GAMM : un modèle multidimensionnel agile à base de graphes pour des entrepôts multi-versions

Redha Benhissen[*,**], Fadila Bentayeb[*,**]
Omar Boussaid[*,**]

[*] Laboratoire ERIC, Université Lyon 2, Bron 69500, France
[**] {redha.benhissen,fadila.bentayeb,omar.boussaid}@univ-lyon2.fr

Résumé. Dans le contexte des big data où le volume de données croît continuellement, nous proposons dans cet article une approche agile d'évolution de schéma dans les entrepôts de données qui permet aux concepteurs d'intégrer de nouvelles sources de données et de tenir compte des nouveaux besoins d'utilisateurs afin d'enrichir les possibilités d'analyse. Notre approche est fondée sur un modèle évolutif en multi-versions basé sur la modélisation ensembliste sous forme de graphe. Un méta-modèle permettra la gestion des versions du schéma de l'entrepôt. Nous proposons également des fonctions d'évolution au niveau schéma (l'évolution des instances n'est pas traitée ici). Nous validons notre approche par un prototype logiciel qui gère à la fois les versions et l'évolution de schéma dans le modèle.

1 Introduction

L'approche d'entreposage de données Inmon (1992); Kimball (1996) constitue un champ de recherche important dans lequel de nombreux problèmes restent à résoudre. Les entrepôts de données (ED) centralisent des données provenant de différentes sources pour répondre aux besoins d'analyses des utilisateurs. Un des points clés de la réussite du processus d'entreposage de données réside dans la définition du modèle de l'entrepôt en fonction des sources de données et des besoins d'analyse. Avec l'avènement des big data nous assistons à une prolifération des sources de données amenant de nouveaux besoins d'analyse. Les big data offrent ainsi de nouvelles opportunités d'analyse engendrées par la diversification des sources de données. Lorsqu'un entrepôt de données est conçu, son schéma restera figé ; si les besoins d'analyse sont amenés à évoluer, leur prise en compte peut s'avérer coûteuse. Le modèle multidimensionnel classique, basés sur le modèle en étoile, présente des limites aux possibilités de changement, et son évolution est complexe que ce soit au niveau schéma ou au niveau données. Ces limites sont liées au modèle en étoile figé : car il est créé pour des besoins d'analyse connus à l'avance. C'est dans ce contexte que nous nous intéressons à l'évolution des entrepôts de données. Même si cette problématique n'est pas nouvelle car de nombreux travaux existent dans la littérature Faisal et al. (2017), elle continue tout de même à poser des questions de recherche notamment en termes d'évolution de schémas et de données, de gestion de ces évolutions ainsi que d'inté-

grité des données.

Dans cet article, nous proposons un modèle d'entrepôt de données agile (GAMM : *Graph-based Agile Multidimenional Model*) qui peut prendre en compte à la fois l'émergence de nouvelles sources de données et de nouveaux besoins d'analyse. GAMM s'appuie sur un méta-modèle qui permet de générer des versions à chaque fois que des mises à jours sont effectuées sur le modèle. GAMM s'inspire de la modélisation ensembliste, notamment *Anchor modeling* (AM), et permet de produire des modèles multidimensionnels sous forme de graphes. Il hérite de l'agilité apportée par AM et des nombreux avantages offerts par les bases de données NoSQL orientées graphes. Nous proposons également plusieurs fonctions d'évolution de schéma de l'entrepôt et illustrons notre propos avec des exemples ainsi que le méta-modèle permettant la gestion des versions générées par ces évolutions. Nous validons notre modèle par un prototype logiciel dans lequel sont implémentés à la fois le méta-modèle et GAMM.

Dans cet article, nous exposons dans la section 2 l'état de l'art sur l'évolution dans les entrepôts de données, avec lesquels nous comparons notre modèle GAMM selon les critères qui nous semblent pertinents. Dans la section 3, nous présentons notre modèle agile GAMM, le principe de la modélisation ensembliste et des graphes sur lesquels se base notre modèle ainsi que les règles de passage d'un modèle multidimensionnel classique vers un modèle multidimensionnel agile à base de graphe. Nous décrivons par la suite dans la section 4 le processus d'évolution de notre modèle élaboré par les fonctions d'évolution mises en place ainsi que le méta-modèle pour la gestion des versions engendrées par ces évolutions. Dans la section 5, nous décrivons notre prototype logiciel en architecture 3-tiers permettant de valider notre modèle, et nous présentons un cas d'usage basé sur les données Northwind [1]. Dans la section 6, nous concluons notre article et ouvrons des perspectives à nos travaux.

2 Travaux connexes

A la fin des années 90, les systèmes décisionnels basés sur la modélisation multidimensionnelle se sont progressivement imposés pour fournir des outils et des méthodes permettant aux décideurs d'avoir une meilleure visibilité de leurs environnements et de leur offrir un accompagnement adéquat pour la prise de décision stratégique Inmon (1992). Le concept de la modélisation multidimensionnelle, notamment sur l'aspect évolutif, est devenu un domaine de recherche à part entière Golfarelli et Rizzi (2018). En effet, l'évolution des modèles multidimensionnels a fait l'objet de plusieurs travaux de recherches proposant ainsi différentes approches. Ces travaux peuvent être regroupés en deux catégories : (i) selon l'évolution des données Bliujute et al. (1998); Mendelzon et Vaisman (2000); Golfarelli et Rizzi (2011); Garani et al. (2016); Phungtua-Eng et Chittayasothorn (2019); Ahmed et al. (2020), (ii) selon l'évolution des schémas, sans prise en charge de l'historique : Hurtado et al. (1999); Blaschka et al. (1999); Bellahsene (2002); Benitez-Guerrero et al. (2004); Mazón et al. (2006); Favre et al. (2007); Talwar et Gosain (2012), et avec prise en charge de l'historique des changements : Morzy et Wrembel (2003); Wrembel et Morzy (2006); Ravat et al. (2006); Rizzi et

1. https ://github.com/Microsoft/sql-server-samples/tree/master/samples/databases/northwind-pubs

Golfarelli (2007); Ahmed et al. (2014, 2021).

Dans le même contexte et pour apporter plus d'agilité aux modèles multidimensionnels classiques, nous nous sommes intéressés à d'autres formalismes de modélisation plus indiqués en matière d'évolution. En effet, la modélisation ensembliste (*Ensemble Modeling*) offre une grande flexibilité. Deux approches se distinguent, à savoir la modélisation d'ancrage «*Anchor Modeling*» (AM), Regardt et al. (2009); Rönnbäck et al. (2010), et la modélisation *Data Vault* (DVM), Hultgren (2012b); Fentaw (2014); Linstedt (2015). Ces deux approches sont assez proches dans le principe avec un avantage pour AM Rönnbäck et Hultgren (2013) en raison d'une modélisation hautement normalisée.

AM a été introduit dans un contexte général de bases de données relationnelles. Cependant, dans Nëmec (2012); Nëmec et Zapletal (2014), les auteurs proposent une modélisation d'ancrage pour le schéma en étoile. Ce modèle a été élaboré dans une perspective de comparaison des performances. En effet, ce travail se focalise plutôt sur l'analyse des performances des requêtes entre le schéma d'ancrage et le schéma en étoile classique.

Dans Hamitouche (2021), les auteurs ont proposé un modèle décisionnel basé sur la technique AM en définissant les règles de passage d'un modèle multidimensionnel classique vers un modèle en AM, et se focalisent sur son aspect évolutif et agile. Toutefois, en restant sur une approche relationnelle, ce modèle présente des contraintes d'intégrités par la présence de valeurs NULL dans les faits après chaque évolution lors de l'ajout de nouvelles dimensions.

Les travaux de la littérature sur les modèles multidimensionnels évolutifs s'appuient sur une modélisation en étoile basée sur des structures relationnelles limitant leur évolution notamment en termes de schéma. En effet, les solutions, adoptées jusqu'à alors, proposaient soit un versionnement implicite en fournissant des extensions temporelles au schéma telle que l'utilisation du *bitmap* pour le partage des données entre plusieurs versions références, ce qui est contraignant en termes de requêtes notamment dans un contexte de big data ; soit par un versionnement explicite par la duplication des données et des entités (éléments du schéma), ce qui est également contraignant en raison de la quantité de données générée et de la structure complexe à gérer Solodovnikova (2007); Faisal et al. (2017). De plus, l'extension des entités relationnelles engendre une perte dans l'intégrité des données générée par la création de valeurs NULL.

Dans notre étude de l'art (TAB 1), nous avons établi une comparaison entre les différents travaux de recherche existants en se basant sur l'aspect évolutif des schémas et des données, l'historique de ces évolutions, la redondances des entités ainsi que l'intégrité des données.

L'évolution des modèles multidimensionnels telle qu'elle est traitée dans les différents travaux de recherche s'avère contraignante tant sur le plan logique que physique. En effet, la modélisation des faits attachés à plusieurs dimensions limite considérablement les possibilités d'évolution en raison d'une conception basée sur un schéma en étoile. De plus, la structure relationnelle utilisée dans la quasi-généralité des travaux de littérature n'est pas adaptée à cette évolution en raison de la structure tabulaire des entités et d'un schéma établi pour un besoin spécifique dont la mise à jour est complexe. Le modèle que nous proposons permet une évolution incrémentale des schémas avec un historique de toutes les précédentes versions. L'ap-

Travaux	Évolution temporelle des données	Evolution des schémas	Historique	Non redondance des données	Données intègres
Bliujute et al. (1998)	√				
Hurtado et al. (1999)		√		√	√
Blaschka et al. (1999)		√		√	√
Mendelzon et Vaisman (2000)	√				
Bellahsene (2002)		√			
Morzy et Wrembel (2003)		√	√		
Bebel et al. (2004)		√	√		
Benitez-Guerrero et al. (2004)		√		√	√
Mazón et al. (2006)		√		√	√
Ravat et al. (2006)	√	√	√		
Favre et al. (2007)		√		√	√
Rizzi et Golfarelli (2007)		√	√		
Golfarelli et Rizzi (2011)	√				
Talwar et Gosain (2012)		√		√	√
Ahmed et al. (2014)		√	√		
Garani et al. (2016)	√				
Phungtua-Eng et al.(2019)	√			√	√
Ahmed et al. (2020, 2021)	√	√	√		
Hamitouche et al.(2021)		√	√		

TAB. 1 – *Tableau comparatif.*

proche adoptée pour la formalisation de notre modèle facilitera, par la suite, l'intégration de la gestion de l'évolution des données.

3 GAMM : Modèle multidimensionnel agile à base de graphes

Afin d'assurer l'évolution d'un entrepôt de données, nous proposons un modèle multidimensionnel agile qui s'appuie, d'une part, sur la modélisation ensembliste, et d'autre part, sur une structure de graphes. La modélisation ensembliste apporte plus de flexibilité aux modèles multidimensionnels au niveau conceptuel en raison de sa forte normalisation privilégiant une évolution incrémentale non destructive. La structure de graphe, quant à elle, aidera à remédier aux contraintes logiques et physiques des modèles classiques. En effet outre la qualité repré-

sentative des données inter-connectées, l'utilisation des graphes a été préférée pour l'absence des contraintes d'intégrité notamment pour la gestion des clés lors de l'évolution des schémas.

3.1 Principe de l'agilité dans un modèle multidimensionnel

Nous définissons l'agilité dans un modèle multidimensionnel comme sa capacité à répondre à de nouveaux besoins d'analyse ainsi qu'à l'intégration de nouvelles sources de données de manière incrémentale tout en conservant un historique des versions précédentes.

Pour concevoir notre modèle multidimensionnel GAMM, nous nous sommes intéressés aussi aux niveaux logique et physique afin d'assurer une agilité à notre modèle notamment en matière d'évolution. En effet, sur le volet logique, nous nous sommes basés sur les mêmes principes de la modélisation AM. Cette dernière se base sur l'idée centrale de la décomposition unifiée, où une entité est décomposée en éléments constitutifs pour des raisons de flexibilité, d'adaptabilité, d'agilité et d'interopérabilité Hultgren (2012a); Rönnbäck et Hultgren (2013). Il s'agit notamment de définir clairement le concept de base (concept métier) permettant d'identifier l'entité, et de lui associer sous forme d'ensemble, les éléments dissociés constituant le contexte et les relations. Cette dissociation se fait sur la base de la nature des données, la fréquence de changement et le type de relations.

Appliquée aux principes de la modélisation multidimensionnelle, la modélisation ensembliste facilite l'évolution des modèles Rönnbäck et Hultgren (2013). L'évolution des schémas et les changements dans l'environnement source ne nécessitent pas de modifications majeures dans le modèle, ni dans les applications existantes. Toutes les versions précédentes du schéma de l'entrepôt de données sont disponibles en tant que sous-ensembles du schéma actuel, et des fragments peuvent être modélisés et appliqués de manière itérative Rönnbäck et al. (2010).

Dans un souci de cohérence, nous utiliserons le modèle de *Ventes au détail* de notre cas d'utilisation comme exemple courant pour présenter les différents éléments de notre proposition. Cet exemple est composé d'un fait SALES et deux dimensions PRODUCT et CUSTOMER. La dimension PRODUCT est décrite par les attributs *Product_Name* et *Unit_price* et un niveau hiérarchique CATEGORY avec l'attribut *Category_Name*. Cette dimension possède donc une seule hiérarchie sur l'axe d'analyse (PRODUCT, CATEGORY). La dimension CUSTOMER est décrite par les attributs *Customer_Name*, *Adress* et *Phone*, ainsi que par un niveau hiérarchique CITY ayant l'attribut *City_Name*. Le fait SALES est indiqué par les mesures *Sales_Amount*. Nous représentons dans la figure 1 le modèle de Ventes au détail en utilisant la technique de modélisation par ancrage.

Anchor Modeling propose une ligne directrice pour la conception du modèle en se basant sur une représentation spécifique tels que les ancres (*Anchors*), les noeuds(*Knots*), les attributs (*Attributes*) et les liens (*Ties*), Regardt et al. (2009); Rönnbäck et al. (2010).

— Une ancre représente un ensemble d'entités principales (concept métier), elle détient l'identité des entités dans le modèle par la génération d'une clé de substitution (*Surrogate key*). *Exemple* : Le fait SALES, les dimensions PRODUCT et CUSTOMER ainsi que les niveaux CATEGORY et CITY.

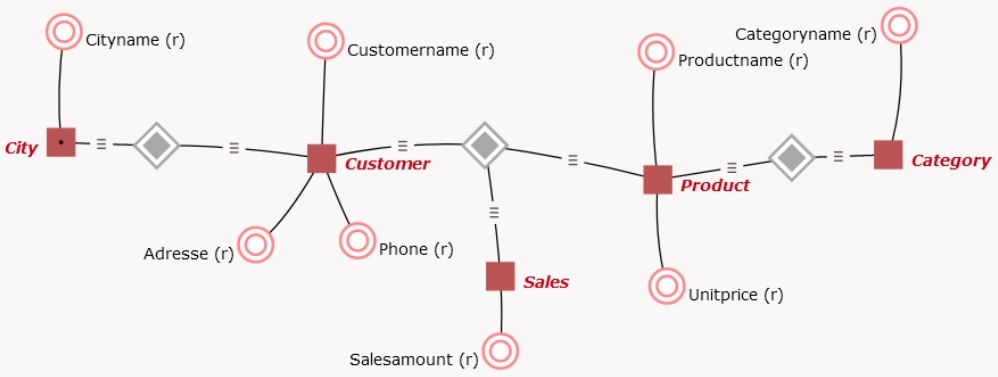

FIG. 1 – *Représentation logique d'un modèle multidimensionnel de Ventes au détail avec la technique Anchor Modeling*

— Les attributs sont utilisés pour décrire les entités principales. Un attribut historisé a un double contour. *Exemlpe* : les attributs PRODUCT_NAME et UNIT_PRICE de la dimension PRODUCT, les attributs CUSTOMER_NAME, ADRESS et PHONE de la dimension CUSTOMER ainsi que les attributs CATEGORY_NAME et CITY_NAME qui décrivent respectivement les niveaux CATEGORY et CITY.
— Les noeuds sont utilisés pour représenter un ensemble déterminé qui décrit l'entité principale. Ils sont utilisés pour éviter la redondance dans les entités.
— Les liens représentent une association entre deux ou plusieurs entités d'ancrage. Un lien historique a un double contour.

De plus, AM utilise des règles de passage du modèle logique vers le modèle physique en créant une table relationnelle pour chaque entité et en incorporant également un étiquetage temporel pour les entités chronologiques.

3.2 Principe de notre approche GAMM

Le modèle GAMM, s'inspirant de la technique *Anchor Modeling* et la formalisation en graphe, utilise les noeuds pour matérialiser les concepts métiers et les arrêtes pour les relations entre ces concepts. Pour ce faire, notre modèle logique (FIG. 2) est caractérisé par la séparation des concepts métier de leurs descripteurs (attributs) permettant une évolution indépendante de chaque entité. En effet, les faits, les dimensions ainsi que les niveaux des hiérarchies sont représentés par des noeuds correspondant aux concepts multidimensionnels de base auxquels sont liés d'autres noeuds représentant les descripteurs. Un étiquetage temporel a été attribué à toutes les entités pour permettre l'identification des différentes instances, et leur appartenance à une version donnée. Outre la qualité représentative des données inter-connectées, l'utilisation des graphes a été préférée pour l'absence des contraintes d'intégrité notamment pour la gestion des clés lors de l'évolution des schémas. Une base de données NoSQL orientée graphe a été retenue en raison de sa concordance et sa proximité avec le modèle multidimensionnel agile en graphe proposé notamment pour ses caractéristiques non relationnelles et évolutives, où les

données peuvent être stockées sans se conformer à un schéma préétabli. Il s'agit de la base de données NoSQL : Neo4j. Dans ce qui suit, nous représentons sous forme de modèle GAMM (FIG. 2), le même exemple AM (FIG. 1). Dans ce modèle, les entités seront représentées par des noeuds et les relations par des arêtes selon les principes des graphes.

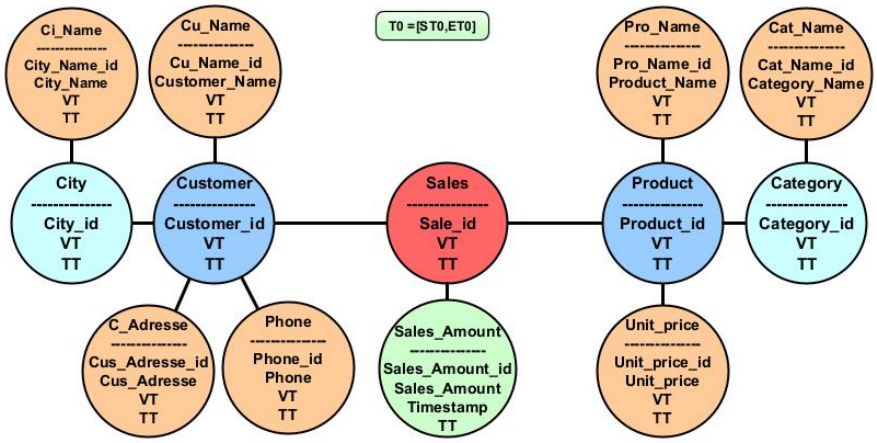

FIG. 2 – *Modèle logique basé sur le modèle GAMM*

Dans notre exemple, nous avons deux dimensions PRODUCT et COSTUMER, un fait SALES, une mesure *Sales_Amount* et des niveaux des hiérarchies CATEGORY pour PRODUCT et CITY pour COSTUMER. Ces Dimensions ainsi que ces niveaux sont caractérisés par des noeuds descripteurs.

Les règles de passage d'un modèle multidimensionnel classique vers un Modèle multidimensionnel agile à base de graphe ont été établies comme suit :

1. Chaque entité du modèle (fait, dimension, mesure, niveau et attribut) est représentée par un noeud propre
2. Toutes les relations sont représentées par des arrêtes
3. Les faits sont en relation directe avec les mesures
4. Les faits sont en relation directe avec les dimensions
5. Les niveaux sont en relation avec une (dimension/un autre niveau) selon leur profondeur
6. Les attributs sont en relation directe avec les (dimensions/niveaux)
7. Les niveaux constituent des hiérarchies selon des axes d'analyse organisés du niveau d'agrégation le plus fin au plus élevé
8. Toutes les entités possèdent un étiquetage chronologique (VT = Valid_Time et TT = Transction_time) conformément aux principes des bases de données temporelles.

Ce modèle sépare les concepts métiers des descripteurs (attributs) afin de permettre une évolution indépendante de chaque entité. En effet, les faits, les dimensions ainsi que les niveaux des hiérarchies sont représentés par des noeuds correspondant aux concepts de base et auxquels sont liés d'autres noeuds représentant les descripteurs. Ainsi, le modèle GAMM est représenté par la fonction suivante

$$GAMM(t) = \{F, D, FAssoc[F, D](t)\}$$

$GAMM(t)$ représente la version du schéma à l'instant t.
$t \in T = [ST, ET]$ représente la période de validité de la version du schéma, où ST (*Starting_Time*) = Date de début de version et ET (*Ending_Time*) = Date de fin de version.
$F = \{f_i(t)\}, i \in [1, *]$: Représente l'ensemble des faits à l'instant t. Chaque fait possède des mesures représentées par des noeuds indépendants
$f_i(t)$: Représente le fait f_i à l'instant t

$D = \{d_j(t)\}, j \in [1, *]$: Représente l'ensemble des dimensions selon lesquelles $f_i(t)$ est analysable à l'instant t
$d_j(t)$: Représente la dimension d_j l'instant t. Chaque dimension peut posséder des hiérarchies représentées par des niveaux et des attributs descripteurs conformément à la figure 3

$FAssoc[F, D](t) : f_i(t) \Longrightarrow \{d_j(t), ST, ET\}$ où $j \in [1, *]$: représente la fonction d'association de l'ensemble des dimensions $\{d_j(t)\}$ avec le fait $f_i(t)$ à l'instant t

Exemple :

$$GAMM(t_0) = \{\{\text{SALES}\}, \{\text{PRODUCT}, \text{STORE}\},$$

$$\{\text{SALES} \Longrightarrow \text{PRODUCT}, \text{SALES} \Longrightarrow \text{STORE}\}\}$$

Les dimensions et les niveaux ont les mêmes caractéristiques, chacun étant constitué d'un noeud principal représentant le concept métier et auquel sont associés des noeuds attributs (descripteurs). Chaque dimension peut avoir plusieurs hiérarchies en fonction des axes d'analyse. Ces axes d'analyse sont définis par les niveaux qui les composent, où les noeuds de premier niveau sont directement associés aux noeuds de la dimension. Cette représentation nous permet d'apporter une agilité dans l'évolution du modèle tant au niveau du schéma qu'au niveau des instances de données tout en conservant l'historique de ces évolutions. Dans cet article, nous nous intéressons à l'évolution du schéma que nous présentons dans la section suivante. L'évolution des instances des données n'est pas traitée ici.

4 Évolution de GAMM

4.1 Types d'évolution à l'aide du modèle GAMM

Le modèle GAMM est appelé à évoluer dans le temps en termes de structure et de données selon de nouveaux besoins d'analyse et des nouvelles sources de données. Les différentes versions du schéma produites grâce au processus d'évolution structurelle sont représentées dans Fig. 3. Nous présentons dans ce qui suit, trois nouveaux besoins d'analyse que nous détaillons

selon l'évolution permise par le modèle GAMM (Les opérations de suppression et de modification ne sont pas décrites dans cet article en raison du manque d'espace).

1. Ajout d'un nouveau niveau COUNTRY : (COSTUMER $\succ$ CITY $\succ$ COUNTRY).
   (le symbole $\succ$ représente un opérateur d'agrégation d'un niveau 1 à un niveau 2). Celui-ci ayant l'attribut *Country_Name* au niveau COSTUMER de la dimension COSTUMER, représenté par le schéma de [Fig. 3 :$GAMM(t_1)$], crée une nouvelle hiérarchie répondant ainsi à un nouveau besoin d'analyse géographique plus détaillée utilisant l'appartenance à un Pays.
   L'ajout de cette nouvelle hiérarchie ne nécessite pas de modification au niveau du fait SALES, ni sur les mesures qui lui sont associées, étant donné que le niveau ajouté ne représente pas un niveau de base et ne correspond pas, par conséquent, au niveau de granularité le plus fin de la dimension COSTUMER. Toutefois, les requêtes d'analyses devront être mises à jour, notamment pour la génération de cubes OLAP construits en fonction de ladite dimension.

2. Ajout d'une nouvelle dimension SUPPLIER
   L'ajout de cette nouvelle dimension ayant les attributs *Company_Name*, *Address* et *Phone*, représentée par le schéma de [Fig. 3 :$GAMM(t_2)$], offre au modèle un nouvel axe d'analyse répondant ainsi à un nouveau besoin d'analyse exprimé par la dimension ajoutée.
   Ce nouvel axe d'analyse augmente le niveau de granularité des mesures présentes dans le fait SALES. Par conséquent, de nouvelles instances de ces mesures seront créées pour répondre à ce nouveau niveau de détails. Les anciennes instances seront conservées pour une historisation. La *Surrogate_Key* permettra de différencier entre ces instances.

3. Ajout d'une nouvelle mesure *Quantity*
   L'ajout de cette nouvelle mesure représenté par le schéma de [Fig. 3 :$GAMM(t_3)$], répond à un nouveau besoin d'analyse à savoir la quantité de ventes. Cette opération d'ajout ne nécessite pas de modification directe au niveau du fait SALES. Cependant, cette nouvelle mesure doit avoir le même niveau de granularité que les mesures déjà associées à la dernière instance du fait SALES.
   Cette opération d'ajout ne nécessite pas de modification directe au niveau du fait SALES. Cependant, la nouvelle mesure doit avoir le même niveau de granularité que les mesures déjà associées à la dernière instance du fait SALES

Le formalisme du modèle GAMM permet toutes les évolutions possibles, à savoir l'ajout d'un attribut, d'une dimension, d'un niveau, d'une mesure et d'un fait (GAMM permet également toutes les suppressions d'entités. toutefois, par souci d'espace, nous avons traité dans cet article que les opérateurs d'ajout). Ces évolutions sont établies par des opérateurs d'évolution détaillés dans ce qui suit.

4.2 Opérateurs d'évolution de schéma dans GAMM

GAMM est un modèle évolutif en multi-versions où chaque version représente l'état du schéma à l'instant t. Les évolutions se font selon la fonction suivant :

$$GAMM(t_{n+1}) = GAMM(t_n) + Evolution\_Operateur(t_{n+1})$$

Où : $GAMM(t_{n+1})$: représente l'état du modèle après l'évolution, $GAMM(t_n)$: représente l'état du modèle à l'instant t_n (avant évolution), et $Evolution\_Operateur(t_{n+1})$: représente l'opérateur d'évolution à appliquer sur une version donnée selon le type d'évolution. Les opérateurs d'évolutions sont définis dans TAB. 2.

Évolution	Opérateur d'évolution	description
Ajout d'un attribut	Add_attribute(att,dim) Add_attribute(att,niv)	Ajout de l'attribut att à la dimension dim ou au niveau niv
Ajout d'une mesure	Add_measure(msr,fct)	Ajout de la mesure msr au fait fct
Ajout d'une dimension	Add_dimension(dim,fct,[att])	Ajout de la dimension dim au fait fct avec l'ensemble des attributes [att]
Ajout d'un niveau	Add_level(niv,dim) Add_level(niv,niv2)	Ajout du niveau niv à la dimension dim ou au niveau niv2
Ajout d'un fait	Add_fact(fait[dim],[msr])	Ajout du fait fact à l'ensemble des dimensions [dim] avec l'ensemble des mesures [msr]

TAB. 2 – *Opérateurs d'évolution de schéma.*

Nous reprenons les trois évolutions citées précédemment sous forme d'exemples pour illustrer la fonction d'évolution.

Exemple 1 : Évolution du modèle par l'ajout d'un niveau à une hiérarchie

$$GAMM(t_1) = GAMM(t_0) + Add\_level(\text{COUNTY}, \text{CITY})$$

Où COUNTRY représente le niveau de la hiérarchie à ajouter au niveau CITY.

Exemple 2 : Évolution du modèle par l'ajout d'une dimension

$$GAMM(t_2) = GAMM(t_1) + Add\_dimension(\text{SUPPLIER}, \text{SALES},$$

$$[\textit{Pro\_Name}, \textit{S\_adress}, \textit{Cat\_Name}])$$

Où SUPPLIER représente la nouvelle dimension de la hiérarchie à ajouter au fait SALES avec les attributs *Pro_Name, S_adress et Cat_Came*.

Exemple 3 : Évolution du modèle par l'ajout d'une mesure

$$GAMM(t_3) = GAMM(t_2) + Add\_measure(\textit{Quantity}, \text{SALES})$$

Où *Quantity* représente la nouvelle mesure à ajouter au fait SALES.

La figure Fig. 3 montre les schémas des différentes versions créées par les évolutions précédentes ainsi que le modèle initial. En effet, Après l'application des trois évolutions, on peut observer que le même modèle contient tous les schémas précédents et que toutes les versions sont des sous-versions du modèle GAMM. Cette caractéristique incrémentale est acquise en adoptant le principe de la modélisation d'ancrage dans la conception de notre modèle. Ainsi, aucune redondance de données n'a lieu.

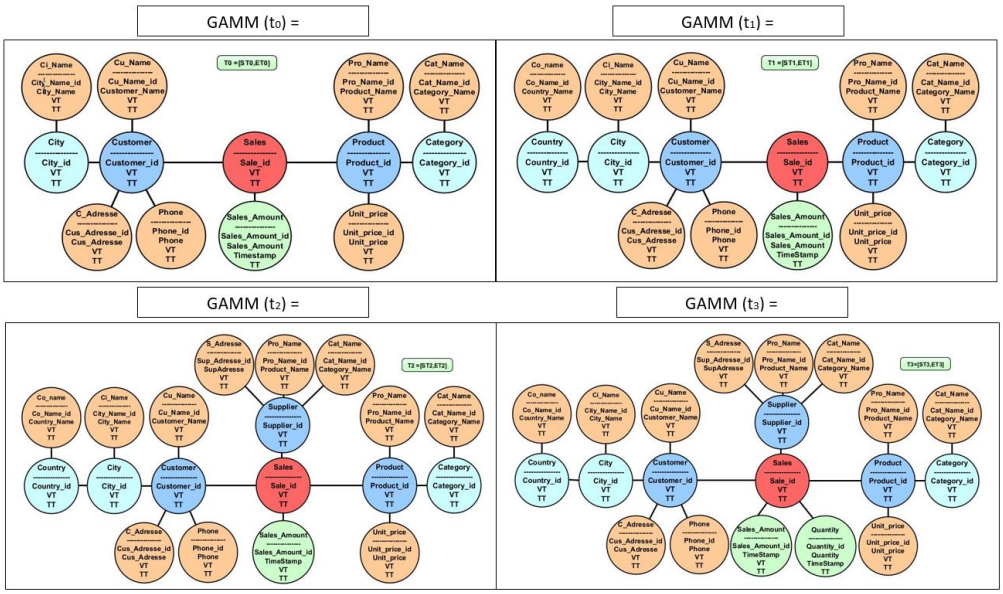

FIG. 3 – *Évolution du modèle par les ajouts d'un niveau, d'une dimension et d'une mesure.*

Pour la gestion et l'identification des versions engendrées par les différentes évolutions, nous avons mis en place un méta-modèle qui sera détaillé dans la section suivante.

4.3 Gestion des versions

Pour gérer les différentes versions issues de l'évolution du modèle GAMM, nous avons défini un méta-modèle qui permet d'identifier toutes les versions présentes dans le modèle ainsi que toutes les entités appartenant à ces versions. Ce méta-modèle est basé sur une classe centrale appelée *Version* contenant l'identifiant de la version, son *Start-Time* et son *Ending-Time* pour l'identification de la période de validité de chaque version. Nous représentons le méta-modèle GAMM par le diagramme représenté dans (FIG 4). Toutes les classes pivotent autour

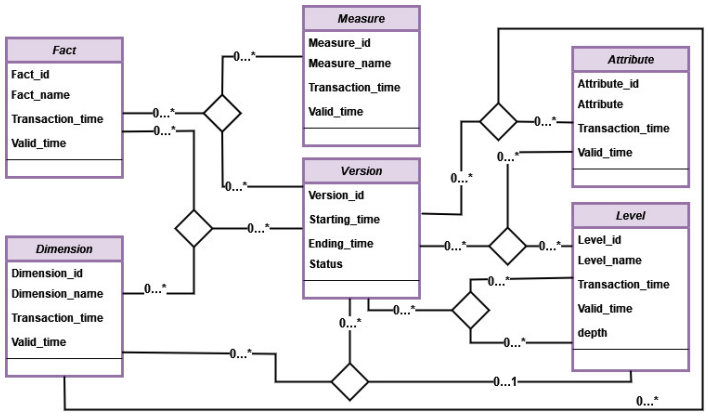

FIG. 4 – *Méta-Modèle de GAMM*

de la classe centrale *Version* permettant de déterminer les faits appartenant aux versions ainsi que les mesures et les dimensions qui lui sont associées. Le diagramme détermine également les niveaux des hiérarchies associées aux dimensions à un moment donné, l'ordre de ces niveaux et les attributs associés aux dimensions/niveaux. A chaque nouvelle création d'entité dans le modèle, une nouvelle version est créée où son *Starting_Time* prend la date de création et son *Ending_Time* prend la valeur NULL jusqu'à la fin de sa validité par la création d'une autre version et son statut prend la valeur *Last*. De même, le *Ending_Time* de la version précédente est mis à jour et son statut prend la valeur *Old*. Toutes les entités de la version précédente déterminant la structure du schéma seront reconduites dans cette nouvelle version en sus de l'entité nouvellement créée.

5 Prototype logiciel

Pour valider notre modèle GAMM, nous avons développé un prototype logiciel qui inclut à la fois le méta-modèle et le les différentes versions générées. Ce prototype a été développé en architecture 3-tiers avec un premier niveau *front-end* basé sur les technologie WEB et utilisant la bibliothèque GoJS pour créer et manipuler des diagrammes et des graphiques interactifs sur le web ; un deuxième niveau *back-end* développé en JavaEE représente le coeur des traitements ; et un troisième niveau responsable de la gestion des données avec une partie méta-données implémentée sur une base de données relationnelle pour la gestion du niveau logique et une partie pour les données implémentées sur la base de données de graphes NoSQL Neo4j pour la gestion du niveau physique. La partie gestion des versions est déjà finalisée. Nous travaillons actuellement sur la partie gestion des données.

La figure Fig. 5 représente la partie *front-end* du prototype où nous avons trois principaux volets. Un volet menu pour les différentes opérations liées à la gestion de la structure, des données et la partie analyse, un second volet pour le parcours des versions, et un volet principal pour l'affichage des différents schémas de versions.

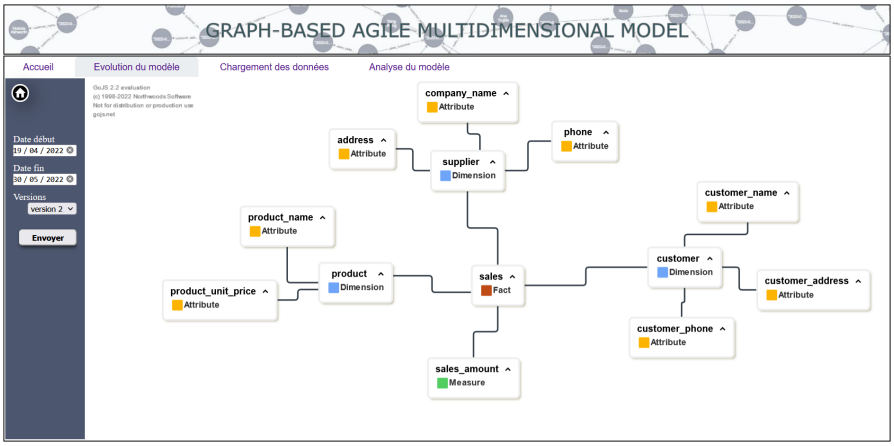

FIG. 5 – *Prototype GAMM*

Un premier travail de validation du modèle notamment sur le concept évolution en version où nous avons opté pour le processus d'instanciation sur Neo4j par plusieurs versions chronologiques où des requêtes ont été appliquées sur chaque version. Pour ce faire, nous avons utilisé l'entrepôt de données Microsoft *Northwind*. Il s'agit de données relatives à une activité commerciale afin d'analyser les montants des ventes et les quantités vendues.

1. Création d'une 1ère version correspondant à GAMM(t_1) caractérisée par l'ajout d'un niveau *Country* avec ST=01/01/2019 et ED=31/12/2019
2. Création d'une 2ème version correspondant à GAMM(t_2) caractérisée par l'ajout de la dimension *Supplier* avec ST=01/01/2020 et ED=31/12/2020
3. Création d'une 3ème version correspondant à GAMM(t_3) caractérisée par l'ajout de la mesure *Quantity* avec ST=01/01/2021 et ED=31/12/2021

Nous avons pu appliquer des requêtes sur chacune des versions ainsi obtenues :

Requête 1 : Affichage du montant des ventes par pays dans la version 1

```
MATCH (cnm:customer_name)←[rel1:customer_name]-(c:customer)←[rel2:sales_customer]
-(s:sales)-[rel3:sales_product]→(p:product)-[rel4:product_name]→(pnm:product_name),
(cou:country)←[rel1:city_country]-(ci:city)←[rel2:customer_city]-(c:customer)
MATCH (sam:sales_amount {customer_id: s.customer_id, product_id:s.product_id})
WHERE sam.valid_date ⩽ date({year:1996,month:12})
RETURN cou.country as ctry,sum(sam.sales_amount) ORDER BY cou.country
```

Requête 2 : Affichage du montant total par client, produit et fournisseur dans la version 2

Requête 3 : Affichage de la quantité vendue par client, produit et fournisseur dans la version 3

```
MATCH (cnm:customer_name)←[rel1:customer_name]-(c:customer)←[rel2:sales_customer]-(s:sales)-[rel3:sales_product]→
(p:product)-[rel4:product_name]→(pnm:product_name),(s:sales)-[rel5:sales_supplier]→(sup:supplier)
-[rel6:supplier_company_name]→(scn:supplier_company_name)
MATCH (sam:sales_amount {customer_id: s.customer_id, product_id:s.product_id , supplier_id:s.supplier_id})
WHERE date({year:1997,month:1}) ⩽ sam.valid_date ⩽ date({year:1997,month:12})
RETURN cnm.customer_name as cname,pnm.product_name as pname, scn.supplier_company_name as sname, sum(sam.sales_amount)
ORDER BY cnm.customer_name, pnm.product_name, scn.supplier_company_name
```

```
MATCH (cnm:customer_name)←[rel1:customer_name]-(c:customer)←[rel2:sales_customer]-(s:sales)-[rel3:sales_product]
→(p:product)-[rel4:product_name]→(pnm:product_name), (s:sales)-[rel5: sales_supplier]→(sup:supplier)-
(rel6:supplier_company_name]→(scn:supplier_company_name)
MATCH (qty:quantity {customer_id: s.customer_id, product_id:s.product_id, supplier_id:s.supplier_id})
WHERE date({year:1998,month:1}) ⩽ qty.valid_date ⩽ date({year:1998,month:12})
RETURN cnm.customer_name as cname,pnm.product_name as pname, scn.supplier_company_name as sname, sum(qty.quantity)
ORDER BY cnm.customer_name, pnm.product_name, scn.supplier_company_name
```

6 Conclusion

L'avènement des big data avec notamment la prolifération des sources de données ont fait émerger de nouvelles questions de recherche dans le domaine des entrepôts de données. Plus précisément, il s'agit de comment faire évoluer le schéma et les instances des données dans un modèle multidimensionnel pour à la fois intégrer de nouvelles sources de données et tenir compte de nouveaux besoins d'analyse. Les ED s'appuient généralement sur le modèle en étoile qui est figé et inadapté aux changements. Pour pallier ce problème, nous avons proposé dans cet article un modèle multidimensionnel agile, appelé GAMM, qui s'appuie sur le formalisme ensembliste AM au niveau conceptuel et sur la structure de graphe au niveau logique et physique. Grâce à des fonctions d'évolution que nous avons définies, GAMM permet l'évolution de schéma dans un entrepôt de données en créant à chaque fois une nouvelle version du modèle multidimensionnel. Nous avons proposé également un méta-modèle pour gérer les différentes versions de GAMM. Nous avons validé GAMM par un prototype logiciel en architecture 3-tiers. Ce travail ouvre de nombreuses perspectives. A court terme, nous étudions l'évolution des instances des données dans GAMM, et à moyen terme nous nous intéresserons aux requêtes décisionnelles croisées. Par ailleurs, nous proposons de mener une réflexion sur des analyses poussées sur des modèles de graphes afin de permettre à l'analyse en ligne de produire des modèles explicatifs et prédicatifs.

Références

Ahmed, W., E. Zimányi, A. A. Vaisman, et R. Wrembel (2020). A temporal multidimensional model and OLAP operators. *Int. J. Data Warehous. Min. 16*(4), 112–143.

Ahmed, W., E. Zimányi, A. A. Vaisman, et R. Wrembel (2021). Schema evolution in multiversion data warehouses. *Int. J. Data Warehous. Min. 17*(4), 1–28.

Ahmed, W., E. Zimányi, et R. Wrembel (2014). A logical model for multiversion data warehouses. In L. Bellatreche et M. K. Mohania (Eds.), *Data Warehousing and Knowledge Discovery - 16th International Conference, DaWaK 2014, Germany*, Lecture Notes in Computer Science. Springer.

Bebel, B., J. Eder, C. Koncilia, T. Morzy, et R. Wrembel (2004). Creation and management of versions in multiversion data warehouse. In H. Haddad, A. Omicini, R. L. Wainwright, et

L. M. Liebrock (Eds.), *Proceedings of the 2004 Symposium on Applied Computing (SAC), Cyprus*. ACM.

Bellahsene, Z. (2002). Schema evolution in data warehouses. *Knowledge and Information Systems 4*(3).

Benitez-Guerrero, E., C. Collet, et M. Adiba (2004). The whes approach to data warehouse evolution. *e-Gnosis Num.002*.

Blaschka, M., C. Sapia, et G. Höfling (1999). On schema evolution in multidimensional databases. In *International Conference on Data Warehousing and Knowledge Discovery*, pp. 153–164. Springer.

Bliujute, R., S. Saltenis, G. Slivinskas, et C. S. Jensen (1998). Systematic change management in dimensional data warehousing. In *Proceedings of the Third International Baltic Workshop on Data Bases and Information Systems, Riga, Latvia*. Citeseer.

Faisal, S., M. Sarwar, K. Shahzad, S. Sarwar, S. W. Jaffry, et M. M. Yousaf (2017). Temporal and evolving data warehouse design. *Sci. Program. 2017*, 7392349 :1–7392349 :18.

Favre, C., F. Bentayeb, et O. Boussaid (2007). Evolution of data warehouses' optimization : A workload perspective. In *International Conference on Data Warehousing and Knowledge Discovery*. Springer.

Fentaw, A. E. (2014). *Data Vault Modelling : An Introductory Guide*. Helsinki : Helsinki Metropolia University of Applied Sciences.

Garani, G., G. K. Adam, et D. Ventzas (2016). Temporal data warehouse logical modelling. *Int. J. Data Min. Model. Manag. 8*(2), 144–159.

Golfarelli, M. et S. Rizzi (2011). Temporal data warehousing : Approaches and techniques. In D. Taniar et L. Chen (Eds.), *Integrations of Data Warehousing, Data Mining and Database Technologies - Innovative Approaches*, pp. 1–18. Information Science Reference.

Golfarelli, M. et S. Rizzi (2018). From star schemas to big data : 20+ years of data warehouse research. In S. Flesca, S. Greco, E. Masciari, et D. Saccà (Eds.), *A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years*, Studies in Big Data. Springer International Publishing.

Hamitouche, Boussaid, B. (2021). Vers une modélisation agile des entrepôts de données basée sur la technique anchor modelling.

Hultgren, H. (2012a). Data vault modelling guide–introductory guide to data vault modelling. g.a.

Hultgren, H. (2012b). Introductory guide to data vault modeling. *Genesee Academy*.

Hurtado, C. A., A. O. Mendelzon, et A. A. Vaisman (1999). Maintaining data cubes under dimension updates. In *Proceedings 15th International Conference on Data Engineering (Cat. No. 99CB36337)*, pp. 346–355. IEEE.

Inmon, W. H. (1992). *Building the Data Warehouse*. USA : John Wiley & Sons, Inc.

Kimball, R. (1996). *The Data Warehouse Toolkit : Practical Techniques for Building Dimensional Data Warehouses*. USA : John Wiley & Sons, Inc.

Linstedt, D. (2015). Data vault basics. *https ://danlinstedt.com*.

Mazón, J.-N., J. Pardillo, et J. Trujillo (2006). Applying transformations to model driven

data warehouses. In A. M. Tjoa et J. Trujillo (Eds.), *Data Warehousing and Knowledge Discovery*, Berlin, Heidelberg, pp. 13–22. Springer Berlin Heidelberg.

Mendelzon, A. et A. Vaisman (2000). Temporal queries in olap. pp. 242–253.

Morzy, T. et R. Wrembel (2003). Modeling a multiversion data warehouse : A formal approach. In *ICEIS 2003, Proceedings of the 5th International Conference on Enterprise Information Systems, Angers, France, April 22-26, 2003*, pp. 120–127.

Němec, R. (2012). The comparison of anchor and star schema from a query performance perspective. *International Journal of Computer and Information Engineering 6*(11), 1421 – 1425.

Němec, R. et F. Zapletal (2014). The design of multidimensional data model using principles of the anchor data modeling : An assessment of experimental approach based on query execution performance. *WSEAS Transactions on Computers 13*, 177–194.

Phungtua-Eng, T. et S. Chittayasothorn (2019). Slowly changing dimension handling in data warehouses using temporal database features. In N. T. Nguyen, F. L. Gaol, T. Hong, et B. Trawinski (Eds.), *Intelligent Information and Database Systems - 11th Asian Conference, ACIIDS 2019, Indonesia*. Springer.

Ravat, F., O. Teste, et G. Zurfluh (2006). A multiversion-based multidimensional model. In A. M. Tjoa et J. Trujillo (Eds.), *Data Warehousing and Knowledge Discovery, 8th International Conference, DaWaK 2006, Krakow, Poland, September 4-8, 2006, Proceedings*, Volume 4081 of *Lecture Notes in Computer Science*, pp. 65–74. Springer.

Regardt, O., L. Rönnbäck, M. Bergholtz, P. Johannesson, et P. Wohed (2009). Anchor modeling. In *International Conference on Conceptual Modeling*, pp. 234–250. Springer.

Rizzi, S. et M. Golfarelli (2007). X-time : Schema versioning and cross-version querying in data warehouses. In R. Chirkova, A. Dogac, M. T. Özsu, et T. K. Sellis (Eds.), *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pp. 1471–1472. IEEE Computer Society.

Rönnbäck, L. et H. Hultgren (2013). Comparing anchor modeling with data vault modeling. *Modeling for the modern Data Warehouse, White paper*.

Rönnbäck, L., O. Regardt, M. Bergholtz, P. Johannesson, et P. Wohed (2010). Anchor modeling—agile information modeling in evolving data environments. *Data & Knowledge Engineering 69*(12).

Solodovnikova, D. (2007). Data warehouse evolution framework. In S. D. Kuznetsov, A. Fomichev, B. Novikov, et D. Shaporenkov (Eds.), *Proceedings of the SYRCoDIS 2007 Colloquium on Databases and Information Systems, Moscow, Russia, May 31 - June 1, 2007*. CEUR-WS.org.

Talwar, K. et A. Gosain (2012). Hierarchy classification for data warehouse : A survey. *Procedia Technology 6*, 460–468.

Wrembel, R. et T. Morzy (2006). Managing and querying versions of multiversion data warehouse. In Y. E. Ioannidis, M. H. Scholl, J. W. Schmidt, F. Matthes, M. Hatzopoulos, K. Böhm, A. Kemper, T. Grust, et C. Böhm (Eds.), *Advances in Database Technology - EDBT 2006*, Volume 3896. Springer.

Summary

In the context of big data where the volume of data is continuously growing, we propose in this paper an agile approach to schema evolution in data warehouses that allows designers to integrate new data sources and take into account new user needs in order to enrich the analysis possibilities. Our approach is based on a multi-versioned evolutionary model using graph-based assembly modelling. A meta-model will allow the management of the versions of the warehouse schema. We also propose evolution functions at the schema level (the evolution of instances is not dealt with here). We validate our approach with a software prototype that manages both versions and schema evolution in the model.