatique au moment de l'insertion des données et le stocke dans une collection sous MongoDB. Il effectue aussi sa mise à jour. Notre outil prend en entrée la requête de l'utilisateur, procède à la réécriture en utilisant le dictionnaire afin d'extraire différents chemins possibles pour chaque prédicat et lance son exécution dans MongoDB.

5.1 Protocole expérimental

Pour l'ensemble de nos expériences nous avons utilisé le système orienté document MongoDB afin de stocker et exécuter nos requêtes. Nous avons choisi de travailler sur des collections de documents synthétisés. Les documents sont construits à partir d'une collection accessible sur internet qui décrit des films proposés par IMDB [2] et composés de 28 attributs qui sont tous liés à la racine (structure plate) : tous les documents sont donc homogènes.

Les caractéristiques des collections générées sont résumées dans le tableau 1. Les documents de la collection sont générés aléatoirement sans ordre particulier : les documents d'un même schéma sont aléatoirement répartis dans la collection. Les groupes d'attributs pour chaque schéma généré sont liés à ce schéma : le groupe est de la forme *"groupe_xy"* où x fait référence au numéro du groupe et y fait référence à un schéma (numérotés alphabétiquement). Exemple : le groupe *"groupe_1E"* fait référence au premier groupe du cinquième schéma généré. Deux schémas différents n'ont donc pas de sous-chemin commun. Pour les niveaux intermédiaires d'imbrication nous générons des attributs intermédiaires entre le groupe et les attributs qui contiennent les valeurs. Exemple : si l'attribut *"movie_title"* fait partie du premier groupe du schéma 2, dans un document généré, le chemin vers cet attribut sera *"groupe_1B.level0.movie_title"*.

Description des requêtes Les requêtes $Q1,\ Q3,\ Q5$ contiennent la forme conjonctive des prédicats alors que les requêtes $Q2,\ Q4,\ Q6$ contiennent la forme disjonctive des prédicats.

$Q1/Q2 : \pi_*(\sigma_{director\_name=\text{“}A\%\text{”}\ (\wedge/\vee)gross>100000}(C))$

$Q3/Q4 : \pi_*(\sigma_{director\_name=\text{“}A\%\text{”}\ (\wedge/\vee)gross>100000(\wedge/\vee)duration<200(\wedge/\vee)title\_year<1950}(C))$

$Q5/Q6 : \pi_*(\sigma_{director\_name=\text{“}A\%\text{”}\ (\wedge/\vee)gross>100000(\wedge/\vee)duration<200(\wedge/\vee)title\_year<1950(\wedge/\vee)}$
$_{pays!=\text{“”}(\wedge/\vee)language=\text{“}English\text{”}(\wedge/\vee)imdb\_score<4(\wedge/\vee)cast\_total\_facebook\_likes>500}(C)$

Paramètre	Valeur
Nombre de schémas par collection	10
Nombre de sous-arbres arbitraires par schéma	{5,6,1,3,4,2,7,2,1,3}
Profondeurs des sous-arbres	{4,2,6,1,5,7,2,8,3,4}
Présence de chaque schéma dans la collection	10 %
Nombre d'attributs dans un schéma	Aléatoire
Nombre d'attributs par sous-arbres	Aléatoire
Tailles des collections utilisées	10 Go, 25 Go, 50 Go, 100 Go
Nombre de documents par collections	12 M, 30 M , 60 M, 120 M

TAB. 1 – *Paramètres de génération d'une collection*

5.2 Évaluation du module de réécriture de requêtes

La première évaluation porte sur le temps d'exécution de la même requête posée sur la collection homogène et la requête réécrite sur la collection hétérogène ; les deux requêtes retournant le même nombre de résultats. L'objectif est d'étudier le coût additionnel de notre solution par rapport à la simple exécution d'une requête sur un jeu de donnée homogène. Nous comparerons aussi ce coût à une interrogation de la collection hétérogène sans réécriture : la somme des coût des requêtes individuelles sur chaque schéma.

2. http ://www.omdbapi.com/

Afin d'évaluer la sélection, nous proposons d'exécuter les 6 requêtes de longueurs variables présentées ci-avant (la projection portera sur tous les attributs).

Nous avons utilisé les mesures suivantes pour chacune des requêtes :

— $QRewritten$: Le temps d'exécution de la requête réécrite par notre système sur la collection hétérogène.

— $QSeparated$: La somme des temps d'exécution des sous-requêtes sur la collection hétérogène ; requêtes sans réécriture sur tous les schémas possibles.

— $QBase$: Le temps d'exécution de la requête sur la collection homogène.

Notons que les requêtes conjonctives $Q1,\ Q3,\ Q5$ retournent au maximum 1% des documents tandis que les requêtes disjonctives $Q2, Q4, Q6$ retournent au minimum 70% des documents.

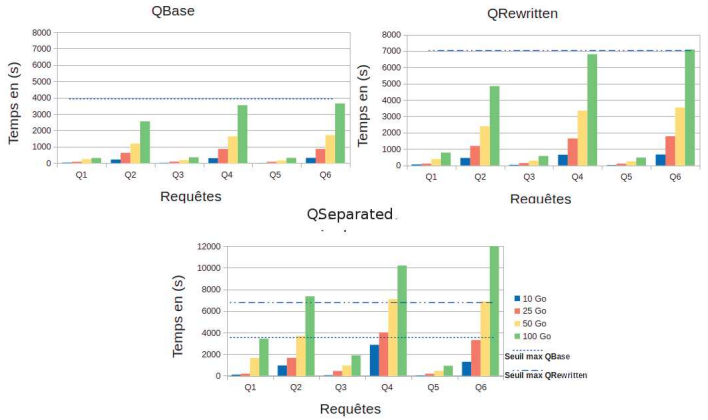

FIG. 2 – *Évaluations de la requêtes réécrite*

Les résultats présentés figure 2 montrent que notre solution ne dépasse jamais, en temps d'exécution, 2 fois en moyenne dans le cas des requête disjonctive et 1 fois et demi dans le cas des requêtes conjonctive, la durée de la requête de base (sur données homogènes) alors que la réécriture ajoute 10 disjonctions par critère de sélection (10 schémas possibles). De plus, les attributs de la collection hétérogènes sont tous imbriqués à différents niveaux qui peuvent atteindre 7 niveaux dans le cas du sixième schéma. Les évaluation montrent aussi de meilleures performances par rapport aux requêtes sur schémas séparés : jusqu'à 2 fois et demi plus rapide. Même si d'autres évaluation seront à mener, les expériences montrent une croissance du temps d'exécution de la requête réécrite linéaire par rapport à la taille de la collection et du temps d'exécution de la requête sur la collection homogène, à l'opposé de la croissance exponentielle observée dans le cas des requêtes cumulées.

5.3 Évaluation du module de création du dictionnaire

Le tableau 2 présente le temps nécessaire pour la création du dictionnaire pour 2, 4, 6, 8 et 10 schémas pour des collections de 100 Go. Le temps nécessaire pour la création du dictionnaire est influencé nettement par le nombre de schémas dans la même collection. Notons cependant qu'il s'agit d'une opération réalisée une seule fois sur la collection lorsque celle ci

existe déjà. En cas d'alimentation continue d'une collection, le dictionnaire sera mis à jour au fur et à mesure de l'arrivée des données (documents). Nous avons aussi étudier l'effet d'une forte hétérogénéité sur un nombre important de schémas en testant jusqu'à 5000 schémas dans la même collection. Le dictionnaire à été généré avec pour chaque attribut 5000 schémas différents. Nous avons aussi évalué le temps mis pour la réécriture de la requête $Q6$ et nous avons un temps de réécriture très intéressant de moins de 1.5 secondes. Enfin, la taille du dictionnaire n'a jamais dépassé 12 Mo ce qui est restreint au vu des collections et de l'hétérogénéité traitées.

Nombre de schémas	2	4	6	8	10
Temps en minutes	96	108	127	143	156
Taille dict généré en Ko	4,154	9,458	13,587	17,478	22,997

TAB. 2 – *Temps de création du dictionnaire selon le nombre de schémas (base de 100 Go)*

6 Conclusion

L'hétérogénéité dans les systèmes orientés documents constitue un défi majeur lors de l'exploitation des données. Nous proposons une approche facilitant l'interrogation de documents à structures hétérogènes en simplifiant l'écriture des requêtes. Notre approche repose sur la construction d'un dictionnaire de données qui indexe tous les schémas d'une collection de documents, et qui est exploitée pour réécrire de manière transparente les requêtes des utilisateurs. Les requêtes réécrites permettent d'obtenir facilement l'ensemble des documents répondants à la requête initiale.

En perspectives de ces travaux, nous allons continuer la validation de l'algorithme de réécriture de requêtes en le validant sur d'autres systèmes orientés documents tels que couchBase. Nous étudions le passage à l'échelle sur des collections réelles de grande taille et nous menons des expériences sur des environnements distribués. A long terme, nous travaillerons sur l'extension du langage de requêtes par plus d'opérateurs (agrégation et jointure) ainsi que sur le support des hétérogénéités sémantique et syntaxique.

Références

Anderson, J. C., J. Lehnardt, et N. Slater (2010). *CouchDB : The Definitive Guide : Time to Relax*. " O'Reilly Media, Inc.".

Beyer, K. S., V. Ercegovac, R. Gemulla, A. Balmin, M. Eltabakh, C.-C. Kanne, F. Ozcan, et E. J. Shekita (2011). Jaql : A scripting language for large scale semistructured data analysis. In *Proceedings of VLDB Conference*.

Bourhis, P., J. L. Reutter, F. Suárez, et D. Vrgoč (2017). Json : data model, query languages and schema specification. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pp. 123–135. ACM.

Chevalier, M., M. El Malki, A. Kopliku, O. Teste, et R. Tournier (2015). Implementation of multidimensional databases with document-oriented nosql. In *(DAWAk'15)*, pp. 379–390.

Chodorow, K. et M. Dirolf (2010). Mongodb : The definitive guide o'reilly media.

Chouder, M. L., S. Rizzi, et R. Chalal (2017). Enabling self-service bi on document stores. In *EDBT/ICDT Workshops*.

Clark, J., S. DeRose, et al. (1999). Xml path language (xpath) version 1.0.

Floratou, A., N. Teletia, D. J. DeWitt, J. M. Patel, et D. Zhang (2012). Can the elephants handle the nosql onslaught ? *Proceedings of the VLDB Endowment 5*(12), 1712–1723.

Florescu, D. et G. Fourny (2013). Jsoniq : The history of a query language. *IEEE internet computing 17*(5), 86–90.

Herrero, V., A. Abelló, et O. Romero (2016). Nosql design for analytical workloads : variability matters. In *Conceptual Modeling : 35th International Conference, ER 2016, Gifu, Japan, November 14-17, 2016, Proceedings 35*, pp. 50–64. Springer.

Hidders, J., J. Paredaens, et J. Van den Bussche (2017). J-logic : Logical foundations for json querying. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pp. 137–149. ACM.

Li, Y., C. Yu, et H. Jagadish (2004). Schema-free xquery. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pp. 72–83. VLDB Endowment.

Ong, K. W., Y. Papakonstantinou, et R. Vernoux (2014). The sql++ query language : Configurable, unifying and semi-structured. *arXiv preprint arXiv :1405.3631*.

Shvaiko, P. et J. Euzenat (2005). A survey of schema-based matching approaches. *Journal on data semantics IV*, 146–171.

Stonebraker, M. (2012). New opportunities for new sql. *Communications of the ACM 5*(11), 10–11.

Tahara, D., T. Diamond, et D. J. Abadi (2014). Sinew : a sql system for multi-structured data. In *2014 SIGMOD*, pp. 815–826. ACM.

Thusoo, A., J. S. Sarma, N. Jain, Z. Shao, P. Chakka, S. Anthony, H. Liu, P. Wyckoff, et R. Murthy (2009). Hive : a warehousing solution over a map-reduce framework. *Proceedings of the VLDB Endowment 2*(2), 1626–1629.

Wang, L., S. Zhang, J. Shi, L. Jiao, O. Hassanzadeh, J. Zou, et C. Wangz (2015). Schema management for document stores. *Proceedings of the VLDB Endowment 8*(9), 922–933.

Summary

Documents oriented DB can store any document, whatever its data schema. This facility generates potential structural heterogeneity of documents which makes it difficult to query data because the same entity can be described according to different schemas. This paper presents an approach of transparent querying of documents oriented DB with structural heterogeneity. For that purpose, we suggest building a dictionary which associates every field with all partial schemas (paths) of the DB allowing to reach it. This dictionary is used to rewrite the user query in disjunctions of partial schemas in order to find all the documents, whatever their real schema. Our experiments show acceptable costs of execution of the rewritten query when compared to the cost of equivalent query on homogeneous schemas