# Modèle multidimensionnel agile pour les données massives

Redha Benhissen

N° d'ordre NNT : 2023LYO20071

THÈSE de DOCTORAT DE L'UNIVERSITÉ LUMIÈRE LYON 2

**École Doctorale : ED 512**

**Informatique et Mathématiques**

Discipline : Informatique

Soutenue publiquement le 5 décembre 2023 par :

Redha BENHISSEN

# Modèle multidimensionnel agile pour les données massives.

Devant le jury composé de :

Olivier TESTE, Professeur des Universités, Université Toulouse 2, Président

Samira SI-SAÏD CHERFI, Professeure des Universités, Conservatoire National Arts et Métiers, Rapporteure

Laurent D'ORAZIO, Professeur des Universités, Université Rennes 1, Rapporteur

Veronika PERALTA COSTABEL, Maîtresse de conférences, Université de Tours, Examinatrice

Fadila BENTAYEB, Professeure des Universités, Université Lumière Lyon 2, Directrice de thèse

Omar BOUSSAÏD, Professeur des Universités, Université Lumière Lyon 2, Co-Directeur de thèse

# Contrat de diffusion

**UNIVERSITÉ LUMIÈRE LYON 2**

**ÉCOLE DOCTORALE N°512**

**INFORMATIQUE ET MATHÉMATIQUES (InfoMaths)**

# THÈSE

pour obtenir le titre de

**Docteur en Informatique**

Présentée et soutenue par

Redha BENHISSEN

---

## Modèle multidimensionnel agile pour les données massives

---

Soutenue publiquement le 05/12/2023

Devant le jury composé de :

| | | |
|---|---|---|
| Pr. Olivier TESTE | Université Toulouse 2 | Président |
| Pr. Fadila BENTAYEB | Université Lumière Lyon 2 | Co-Directrice |
| Pr. Omar BOUSSAID | Université Lumière Lyon 2 | Co-Directeur |
| Pr. Samira SI-SAID CHERFI | Conservatoire National des Arts et Métiers | Rapportrice |
| Pr. Laurent D'ORAZIO | Université de Rennes 1 | Rapporteur |
| Mcf. Veronika PERALTA COSTABEL | Université de Tours | Examinatrice |

# Remerciements

La réalisation de cette thèse, bien que souvent perçue comme un voyage solitaire, a en réalité été façonnée par un environnement riche en interactions.

Je tiens à exprimer mes sincères remerciements à ceux qui ont dirigé ces travaux en me prodiguant leurs conseils, leur bienveillance et leur patience : Fadila BENTAYEB et Omar BOUSSAID.

Je souhaite également adresser ma gratitude aux rapporteurs de ce document, Samira SI-SAID CHERFI et Laurent D'ORAZIO, ainsi qu'aux autres membres du jury, Olivier TESTE et Veronika PERALTA COSTABEL, pour l'intérêt qu'ils ont porté à mes travaux.

Un remerciement chaleureux est également destiné à toute l'équipe du laboratoire ERIC pour leur accueil, en particulier à Jérôme DARMONT et Julien JACQUES, qui ont successivement présidé le laboratoire.

Enfin, et surtout, je tiens à exprimer ma profonde reconnaissance envers ma famille. Un hommage particulier est dédié à mon regretté père, Omar, que DIEU l'accueille dans son vaste paradis, ainsi qu'à ma mère, Yassia, que DIEU la préserve. Un merci affectueux à mon épouse, Nachida, et à mes enfants, Wassim, Ines, Djad et Djaouad. Je suis également reconnaissant envers ma sœur, Imène, ainsi que mes frères, Fethi, Amine et Walid, qui ont été présents à chaque étape de cette période importante de ma vie.

**Résumé**

La thèse intitulée « Modèle Multidimensionnel Agile pour les Données Massives » explore le défi inhérent à l'analyse et à l'évolution des Big Data dans le contexte des entreprises modernes. L'avancée rapide des technologies d'analyse des Big Data, telles que la Business Intelligence (BI), l'apprentissage automatique et l'intelligence artificielle, a un impact profond sur le processus de prise de décision au sein des entreprises. Cette évolution pousse les entreprises à repenser leurs architectures d'information afin de gérer et d'analyser des volumes massifs de données. Les entrepôts de données jouent un rôle central dans cette transformation, en facilitant l'exploration et l'analyse approfondie des données.

Cependant, les modèles multidimensionnels traditionnels présentent des limites face à la diversité et à l'évolution des sources de données ainsi qu'aux besoins changeants d'analyse. C'est dans ce contexte que cette thèse propose le modèle multidimensionnel agile à base de graphes, nommé GAMM. Les objectifs de la thèse consistent à rendre les modèles multidimensionnels plus flexibles, à gérer l'évolution des schémas et des données, et à automatiser ce processus tout en préservant la cohérence des analyses.

Trois contributions majeures ont été apportées. La première contribution se concentre sur l'évolution des schémas multidimensionnels. Le modèle GAMM est développé pour permettre l'évolution des schémas sous forme de versions multiples, en utilisant une base de données orientée graphe de type NoSQL pour une flexibilité accrue. Chaque version de schéma est associée à un intervalle de temps, facilitant la navigation à travers les différentes versions. La deuxième contribution traite de la gestion temporelle des données dans les entrepôts de données multi-versions. L'aspect temporel est introduit pour conserver l'historique d'évolution des données, y compris les modifications au niveau des dimensions. Des étiquetages temporels sont utilisés pour assurer la cohérence des analyses au fil du temps. La troisième contribution consiste en l'automatisation de l'évolution des schémas. Des techniques d'exploration de données sont intégrées pour détecter automatiquement des schémas multidimensionnels à partir de diverses sources de données. Cette automatisation améliore l'efficacité et la précision de la gestion de l'évolution des schémas, tout en facilitant la création de niveaux de hiérarchie sur plusieurs dimensions. Cette approche automatisée rend les analyses multidimensionnelles plus pertinentes et précises pour la prise de décision.

Les travaux développés dans le cadre de cette thèse proposent une approche novatrice pour l'évolution des entrepôts de données multidimensionnels dans un environnement de Big Data. Le modèle proposé permet d'intégrer de nouvelles sources de données, de s'adapter aux besoins d'analyse changeants, et de préserver l'historique de ces évolutions. Il offre ainsi une solution élégante pour la flexibilité et la cohérence des schémas et des données dans un environnement en constante évolution.

**Mots-clés :** Entrepôt de données, Modèle multidimensionnel, Schéma en multi-version, Entrepôt de données temporelles, Évolution automatique des schémas, Bases de données orientées graphes, NoSQL.

**Abstract**

The thesis titled « Agile Multidimensional Model for Big Data » delves into the inherent challenge of analyzing and evolving Big Data in the context of modern enterprises. The rapid advancement of Big Data analytics technologies, such as Business Intelligence (BI), machine learning, and artificial intelligence, profoundly impacts the decision-making process within businesses. This evolution compels enterprises to rethink their information architectures to manage and analyze massive volumes of data effectively. Data warehouses play a pivotal role in this transformation by facilitating the exploration and in-depth analysis of data.

However, traditional multidimensional models have limitations in the face of the diversity and evolution of data sources as well as changing analysis needs. In this context, this thesis proposes the agile multidimensional model based on graphs, named GAMM. The objectives of the thesis are to make multidimensional models more flexible, manage evolving schemas and data, and automate this process while preserving analysis coherence.

Three major contributions have been made. The first contribution focuses on the evolution of multidimensional schemas. The GAMM model is developed to enable schema evolution in the form of multiple versions, using a NoSQL graph database for increased flexibility. Each schema version is associated with a time interval, facilitating navigation across different versions. The second contribution deals with the temporal management of data in multi-version data warehouses. Temporality is introduced to preserve the historical evolution of data, including changes at the dimension level. Temporal labeling is used to ensure the consistency of analyses over time. The third contribution involves the automation of schema evolution. Data exploration techniques are integrated to automatically detect multidimensional schemas from various data sources. This automation improves the efficiency and accuracy of schema evolution management while facilitating the creation of hierarchy levels across multiple dimensions. This automated approach makes multidimensional analyses more relevant and precise for decision-making.

The work conducted within the scope of this thesis presents an innovative approach to the evolution of multidimensional data warehouses in a Big Data environment. The proposed model enables the integration of new data sources, adaptation to changing analytical needs, and preservation of the history of these evolutions. It provides an elegant solution for flexibility and consistency in schemas and data within a constantly evolving environment.

**Keywords :** Data warehouse, Multidimensional model, Multi-version schema, Temporal data warehouse, Automatic schema evolution, Graph-based database, NoSQL.

# Table des matières

# Liste des tableaux

# Table des figures

# Acronymes

**BI** Business Intelligence. 1–3, 135

**Big Data** données massives. 1–5, 135, 137, 139

**DFM** Dimensional Fact Model. 45, 99

**DInst** Data Instance. 51

**DT** Decision_Time. 19

**ER** Entité-Relation. 50

**ERIC** Entrepôts, Représentation et Ingénierie des Connaissances. 2, 3

**ET** Ending_Time. 6, 51, 64, 81–84, 116, 117

**ETL** Extract, Transform, and Load. 2–4, 13, 18, 71, 79

**FD** From_Date. 7, 20, 46, 51, 59, 60, 71–77, 90

**FF** Facteur de Filtrage. 114, 125, 139

**GAMM** Graph-based Agile Multidimensional Model. i, ii, v, vi, 1, 6–9, 43, 44, 50–53, 55, 57, 59, 61–87, 89–93, 95, 96, 98, 99, 101–103, 108–110, 112, 113, 116–118, 133, 134, 136–139

**GTDW** Graph Temporal Data Warehouse. v, 51, 59–61

**NoSQL** Not only SQL. 1, 4, 6, 8, 11, 24, 40, 41, 59, 79, 117, 136

**OLAP** Online Analytical Processing. 1–3, 6, 13, 14, 17, 33, 59, 75, 88, 99, 108, 127, 136, 138, 139

**SCD** Slowly Changing Dimension. 22, 34, 39

**SQL** Structured Query Language. 17, 21, 24, 34, 36

**SSB** Star Schema Benchmark. ii, iv, vi, 6, 7, 111–115, 117–120, 122–126, 133

**ST** Starting_Time. 6, 51, 64, 81–84, 116, 117

**SV** Schema Version. 6, 51, 83

**T** période temporelle. 6, 51, 64, 81, 90, 117

**TD** To_Date. 7, 20, 46, 51, 59, 60, 71–77, 90

**TT** Transaction_Time. 6, 19, 33, 34, 51, 59, 60, 71–73, 75, 83, 84, 87, 88

**VT** Valid_Time. 7, 19, 33, 34, 51, 59, 60, 71–73, 75, 83, 90

CHAPITRE 1

# INTRODUCTION

## 1.1 Contexte général

L'analyse des données massives (Big Data) a évolué, au fil des années, grâce à l'adoption de technologies modernes et à l'importance croissante accordée à l'analyse avancée. Il n'existe pas de technologie unique qui englobe l'analyse des Big Data. Les entreprises recourent à différentes technologies pour tirer le meilleur parti des informations qu'elles collectent. Parmi ces technologies figurent la Business Intelligence (BI), l'apprentissage automatique, l'intelligence artificielle, etc. Ces tendances technologiques sont susceptibles de stimuler la demande en matière d'analyse des Big Data dans un avenir proche. Aujourd'hui, les entreprises se concentrent sur la refonte de l'architecture de leur système d'information, la consolidation des données et l'évolution des systèmes existants afin d'accéder à de vastes quantités de données pour les besoins de leurs travaux. Le Big Data a un impact considérable sur les entreprises, car il les aide à gérer et à analyser efficacement ces masses de données afin de mieux maîtriser leur environnement.

Dans le domaine de la BI, l'entreposage des données et la technologie Online Analytical Processing (OLAP) se sont révélés fort utiles pour des analyses efficaces des données. Les entrepôts de données sont des systèmes de stockage et de gestion de données pour faciliter la prise de décision. Ils sont utilisés pour collecter, organiser, intégrer et analyser de grandes quantités de données provenant de différentes sources, telles que des bases de données de production, des capteurs, des appareils de mesure, etc. Ils jouent un rôle essentiel dans la prise de décision dans de nombreux domaines, y compris la science en général, la médecine, l'économie, les sciences sociales et bien d'autres. L'un des objectifs clés des entrepôts de données est de faciliter l'exploration et l'analyse des données pour générer de nouvelles connaissances. Les acteurs en charge de la décision peuvent effectuer des requêtes complexes sur les données, appliquer des algorithmes d'analyse avancés et visualiser les résultats de manière interactive. Cela permet d'identifier des tendances, des corrélations et des modèles cachés dans les données qui pourraient conduire à de nouvelles découvertes (des *insights*) significatives.

Outre les entrepôts de données, les lacs de données sont devenus des supports de données essentiels dans le domaine de l'analyse des Big Data. En effet, un lac de

données est un *repository* de données brutes, non transformées, conservées dans leur format d'origine ; elles sont de différentes natures et structures ; et sans avoir besoin de les modéliser au préalable. Ces deux supports de données (les entrepôts de données et les lacs de données) constituent le noyau des systèmes modernes de prise de décision ; ils jouent un rôle clé dans la gestion, l'intégration et l'analyse des Big Data, permettant aux entreprises d'obtenir des informations précieuses pour la prise de décisions stratégiques.

La structuration des entrepôts de données s'effectue à travers des modèles multidimensionnels (Kimball, 1996). Les données sont alors organisées en fonction des sujets analysés, représentés par des « faits » (objets à observer), et des axes d'analyse (objets d'observation), représentés par des « dimensions » (tels le temps, le lieu, les produits, etc.). Les « faits » sont des regroupements d'indicateurs appelés « mesures » (tels que les ventes, les revenus, etc.), tandis que les « dimensions » déterminent le niveau de détails des mesures et peuvent être hiérarchisées par niveau. Les hiérarchies permettent de projeter l'analyse à différents niveaux de granularité, offrant ainsi des possibilités d'analyse ascendante et descendante. Cette structure permet une analyse efficace à plusieurs niveaux et facilite l'exploration des données selon différents angles.

Les entrepôts de données et les systèmes OLAP basés sur le modèle multidimensionnel représentent les données dans un espace à $n$ dimensions, communément appelé « cube de données » (ou hypercube de données). Cette représentation permet de naviguer dans les données à travers des opérations OLAP, telles que le *drill-down*, le *roll-up*, le *rotate*, le *slicing*, etc. Une caractéristique fondamentale du modèle multidimensionnel est sa capacité à visualiser les données selon différentes perspectives et à différents niveaux de détails. Les opérations OLAP concrétisent ces perspectives et niveaux de détails en fournissant un environnement interactif pour l'analyse des données.

L'architecture générale de l'entrepôt de données se compose de trois principaux niveaux (Vaisman and Zimányi, 2022). Au premier niveau, appelé « *back-end* », on retrouve les outils d'extraction, de transformation et de chargement (Extract, Transform, and Load) (ETL), qui sont utilisés pour alimenter l'entrepôt de données à partir de bases de données opérationnelles et de plusieurs autres sources de données. Au deuxième niveau, appelé « niveau entrepôt de données », on trouve un entrepôt de données d'entreprise et/ou plusieurs *data marts* (ou magasins de données), ainsi qu'un référentiel de méta-données qui stocke des informations sur les données de l'entrepôt. Il est important de noter qu'un entrepôt de données d'entreprise est centralisé, c.à.d., il peut couvrir plusieurs activités (voire l'ensemble) d'une organisation, tandis qu'un *data mart* est un mini entrepôt de données spécialisé ciblant un domaine fonctionnel ou départemental particulier au sein d'une organisation. Au troisième niveau, appelé « niveau OLAP », on peut créer des vues multidimensionnelles des données (cubes de données), ce qui permet aux utilisateurs d'effectuer des analyses métiers de type BI, OLAP, fouille de données, etc.

Ainsi, l'entrepôt de données est construit en collectant des données à partir de diverses sources, généralement, hétérogènes qui peuvent être structurées, peu ou pas

structurées. Ce support est alors alimenté à l'aide d'un processus ETL, permettant aussi d'assurer des rafraîchissements périodiques des données. Cependant, les modèles multidimensionnels classiques, basés sur les modèles en étoile, présentent certaines limites face à la diversification et à l'hétérogénéité des sources de données, aux besoins d'analyse changeants et à la nécessité de prendre en compte la sémantique dans le processus décisionnel. En effet, l'avènement du Web et l'abondance de données multimédias ont largement contribué à l'émergence de données qualifiées dans la littérature de « données complexes » qui ne sont souvent ni numériques ni symboliques (Bentayeb, 2011).

Cette thèse, intitulée « *Modèle Multidimensionnel Agile pour les Données Massives* », a été élaborée au sein du laboratoire Entrepôts, Représentation et Ingénierie des Connaissances (ERIC) de l'université Lumière Lyon 2. Le laboratoire ERIC est spécialisé dans les domaines de la modélisation des grands entrepôts de données complexes, la fouille des données massives et les processus d'aide à la décision.

## 1.2 Problématique et axes de recherche

Avec l'avènement des Big Data, les plates-formes de la BI ont évolué pour placer les données au cœur des actions des utilisateurs. Une quantité massive de données impose aux entreprises d'élargir continuellement leurs besoins d'analyses à des fins décisionnelles. C'est pourquoi le processus d'aide à la décision et les structures multidimensionnelles classiques ne doivent pas ignorer cette évolution lors de l'utilisation des solutions analytiques afin de prendre des décisions éclairées

Un schéma multidimensionnel est conçu selon des besoins d'analyse définis au préalable. La prise en charge de nouveaux besoins d'analyse implique la modification du schéma multidimensionnel. Toutefois, le modèle multidimensionnel classique est peu évolutif en général, dans le sens où les données sont emmagasinées sous forme de « silo », dont la modification du schéma est souvent complexe, chronophage et parfois impossible. La modification de schéma peut altérer le bon fonctionnement du système déjà en place notamment pour l'historisation des données intégrées.

L'intégration de nouvelles sources de données nécessite non seulement une adaptation du processus ETL, mais aussi celle du schéma physique de l'entrepôt de données. Toutefois, la modification du schéma physique peut s'avérer contraignante, voire impossible, notamment dans un contexte de bases de données relationnelles (partie que nous développerons dans la Section 2 relative à l'état de l'art).

De plus, la modification de schéma multidimensionnel et la conservation des données historisées dans l'entrepôt, impliquent souvent la création de plusieurs versions du schéma avec différents niveaux de granularité pour les données agrégées. C'est pourquoi, le processus OLAP doit être en mesure de s'adapter à ces évolutions afin de permettre une navigation fluide et cohérente dans les données à travers les différentes versions existantes.

Afin d'avoir une représentation fidèle du contexte d'analyse, il est nécessaire d'établir un processus d'actualisation régulière des données à partir des sources hétérogènes à l'aide d'un processus d'ETL. Un décalage entre les données sources et celles présentes dans l'entrepôt de données peut compromettre la cohérence des analyses, d'où l'importance du contrôle des nouvelles données dans le modèle multidimensionnel. Traditionnellement, le processus d'actualisation fournit régulièrement à l'entrepôt de données des données collectées et transformées. Celles-ci sont ensuite intégrées dans des structures multidimensionnelles adaptées aux différents scénarios de l'analyse décisionnelle. Cependant, une fois en service, l'entrepôt de données n'est pas seulement alimenté en nouveaux faits ; il peut également y avoir des changements dans les données des dimensions, tels que des modifications dans les relations hiérarchiques ou des attributs descriptifs. Une gestion appropriée de ces changements est un facteur clé pour assurer la cohérence des systèmes d'entreposage de données.

Depuis l'avènement des entrepôts de données, le processus d'entreposage constitue un domaine de recherche important, nécessitant la résolution de nombreux verrous scientifiques et technologiques, notamment ceux liés à l'aspect évolutif des entrepôts de données. Le problème de l'évolution des modèles multidimensionnels a fait l'objet de nombreuses études de recherche proposant différentes approches. Ces travaux peuvent être regroupés en deux principales catégories : la première concerne l'évolution des schémas, tandis que la seconde cible l'évolution temporelle des données.

C'est au vu de ces travaux que l'idée est venue de proposer un modèle multidimensionnel agile capable d'intégrer de nouvelles sources de données, tout en étant flexible face aux nouveaux besoins d'analyse, particulièrement dans un contexte de Big Data. Dans le cadre de nos travaux de recherche, nous nous sommes concentrés sur le développement de techniques visant à rendre les modèles multidimensionnels plus flexibles, tant au niveau conceptuel que physique. Nous avons également étudié l'évolution temporelle et incrémentale des données, les différents processus d'évolution des schémas et des données, ainsi que les méthodes d'interrogation des données dans un environnement évolutif.

Une des solutions envisageables au niveau physique est l'adoption de bases de données de type Not only SQL (NoSQL) dans le processus d'entreposage. En effet, dans un environnement de Big Data, les bases de données relationnelles sont peu appropriées pour gérer d'énormes volumes de données, et présentent certaines limites en termes de stockage, de passage à l'échelle et d'analyse. Il devient alors essentiel de recourir à de nouvelles solutions telles que les bases de données NoSQL. Celles-ci diffèrent des bases de données relationnelles, notamment pour le stockage de données non structurées, sans pour autant être liées à un schéma fixe prédéfini. Cela apporte un avantage considérable aux systèmes en termes d'évolutivité. Le NoSQL est utilisé pour le Big Data et les applications web en temps réel. Les géants de la technologie collectent chaque jour plusieurs teraoctets de données. Le journal « Statista Digital Economy Compass », révèle que 33 zettaoctets [1] de données numériques ont été créées dans le monde au

1. $1Zo = 10^{21}$ octets.

cours de l'année 2018 [2].

## 1.3 Objectifs

L'objectif principal de cette thèse est de proposer un modèle multidimensionnel agile adapté aux Big Data, capable de répondre aux nouveaux besoins d'analyse et de prendre en compte l'évolution des sources de données de manière flexible. Pour atteindre cet objectif, nous envisageons de concevoir un modèle qui permettra la prise en compte de nouvelles sources de données de manière fluide, sans compromettre la structure déjà existante. Nous nous intéressons aux différentes techniques et approches qui permettent de gérer les changements dans les sources de données, tant au niveau du schéma que celui du contenu.

Nous souhaitons également examiner les mécanismes d'évolution, à la fois manuels et automatiques, en prenant en compte la temporalité des versions de schéma et de données. Cela implique d'explorer les possibilités de modification du schéma multidimensionnel de manière à faciliter son évolution tout en préservant la cohérence des données historisées.

Nous définissons l'agilité dans un modèle multidimensionnel comme étant la capacité à construire des entrepôts de données de manière évolutive tout en accordant la priorité aux objectifs métier. Il s'agit de répondre de manière flexible et progressive aux nouveaux besoins d'analyse après intégration de nouvelles sources de données, et tout en préservant les constructions précédentes. Cette flexibilité offre une valeur ajoutée accrue aux décideurs et leur permet de mieux gérer leur environnement.

## 1.4 Contributions

Afin de répondre à la problématique soulevée dans le cadre de cette thèse, nous avons élaboré un modèle multidimensionnel agile destiné à gérer les données massives. Cette proposition englobe un formalisme, un méta-modèle, des règles de passage, un cadre de requêtes, ainsi que des études de validation. Fondé sur des structures de graphes, le modèle proposé est évolutif et se décline en plusieurs versions, permettant ainsi une évolution incrémentale tant au niveau des schémas que des données ; il permet aussi de conserver l'historique des données pour d'ultérieures consultations. En résolvant les problèmes conceptuels et physiques inhérents aux modèles multidimensionnels classiques, ce modèle constitue une avancée significative. En outre, nous avons également proposé une approche d'automatisation de l'évolution au sein de notre modèle.

Dans ce cadre, nos contributions dans cette thèse s'articulent autour de trois approches complémentaires pour l'évolution des modèle multidimensionnels, à la fois en termes de schémas et de données, ainsi que les techniques inhérentes.

2. https://www.statista.com/study/52194/digital-economy-compass/

1 - La première contribution se concentre sur l'étude de l'évolution des modèles multidimensionnels au niveau des schémas. L'objectif est de résoudre le problème d'inflexibilité rencontré dans les modèles multidimensionnels classiques. Pour ce faire, nous avons mis au point un méta-modèle de schéma dimensionnel agile afin de gérer son évolution selon plusieurs versions. Les données sont stockées dans une base de données NoSQL orientée graphe pour offrir davantage de flexibilité. Des requêtes simples et croisées permettent aux processus OLAP de naviguer aisément à travers les différentes versions de données existantes. Dans ce cadre, nous avons proposé le modèle intitulé Graph-based Agile Multidimensional Model (GAMM). C'est un modèle multidimensionnel agile à base de graphes, qui s'articule autour d'une approche flexible d'évolution de schémas dans les entrepôts de données.

Le modèle proposé permet aux concepteurs d'intégrer de nouvelles sources de données et de répondre à des nouveaux besoins d'utilisateurs afin d'enrichir les analyses de l'entrepôt de données. Il s'agit d'une approche basée sur un modèle à schémas évolutifs en multi-versions et un entrepôt de données stocké dans une base de données orientée graphes de type NoSQL. Chaque version de schéma (*Schema Version*) (SV) est associée à une période temporelle (T) caractérisée par un un temps de début Starting_Time (ST) et un temps de fin Ending_Time (ET), et correspond à une instance de données extraite de l'entrepôt de données temporelles basé sur les graphes en utilisant le paramètre temporel Transaction_Time TT introduit au niveau des noeuds de type *Fait*.

Sur le plan conceptuel, cette approche constitue une extension de la modélisation multidimensionnelle classique qui repose sur les concepts de « **fait, mesure, dimension, hiérarchie et niveau** » . Pour prendre en compte l'évolution chronologique des schémas et des données et pour permettre l'historisation des données, nous avons introduit un étiquetage temporel, et défini un cadre formel détaillé.

Un méta-modèle a été proposé pour l'identification et la gestion des différentes versions du schéma. Des opérateurs d'évolution de schéma et des règles de passage ont été définis. Un mécanisme explicite et implicite a été établi pour les requêtes simples et les requêtes croisées. Une validation fonctionnelle de notre approche ainsi que des analyses de performances ont été réalisées à l'aide d'un prototype que nous avons dévelopé, et enfin à la réalisation de deux études de cas basées sur le Star Schema Benchmark (SSB) [3] [4].

Ce travail a été acté par deux publications respectivement nationale (EDA 2022) (Benhissen et al., 2022) et internationale (DOLAP 2023) (Benhissen et al., 2023a).

2 - Notre deuxième contribution se focalise sur l'analyse de l'évolution temporelle des données au sein d'un entrepôt de données multi-versions, dans le but de représenter de manière précise le contexte d'analyse. L'objectif de cette contribution est de résoudre le problème de la temporalité des informations et des relations hiérarchiques

3. https ://jorgebarbablog.wordpress.com/2016/03/21/how-to-load-the-ssb-schema-into-an-oracle-database/

4. https ://github.com/Kyligence/ssb-kylin

dans les entrepôts de données, afin d'assurer une gestion adéquate des changements d'instances et d'assurer leur pertinence en fonction du temps.

Dans cette partie de notre travail, en poursuivant le développement du modèle GAMM, nous avons proposé une approche temporelle pour les entrepôts de données multi-versions orientés graphes, permettant de conserver l'historique évolutif des données, y compris les changements au niveau des dimensions. Nous avons appliqué un étiquetage temporel de type Valid_Time (VT) aux niveau des noeuds *fait* ainsi qu'un étiquetage temporel de type From_Date (FD) et To_Date (TD) au niveau des relations dans les instances des dimensions, permettant ainsi l'identification de la période de validité de ces relations. Cela garantit des analyses cohérentes et des résultats concordants selon les changements survenus.

Le formalisme du modèle GAMM ainsi que les règles de passage ont été adaptés pour prendre en charge l'évolution des instances de données. Nous avons également proposé des requêtes temporelles telles que l'union, la jointure, la différence et l'agrégation, afin de prendre en compte la temporalité des données dans les analyses. L'approche a été validée à travers une étude de cas basée sur le benchmark SSB, illustrant des changements au niveau des instances de données.

Ce travail a été acté par une publication internationale dans la conférence (DATA 2023) (Benhissen et al., 2023b).

3 - Notre troisième contribution porte sur l'automatisation de l'évolution des schémas dans les entrepôts de données multi-versions. La combinaison du processus de stockage des données et de l'exploration de données offre de grandes opportunités pour améliorer le traitement des données agrégées. En effet, certaines techniques d'exploration de données peuvent être utilisées comme des opérateurs d'agrégation et pour découvrir des modèles cachés dans les données (Bentayeb and Favre, 2009).

De plus, ces techniques d'exploration de données jouent un rôle prépondérant dans la détection automatique de schémas multidimensionnels à partir de diverses sources de données. Leur pouvoir descriptif et prédictif est exploité pour extraire des connaissances à partir des données, ce qui facilite l'identification et l'évolution de nouvelles dimensions, hiérarchies et mesures. Cela permet à notre approche d'évolution de schéma de s'adapter dynamiquement aux nouvelles sources de données et aux tendances changeantes dans les données agrégées.

Dans notre approche, nous avons intégré certaines techniques d'exploration de données pour automatiser le processus d'évolution des schémas. Cela permet une adaptation dynamique des schémas en fonction des changements dans les données et des nouvelles sources de données. L'objectif est d'améliorer l'efficacité et la précision de la gestion de l'évolution des schémas dans les entrepôts de données multi-versions.

Dans ce contexte, nous avons développé une approche automatisée pour l'évolution des schémas, appliquée au modèle GAMM. Cette approche repose sur l'utilisation d'al-

gorithmes de partitionnement de graphes, en particulier ceux basés sur la modularité tels que Louvain et Newman. Elle vise à identifier de nouveaux niveaux de hiérarchie correspondant aux clusters détectés dans le graphe. Ainsi, une nouvelle version du schéma de l'entrepôt de données est générée en suivant les fonctions d'évolution préétablies, tandis que les anciennes versions du schéma sont préservées.

Ce processus peut être automatisé et appliqué après chaque intégration de nouvelles sources de données, ce qui facilite la détection continue de connaissances cachées et améliore considérablement l'adaptabilité aux changements. Cette automatisation permet non seulement de gérer efficacement l'évolution des schémas en réponse aux nouvelles données, mais elle offre également la possibilité de détecter de manière proactive des connaissances cachées et des modèles émergents. De plus, cette approche permet la création simultanée de niveaux de hiérarchie sur plusieurs dimensions, ce qui enrichit la structure des données orientées décision et améliore leur représentation.

Grâce à cette automatisation, l'analyse multidimensionnelle devient plus pertinente et plus précise. Les nouvelles dimensions et hiérarchies détectées en fonction des données fraîchement intégrées permettent d'explorer les relations entre les données sous différents angles. Les opérateurs d'agrégation basés sur l'exploration de données contribuent à extraire des informations plus profondes et pertinentes, offrant ainsi des informations plus précieuses pour la prise de décision.

## 1.5 Organisation du mémoire

Le présent mémoire est structuré en six grandes parties, détaillées comme suit :

1. La première partie de notre travail se consacre à une exploration approfondie de l'état actuel de la recherche dans le domaine de l'évolution des modèles multidimensionnels. Cette partie est divisée en deux parties interconnectées. La première présente les bases des entrepôts de données, des bases de données temporelles et des entrepôts multidimensionnels. Elle explore aussi les bases de données NoSQL orientées graphes et les algorithmes de partitionnement de graphes. La deuxième partie examine les travaux pertinents dans la littérature, notamment les approches pour l'évolution des modèles multidimensionnels, en mettant en évidence leurs avantages et limites.
2. La deuxième partie est consacrée à notre modèle multidimensionnel agile à base de graphes intitulé GAMM. Nous présentons l'approche globale du modèle, sa formalisation algébrique pour l'évolution du schéma et des données, le méta-modèle utilisé pour la gestion des différentes versions ainsi que le modèle physique en graphe pour le stockage de l'entrepôt de donnée temporelle.
3. Dans la troisième partie, nous détaillons les mécanismes et les procédures d'évolution dans le modèle GAMM, tant au niveau du schéma que des données. Nous explorons les fonctions d'évolution du schéma, la gestion du méta-modèle lors des évolutions et le processus d'évolution des données, notamment les changements dans les dimensions.

4. Nous abordons dans la quatrième partie, les méthodes d'interrogation des versions et des données dans le modèle GAMM. Nous examinons les requêtes explicites et implicites pour interroger une version ou plusieurs versions de schémas (*cross-queries*), les requêtes temporelles pour obtenir des résultats cohérents malgré les changements dans les dimensions, ainsi que l'interrogation du métamodèle pour identifier les différents schémas.
5. Dans la cinquième partie, nous explorons l'approche d'automatisation de l'évolution de schéma dans le modèle GAMM en utilisant des techniques de partitionnement de graphes. Nous expliquons comment cette approche permet d'identifier de nouveaux niveaux de hiérarchie correspondant à des clusters détectés et de créer automatiquement de nouvelles versions du schéma.
6. Dans cette dernière partie, nous présentons plusieurs études de cas en guise de validation du modèle GAMM. Nous couvrons l'évolution du schéma, l'évolution des données, l'automatisation du processus d'évolution, ainsi que les tests de performances pour évaluer l'efficacité de notre approche.

Chaque partie contribue à la compréhension, au développement et à la validation du modèle GAMM dans le contexte de l'évolution des entrepôts de données multidimensionnels.

Enfin, nous concluons ce mémoire et présentons un bilan général de l'ensemble de nos contributions en faisant apparaître les perspectives de recherche que nous envisageons d'étudier à l'avenir.

CHAPITRE 2

# CONTEXTE ET ÉTAT DE L'ART

## Sommaire

## 2.1 Introduction

La première partie de ce chapitre est dédiée à la présentation du contexte et des concepts fondamentaux qui sous-tendent notre recherche. Elle dévoile une vue d'ensemble des entrepôts de données de par leurs concepts clés, leur architecture ainsi que le processus qui les anime. Par ailleurs, cette partie aborde également les fondements des bases de données temporelles, mettant en lumière l'importance de l'aspect temporel dans la gestion et l'analyse des données. De plus, elle explore en détail les entrepôts de données multidimensionnels, en insistant sur leurs caractéristiques et leurs opérateurs temporels qui sont au cœur de notre étude. Une incursion dans l'univers des bases de données NoSQL orientées graphes apporte une perspective éclairante sur la manière dont ces systèmes gèrent les données inter-connectées. Enfin, une exploration des algorithmes de partitionnement de graphes complète cette partie mettant en évidence leur rôle crucial dans la répartition efficace des données dans un environnement de graphes.

La deuxième partie de ce chapitre est consacrée à une revue détaillée des travaux pertinents dans la littérature. Elle examine les diverses approches liées à l'évolution des modèles multidimensionnels, explore les deux grandes familles de recherches dans ce domaine. Cette partie met en lumière les méthodes existantes, leurs avantages et leurs limites, tout en soulignant les lacunes qui ont motivé nos propres recherches. Cette partie porte aussi une réflexion sur le positionnement de nos travaux au sein du paysage actuel de la recherche dans le domaine de l'évolution des modèles multidimensionnels.

En somme, ce chapitre crée un socle solide pour l'ensemble de notre étude. En déployant une vue panoramique du domaine et en explorant les travaux existants, il prépare le terrain pour une analyse approfondie des défis, des solutions et des contributions qui seront exposés dans les chapitres à venir.

## 2.2 Contexte de l'étude

### 2.2.1 Vue d'ensemble sur les entrepôts de données

#### 2.2.1.1 Concepts

Les entrepôt de données ont vu le jour à la fin des années 80 et au début de celles de 90 avec la proposition d'une définition fondatrice de Bill Inmon : « un entrepôt de données est une collection de données orientée sujet, intégrée, variable dans le temps et non volatile en soutien au processus de prise de décision » (Inmon, 1992). Cette notion a été consolidée quelques temps plus tard avec la publication du livre de référence de Ralph Kimball, qui a introduit le schéma en étoile comme modèle de base pour la modélisation multidimensionnelle des données dans les entrepôts de données relationnels (Kimball, 1996).

Défini comme un référentiel de stockage de données granulaires, structurées ou semi-structurées, provenant de diverses sources à travers un processus d'intégration, un entrepôt de données joue un rôle crucial dans la prise de décisions stratégiques

au sein d'une organisation. Les données stockées dans l'entrepôt de données sont non volatiles, c'est-à-dire qu'elles ne sont pas modifiées en temps réel, mais plutôt mises à jour périodiquement pour refléter les changements et les évolutions dans l'entreprise. Ces données sont orientées pour répondre à des besoins spécifiques et prédéfinis, visant à fournir des informations pertinentes pour la haute direction (Kimball and Ross, 2013).

Un entrepôt de données contient des données d'entreprise *(business data)* à différents niveaux de granularité, allant des agrégations globales aux fins détails. Ces données historiques, collectées et intégrées depuis diverses sources, incluent souvent des informations transactionnelles, opérationnelles et même des méta-données. L'objectif ultime d'un entrepôt de données est de faciliter des analyses approfondies et des prises de décision, en permettant aux utilisateurs d'explorer, de consolider et de croiser ces données de manière intuitive (Vaisman and Zimányi, 2022).

L'entrepôt de données joue un rôle vital dans le processus de l'informatique décisionnelle en fournissant un environnement structuré pour l'analyse, la modélisation et la génération de rapports. Les données de l'entrepôt de données peuvent être utilisées pour des analyses multidimensionnelles, des requêtes *ad hoc* et des rapports pré-configurés. Grâce à sa capacité de stocker des données historiques, l'entrepôt de données permet également aux organisations de suivre les tendances et les évolutions au fil du temps, ce qui est essentiel pour identifier les opportunités et les défis futurs.

#### 2.2.1.2 Architecture des entrepôts de données

Il existe principalement deux approches dans le domaine de l'entreposage des données : "l'approche Inmon" et "l'approche Kimball". Chacune d'elles possède sa propre philosophie et ses principes. Les principales distinctions entre ces deux modèles sont les suivantes :

**Approche Top-Down (Inmon)** (Inmon, 2005) : Le modèle Inmon adopte une approche *top-down*, mettant l'accent sur la construction d'un « Entrepôt de Données Corporatif ». Dans cette perspective, les données sont intégrées et centralisées au sein d'un unique entrepôt de données.

**Approche Bottom-Up (Kimball)** (Kimball and Ross, 2013) : Le modèle Kimball adopte une approche *bottom-up*, dans laquelle les data-marts sont conçus de manière indépendante pour répondre aux besoins spécifiques des utilisateurs. Par la suite, ces data-marts sont perçus comme un ensemble équivalent à un entrepôt de données.

Certaines organisations optent pour une approche hybride en combinant des éléments des architectures d'Inmon et de Kimball. Elles peuvent, par exemple, utiliser des schémas dimensionnels pour certains domaines spécifiques, tout en adoptant une approche plus normalisée pour d'autres parties de l'entrepôt. La littérature présente plusieurs nouvelles propositions d'approches d'entreposage qui enrichissent et élargissent les options existantes.

Ci-dessous, nous présentons l'architecture globale d'un entrepôt de données centra-

lisé telle qu'elle est exposée dans (Vaisman and Zimányi, 2022). Celle-ci est constituée de trois principaux tiers, ainsi que d'un niveau d'analyses. Au premier tiers, également appelé « back-end », se situent les outils d'extraction, de transformation et de chargement (ETL), utilisés pour approvisionner l'entrepôt de données à partir de bases de données opérationnelles et de diverses autres sources de données.

Au deuxième tiers, appelé « niveau entrepôt de données », il y a un entrepôt de données d'entreprise et/ou plusieurs data-marts, ainsi qu'un référentiel de méta-données contenant des informations sur l'entrepôt de données et son contenu. Il est important de noter que l'entrepôt de données d'entreprise est centralisé et couvre l'ensemble de l'organisation, tandis qu'un data-mart est un mini-entrepôt de données spécialisé qui cible un domaine fonctionnel ou un département spécifique au sein de l'organisation.

Le troisième tiers, appelé « niveau OLAP », est destiné à la création des vues multidimensionnelles des données ; ce qui permet aux utilisateurs d'effectuer des analyses métiers de type OLAP, fouille de données, etc.

Le schéma global de cette architecture est représenté dans (Vaisman and Zimányi, 2022) comme suit :

**Sources de données** : Les données sont extraites de diverses sources, telles que les systèmes opérationnels (ou de production), les bases de données, les fichiers plats, etc. Ces sources peuvent être internes ou externes à l'organisation.

**Zone de Staging** : Les données extraites sont chargées dans une zone de staging (zone d'attente, ou ODS : *Operating Data Staging*) où elles sont nettoyées, transformées et préparées pour le chargement dans l'entrepôt de données.

**ETL** (Extract, Transform, and Load) : le processus qui consiste à extraire les données des sources, les transformer pour les rendre conformes aux besoins de l'entrepôt de données (nettoyage, agrégation, conversion de formats...) et enfin les charger dans l'entrepôt.

**Entrepôt de données** : Celui-ci est le cœur de l'architecture. C'est une base de données centrale qui stocke les données intégrées, historisées et prêtes à l'analyse. Il est généralement organisé selon un modèle multidimensionnel pour faciliter les requêtes analytiques.

**Data-mart** : Les data-marts sont des mini-entrepôts de données qui sont conçus pour répondre aux besoins d'analyse d'un groupe spécifique d'utilisateurs ou d'un département. Ils permettent de fournir des données plus ciblées pour des analyses spécifiques.

**Méta-données** : Les méta-données sont des informations descriptives sur les données. Elles permettent de comprendre la signification des données, leur provenance,

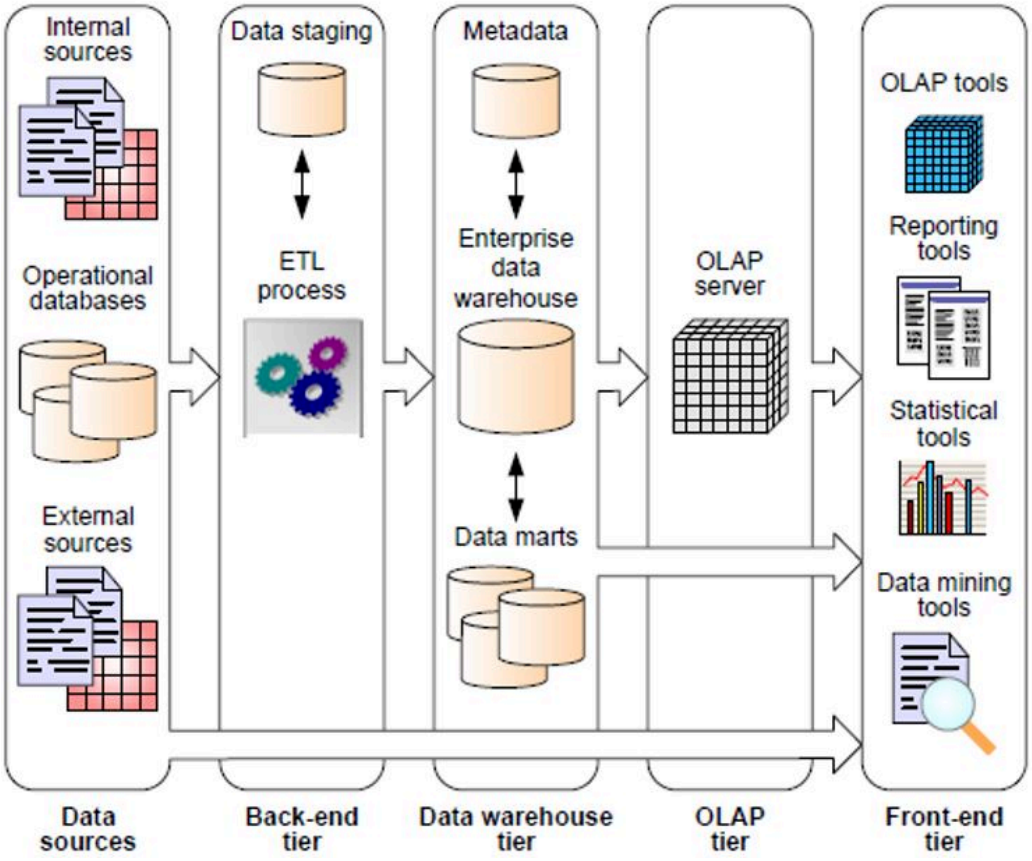



FIGURE 2.1 – Architecture typique des entrepôts de données (Vaisman and Zimányi, 2022)

leur qualité, etc. Les méta-données sont essentielles pour la gestion et la documentation de l'entrepôt de données.

**Niveau OLAP** : Pour faciliter les analyses avancées et les opérations de traitement analytique en ligne, un serveur OLAP est mis en œuvre (par exemple : MOLAP : Multidimensional OLAP). Ce serveur propose des fonctionnalités telles que le regroupement, l'agrégation, l'application de filtres, etc. Il est à noter que les opérations OLAP peuvent être exécutées directement au niveau de la base de stockage de l'entrepôt de données (ROLAP : Relational OLAP), ou elles peuvent s'appuyer sur les deux systèmes (HOLAP : Hybride OLAP).

**Niveau analyse** : Les outils d'analyse sont des logiciels qui permettent d'interroger, visualiser et analyser les données de l'entrepôt. Cela peut inclure des outils de business intelligence, des tableaux de bord, des outils de data mining, etc.

Cette architecture typique permet de créer un environnement solide pour l'analyse de données, la génération de rapports et la prise de décisions au sein d'une organisation.

#### 2.2.1.3 Modélisation multidimensionnelle

La modélisation multidimensionnelle est une approche de conception de bases de données destinée à faciliter l'analyse et la prise de décision en mettant l'accent sur les aspects analytiques plutôt que transactionnels. Elle est principalement utilisée dans les entrepôts de données pour organiser et représenter les données de manière à faciliter l'exploration et l'agrégation des informations (Kimball, 1996; Kimball and Ross, 2013). Voici les principaux concepts de la modélisation multidimensionnelle :

**Faits** : Éléments centraux décrivant le sujet d'analyse, les faits sont décrits par les dimensions et regroupent les indicateurs sous forme de mesures numériques ou quantitatives des données que l'on souhaite analyser. Par exemple, dans un entrepôt de données de ventes, les ventes constituent un fait.

**Mesures** : Elles sont les attributs numériques associés aux faits. Elles représentent les indicateurs que l'on souhaite observer. Elles sont décrites par les dimensions des faits. Par exemple, dans l'entrepôt de données de ventes, les mesures pourraient être le montant des ventes, la quantité vendue, le chiffre d'affaires, etc.

**Agrégation** : Les mesures peuvent être agrégés selon différentes combinaisons de dimensions. Cela permet d'obtenir des résultats à des niveaux différents de granularité. Par exemple, le chiffre d'affaires peut être agrégé par mois, par trimestre, par produit, etc.

**Dimensions** : Elles sont les attributs qualitatifs ou catégoriques utilisés pour décrire les faits. Elles permettent de classer et de filtrer les données. Dans un entrepôt de données de ventes, les dimensions pourraient être Temps, Produit, Client, etc.

**Niveaux** : Ils représentent les différents niveaux de granularité dans une hiérarchie de dimension. Par exemple, une dimension temporelle pourrait avoir des niveaux tels que Année, Mois, Jour, etc.

**Attributs de decription** : Ils sont les différentes modalités que peuvent prendre les dimensions. Par exemple, pour la dimension Temps, les attributs pourraient être les différentes "dates".

**Hiérarchies** : Représentent les axes d'analyse. Elles organiseent les niveaux des dimensions ; du niveau le plus détaillé au niveau le plus agrégé. Par exemple, une hiérarchie Temps pourrait être organisée en Année, Mois, et Jour.

#### 2.2.1.4 Schéma multidimensionnel

Nous présentons ci-après, les différents schémas de modélisation multidimensionnelle (Kimball, 1996; Kimball and Ross, 2013), basés sur les principaux concepts présentés dans dans la Sous-section 2.2.1.3 :

**Schéma en étoile** (voir figure 2.2) : Dans ce schéma, le modèle est organisé autour d'une table de faits centrale, entourée de tables de dimensions. Chaque table de dimension est reliée à la table de faits par une clé étrangère et contient des attributs de description. La table de faits ne contient que des clefs étrangères et des mesures. Ce schéma est simple et efficace pour des modèles de données simples.

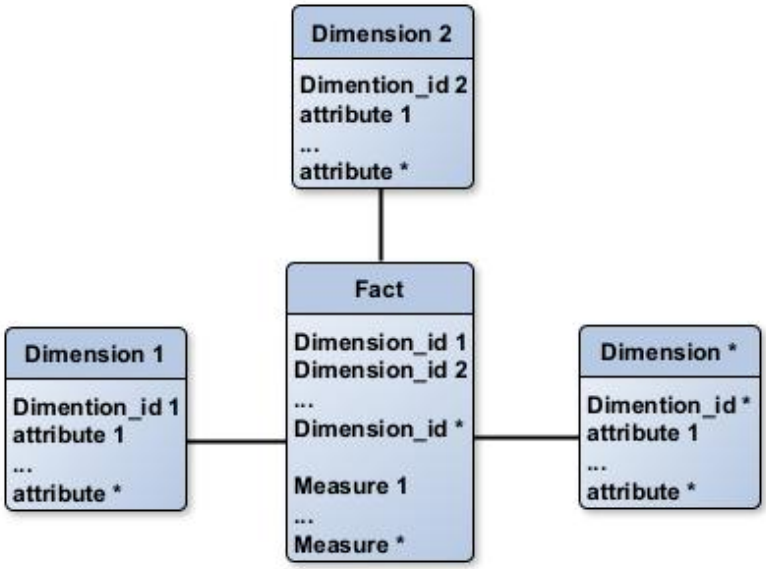



FIGURE 2.2 – Exemple d'un schéma multidimensionnel en étoile

**Schéma en flocon de neige** (voir figure 2.3) : Ce schéma est une variation du schéma en étoile où les tables de dimensions sont normalisées en sous-tables pour créer des hiérarchies. Cela réduit la redondance des données mais génère de nouveaux paliers d'observation (les niveaux) à l'aide de requêtes complexes.

**Schéma en constellation** (voir figure 2.4) : Ce type de schéma combine plusieurs schémas en étoile pour traiter des besoins d'analyse plus complexes. Les étoiles sont reliées entre elles, de façon explicite ou implicite, par des dimensions partagées (dimensions conformes) ou des relations entre les mesures appartenant à différentes tables de faits.

Chacun de ces schémas a ses avantages et ses inconvénients ; le choix du type de schéma dépend des besoins spécifiques de l'organisation et de la complexité des données à modéliser.

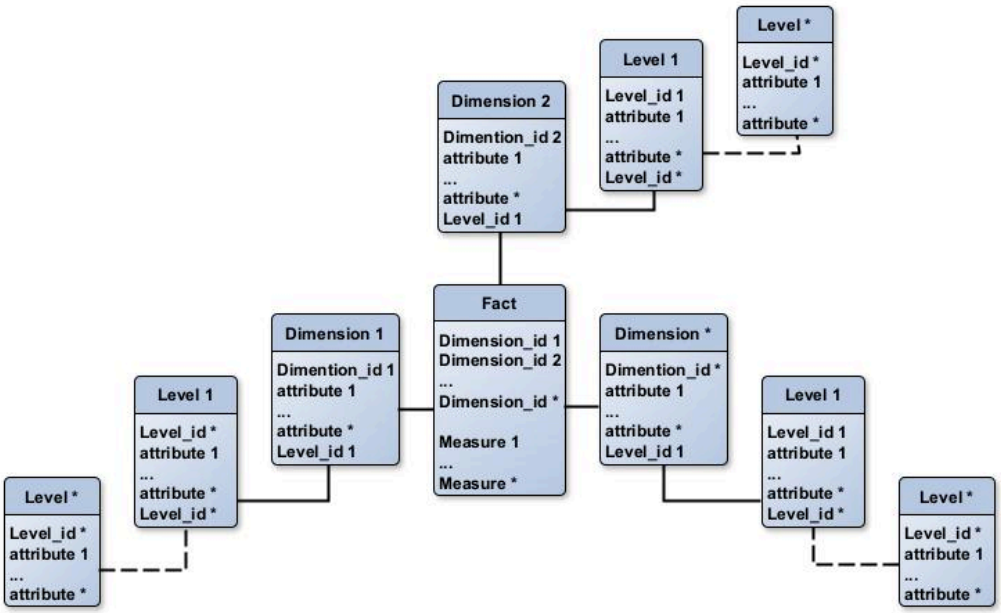



FIGURE 2.3 – Exemple d'un schéma multidimensionnel en flocon de neige

#### 2.2.1.5 Niveau logique d'un entrepôt de données

Les différentes approches mettant en œuvre un modèle logique sont les suivantes (Vaisman and Zimányi, 2022) :

**Systèmes relationnal OLAP (ROLAP)** : Ils représentent les données multidimensionnelles dans des bases de données relationnelles (Tables, Colonnes, Enregistrements, etc.), et utilisent des fonctions Structured Query Language (SQL) et des méthodes spéciales pour les opérations OLAP. Des agrégats pré-calculés sont utilisés pour améliorer les performances, mais cela a pour des avantage une utilisation importante de l'espace de stockage. Cependant, les requêtes SQL peuvent être complexes.

**Systèmes multidimensional OLAP (MOLAP)** : Ils représentent les données dans des structures de données de type tableaux à plusieurs arêtes, combinées à des techniques d'indexation et de hachage. Les opérations OLAP sont simples et rapides à réaliser, cependant, la capacité de stockage est généralement inférieure à celle des systèmes ROLAP.

**Systèmes hybrid OLAP (HOLAP)** : Ils combinent les approches ROLAP et MOLAP pour bénéficier de la capacité de stockage de ROLAP et de la rapidité des traitements de MOLAP. Par exemple, les données détaillées sont stockées dans une base de données relationnelle, tandis que les agrégats sont conservés dans un magasin MOLAP.

La plupart des outils OLAP actuels prennent en charge une combinaison de ces modèles, mais reposent souvent sur un entrepôt de données mis en œuvre sur un sys-

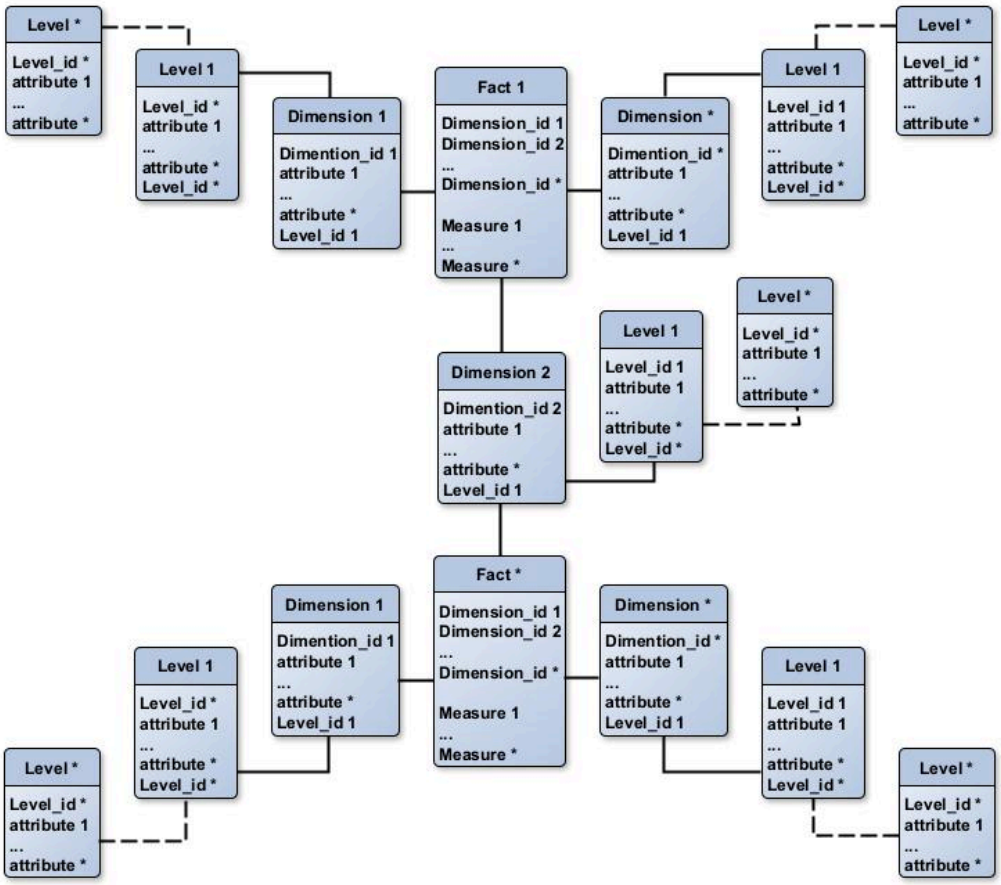



FIGURE 2.4 – Exemple d'un schéma multidimensionnel en constellation

tème de gestion de base de données relationnelle.

#### 2.2.1.6 Processus ETL

Le processus ETL (Extract, Transform, and Load) est utilisé pour alimenter un entrepôt de données en le chargeant par des données à partir de sources hétérogènes. Voici les étapes typiques du processus ETL (Kimball and Ross, 2013) :

1. **Extraction (*Extract*)** : Cette étape consiste à extraire les données des sources de données, qu'il s'agisse de bases de données opérationnelles, de fichiers plats, de services web, etc. Elles sont collectées en fonction des critères définis, tels que des plages de dates ou des filtres spécifiques.

2. **Transformation (*Transform*)** : Les données extraites sont au départ hétérogènes en général. Elles peuvent nécessiter des transformations pour les rendre homogènes et conformes au modèle de données de l'entrepôt. Cela comprend

des opérations telles que la normalisation, le filtrage, la conversion de formats, l'enrichissement de données, la gestion des valeurs manquantes, etc. L'objectif est d'obtenir des données propres et de qualité avant leur chargement dans l'entrepôt.

3. **Chargement (*Load*)** : Une fois que les données aient été extraites et transformées, elles sont chargées pour la première fois dans l'entrepôt de données, ou pour les rafraichir les fois d'après. Cela implique généralement l'utilisation d'un processus de chargement en masse (*bulk loading*) pour optimiser les performances. Les tables de faits et les tables de dimensions sont créées et alimentées avec des données appropriées.

   Il est à noter qu'après le chargement initial, il peut être nécessaire de mettre à jour les données dans l'entrepôt de données en fonction des changements survenus dans les sources de données, ce qui inclut des mises à jour, des suppressions ou des ajouts de nouvelles données (*Fact and Dimension Modification and Refresh*). Ces opérations de modification et de rafraîchissement des faits et des dimensions garantissent que l'entrepôt de données reste à jour et reflète les dernières informations disponibles. Les données de l'entrepôt doivent être régulièrement actualisées pour refléter les changements opérationnels et maintenir la pertinence des analyses. Cela peut être effectué selon un calendrier planifié ou en temps réel, en fonction des besoins de l'entreprise.

### 2.2.2 Bases de données temporelles

Les bases de données temporelles sont conçues spécifiquement pour gérer des données qui évoluent dans le temps et pour permettre des requêtes temporelles avancées. Elles sont utilisées lorsque il y a besoin de stocker et de manipuler des informations historiques et de suivre les changements dans les données au fil du temps.

Le principe fondamental des bases de données temporelles est d'associer à chaque enregistrement une valeur temporelle à travers un mécanisme d'horodatage. Ce dernier permet de suivre les changements et la validité des données au fil du temps, et de les interroger en fonction de ces informations temporelles (Böhlen et al., 2017).

L'horodatage des données est un mécanisme qui consiste à associer une date (et éventuellement une heure) à une donnée. Son objectif principal est de marquer un instant de référence où une opération a été effectuée ou d'enregistrer la période de validité de la donnée. Ainsi, un horodatage peut être soit ponctuel, indiquant un moment précis, soit un intervalle de temps, couvrant une durée spécifique. Le choix entre les deux dépend du type de données et des besoins des utilisateurs (Böhlen et al., 2017).

Dans un horodatage ponctuel, chaque enregistrement, dans la base de données, est associé à un « *timestamp* » qui représente un moment précis, tel que la date de validité Valid_Time (VT) (Jensen and Snodgrass, 2018b), la date de transaction ou de chargement Transaction_Time (TT), la date de décision Decision_Time (DT) (Jensen and

Snodgrass, 2018a), etc. Ces horodatages sont utiles pour enregistrer des événements spécifiques dans la vie réelle et sont couramment utilisés dans les bases de données pour les opérations de suivi, de journalisation et de prise de décision .

Valid_Time représente le moment où une donnée est vraie par rapport à la réalité, tandis que Transaction_Time représente le moment où une données est stockée dans un base de données(Jensen and Snodgrass, 2018b,a). Le modèle de données bitemporales prend en charge à la fois le temps de validité et le temps de transaction (Böhlen et al., 2017).

Exemple :

TABLE 2.1 – Liste des œuvres d'art.

| Artwork_id | Name | Artist | Type | Valid_Date | Transaction_Date |
|---|---|---|---|---|---|
| Artwork001 | Guernica | Pablo Picasso | Painting | 04/06/1937 | 01/07/2020 |

Le tableau 2.1 présente un extrait d'une liste d'œuvres d'art. L'exemple illustre que la célèbre toile de PABLO PICASSO, dénommée « Guernica », possède deux références temporelles. La première représente la date de sa création, indiquée par le *timestamp* Valid_Time, tandis que la seconde correspond à la date de son chargement dans la base de données, représentée par le *timestamp* Transaction_Time.

Dans un horodatage à intervalles de temps, chaque enregistrement de données est défini par une période temporelle avec un temps de début From_Date (FD) et un temps de fin To_Date (TD). Ces horodatages sont utilisés pour enregistrer des événements qui s'étendent sur une période donnée plutôt que sur un instant précis, tels que les plages de validité, les périodes d'activité, les historiques d'états, etc. Cela permet de représenter les enregistrements valables pendant une période spécifique (Lorentzos, 2018).

Exemple :

TABLE 2.2 – Table d'affectation des employés.

| Sgt_Key | Employee_id | Name | Department | From_Date | To_Date |
|---|---|---|---|---|---|
| SK_001 | Employ001 | Marc Bloom | Mecanic | 02/01/2022 | 01/07/2023 |
| SK_002 | Employ001 | Marc Bloom | Marketing | 01/07/2023 | 31/12/9999 |

Le tableau 2.2 présente un extrait des affectations d'employés à des départements dans une entreprise. L'employé MARC BLOOM était initialement affecté au département MECANIC à partir du 02/01/2022, puis il a été réaffecté au département MARKETING le 01/07/2023. Ainsi, les périodes pendant lesquelles l'employé est affilié à chacun des départements sont représentées en utilisant un horodatage de type intervalles de temps.

Les bases de données temporelles conservent un historique temporel complet des données et peuvent également gérer des intervalles de validité pour les enregistrements (Date, 2015). Cela permet de reconstituer l'état des données à n'importe quel moment donné dans le passé.

Les bases de données temporelles offrent des fonctionnalités permettant d'effectuer des requêtes basées sur le temps. Ces fonctionnalités permettent d'obtenir des résultats correspondant à une période spécifique, d'analyser les tendances temporelles, ou de récupérer des versions antérieures des enregistrements. De plus, les horodatages peuvent être utilisés pour effectuer des comparaisons en se basant sur les treize relations d'intervalle de base d'Allen, telles que « *before* », « *meets* », « *during* », etc. (Allen, 1983).

Les bases de données temporelles sont utilisées dans de nombreux domaines, tels que la finance, la gestion des données scientifiques, la gestion de l'historique des versions de logiciels, la gestion des données environnementales, etc. Elles offrent des fonctionnalités puissantes pour analyser et comprendre l'évolution des données dans le temps (Böhlen et al., 2017).

Le support temporel a été intégré dans le standard SQL en 2011 (Kulkarni and Michels, 2012), et il a été implémenté dans certains systèmes de gestion de base de données (SGBD) tels que SQL Server 2016, Oracle 12c, IBM DB2 et Teradata (Poscic et al., 2018). Outre ces SGBD traditionnels, il existe également plusieurs systèmes natifs de bases de données temporelles disponibles, tels que TimescaleDB, InfluxDB, Prometheus, etc. Chacun de ces systèmes offre ses propres fonctionnalités et caractéristiques spécifiques, mais ils ont tous pour objectif commun de stocker et de gérer efficacement des données temporelles.

### 2.2.3 Entrepôts de données temporelles

Un entrepôt de données temporelles constitue un type spécifique d'entrepôt de données spécialement conçu pour le stockage et l'analyse de données qui varient selon le temps. Son objectif premier est de capter, intégrer et analyser des informations historiques liées aux données, facilitant ainsi la prise de décision en se basant sur des tendances et des modèles temporels. Grâce à cet entrepôt de données, les utilisateurs peuvent effectuer des requêtes et des analyses orientées temps, permettant ainsi de comprendre comment les données ont évolué et se sont comportées au fil du temps (Vaisman and Zimányi (2022); Ahmed et al. (2015)).

L'entrepôt de données temporelles offre une capacité d'exploration des variations et des cycles temporels, ce qui est crucial pour de nombreux domaines d'application. Il permet aux utilisateurs de visualiser les changements au fil du temps, de détecter les fluctuations saisonnières, d'analyser les performances passées et de prévoir les tendances futures. En outre, il permet l'application de requêtes temporelles sophistiquées, telles que les sélections, les projections et les regroupements basés sur des intervalles de temps spécifiques (Vaisman and Zimányi (2022); Ahmed et al. (2015)).

Dans le domaine des entrepôts de données temporelles, l'horodatage peut être de deux types principaux : ponctuel ou intervalle. Ces deux principes d'horodatage sont utilisés pour enregistrer et représenter les informations temporelles associées aux données stockées dans l'entrepôt (voir sous-section 2.2.2). L'utilisation de l'horodatage de type ponctuel ou intervalle dépend du contexte et des besoins spécifiques de l'application. Les entrepôts de données temporelles utilisent ces principes d'horodatage pour permettre des analyses historiques, des requêtes temporelles et la gestion de données qui évoluent dans le temps (Golfarelli and Rizzi (2018); Vaisman and Zimányi (2022)).

L'utilisation d'une base de données temporelle est au cœur de la gestion des données temporelles dans un entrepôt de données. Une base de données temporelle est conçue pour stocker et interroger efficacement des données avec des composantes temporelles. Elle ajoute des fonctionnalités spécifiques pour la gestion du temps, telles que l'horodatage des enregistrements et le suivi des changements temporels (Poscic et al. (2018); Phungtua-Eng and Chittayasothorn (2019)).

### 2.2.4 Changements dans les dimensions

Outre l'insertion de nouvelles entités (faits et/ou dimensions) dans l'entrepôt de données, les données existantes peuvent être modifiées par des opérations de mise à jour. Ces modifications peuvent inclure des changements d'attributs existants ou des modifications des relations entre les hiérarchies, ce qui implique également des mises à jour des clés. Si elles ne sont pas gérées correctement, ces opérations de mise à jour peuvent altérer les analyses produites par l'entrepôt de données.

Ces changements dans les dimensions nécessitent un traitement temporel des requêtes afin d'éviter toute incohérence dans les résultats. Pour faire face à ces changements, Kimball et Ross ont étudié le problème des changements de dimensions et ont proposé des techniques de traitement en fonction de la vitesse des changements Kimball and Ross (2013). Pour les dimensions à changement lent (Slowly Changing Dimension) (SCD), ils ont suggéré les techniques suivantes :

(i) Conserver la valeur originale de l'attribut, de sorte que les faits soient toujours regroupés en fonction de cette valeur originale (Type 0).

(ii) Remplacer l'ancienne valeur de l'attribut par une nouvelle valeur. Les faits seront alors associés à la valeur actuelle de l'attribut (Type 1).

(iii) Ajouter une nouvelle ligne de dimension avec une nouvelle valeur de l'attribut/-clé étrangère, en tenant compte de l'aspect temporel pour que les faits soient associés à la valeur de l'attribut en fonction de la durée de sa véracité, délimitée par une date de début et une date de fin (Type 2).

(iv) Ajouter une nouvelle colonne pour conserver les valeurs actuelles et précédentes de l'attribut (Type 3).

Pour les dimensions qui évoluent rapidement, Kimball et Ross (Kimball and Ross (2013)) ont proposé une technique impliquant l'ajout d'une mini-dimension. Ainsi, les attributs fréquemment analysés ou ceux qui évoluent rapidement peuvent être séparés dans une dimension distincte.

En outre, d'autres techniques hybrides ont été proposées, combinant tout ou partie des différentes approches mentionnées ci-dessus pour répondre aux exigences de la préservation des attributs historiques et de l'établissement de rapports. Ces approches permettent de gérer efficacement les opérations de mise à jour dans l'entrepôt de données, assurant ainsi la cohérence et l'exactitude des analyses réalisées sur les données.

### 2.2.5 Opérateurs temporels dans les entrepôts de données

Les opérations temporelles dans les entrepôts de données permettent de manipuler des données temporelles en prenant en compte les aspects temporels tels que les intervalles, les points dans le temps, les durées, etc. Voici les principales opérations temporelles définies dans Vaisman and Zimányi (2022) et Ahmed et al. (2015) :

1. **Jointure temporelle** : Elle combine des enregistrements de différentes tables en utilisant des conditions temporelles. Elle est utilisée pour récupérer les enregistrements qui se chevauchent dans le temps ou qui ont des intervalles de validité communs.
2. **Union temporelle** : L'union temporelle fusionne deux ensembles de données temporelles en conservant les intervalles de validité communs et en combinant les autres intervalles. Cela permet de regrouper des données temporelles similaires provenant de différentes sources.
3. **Différence temporelle** : La différence temporelle permet de trouver les intervalles de temps qui existent dans un ensemble de données temporelles mais qui ne se chevauchent pas avec un autre ensemble de données temporelles. Cela permet d'identifier les écarts et les variations entre les ensembles de données.
4. **Agrégation temporelle** : L'agrégation temporelle permet de regrouper les données en fonction d'intervalles de temps spécifiques, tels que par jour, par semaine, par mois, etc. Cela permet de résumer les données temporelles sur des périodes plus larges et de calculer des mesures agrégées telles que la somme, la moyenne, le maximum, le minimum, etc.

Il est important de noter que ces opérations temporelles peuvent être mises en œuvre de différentes manières en fonction des systèmes de gestion de bases de données utilisés et des langages de requête supportés.

### 2.2.6 Bases de données NoSQL orientées graphes

Les bases de données orientées graphes sont spécialement conçues pour la gestion de données inter-connectées, où les relations entre les entités jouent un rôle aussi crucial que les entités elles-mêmes. Elles trouvent une vaste application dans la modélisation de domaines complexes tels que les réseaux sociaux, les systèmes de recommandation de contenus, la gestion des connaissances, et bien d'autres encore... Cette conception

repose sur l'utilisation de structures de graphes, où les nœuds symbolisent les diverses entités et les arêtes représentent les liens entre ces entités. L'une des caractéristiques majeures de ces bases de données est leur capacité à exécuter efficacement des requêtes de parcours de graphes, ce qui en fait est une solution privilégiée pour les scénarios où les relations complexes entre les données sont prédominantes (Robinson et al. (2015)).

Ces bases de données appartiennent à la catégorie des systèmes de gestion de bases de données NoSQL (Not only SQL), au même titre que les bases de données orientées documents, orientées colonnes, et orientées clés-valeurs. Elles se distinguent des bases de données relationnelles classiques en abandonnant l'utilisation du langage SQL comme principal moyen d'interaction. Leur architecture est optimisée pour la gestion efficiente de grandes quantités de données non, peu ou pas structurées, tout en offrant une grande évolutivité (Kaur and Rani (2013)).

#### 2.2.6.1 Modèles de données des bases de données orientées graphes

Les bases de données orientées graphes proposent principalement deux modèles de données pour représenter et stocker les informations sous forme de graphes. Ils s'agit du Modèle de Graphes de Propriétés (*Property Graph Model*) ainsi que du Modèle de Graphes à Triplets (*Triple Store Model*) (Pokorný (2015); Heath and Bizer (2022)) :

**Modèle de Graphes de Propriétés (*Property Graph Model*)** (Robinson et al. (2015)) : L'information dans un graphe de propriété est modélisée grâce à trois blocs de base (nœuds, relations- arêtes et propriétés) (voir figure 2.5) :

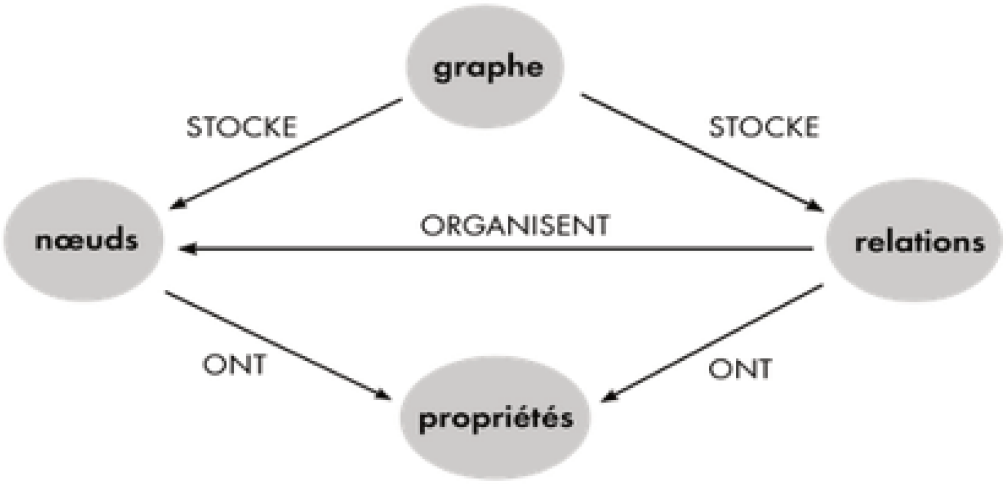



FIGURE 2.5 – Terminologie de base pour les graphes de propriétés.

Dans ce modèle, chaque nœud et arête (relation) du graphe peut posséder des propriétés. Chaque propriété peut être associée à une valeur. De plus, elles peuvent avoir

des étiquettes (*labels*) qui fournissent des informations supplémentaires aux nœuds et aux arêtes. Ce modèle est largement utilisé dans des bases de données orientées graphes populaires telles que Neo4j. Ils existent des extensions au graphe de propriété tel que le Modèle de Graphes de Valeurs où les nœuds et les arêtes du graphe peuvent stocker des valeurs complexes, telles que des tableaux, des enregistrements ou d'autres structures de données. Cela permet de modéliser des données plus riches et diverses au sein du graphe.

**Modèle de Graphes à Triplets (*Triple Store Model*)** (Heath and Bizer (2022)) : Le modèle de graphes à triplets est basé sur le modèle RDF (*Resource Description Framework*). Il stocke les données sous forme de triplets (sujet, prédicat, objet), où chaque triplet représente une assertion. Ce modèle est largement utilisé pour les données sémantiques et les graphes de connaissances.

Ces modèles de données sont conçus pour répondre à différents besoins de représentation et de stockage des données dans les bases de données orientées graphes.

#### 2.2.6.2 Principales bases de données orientées graphes

Il existe plusieurs bases de données orientées graphes, chacune offrant des caractéristiques spécifiques ainsi que des langages d'interrogation adaptés à la manipulation de données inter-connectées.

Le tableau 2.3 représente une comparaison entre différentes bases de données orientées graphes en fonction de critères tels que la popularité, le langage de requêtes, la gratuité, ainsi que d'autres facteurs importants (Pokorný (2015); Fernandes and Bernardino (2018)) :

Les bases de données présentées dans le tableau 2.5 illustrent la diversité des approches et des fonctionnalités disponibles pour la gestion et l'analyse de données. Chacune offre des avantages que ce soit en termes de langage de requêtes, de schéma de données ou de compatibilité avec d'autres outils et services.

Il est important de noter que chaque base de données a ses avantages et inconvénients ; et le choix dépendra des besoins spécifiques de l'application à utiliser. La popularité peut être un indicateur de la maturité de la base de données et de la disponibilité d'une communauté active pour le support. La flexibilité de modélisation se réfère à la capacité de la base de données à gérer différents types de graphes et de structures de données. Le langage de requêtes est essentiel pour interagir avec la base de données. La gratuité peut être un facteur décisif, mais il est important de noter que certaines fonctionnalités avancées peuvent nécessiter des licences payantes. Enfin, chaque base de données peut avoir des caractéristiques uniques qui peuvent influencer son choix en fonction des besoins spécifiques.

TABLE 2.3 – Comparative des différentes bases de données orientées graphes

| **Base de Données** | **Popularité** | **Langage de Requête** | **Gratuité** | **Autres Caractéristiques** |
|---|---|---|---|---|
| Neo4j | Très populaire | Cypher | Version communautaire gratuite, éditions payantes avec fonctionnalités avancées | Modélisation flexible de graphes de propriétés |
| Amazon Neptune | Populaire | Gremlin et SPARQL | Payant, tarification AWS | Service géré, intégration facile avec AWS |
| ArangoDB | Bien établi | AQL (ArangoDB Query Language) | Version communautaire gratuite, éditions payantes avec fonctionnalités avancées | Modèles de documents, clés-valeurs et graphes |
| TigerGraph | En croissance | GSQL (Graph Structured Query Language) | Version gratuite de développement, éditions payantes avec fonctionnalités avancées | Schéma dynamique pour graphes |
| JanusGraph | Populaire | Gremlin | Gratuit et open source | Prise en charge du stockage distribué |
| Virtuoso | Bien établi | Prise en charge de SPARQL | Version communautaire gratuite, éditions payantes avec fonctionnalités avancées | Prise en charge des données RDF |

#### 2.2.6.3 Flexibilité des bases de données orientées graphes

Les bases de données orientées graphes sont souvent considérées comme flexibles pour plusieurs facteurs (Robinson et al. (2015); Akoka et al. (2021)) :

1. **Absence de schéma prédéfini** : Contrairement aux bases de données relationnelles traditionnelles qui nécessitent un schéma rigide prédéfini avant l'insertion des données, les bases de données orientées graphes permettent aux utilisateurs de créer et de modifier la structure du graphe au fur et à mesure que les données évoluent. Cela signifie qu'il n'y a pas de contrainte de besoin d'un schéma strict dès le départ, ce qui permet une flexibilité considérable lors de la conception et de l'évolution du modèle de données.

2. **Modélisation naturelle des données** : Les bases de données orientées graphes permettent de modéliser les données de manière naturelle en utilisant des nœuds, des arêtes et des propriétés. Cette modélisation flexible reflète souvent la structure réelle des données dans le monde réel, ce qui rend la représentation des données plus intuitive et flexible.

3. **Évolutivité du schéma** : Les bases de données orientées graphes permettent d'ajouter de nouveaux types de nœuds, d'arêtes ou de propriétés au fur et à mesure que de nouveaux besoins apparaissent, sans avoir à réorganiser ou à migrer la structure existante. Cela facilite l'évolutivité du schéma en fonction des changements induis pas des exigences métier.

4. **Absence de contraintes d'intégrité rigides** : Contrairement aux bases de données relationnelles qui appliquent des contraintes d'intégrité strictes, les bases de données orientées graphes offrent souvent plus de flexibilité en matière de contraintes. Cela peut être un avantage lorsque les données sont complexes ou mal structurées, car cela permet aux utilisateurs de représenter et de gérer ces données sans être limités par des contraintes rigides.

5. **Représentation de données variées** : Les bases de données orientées graphes peuvent stocker des données variées, y compris des propriétés riches et complexes, des données spatiales et/ou temporelles. Cette capacité à représenter divers types de données contribue à la flexibilité générale de la base de données.

Il est important de noter que la flexibilité offerte par les bases de données orientées graphes peut être à la fois un avantage et un défi, car elle nécessite une gestion proactive de la qualité des données et des modèles appropriés pour garantir la cohérence et la précision des informations stockées.

#### 2.2.6.4 Discussion

Les bases de données orientées graphes offrent une solution puissante pour la modélisation et l'analyse des données inter-connectées. Elles sont adaptées à une gamme

variée de cas d'utilisation et continuent d'évoluer pour relever les défis liés à la gestion et à l'analyse de graphes de grande taille. Le choix d'une base de données orientée graphes dépendra des besoins spécifiques et des exigences en termes de performances, de scalabilité et de convivialité.

### 2.2.7 Partitionnement de graphes

Il existe dans la littérature, divers types d'algorithmes de partitionnement de graphes qui ont été développés pour résoudre une multitude de problèmes. Ces algorithmes visent à diviser un graphe en sous-ensembles ou communautés, où les nœuds d'un sous-ensemble ont des liens forts entre eux, tandis que les liens entre les sous-ensembles sont faibles (Fortunato and Hric (2016); Newman (2016)). Ces approches de partitionnement de graphes peuvent être regroupées en différentes familles, chacune ayant ses propres caractéristiques et techniques :

#### 2.2.7.1 Algorithmes de partitionnement de graphes basés sur la modularité

Les algorithmes basés sur la modularité visent à identifier des partitions de graphe qui maximisent la qualité des communautés détectées. La modularité mesure la différence entre le nombre de liens observés entre les nœuds d'une même communauté et le nombre attendu dans un modèle aléatoire. L'objectif est de trouver des partitions qui maximisent cette différence :

**Algorithme de Louvain** (Blondel et al. (2008)) : C'est un algorithme de partitionnement de graphe qui optimise la modularité en déplaçant les sommets entre les communautés. Il se compose de deux étapes : l'optimisation locale et l'agrégation des communautés.

**Algorithme de Girvan-Newman** (Newman and Girvan (2004)) : Cet algorithme utilise la notion de « *betweenness centrality* » pour supprimer itérativement les arêtes ayant la plus grande contribution à la modularité globale du graphe. Cela conduit à la formation de communautés disjointes.

**Fast Greedy** (Clauset et al. (2004)) : Il fusionne itérativement des communautés de nœuds voisins pour maximiser la modularité du graphe. À chaque étape, il identifie la paire de communautés dont la fusion entraînerait la plus grande augmentation de modularité et les fusionne. Ce processus se répète jusqu'à ce qu'aucune fusion supplémentaire ne puisse augmenter la modularité. L'algorithme priorise la création de communautés plus grandes, ce qui peut conduire à des résultats de détection de communautés rapides et efficaces.

#### 2.2.7.2 Algorithmes basés sur le partitionnement spectral

Les algorithmes de partitionnement de graphes basés sur le partitionnement spectral utilisent les propriétés spectrales du graphe (Matrice d'adjacence, Matrice Laplacienne) pour réaliser la partition. Voici quelques-uns des algorithmes les plus couram-

ment utilisés dans cette catégorie :

**Partitionnement spectral** (Fiedler (1973)) : Cet algorithme utilise les vecteurs propres de la matrice de Laplace du graphe pour effectuer la partition. Il projette les sommets du graphe dans un espace de dimension réduite en utilisant les vecteurs propres associés aux plus petites valeurs propres. Ensuite, il applique des méthodes de *clustering*, comme k-means, pour regrouper les sommets projetés.

**Ratio Cut** (Ng et al. (2001)) : Cet algorithme cherche à minimiser le ratio *cut*, qui mesure le nombre d'arêtes entre les partitions, normalisé par la taille des partitions. Il utilise les vecteurs propres de la matrice de Laplace normalisée du graphe pour réaliser la partition en séparant les sommets selon une coupe qui minimise le ratio *cut*.

**Normalized Cut** (Shi and Malik (2000)) : Cet algorithme cherche à minimiser le *normalized cut* qui est une mesure de la similarité entre les partitions basée sur les poids des arêtes coupées et les poids arêtes internes. Il utilise les vecteurs propres de la matrice de Laplace normalisée du graphe pour effectuer la partition en séparant les sommets selon une coupe qui minimise le *normalized cut*.

#### 2.2.7.3 Algorithmes basés sur la modélisation probabiliste

Les méthodes de modélisation probabiliste pour le partitionnement de graphes visent à modéliser les relations entre les nœuds d'un graphe à l'aide de distributions probabilistes. Ces méthodes utilisent des modèles statistiques pour capturer les tendances de connexion entre les nœuds et identifier les structures de communautés. Voici quatre principaux algorithmes de modélisation probabiliste pour le partitionnement de graphes :

**Stochastic Block Models** (SBM) (Holland et al. (1983)) : Le modèle de bloc stochastique (SBM) est l'une des méthodes de modélisation probabiliste les plus populaires pour le partitionnement de graphes. Il attribue des probabilités d'arêtes entre les groupes de nœuds (communautés) et génère ensuite des arêtes du graphe en fonction de ces probabilités. Les paramètres du SBM peuvent être estimés à partir du graphe observé pour détecter les communautés.

**Latent Dirichlet Allocation** (LDA) (Blei et al. (2003)) : Bien que principalement utilisé dans le traitement du langage naturel, LDA peut également être adapté au partitionnement de graphes. LDA attribue des distributions de sujets latents aux nœuds, où chaque nœud appartient à plusieurs sujets avec certaines probabilités. Les nœuds partageant des sujets similaires sont regroupés en communautés.

**Mixed-Membership Stochastic Block Models (MMSB)** (Airoldi et al. (2008)) : Le MMSB étend le SBM en permettant à chaque nœud d'appartenir à plusieurs groupes (communautés) avec des poids de mélange. Il modélise les interactions entre les nœuds en tenant compte des affiliations de groupes multiples, ce qui en fait une méthode adaptée aux graphes où les nœuds peuvent avoir des rôles multiples.

#### 2.2.7.4 Algorithmes de Bisection

Les algorithmes de Bisection visent à diviser le graphe en deux parties équilibrées tout en minimisant la coupure entre ces parties. Ils sont souvent utilisés dans des applications de partitionnement de tâches ou de ressources :

**Spectral Bisection** (Pothen et al. (1990)) : Cet algorithme projette les nœuds dans un espace de valeurs propres de la matrice Laplacienne pour obtenir une bisection équilibrée du graphe. Il identifie un vecteur propre associé à la seconde plus petite valeur propre pour réaliser la bisection.

**Kernighan-Lin** (Kernighan and Lin (1970)) : Cet algorithme itératif optimise l'équilibrage et la coupure en permutant des nœuds entre deux partitions. Il sélectionne des paires de nœuds à échanger en utilisant une stratégie gloutonne.

**Fiduccia-Mattheyses** (Fiduccia and Mattheyses (1982)) : Aussi appelé FM, cet algorithme déplace les nœuds entre les partitions pour améliorer l'équilibrage et la coupure en utilisant des passes ascendantes et descendantes.

#### 2.2.7.5 Algorithmes de Recuit Simulé

Les algorithmes de recuit simulé sont des méta-heuristiques inspirées du processus de refroidissement d'un matériau. Ils explorent l'espace de solutions en effectuant des mouvements probabilistes, avec une probabilité de plus en plus faible de choisir des solutions moins optimales au fur et à mesure du processus :

**METIS** (Karypis and Kumar (1998)) : METIS utilise une variante de l'algorithme de recuit simulé pour partitionner les graphes en optimisant la coupure. Il explore les partitions en déplaçant les nœuds entre les blocs pour réduire la coupure totale.

**Simulated Annealing** (Kirkpatrick et al. (1983)) : L'algorithme de recuit simulé classique effectue des mouvements stochastiques pour explorer l'espace de solutions. La température (paramètre qui contrôle la probabilité d'acceptation) est progressivement réduite, ce qui permet de converger vers une solution optimale.

**Ant Colony Optimization** (Dorigo and Stützle (2004)) : Inspiré du comportement des fourmis cherchant la meilleure route, cet algorithme explore l'espace de solutions en utilisant des agents virtuels pour détecter les communautés.

#### 2.2.7.6 Algorithmes de Propagation de Label

Ces algorithmes attribuent des étiquettes (labels) aux nœuds et les propagent itérativement entre les voisins pour détecter les communautés :

**Label Propagation Algorithm (LPA)** (Raghavan et al. (2007)) : Les nœuds adoptent les étiquettes majoritaires de leurs voisins, itérativement. Il est efficace pour détecter des communautés locales mais pas avec des structures hiérarchiques.

**LabelRankT** (Xie et al. (2013)) : Il attribue des scores de classement aux nœuds en se basant sur la propagation d'étiquettes, puis classe les nœuds en fonction de leurs scores. Il est robuste aux bruits et aux étiquettes incorrectes.

**BigClam (Big Community Algorithm)** (Yang and Leskovec (2013)) : Modélise les relations sociales en attribuant des affiliations communautaires aux nœuds et optimise une fonction d'objectif basée sur la probabilité d'observation des arêtes.

#### 2.2.7.7 Approches de Coupe de Graphes

Ces approches visent à minimiser le nombre d'arêtes coupées tout en divisant le graphe en plusieurs sous-graphes. Ces méthodes sont particulièrement utilisées dans des applications telles que la conception de circuits intégrés, la segmentation d'images et la partition de réseaux de communication.

**Min-Cut / Max-Flow** (Ford and Fulkerson (1962)) : Ces algorithmes résolvent le problème de partitionnement en trouvant la coupe minimale ou le flux maximal dans le graphe. Ils sont utilisés pour partitionner des réseaux de communication et des circuits électroniques.

**Multilevel Partitioning** (Karypis and Kumar (1998)) : Utilise une approche descendante pour réduire la taille du graphe tout en préservant sa structure, puis applique des algorithmes de coupe de graphe sur les niveaux réduits.

**Karger's Min-Cut Algorithm** (Karger and Stein (1996)) : Utilise la technique du contrat aléatoire pour contracter les nœuds du graphe de manière aléatoire, réduisant ainsi le problème de partitionnement à la recherche d'une coupe minimale dans le graphe réduit.

Le tableau 2.4 synthétise les familles d'algorithmes de partitionnement de graphes, en se basant sur différents critères tels que le type, les approches utilisées et les domaines d'utilisation (Fortunato and Hric (2016); Newman (2016)) :

Il est important de noter que certains algorithmes peuvent appartenir à plusieurs catégories, car ils utilisent des techniques combinées pour atteindre les objectifs de partitionnement. De plus, de nouveaux algorithmes de partitionnement de graphes sont régulièrement proposés dans la littérature scientifique ; et cette liste n'est donc pas exhaustive. Le choix de l'algorithme dépendra des spécificités du graphe, des objectifs de partitionnement et des contraintes du problème à résoudre.

TABLE 2.4 – Comparaison des Familles d'Algorithmes de Partitionnement de Graphes

| **Famille** | **Type** | **Approches Utilisées** | **Domaines d'Utilisation** |
|---|---|---|---|
| Algorithmes basés sur la Modularité | Optimisation combinatoire | Maximisation de la modularité, itérations et déplacement de nœuds | Réseaux sociaux, analyse de réseaux, biologie |
| Algorithmes de Bisection | Optimisation combinatoire | Décomposition spectrale, itérations de permutation | Conception de circuits, calcul parallèle, optimisation |
| Algorithmes de Recuit Simulé | Métaheuristique | Recherche probabiliste, exploration itérative | Partitionnement de graphe, optimisation combinatoire |
| Algorithmes de Propagation de Label | Heuristique | Propagation itérative des étiquettes, majorité des voisins | Classification de documents, réseaux sociaux |
| Algorithme de Partitionnement Spectral | Analyse matricielle | Décomposition en valeurs propres, clustering | Apprentissage non supervisé, traitement d'image |
| Méthodes de Modélisation Probabiliste | Modélisation probabiliste | Estimation de maximum de vraisemblance, approches statistiques | Réseaux sociaux, détection de communautés |
| Approches de Coupe de Graphe | Optimisation combinatoire | Minimisation du nombre d'arêtes coupées | Conception de circuits, partitionnement de données |

## 2.3 Étude de l'art

### 2.3.1 Introduction

L'évolution des modèles multidimensionnels a suscité de nombreux travaux de recherche, pouvant être classés en deux grandes catégories : l'évolution des données et l'évolution des schémas.

Dans la première catégorie, l'évolution des données, les recherches se focalisent sur la gestion des changements et des mises à jour des données dans l'entrepôt multidimensionnel. Cela inclut notamment la gestion des mises à jour des faits, des dimensions et des hiérarchies, ainsi que la gestion des différentes versions des données. Les travaux de cette catégorie visent à assurer la cohérence des données au fil du temps et à permettre leur mise à jour de manière efficace et transparente pour faciliter les interrogations ultérieures.

Dans la seconde catégorie, l'évolution des schémas, l'accent est mis sur les changements structurels du modèle multidimensionnel lui-même. Cela inclut la modification des dimensions existantes, l'ajout de nouvelles dimensions, la création de nouvelles hiérarchies ou la suppression de hiérarchies obsolètes. Les recherches dans cette catégorie visent à rendre le schéma plus flexible et adaptable aux nouveaux besoins d'analyse, tout en préservant la cohérence des données existantes. Ces efforts permettent aux entrepôts de données de s'adapter aux évolutions des besoins métier et d'assurer des analyses pertinentes et en temps réel.

### 2.3.2 Travaux connexes sur l'évolution des données dans les entrepôts de données

Il existe plusieurs travaux de recherche dans la littérature qui ont abordé de nombreuses facettes des entrepôts de données temporelles (Faisal et al., 2017; Golfarelli and Rizzi, 2018), notamment le type de temporalité, le niveau conceptuel et logique, l'évolution des données, les changements de dimensions, les mesures différées, la mise en œuvre des approches, les requêtes temporelles et l'agrégation dans les relations temporelles. Nous pouvons citer les travaux suivants :

(Bliujute et al., 1998) : Dans cet article intitulé «*Maintaining Temporal Information in Data Warehouses*», les auteurs ont proposé une approche pour gérer les aspects temporels des données dans les entrepôts. Ils ont introduit le concept de suppression de la dimension temporelle et son remplacement par des étiquettes temporelles au niveau de chaque instance de données. Ils ont exploré l'utilisation des étiquettes VT (Valid_Time) et TT (Transaction_Time) pour représenter les périodes de validité des données et les périodes pendant lesquelles les données sont présentes dans l'entrepôt de données.

(Mendelzon and Vaisman, 2000) : Dans «*Temporal Data Warehousing*», les auteurs ont proposé un modèle multidimensionnel temporel permettant des requêtes OLAP

temporelles. Ils ont exploré les différentes manières d'intégrer des informations temporelles dans les dimensions et les faits de l'entrepôt ; ce qui permet aux utilisateurs d'analyser les données en fonction du temps.

(Golfarelli and Rizzi, 2007) : («*Managing Late Measurements in Data Warehouses*»), les auteurs ont proposé ici des solutions de conception alternatives pour prendre en charge différents types de requêtes en présence de mesures tardives dans les entrepôts de données. Ces solutions permettent une analyse historique significative en prenant en compte les données qui arrivent tardivement dans l'entrepôt de données. Ces solutions sont basées sur l'application de la distinction entre le temps de transaction et le temps valide dans le schéma qui représente le fait d'intérêt.

(Faisal and Sarwar, 2014) : Dans leur étude «*Handling slowly changing dimensions in data warehouses*», les auteurs ont comparé les performances des SCD de type 2 et hybrides dans un entrepôt de données temporelles. Ils ont examiné comment ces deux approches gèrent les changements dans les dimensions au fil du temps et ont évalué leur impact sur les requêtes OLAP temporelles.

(Saroha and Gosain, 2015) : Dans «*Handling Retroactive and Proactive Updates in Bi-Temporal Data Warehouse*», les auteurs ont proposé une approche pour gérer les mises à jour rétro-actives et pro-actives dans un entrepôt de données bi-temporelles. Ils ont utilisé à la fois l'horodatage VT et TT pour suivre les changements dans les données au fil du temps, en permettant ainsi des analyses précises de l'historique des données.

(Garani et al., 2016) : Dans leur travail intitulé «*Logical Modeling of Temporal Data Warehouse*», les auteurs ont proposé une approche de modélisation logique pour les entrepôts de données temporelles. Ils ont introduit le schéma de *starnest* temporel, dans lequel le temps est traité comme des attributs temporels dans chaque dimension temporelle, plutôt que comme une dimension séparée.

(Phungtua-Eng and Chittayasothorn, 2019) : («*Resolving Slowly Changing Dimensions in Data Warehouses using Temporal Database Features*»), les auteurs ont exploré les caractéristiques des bases de données temporelles pour résoudre le problème des changements dans les dimensions SCD, en utilisant notamment SQL. Ils ont suggéré l'utilisation de tables d'état dans le SGBD Oracle pour conserver l'historique des données et suivre les changements dans les dimensions au fil du temps. Cette approche a permis de maintenir une vue complète et cohérente de l'évolution des données dans l'entrepôt, facilitant ainsi les analyses historiques.

(Ahmed et al., 2015, 2020) : Ces auteurs ont présenté un modèle logique et une technique d'interrogation spécifiquement conçus pour les entrepôts de données temporelles. Ils ont exploré les différentes manières de modéliser les données temporelles et ont proposé une approche pour réaliser des requêtes temporelles efficaces dans ce contexte. Leur modèle logique a permis de représenter les aspects temporels des données de manière claire et structurée, facilitant ainsi l'interrogation et l'analyse des

données en fonction du temps.

Ces travaux de recherche ont tous contribué à améliorer notre compréhension de la gestion des données temporelles dans les entrepôts de données. Diverses approches ont été explorées pour représenter et manipuler les aspects temporels des données, permettant ainsi aux utilisateurs de réaliser des analyses plus fines en fonction du temps.

### 2.3.3 Travaux connexes sur l'évolution des schémas dans les entrepôts de données

La deuxième famille des travaux sur l'évolution des modèles multidimensionnels se focalise sur les changements dans les schémas de ces modèles et leur impact sur les analyses. Cette famille est constituée de deux sous-familles distinctes.

La première aborde l'évolution des schémas sans prendre en compte l'historique des changements. Dans ces approches, seule l'état final du modèle est pris en considération, et les modifications antérieures sont ignorées. Cela signifie que les versions précédentes du schéma ne sont pas conservées, et seul le schéma actuel est utilisé pour les analyses. Cette approche peut être simple à mettre en œuvre, mais elle présente des limites en termes de traçabilité des changements et de réversibilité.

La deuxième sous-famille se focalise sur l'évolution du modèle multidimensionnel en un ensemble de versions, en prenant en charge l'historique complet du schéma. Chaque version représente un état différent du modèle à un moment donné dans le temps, ce qui permet de conserver l'historique des évolutions. Cette approche de multi-versions offre une traçabilité complète des changements et permet de revenir à des versions antérieures du schéma si nécessaire. Cela rend le modèle global consultable à pour n'importe quel moment, offrant ainsi de nouvelles opportunités d'analyses et de compréhension de l'évolution des données au fil du temps.

Dans le contexte de modèles d'évolution de schéma sans historisation, différentes approches ont été proposées pour gérer les modifications dans la structure du modèle multidimensionnel.

Certains travaux se concentrent sur les opérations de mise à jour des dimensions et de leurs hiérarchies. Par exemple, des opérateurs ont été développés pour faire évoluer le modèle en modifiant les dimensions et leurs hiérarchies (Hurtado et al., 1999), ainsi que les dimensions et les faits (Blaschka et al., 1999). Ces opérations visent à adapter le modèle aux nouveaux besoins d'analyse et aux changements dans les sources de données.

D'autres travaux se sont intéressés à la création de nouveaux niveaux dans les hiérarchies des dimensions. Ces approches permettent de mettre à jour le modèle de l'entrepôt sans remettre en cause la cohérence de l'analyse des données existantes. Par exemple, certains travaux utilisent des ressources lexicales, telles que *WordNet*, pour enrichir automatiquement les hiérarchies des dimensions (Mazón et al., 2006).

D'autres approches exploitent les connaissances des utilisateurs eux-mêmes pour enrichir les hiérarchies (Favre et al., 2007). Il est également possible de définir un ensemble d'opérations d'évolution et de contraintes au niveau des hiérarchies pour assurer l'intégrité des données et la cohérence du schéma (Talwar and Gosain, 2012).

En outre, certains travaux considèrent l'entrepôt de données comme un ensemble de vues matérialisées construites à partir des sources de données. Dans cette vision, la maintenance des vues matérialisées est directement induite par l'évolution des sources de données (Bellahsene, 2002). Ainsi, toute modification dans les sources de données entraîne automatiquement une mise à jour du modèle de l'entrepôt.

S'agissant de modèles d'évolution de schéma avec historisation, l'approche adoptée se base sur la création de nouvelles versions du modèle après chaque évolution, tout en préservant les versions précédentes. Ainsi, l'entrepôt de données est constitué d'un ensemble de versions ; chacune étant valide durant une période de temps spécifique, déterminée par un temps de début de validité et un temps de fin de validité.

Plusieurs travaux de recherche se sont intéressés à cette approche. Parmi eux, la suite de travaux de T. Morzy, R. Wrembel et B. Babel dans (Morzy and Wrembel, 2003, 2004; Wrembel and Bebel, 2005; Wrembel and Morzy, 2006). Les auteurs se sont concentrés sur la modélisation d'un entrepôt de données multi-versions. Ils ont abordé la création et la gestion des versions dans cet entrepôt, la gestion des requêtes croisées, ainsi que la gestion des méta-informations. Ces travaux ont proposé des approches pour faire évoluer chronologiquement le schéma d'un entrepôt de données, en utilisant le versionnement réel et alternatif avec une séparation physique. Ces versions sont valides pendant des périodes de temps spécifiques, définies par leurs date de début et date de fin. Ils ont également introduit la notion de versions alternatives créées par l'utilisateur pour les analyses de type «*what-if*». Des requêtes SQL avancées ont été développées pour interroger ce type d'entrepôts. Ces requêtes permettent d'interroger la dernière version, une version ultérieure ou un ensemble de versions réelles ou alternatives, ainsi que la fusion de plusieurs résultats partiels.

Dans (Ravat et al., 2006), les auteurs ont proposé un modèle multidimensionnel basé sur les multi-versions. Ils ont défini un schéma en constellation comme un ensemble de versions de schémas en étoiles. Une version en étoile est associée à un intervalle temporel et est composée de versions de dimension (une version par dimension qui est composée d'un schéma et de ses instances) associées à une version de fait (définie par un schéma et ses instances). Une version de fait ou une version de dimension est définie par une fonction de mise en correspondance. Cette fonction est formalisée par une expression algébrique relationnelle sur les données de l'entrepôt de données relationnel pour alimenter les versions.

D'autres travaux, comme ceux de Stefano Rizzi et Matteo Golfarelli (Rizzi and Golfarelli, 2007), ont proposé l'utilisation de graphes pour représenter le modèle multidimensionnel avec historisation. Ils ont défini une algèbre d'opérations permettant de déterminer un schéma consultable sur plusieurs versions. Chaque fois qu'une nou-

velle version est créée, un schéma augmenté est créé pour augmenter la flexibilité des requêtes croisées.

Par ailleurs, les travaux de Ahmed et al dans (Ahmed et al., 2014; Ahmed and Zimányi, 2015) se sont intéressés à un modèle permettant la gestion des entrepôts de données multi-versions. Ils ont proposé trois approches : celle des tables mono-versions (*STV - Single Tables Versions*), l'approche des tables multi-versions (*MTV - Multiple Tables Versions*) et enfin celle des tables hybrides (*HTV - Hybrid Tables Versions*). Ces approches visent à gérer les changements de structures des dimensions et des tables de faits, en différenciant la fréquence des changements entre ces deux types d'entités. Ils ont proposé de représenter les dimensions par des STV, dont la mise à jour se fait après chaque changement, tandis que les tables de faits sont représentées par des MTV, avec la création d'une nouvelle version après chaque changement.

Dans (Ahmed et al., 2021), les mêmes auteurs ont présenté un modèle à évolution de schéma en multi-versions en complément au modèle temporel présenté dans (Ahmed et al., 2020). Ils ont étendu leur approche en définissant la sémantique des opérateurs de modification de schéma (*SMO - Schema Modification Operators*) qui permettent de dériver différentes versions de schéma. Ces opérateurs permettent d'effectuer des modifications structurelles sur le schéma de l'entrepôt ; ce qui facilite la gestion et l'évolution du modèle multidimensionnel au fil du temps. Ainsi, les deux modèles sont complémentaires et prennent en charge l'évolution temporelle du contenu et l'évolution des schémas.

L'approche de multi-versions est particulièrement intéressante car elle permet de gérer efficacement les changements constants dans les sources de données et les besoins d'analyse soumis à des changements. En conservant l'historique du modèle, les analyses peuvent être effectuées sur des versions antérieures du schéma pour comprendre les tendances passées ou les évolutions historiques des données. Cela offre une flexibilité et une adaptabilité accrues aux changements, tout en préservant la traçabilité et la qualité des analyses.

### 2.3.4 Discussion et positionnement

En analysant ces travaux de recherche, il devient évident que les critères essentiels à considérer dans un modèle multidimensionnel évolutif et flexible sont l'évolution des schémas et celle des données, la traçabilité de ces évolutions, ainsi que la prise en charge et la complexité des requêtes croisées. Le tableau 2.5 récapitule et compare de manière synthétique les divers travaux de recherche que nous avons examinés dans notre étude bibliographique :

Ces différentes approches visent à rendre le modèle multidimensionnel plus agile et évolutif face aux changements dans les données et les besoins d'analyse. Chaque approche a ses avantages et ses limites en fonction du contexte d'utilisation et des contraintes spécifiques de l'entrepôt de données. L'objectif commun de ces travaux est

TABLE 2.5 – Comparative des travaux antérieurs

| **Étude** | **Évolution temporelle des données** | **Évolution des schémas** | **Historique d'évolution des schémas** | **Requêtes croisées** |
|---|---|---|---|---|
| Bliujute et al. (1998) | √ | | | |
| Hurtado et al. (1999) | | √ | | |
| Blaschka et al. (1999) | | √ | | |
| Mendelzon and Vaisman (2000) | √ | | | |
| Bellahsene (2002) | | √ | | |
| Morzy and Wrembel (2003, 2004); Wrembel and Bebel (2005); Wrembel and Morzy (2006) | | √ | √ | √ |
| Benitez-Guerrero et al. (2004) | | √ | | |
| Ravat et al. (2006) | | √ | √ | √ |
| Mazón et al. (2006) | | √ | | |
| Favre et al. (2007) | | √ | | |
| Rizzi and Golfarelli (2007) | | √ | √ | √ |
| Talwar and Gosain (2012) | | √ | | |
| Faisal and Sarwar (2014) | √ | | | |
| Ahmed et al. (2014); Ahmed and Zimányi (2015) | | √ | √ | √ |
| Saroha and Gosain (2015) | √ | | | |
| Garani et al. (2016) | √ | | | |
| Phungtua-Eng and Chittayasothorn (2019) | √ | | | |
| Ahmed et al. (2020, 2021) | √ | √ | √ | √ |

de proposer des solutions permettant de gérer efficacement l'évolution du modèle multidimensionnel tout en préservant la cohérence des analyses.

Les entrepôts de données temporelles jouent un rôle essentiel dans la gestion des changements de contenus et dans l'offre d'analyses temporelles malgré les variations dans les dimensions. En conservant l'historique des données et en associant des étiquettes temporelles aux enregistrements, les entrepôts de données temporelles permettent des analyses rétrospectives précises et la prise en compte des différentes versions des données au fil du temps.

D'un autre côté, les entrepôts multi-versions de données sont également conçus pour prendre en charge les changements de schémas et répondre aux nouveaux besoins d'analyse. L'évolution des modèles multidimensionnels et la gestion des modifications structurelles, telles que l'ajout de nouvelles dimensions, la suppression de dimensions obsolètes ou la création de nouvelles hiérarchies, permettent aux entrepôts de données de rester flexibles et adaptables aux changements dans l'environnement commercial.

En combinant une approche temporelle dans les entrepôts de données pour la gestion des changements de contenu avec l'approche multi-versions pour la prise en charge des changements de schémas, les entrepôts de données offrent un cadre complet pour gérer l'évolution des données et des besoins analytiques. Cette combinaison permet d'assurer des analyses cohérentes et pertinentes dans un contexte évolutif.

Les travaux de recherches menées dans (Ahmed et al., 2020, 2021) ont introduit deux approches complémentaires pour aborder les évolutions des données et de la structure au sein des entrepôts. Ces approches ont permis de relever les défis posés par les modifications de contenus et des schémas, en garantissant la souplesse et l'adaptabilité des entrepôts de données aux évolutions des exigences métier et des sources de données. Grâce à ces approches, les entrepôts de données sont en mesure de fournir des analyses qui couvrent tant les aspects historiques que les informations actuelles, tout en maintenant l'intégrité et la cohérence des données.

Cependant, il est important de souligner que ces travaux, tout comme l'ensemble des autres recherches portant sur l'évolution au sein des entrepôts de données, s'appuient généralement sur le modèle relationnel. Ce modèle se caractérise par l'utilisation de tables de données, présente intrinsèquement des défis liés à l'évolution des schémas en raison de sa structure et de son architecture particulières.

En effet, les travaux précédemment présentés ont apporté de nombreuses solutions pour gérer l'évolution des données dans les entrepôts de données et résoudre les problèmes liés aux changements lents de dimensions SCD, notamment en intégrant le support temporel dans SQL :2011 (Kulkarni and Michels, 2012) et en le mettant en œuvre dans certains systèmes de gestion de bases de données (SGBD) tels que SQL Server 2016, Oracle 12c, IBM DB2 et Teradata (Poscic et al., 2018). Cependant, des contraintes persistent en ce qui concerne l'évolution des schémas en raison de la nature rigide du modèle relationnel, des contraintes d'intégrité et des défis liés à la gestion et

à la consultation des versions.

Les bases de données relationnelles sont fondées sur un modèle rigide où les schémas (tables, colonnes, relations, etc.) doivent être définis à l'avance avec des types spécifiques de données. Ainsi, toute modification du schéma induit généralement des altérations dans la structure des tables existantes, ce qui peut être difficile à gérer et à maintenir.

Ces bases de données utilisent des contraintes d'intégrité pour garantir la cohérence des données, telles que les clés primaires et les clés étrangères. Cependant, ces contraintes peuvent devenir complexes à maintenir en cas d'évolution des schémas, car les modifications structurelles peuvent affecter la validité des contraintes existantes.

Dans le contexte de l'évolution des schémas en multi-versions, une approche courante, préconisée notamment dans les travaux d'(Ahmed et al., 2020, 2021), consiste à créer une nouvelle version des tables de faits à chaque modification du schéma. Cela devient particulièrement pertinent lorsqu'il y a des changements du niveau d'agrégation des mesures entre différentes versions du schéma. Ainsi, les requêtes pour chaque version sont exécutées sur la table des faits correspondante à cette version. Cependant, cette approche peut devenir complexe voire impossible à gérer, surtout en présence d'un grand nombre de versions. Les requêtes croisées deviennent également plus complexes, nécessitant l'union de multiples requêtes sur différentes versions de la table des faits et un alignement précis du niveau de granularité des résultats.

En résumé, bien que des avancées significatives aient été réalisées pour gérer l'évolution des données et des schémas dans les entrepôts de données, les défis liés à la rigidité du modèle relationnel et à la complexité de la gestion des versions persistent, nécessitant une attention continue dans la recherche de solutions efficaces et adaptables.

C'est dans ce contexte que nous nous sommes intéressés aux problématiques de recherche liées à la flexibilité de l'évolution dans les modèles multidimensionnels. Pour surmonter les limitations inhérentes au modèle relationnel dans le domaine de l'entreposage de données et pour répondre aux défis de l'évolution des modèles multidimensionnels ; nous avons orienté notre attention vers les systèmes de gestion de bases de données de type NoSQL (Not only SQL), plus précisément les bases de données orientées graphes.

Le choix des bases de données orientées graphes s'est imposé naturellement en raison de leur similitude avec le modèle relationnel. En effet, le modèle en graphes repose sur les concepts de nœuds et d'arêtes, qui peuvent correspondre respectivement aux notions d'entité et d'association dans le modèle relationnel.

En plus de cette similitude conceptuelle, le modèle en graphes offre des avantages en termes de scalabilité, principalement en raison de l'absence d'un schéma préétabli et de contraintes d'intégrité rigides. Les modèles proposés par (Campos et al., 2016; Debrouvier et al., 2021) ont démontré des résultats prometteurs en ce qui concerne la

flexibilité et l'évolution des bases de données NoSQL orientées graphes. De plus, ces bases de données se révèlent particulièrement adaptées à la gestion de données hétérogènes fortement connectées et peu ou pas structurées, comme souligné par (Pokorný, 2015).

L'utilisation des bases de données orientées graphes ouvre ainsi de nouvelles perspectives pour la gestion de l'évolution des schémas et des données dans le domaine de l'entreposage de données, offrant à la fois la flexibilité nécessaire pour faire face aux changements constants et la capacité de gérer des données complexes et interconnectées.

En réponse à la problématique soulevée dans le cadre de cette thèse et en considération des éléments discutés précédemment, nous proposons dans le cadre de ce mémoire un modèle multidimensionnel agile adapté pour la gestion de données massives, capable de répondre aux nouveaux besoins d'analyse et de prendre en compte l'évolution des sources de données de manière flexible.

Ce modèle englobe un formalisme complet, un méta-modèle détaillé, des règles de transitions entre les versions, un cadre de requêtes optimisé et des études de validation approfondies. En s'appuyant sur des structures de graphes, ce modèle présente une évolutivité inhérente et peut être déployé sous différentes versions, permettant ainsi une évolution progressive à la fois au niveau des schémas et des données. De plus, il offre la capacité de conserver un historique complet pour des consultations ultérieures.

En surmontant les défis inhérents aux modèles multidimensionnels traditionnels, ce modèle représente une avancée significative dans le domaine. De plus, nous mettrons en avant une approche automatisée pour gérer l'évolution au sein de notre modèle, ce qui simplifie et accélère le processus de modification et de mise à jour.

## 2.4 Conclusion

Ce chapitre a établi les fondations nécessaires à la compréhension et à l'exploration des domaines clés abordés dans cette thèse. Nous avons exposé un aperçu approfondi des entrepôts de données, en mettant en lumière leurs concepts, architectures et processus fondamentaux. De plus, nous avons exploré les bases de données temporelles, les entrepôts de données multidimensionnels, les bases de données NoSQL orientées graphes ainsi que les algorithmes de partitionnement de graphes, fournissant ainsi un contexte intéressant pour la suite de notre étude.

En outre, nous avons passé en revue les travaux de recherche pertinents liés à l'évolution des modèles multidimensionnels. Cette exploration a permis de cerner les approches existantes, leurs avantages et leurs limites, et de positionner nos propres recherches au sein de ce paysage en constante évolution. Cette synthèse des connaissances antérieures nous a inspiré pour poursuivre nos travaux en abordant certains défis non résolus et pour proposer des solutions novatrices dans le domaine de l'évolution des

modèles multidimensionnels.

Dans les chapitres qui suivants, nous approfondirons nos analyses, méthodes et contributions pour relever les enjeux identifiés et promouvoir une gestion évolutive et agile des données dans le contexte des entrepôts de données multidimensionnels agiles.

CHAPITRE 3

# MODÈLE MULTIDIMENSIONNEL AGILE À BASE DE GRAPHES

## Sommaire

## 3.1 Introduction

Dans ce chapitre, nous présentons notre modèle multidimensionnel agile, appelé GAMM (*Graph-based Agile Multidimensional Model*), qui permet l'évolution des entrepôts de données tant au niveau schéma qu'au niveau données. GAMM est basé sur une structure de graphe et sur le concept d'entrepôts multi-versions. Il offre ainsi aux concepteurs la possibilité d'intégrer dans l'entrepôt de nouvelles sources de données et de nouveaux besoins d'analyse afin d'enrichir ses capacités analytiques permettant de répondre de manière flexible aux nouvelles attentes des utilisateurs.

Sur le plan conceptuel, notre approche étend la modélisation multidimensionnelle classique en utilisant les concepts de base tels que les « faits », les « mesures », les « dimensions », les « hiérarchies » et les « niveaux ». Étant donnée l'évolution chronologique des schémas et des données, ainsi qu'à des fins d'historisation, nous avons introduit un étiquetage temporel et défini un cadre formel détaillé. De plus, nous avons proposé un méta-modèle pour l'identification et la gestion des différentes versions du schéma de l'entrepôt. GAMM permet une évolution progressive et itérative, aussi bien au niveau des schémas que des données. De plus, il conserve un historique complet pour chaque version du schéma, ce qui facilite la consultation des données antérieures.

En prenant en compte les limites des modèles multidimensionnels traditionnels liées à l'évolution des besoins d'analyse dans un contexte de Big Data, aux contraintes d'adaptation aux nouvelles sources de données, à la limitation d'évolution du modèle multidimensionnel classique et à la difficulté de gestion des versions du schéma, ainsi qu'à la cohérence des analyses compromise par les décalages de données et les changements dans les dimensions, notre modèle agile propose une solution innovante pour la gestion de données massives. Il offre une flexibilité accrue, ce qui permet de s'adapter plus facilement aux besoins d'analyse en constante évolution.

Ce chapitre est organisé comme suit : nous présentons tout d'abord deux exemples illustrant respectivement une évolution du schéma et une évolution des données dans un entrepôt (voir section 3.2). Ensuite, nous décrivons en détail notre modèle GAMM (voir section 3.3). Puis, nous présentons respectivement notre méta-modèle pour la gestion des versions du schéma dans la Section 3.4, la formalisation du modèle GAMM dans la Section 3.5 et l'entrepôt de données temporelles en graphe pour le stockage des données dans la Section 3.6. Enfin, nous concluons ce chapitre dans la Section 3.7.

## 3.2 Motivation

Nous présentons ci-après deux cas d'évolution dans un modèle multidimensionnel, illustrant respectivement l'évolution de schémas ainsi que l'évolution temporelle des données, notamment les changements au niveau des dimensions. Dans un souci de cohérence, nous utiliserons cette configuration de schéma comme exemple motivant pour présenter en détail notre proposition dans les chapitres 3, 4 et 5.

### 3.2.1 Exemple d'évolution de schéma

Considérons un entrepôt de données dont le schéma initial $V_0$ à l'instant $t_0$ est représenté dans la Figure 3.1. Ce schéma est composé du fait SALES et des trois dimensions PRODUCT, CUSTOMER et DATE comme suit :

— La dimension PRODUCT est décrite par les attributs *Pro_name* et *Unit_Price*, ainsi que par la hiérarchie CATEGORY. Cette dimension possède donc une seule hiérarchie sur l'axe d'analyse (PRODUCT, CATEGORY).

— La dimension CUSTOMER est décrite par les attributs *Cus_name*, *Address* et *Phone*, ainsi que par la hiérarchie CITY.

— La dimension DATE comporte deux niveaux de hiérarchie, à savoir MONTH et YEAR.

— Le fait SALES contient la mesure *Sales_Amount*.

Nous avons utilisé le formalisme du modèle dimensionnel des faits (*Dimensional Fact Model*) (DFM) Golfarelli et al. (1998) pour représenter les schémas conceptuels de notre exemple.

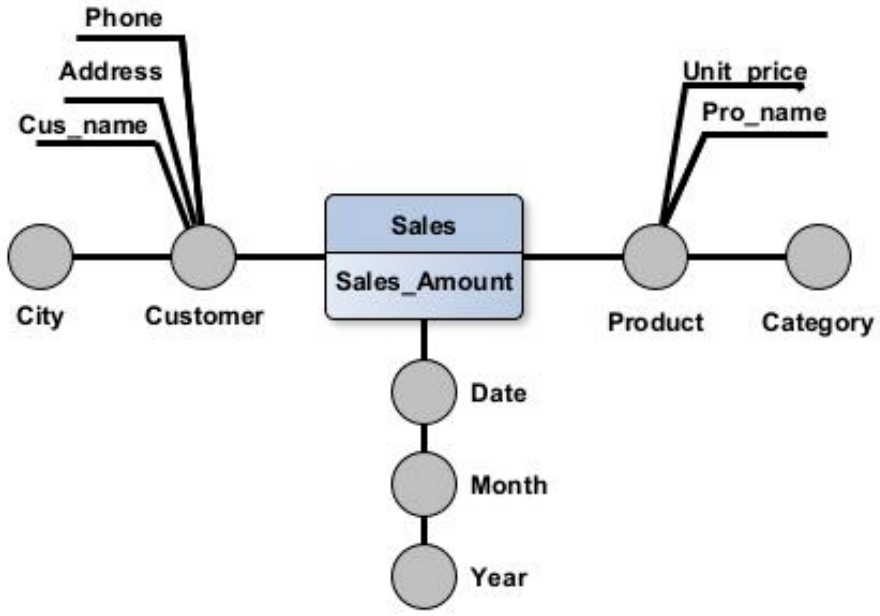



FIGURE 3.1 – Version de schéma $V_0$ à l'instant $t_0$

Supposons que ce schéma subisse plusieurs modifications en fonction de l'évolution

de nos besoins d'analyse et selon la disponibilité des sources de données. Ces modifications sont représentées comme suit :

1. À l'instant $t_1$, un nouveau niveau COUNTRY est ajouté à la dimension CUSTOMER, créant ainsi la version de schéma $V_1$. Cette modification permet de créer une nouvelle hiérarchie sur l'axe d'analyse (CUSTOMER, CITY, COUNTRY), répondant ainsi à un nouveau besoin d'analyse géographique plus agrégé à l'aide du niveau COUNTRY.
2. À l'instant $t_2$, une nouvelle dimension SUPPLIER avec les attributs *Sup_name*, *Address* et *Phone* est ajoutée, créant ainsi une nouvelle version de schéma $V_2$. Cette modification crée un nouvel axe d'analyse permettant d'augmenter le niveau de granularité de la mesure présente dans le fait SALES, en permettant d'analyser les ventes en fonction des fournisseurs.
3. À l'instant $t_3$, la dimension CUSTOMER est supprimée et la nouvelle mesure *Quantity* est ajoutée, créant ainsi la version de schéma $V_3$. L'ajout de cette nouvelle mesure répond à un nouveau besoin d'analyse, à savoir la possibilité de déterminer la quantité des ventes. Cette mesure n'est représentée que par les dimensions PRODUCT, SUPPLIER et DATE, ce qui implique la suppression de la dimension CUSTOMER du schéma. Cette opération diminue le niveau de granularité des mesures associées au fait SALES.

Le schéma conceptuel après ces changements est représenté dans la Figure 3.2 (la dimension CUSTOMER et ses hiérarchies ont été maintenues sur le diagramme avec des lignes discontinues à des fins d'explication).

Les changements apportés au schéma initial $V_0$ ont créé trois versions de schéma : $V_1$, $V_2$ et $V_3$. Chaque version du schéma reflète l'évolution du modèle en fonction des besoins d'analyse. Ces évolutions permettent de répondre à de nouveaux cas d'utilisation et de mieux représenter les données pour des analyses spécifiques.

### 3.2.2 Exemple d'évolution de données

Considérons également les tableaux 3.1, 3.2, 3.3 et 3.4 représentant un exemple d'instances de données correspondant au schéma représenté dans la Figure 3.1. Notons que les instances de données des dimensions dans cet exemple sont représentées en utilisant la technique de Type 2 citée dans la sous-section 2.2.4 qui consiste à ajouter une date de début From_Date (FD) et une date de fin To_Date (TD) pour délimiter la durée de validité de chaque valeur de l'attribut ou des liens dans les hiérarchies.

En général, parmi les techniques citées dans la sous-section 2.2.4, celle de type 2 est considérée comme la plus représentative du monde réel et offre des analyses cohérentes lorsqu'il s'agit de gérer les changements dans les dimensions d'un entrepôt de données. Cette méthode consiste à ajouter une nouvelle ligne avec la nouvelle valeur tout en tenant compte de l'intervalle de temps de la véracité de ces données.

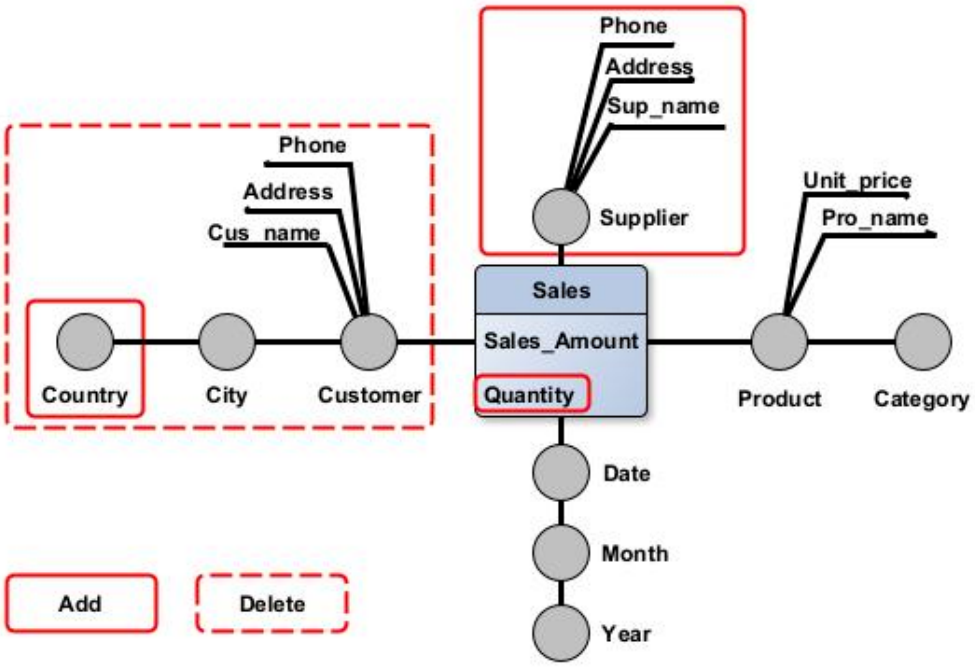



FIGURE 3.2 – Version de schéma $V_3$ à l'instant $t_3$

En revanche, les autres techniques présentent certaines limites :

La technique de type 0, qui consiste à maintenir la valeur originale, peut entraîner des pertes d'informations importantes, car les données postérieures sont écrasées et ne sont plus accessibles pour des analyses sur les données historiques.

La technique de type 1, qui consiste à remplacer la valeur originale par la nouvelle valeur, peut également entraîner des pertes d'informations, car les données antérieures sont perdues et ne peuvent pas être récupérées.

La technique de type 3, qui consiste à ajouter une nouvelle colonne pour stocker la valeur antérieure, est utile pour conserver les informations historiques, mais cela peut entraîner des modifications fréquentes du schéma, ce qui rend la maintenance complexe et peut ne pas être adapté à une évolution continue.

Enfin, la technique d'ajout d'une mini-dimension est destinée à être utilisée dans un cas particulier et ne constitue pas une solution générale au problème des changements de dimensions. Le choix de la technique appropriée dépend des besoins spécifiques du projet et des caractéristiques de l'ensemble de données.

Les instances de la dimension CUSTOMER présentées dans le tableau 3.1 indiquent que la cliente *Mary Saveley* vit dans la ville de *Paris* depuis le *01/01/2020*. De même, les instances de la dimension PRODUCT présentées dans le tableau 3.2 indiquent que le

produit *Mozzarella* est classé dans la catégorie *Fresh* depuis le *01/01/2020*. Le tableau 3.3 représente un exemple d'instances des niveaux de hiérarchie City et Category. De plus, le tableau 3.4 présente un exemple d'instances du fait Sales, qui décrit la mesure *Sales_Amount* pour la cliente *Mary Saveley* et le produit *Mozzarella*.

Table 3.1 – La dimension Customer

| Surrogate_Key | Customer_ID | Cust_Name | City_ID | From_Date | To_Date |
|---|---|---|---|---|---|
| SkCust001 | Cust001 | Mary Saveley | City001 | 01/01/2020 | 31/12/9999 |

Table 3.2 – La dimension Product

| Surrogate_Key | Product_ID | Pro_Name | Category_ID | From_Date | To_Date |
|---|---|---|---|---|---|
| SkProd001 | Prod001 | Mozzarella | Categ001 | 01/01/2020 | 31/12/9999 |

Table 3.3 – Niveaux City et Category

| City_ID | City_Name |
|---|---|
| City001 | Paris |
| City002 | Lyon |

| Category_ID | Category_Name |
|---|---|
| Categ001 | Fresh |
| Categ002 | Dairy |

Table 3.4 – le fait Sales

| Sale_ID | Customer_ID | Product_ID | Order_Date | Sales_Amount |
|---|---|---|---|---|
| Sale001 | Cust001 | Prod001 | 01/03/2020 | 1500 |
| Sale002 | Cust001 | Prod001 | 01/05/2020 | 2200 |
| Sale003 | Cust001 | Prod001 | 01/09/2020 | 1800 |
| Sale004 | Cust001 | Prod001 | 01/11/2020 | 2000 |

Supposons que la cliente *Mary Saveley*, qui vivait dans la ville de *Paris* depuis le *01/01/2020*, a déménagé à *Lyon* le *01/08/2020*. Les instances de la dimension Customer présentées dans le tableau 3.5 indiquent les deux périodes de temps pendant lesquelles la cliente est associée à une ville, avec les intervalles de validité correspondants. De même, le produit *Mozzarella*, qui était classé dans la catégorie *Fresh* depuis le *01/01/2020*, a été reclassé dans la catégorie *Dairy* le *01/04/2020*. Les instances de la dimension Product présentées dans le tableau 3.6 indiquent les deux périodes de temps pendant lesquelles le produit est associé à une catégorie différente, avec les intervalles de validité correspondants.

Table 3.5 – La dimension Customer

| Surrogate_Key | Customer_ID | Cust_name | City_ID | From_Date | To_Date |
|---|---|---|---|---|---|
| SkCust001 | Cust001 | Mary Saveley | City001 | 01/01/2020 | 31/07/2020 |
| SkCust002 | Cust001 | Mary Saveley | City002 | 01/08/2020 | 31/12/9999 |

Table 3.6 – La dimension Product

| Surrogate_Key | Product_ID | Product_Name | Categ_ID | From_Date | To_Date |
|---|---|---|---|---|---|
| SkProd001 | Prod001 | Mozzarella | Categ001 | 01/01/2020 | 31/03/2020 |
| SkProd002 | Prod001 | Mozzarella | Categ002 | 01/04/2020 | 31/12/9999 |

Le tableau 3.4 présente quelques instances du fait Sales, décrivant le *Sales_Amount* pour la cliente *Mary Saveley* et le produit *Mozzarella*. Ces montants représentent les achats de la cliente à la fois lorsqu'elle vivait à *Paris* et lorsqu'elle vivait à *Lyon*. De plus, ces montants représentent également les ventes du produit *Mozzarella* à la fois lorsqu'il était affilié à la catégorie *Fresh* et lorsqu'il était affilié à la catégorie *Dairy*. Ces changements dans les dimensions nécessitent un traitement temporel des requêtes afin d'éviter toute incohérence dans les résultats.

La date *31/12/9999* est souvent utilisée pour représenter une date indéfinie ou une date maximale dans certains systèmes. Dans le contexte des relations temporelles, cette date peut être utilisée pour indiquer que la relation est encore valide et qu'il n'y a pas de date de fin spécifiée. Ainsi, lorsque la date de validité d'une relation est fixée à *31/12/9999*, cela signifie que la relation est considérée comme toujours valide et qu'elle n'a pas encore expiré.

A travers ces deux exemples d'évolution et compte tenu des contraintes conceptuelles et physiques liées à l'évolution du modèle multidimensionnel, il est clair que le modèle multidimensionnel classique, basé sur le schéma en étoile, ainsi que ses variantes, soit limité en matière de changement et son évolution s'avère assez complexe. Ces limitations sont liées au modèle en étoile fixe, faisant de l'entrepôt un silo de données, dont l'architecture est adaptée à des besoins d'analyse connus au préalable.

En effet, l'évolution en terme de schéma implique que chacune de ses versions est valide durant une période de temps bien précise et nécessite l'identification de cette période de validation, ainsi que les instances correspondantes. De même, ces évolutions de données représentées par ces changements de dimensions nécessitent un traitement temporel des requêtes afin d'éviter toute discordance dans les résultats. C'est dans ce contexte que nous avons proposé un modèle multidimensionnel agile que nous détaillons dans ce qui suit.

## 3.3 Architecture de GAMM

GAMM est une approche flexible pour l'évolution des schémas et des données dans les entrepôts de données. Ce modèle permet aux concepteurs d'intégrer de nouvelles sources de données et de répondre aux nouvelles attentes des utilisateurs, afin d'enrichir les capacités analytiques de l'entrepôt. L'approche repose sur un modèle évolutif à plusieurs versions et sur un entrepôt de données stocké dans une base de données orientée graphe unique et globale, comme illustré dans la Figure 3.3.

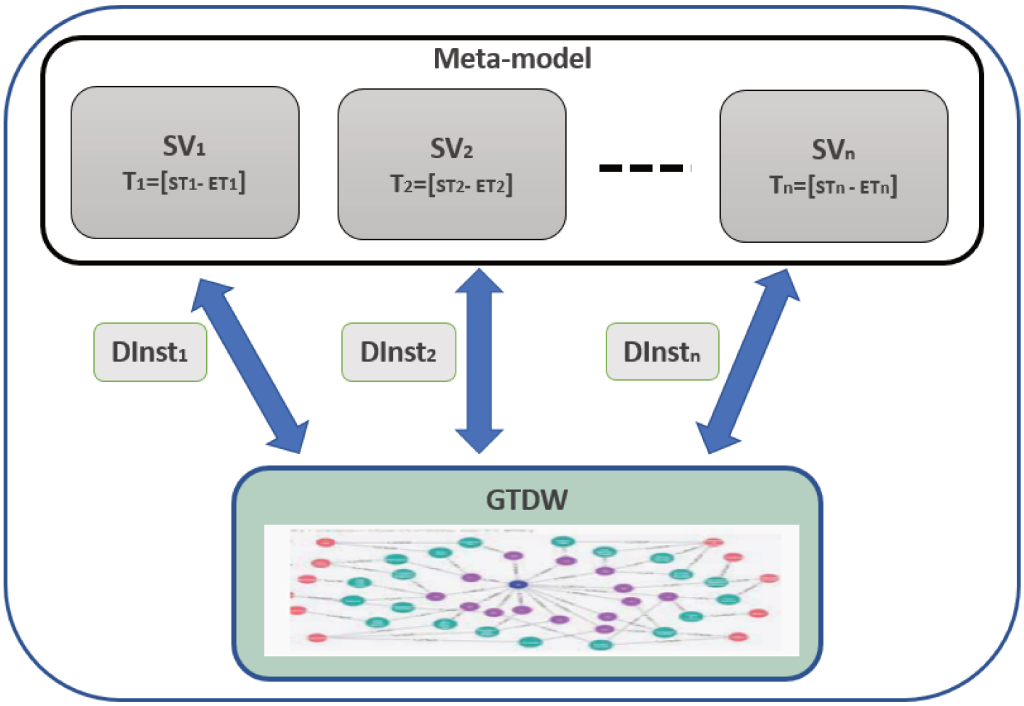



FIGURE 3.3 – Architecture de GAMM

Afin de surmonter les contraintes engendrées par l'utilisation d'un modèle Entité-Relation ER, la représentation en graphe a été adoptée pour offrir une plus grande flexibilité au modèle, notamment en termes d'évolution Akoka et al. (2021). En effet, l'utilisation de graphes présente plusieurs avantages, tels que la représentation de données inter-connectées de manière précise et l'utilisation de relations porteuses d'informations, notamment pour la notion de temporalité dans les liens d'agrégation. De plus, les graphes ont été privilégiés en raison de l'absence de contraintes d'intégrité, de l'absence de schéma préétabli et de leur capacité à représenter chaque valeur d'un n-uplet (ou l'ensemble des n-uplets) à l'aide d'un nœud du graphe.

Cette approche diffère de celle basée sur les tables relationnelles, qui sont des schémas d'ordre tabulaire composés de lignes horizontales et de colonnes verticales, où l'ajout ou la suppression d'un élément affecte l'ensemble de la structure.

Pour permettre une évolution du schéma en multi-versions, chaque version de schéma, notée SV, est valide pendant une période de temps T caractérisée par un temps de début ST et un temps de fin ET. Chaque version correspond à une instance de données (DInst) extraite de l'entrepôt de données temporelles basé sur les graphes (*Graph Temporal Data Warehouse*) (GTDW) en utilisant le temps de chargement des données dans la version, représenté par le paramètre temporel TT (voir sous-section 2.2.2), introduit au niveau des nœuds de type Fait. Un méta-modèle est mis en place pour la gestion des versions de schéma, dont les périodes de validité sont séquentiellement ordonnées.

Afin de conserver l'historique évolutif des données, y compris les changements au niveau des dimensions, et conformément aux principes des bases de données temporelles détaillés dans la sous-section 2.2.2, nous avons appliqué un étiquetage temporel de type VT aux nœuds de type *Fait*, ainsi qu'un étiquetage temporel de type FD et TD au niveau des relations dans les instances des dimensions. Cela permet d'identifier la période de validité de ces relations. Grâce à ces étiquetages temporels, nous sommes en mesure de garantir des analyses cohérentes et des résultats concordants, y compris en présence de changements dans les données.

Rappelons que dans un horodatage ponctuel, Valid_Time (VT) représente le moment où une donnée est vraie par rapport à la réalité, tandis que Transaction_Time (TT) représente le moment où cette donnée est stockée dans une base de données. D'autre part, dans un horodatage à intervalles de temps, From_Date (FD) représente le temps de début de la période de validité de la donnée, tandis que To_Date (TD) représente le temps de fin de cette période temporelle de validité. Ces étiquetages temporels sont essentiels pour comprendre la validité et l'évolution des données dans le temps (voir sous-section 2.2.2).

L'étiquetage que nous utilisons permettra d'identifier les différentes versions du schéma et les instances de données correspondant à chaque version. Grâce à cela, nous serons en mesure d'effectuer des analyses homogènes et d'obtenir des résultats cohérents, même en présence de changements dans les données notamment pour ceux dans les dimensions. Dans le chapitre consacré aux requêtes dans le modèle GAMM, nous fournirons plus de détails sur l'utilisation de cet étiquetage temporel (voir le chapitre 5).

## 3.4 Méta-modèle de GAMM

Nous avons développé un méta-modèle qui permet d'identifier toutes les structures de schémas existantes dans le modèle afin de gérer les multiples versions de schéma résultant de l'évolution du GAMM. Ce méta-modèle repose sur une classe appelée *Version*, qui comprend l'identification de la version, une date de début ST et une date de fin ET pour spécifier la durée de validité, ainsi que l'état de chaque version. Pour illustrer cette structure, nous présentons un diagramme du méta-modèle dans la VOIR FIGURE 3.4 ci-aprés. Nous avons fait le choix de ne pas représenter les noms des relations afin de ne pas encombrer le schéma.

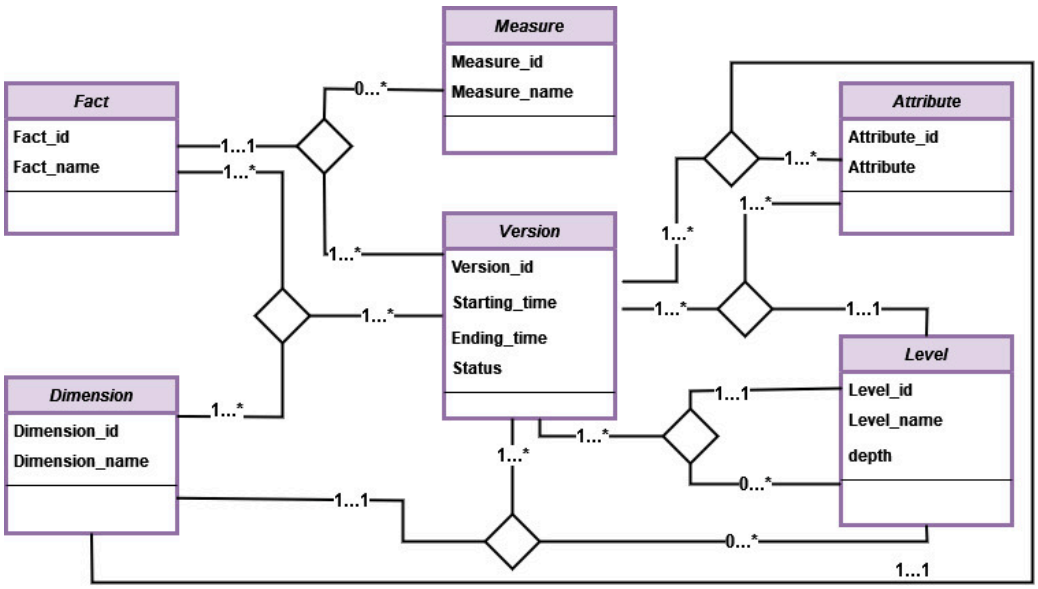



FIGURE 3.4 – Méta-modèle de GAMM

La classe *Version* joue un rôle central dans l'organisation de chaque version de schéma, agissant comme un pivot autour duquel toutes les autres classes sont liées. Le diagramme du méta-modèle définit de manière efficace les faits, les mesures et les dimensions associés à chaque version de schéma.

De plus, le diagramme indique les attributs liés aux dimensions/niveaux, l'ordre dans lequel ces niveaux sont établis, ainsi que les niveaux de hiérarchie liés aux dimensions à un moment donné. L'extraction du schéma de chaque version à partir du méta-modèle est réalisée à l'aide d'un algorithme et de requêtes spécifiques qui seront présentés dans le chapitre 5 relatif à l'interrogation des versions et des données dans le modèle GAMM.

Ce méta-modèle offre donc une structure claire et cohérente pour la gestion des versions de schéma dans le GAMM, permettant une gestion efficace des évolutions et des changements dans le modèle multidimensionnel.

## 3.5 Formalisation de GAMM

Le formalisme du modèle GAMM décrit celui-ci en soulignant la souplesse qui permet une évolution du modèle à la fois au niveau du schéma et des instances de données, tout en préservant l'historique de ces changements. Sur le plan conceptuel, cette approche étend la modélisation multidimensionnelle traditionnelle utilisant les concepts de *fait, mesure, dimension, niveau* et *hiérarchie*. Pour prendre en compte l'évolution chronologique du schéma et des données dans le temps, ainsi que dans le but d'assurer une historisation correcte, un étiquetage temporel a été introduit conformément aux définitions suivantes :

**Definition 1 :** GAMM est représenté comme suit :

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

$GAMM(t)$ représente la version du schéma à un instant $t$.
$t \in T = [ST, ET]$ représente la période de validité de la version du schéma, où $ST$ est le date de début de la version (*Starting_Time*) et $ET$ est le la date de la fin de la version (*Ending_Time*).
$[F] = \{f_i(t)\}, i \in [1, *]$, représente l'ensemble des faits à un instant $t$.
$f_i(t)$ représente le fait $f_i$ à un instant $t$.
$[D] = \{d_j(t)\}, j \in [1, *]$, représente l'ensemble des dimensions selon lesquelles l'ensemble des faits $\{f_i(t)\}$ sont analysables à un instant $t$.
$d_j(t)$ représente la dimension $d_j$ à un instant $t$.
$FAssoc[F, D](t) : f_i(t) \Longrightarrow \{d_j(t), ST, ET\}$, où $j \in [1, *]$, représente la fonction d'association de l'ensemble des dimensions $\{d_j(t)\}$ à l'ensemble des faits $\{f_i(t)\}$ à l'instant $t$.
$[H] = \{h_k(t)\}, k \in [0, *]$, représente l'ensemble des hiérarchies constituant les axes d'analyse à un instant $t$.
$h_k(t)$ représente la hiérarchie $h_k$ à un instant $t$.

Exemple :

$$GAMM(t_0) = \{\{\text{Sales}\}, \{\text{Customer}, \text{Product}\}, \{\text{Sales} \Longrightarrow \text{Customer}, \text{Product}\},$$

$$\{\text{Customer—City}, \text{Product—Gategory}\}, ST_0, ET_0\}$$

Avec $ST_0 =< t_0 < ET_0$, $ST_0 = 01/01/2020$ et $ET_0 = 31/12/2020$.

**Definition 2 :** Une mesure est un indicateur permettant d'analyser un sujet représentant un fait métier ; peut être agrégée et se définit comme suit :

$$M(t) = \{M\_label, I_k^m\}.$$

*M_label* représente le nom de la mesure.
$I_k^m$, $k \in [1, *]$, représente l'ensemble des instances de la mesure $M$.

Exemple :

$$M(t_o) = \{\text{Sales\_Amount}, I_0^m : \{1500\}\}.$$

**Definition 3 :** Un fait représente un sujet analysé par GAMM. Il est défini comme suit :

$$F(t) = \{F\_label, [M], MAssoc[F, M](t), [I^f]\}.$$

*F_label* représente le nom du fait.
$[M] = \{m_k(t)\}$, $k \in [1, *]$, représente l'ensemble des mesures associées au fait à un instant $t$.
$m_k(t)$ représente une mesure $m_k$ à un instant $t$.
$MAssoc[F, M](t) : f_i(t) \Longrightarrow \{m_k(t), ST, ET\}$ représente une fonction d'association de l'ensemble des mesures $\{m_k(t)\}$ au fait $f_i(t)$ à un instant $t$, où $ST$ est le *Starting_Time*

et $ET$ est le $Ending\_Time$ de la version.
$[I^f] = \{i_l^f\}$, $l \in [1, *]$ représente l'ensemble des instances du fait $f$. Chaque instance $i^f = \{[I_k^m], VT, TT\}$.
$VT = Valid\_Time$ représente le moment où une instance de fait est vraie par rapport à la réalité.
$TT = Transaction\_Time$ représente le moment où une instance de fait est stockée dans le modèle.

Exemple :

$$F(t_o) = \{\text{SALES}, M_o, MAssoc[F, M](t_o), i_0^f\}.$$

$M_o$ : {SALES_AMOUNT}.
$MAssoc[F, M](t_o)$ : {SALES $\Longrightarrow$ SALES_AMOUNT, 01/01/2020, 31/12/2020}.
$i_0^f$ : {$I_0^m$ : {1500}, 05/01/2020, 15/06/2020}.

**Représentation graphique d'un nœud de type fait :**

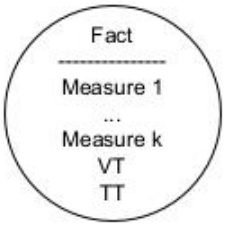



FIGURE 3.5 – nœud de type Fait

Exemple :


FIGURE 3.6 – Exemple d'instanciation d'un nœud de type Fait

**Definition 4 :** Un attribut est un élément de description d'une dimension ou d'un niveau de hiérarchie auquel il est associé. Il est défini comme suit :

$$A(t) = \{A\_Label, Attribute, [I^a]\}.$$

*A_Label* représente le nom du nœud attribut.
*Attribute* représente le nom de l'attribut de description.
$[I^a] = \{i_k^a\}$, $k \in [1, *]$, représente l'ensemble des instances de l'attribut $A$. Chaque instance $i^a = \{\text{valeur de l'attribut}\}$.

Exemple :

$$A(t_0) = \{Product\_Name, name, i_0^a : \{Mozzarella\}\}.$$

**Definition 5 :** Un niveau représente le degré de détails d'une perspective d'analyse selon une hiérarchie donnée. Il est défini comme suit :

$$L(t) = \{L\_Label, L\_ID, [A], [I^l], LAssoc[L, A](t), Rel[[I^l], [I^a]]\}.$$

*L_Label* représente le nom du nœud niveau.
$L\_ID$ représente l'identifiant du niveau.
$[A] = \{a_i(t)\}, i \in [1, *]$, représente l'ensemble des attributs associés au niveau à un instant $t$.
$[I^l] = \{i_j^l\}$, $j \in [1, *]$, représente l'ensemble des instances du niveau $L$. Chaque instance $i^l = \{level\_id\}$.
$LAssoc[L, A](t) : L_j(t) \Longrightarrow \{a_i(t), ST, ET\}$ représente la fonction d'association de l'ensemble des attributs $\{a_i(t)\}$ et le niveau $L_j(t)$ à un instant $t$, où $ST$ est la date de début (*Starting_Time*), $ET$ est la date de fin *Ending_Time* et $j \in [0, *]$.
$Rel[[I^l], [I^a]]$ représente les relations entre les instances des niveaux [L] et les attributs [A], où $Rel[[i_m^l], [i_n^a]] = \{$Relation_Label, From_Date, To_Date$\}$.
*Relation_Label* représente le nom de la relation.
$From\_Date$ représente la date de début de la relation.
$To\_Date$ représente la date de fin de la relation.
$[I^a] = \{i_k^a\}$, $k \in [1, *]$, représente l'ensemble des instances de l'attribut $A$, associés au niveau $L$ à l'instant $t$.

Exemple :

$$L(t_0) = \{\textsc{Category}, Category\_Id, A_0, i_0^l, i_0^a, LAssoc[L, A_0](t_0), Rel[[I_0^l], [I_0^a]]\}.$$

$A_0 : \{Category\_Name\}$.
$i_0^l : \{Categ001\}$
$i_0^a : \{Fresh\}$.
$LAssoc[L, A_0](t_0) : \{\textsc{Category} \Longrightarrow Category\_Name, 01/01/2020, 31/12/2020\}$.
$Rel[[I_0^l], [I_0^a]] = \{Category\_To\_Name, 05/01/2020, 31/12/9999\}$

**Definition 6 :** Une dimension est un axe d'analyse selon lequel un sujet est analysé. Elle détermine le niveau de détails des mesures et se définit comme suit :

$$D(t) = \{D\_Label, D\_ID, [A], [I^d], [I^a], DAssoc_a[D, A](t), Rel[[I^d], [I^a]]\}.$$

$[I^a] = \{i_k^a\}$, $k \in [1, *]$, représente l'ensemble des instances de l'attribut $A$, associés à une dimension $d_j(t)$ à l'instant $t$.

*D_Label* représente le nom du nœud dimension.
*D_ID* représente l'identifiant de la dimension.
$[A] = \{a_i(t)\}, i \in [1, *]$, représente l'ensemble des attributs associés à la dimension à un instant $t$.
$[I^d] = \{i_j^d\}$, $j \in [1, *]$, représente l'ensemble des instances de la dimension $D$. Chaque instance $i^d = \{id\}$.
$[I^a] = \{i_k^a\}$, $k \in [1, *]$, représente l'ensemble des instances de l'attribut $A$, associés à la dimension $D$ à l'instant $t$.
$DAssoc_a[D, A](t) : d_j(t) \Longrightarrow \{a_i(t), ST, ET\}$ représente la fonction d'association de l'ensemble des attributs $\{a_i(t)\}$ et de la dimension $d_j(t)$ à un instant $t$, où $ST$ est la date de début (*Starting_Time*) et $ET$ est la date de fin *Ending_Time*.
$Rel[[I^d], [I^a]]$ représente les relations entre les instances de la dimension D et les attributs [A], où $Rel[[i_m^d], [i_n^a]]$ = {Relation_Label, From_Date, To_Date}.
*Relation_Label* représente le nom de la relation.
*From_Date* représente la date de début de la relation.
*To_Date* représente la date de fin de la relation.

Exemple :

$D(t_0) = \{\text{PRODUCT}, Product\_ID, A_0, i_0^d, i_0^a, DAssoc_a[D, A](t_0), Rel[[I_0^d], [I_0^a]]\}$.

$A_0 : \{Product\_Name\}$.
$i_0^d : \{Prod001\}$
$i_0^a : \{Mozzarella\}$
$DAssoc[D, A](t_o) : \{\text{PRODUCT} \Longrightarrow Product\_Name, 01/01/2020, 31/12/2020\}$.
$Rel[[I_0^d], [I_0^a]] = \{Product\_To\_Name, 05/01/2020, 31/12/9999\}$

**Représentation graphique de nœuds de type dimension-attribut/niveau-attribut :**

Exemple :

**Definition 7 :** Une hiérarchie est une projection de l'analyse par niveau (de granularité) selon l'axe défini par la dimension. Elle est organisée du niveau de granularité le plus fin au niveau de granularité le plus gros, offrant ainsi des possibilités d'analyse pour les regroupements ascendants grâce à l'opérateur *roll-up* et les regroupements descendants grâce à l'opérateur *drill-down* ou autres. La hiérarchie est définie comme suit :

$$H(t) = \{D, [L], R_h[D/L_j, L_k](t), Rel[[I^{d/l_j}], [I^{l_k}]]\}.$$

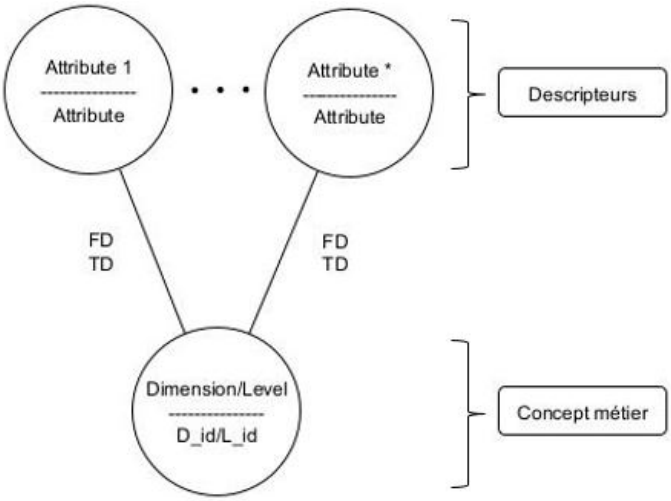



FIGURE 3.7 – nœuds de type dimension/niveau-attribut

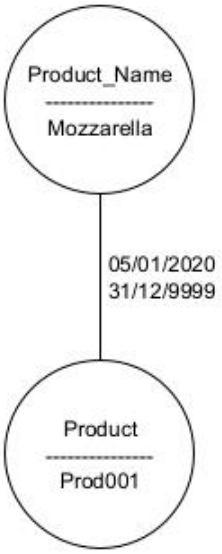



FIGURE 3.8 – Exemple d'instanciation de nœuds de type dimension-attribut

$D$ : représente la dimension à laquelle appartient la hiérarchie.
$[L] = \{l_i(t)\}, i \in [1, *]$, représente l'ensemble des niveaux constituant la hiérarchie H(t) à un instant $t$.
$R_h[D/L_j, L_k](t) : d(t)/l_j(t) \implies \{l_k(t), ST, ET\}$ représente la fonction d'agrégation entre la dimension $d_i(t)$ et les différents niveaux $\{l(t)\}$ constituant la hiérarchie $H(t)$ à un instant $t$, où $j \in [1, *], k \in [1, *]$ et $j \neq k$, $ST$ est la date de début (*Starting_Time*), $ET$ est la date de fin *Ending_Time* de la version.
$Rel[[I^{d/l_j}], [I^{l_k}]]$ représente les relations entre les instances de la dimension D et des niveaux [L], où $Rel[[i_m^{d/l}], [i_n^l]]$ = {Relation_Label, From_Date, To_Date}.
*Relation_Label* représente le nom de la relation.
*From_Date* représente la date de début de la relation.

*To_Date* représente la date de fin de la relation.

Considérons que $D_i(t) \in D$ avec $i \in [1, *]$, $\forall\ H_j(t) \in H$ où $j \in [1, *]$, et $L_1^{h_j}(t)$ est directement liée à $D_i(t)$ de sorte que :

$$D_i(t) \prec L_1^{h_j}(t) \prec L_2^{h_j}(t) \prec ... \prec L_k^{h_j}(t) \prec All.$$

$D_i(t)$ représente le niveau d'agrégation le plus fin de l'axe d'analyse.
$L_k^{h_j}(t)$ représente le niveau d'agrégation le plus fin de la hiérarchie $h_j$.
*All* représente le niveau d'agrégation global de la dimension.

Exemple : Projection de l'analyse par COUNTRY et CITY pour la dimension CUSTOMER peut être effectuée comme suit :

$$D(t) = (\text{CUSTOMER}) \prec L_1^{h_j}(t) = (\text{CITY}) \prec L_2^{h_j}(t) = (\text{COUNTRY}) \prec All.$$

**Représentation graphique des nœuds de Hiérarchies :**

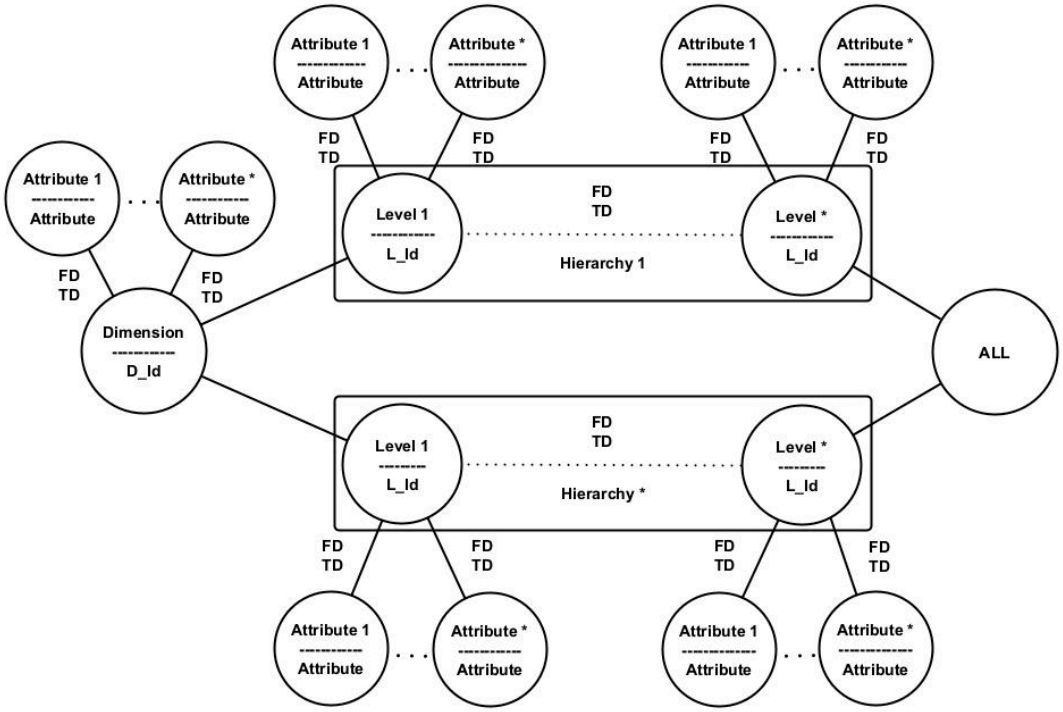



FIGURE 3.9 – Nœuds de Hiérarchies

Exemple :

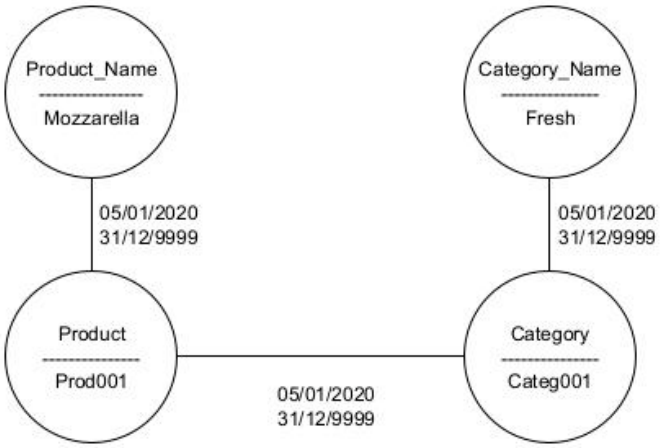



FIGURE 3.10 – Exemple d'instanciation de nœuds de Hiérarchie

## 3.6 Entrepôt de données temporelles en graphes

GTDW est un entrepôt de données temporelles basé sur les graphes. Sa particularité réside dans la séparation des concepts métier de leurs attributs de description. Les faits, les dimensions et les niveaux d'hiérarchies sont représentés par des nœuds correspondant aux concepts métiers. Les nœuds représentant les descripteurs sont liés aux dimensions/niveaux, permettant ainsi à chaque entité d'avoir une évolution indépendante. GTDW est implémenté dans une base de données orientée graphes de type NoSQL, en se basant sur la formalisation du modèle GAMM. Les données sont présentées sous forme d'un graphe de sommets et d'arêtes, en utilisant des pointeurs physiques entre les nœuds. Cette approche évite les jointures dans les requêtes, notamment dans un contexte de big data. De plus, les bases de données orientées graphes se caractérisent par l'absence de type de données et offrent la possibilité d'intégrer des informations dans les relations entre les données.

Un étiquetage temporel de type horodatage ponctuel a été appliqué au niveau des instances des nœuds représentant les faits, tandis qu'un étiquetage temporel de type intervalle de temps a été utilisé au niveau des instances des relations dans les axes de dimensions (dimensions/niveaux/attributs). Cette représentation nous permet de manipuler et de filtrer les données temporelles de manière précise, en particulier pour la navigation dans les données à travers des opérations OLAP. Ainsi, nous pouvons obtenir des résultats précis en fonction des intervalles de temps spécifiés lors de l'exploration des données.

Ainsi, des paramètres temporels de type horodatage ponctuel (VT et TT) ont été associés à tous les nœuds de type faits afin de permettre l'identification chronologique des différentes instances. De plus, des paramètres temporels de type intervalle de temps (FD/TD) ont été attribués à toutes les relations entre les nœuds représentant les dimensions/niveaux/attributs pour déterminer la période de validité de ces relations. Grâce à cette approche, GTDW peut traiter les requêtes temporelles de manière co-

hérente et fournir des résultats concordants, même en présence de changements dans les dimensions.

Conformément au formalisme présenté dans la section 3.5 et aux caractéristiques du GTDW, nous avons établi les règles de transformation présentées dans la sous-section suivante.

### 3.6.1 Règles de transformation

Les faits, les concepts métiers du modèle multidimensionnel, ainsi que les attributs de description seront représentés par des nœuds, tandis que les relations entre ces entités seront représentées par des arêtes. Les règles de transformation d'un modèle multidimensionnel classique en un modèle multidimensionnel temporel basé sur les graphes sont définies comme suit :

1. Chaque n-uplet d'un fait est représenté par son propre nœud.
2. Chaque entité (concept métier/descripteur) d'une dimension, d'un niveau ou d'un attribut est représentée par son propre nœud.
3. Toutes les relations sont représentées par des arêtes.
4. Les nœuds de faits contiennent les mesures.
5. Les nœuds de faits sont directement liés aux nœuds de dimensions.
6. Les nœuds de niveaux sont liés à une dimension ou à un autre nœud de niveau en fonction de leur profondeur.
7. Les nœuds d'attributs sont directement liés aux nœuds de dimensions/niveaux.
8. Les niveaux forment des hiérarchies selon des axes d'analyse organisés du niveau d'agrégation le plus fin au plus gros.
9. Toutes les arêtes entre les valeurs (concept métier/descripteur) d'une dimension, d'un niveau ou d'un attribut sont étiquetées chronologiquement avec FD et TD pour déterminer la durée de vie de la relation.
10. Tous les nœuds de type fait sont étiquetés chronologiquement avec VT et TT conformément aux principes des bases de données temporelles

En appliquant les règles établies sur le schéma de notre exemple de motivation (voir Figure 3.2), nous obtenons le schéma représenté dans la Figure 3.11. Les nœuds représentant les concepts métiers sont séparés des nœuds de description (attributs) afin de permettre une évolution indépendante de chaque entité.

Le nœud Sales dans la Figure 3.11 contient la mesure *Sales_Amount* et représente les nœuds de type fait. Les nœuds Product, Customer et Date représentent les nœuds de type dimensions, tandis que les nœuds Category (pour Product) et City (pour Customer) représentent les nœuds de type niveaux de hiérarchie. Les dimensions et les niveaux sont décrits par des nœuds représentant les attributs.

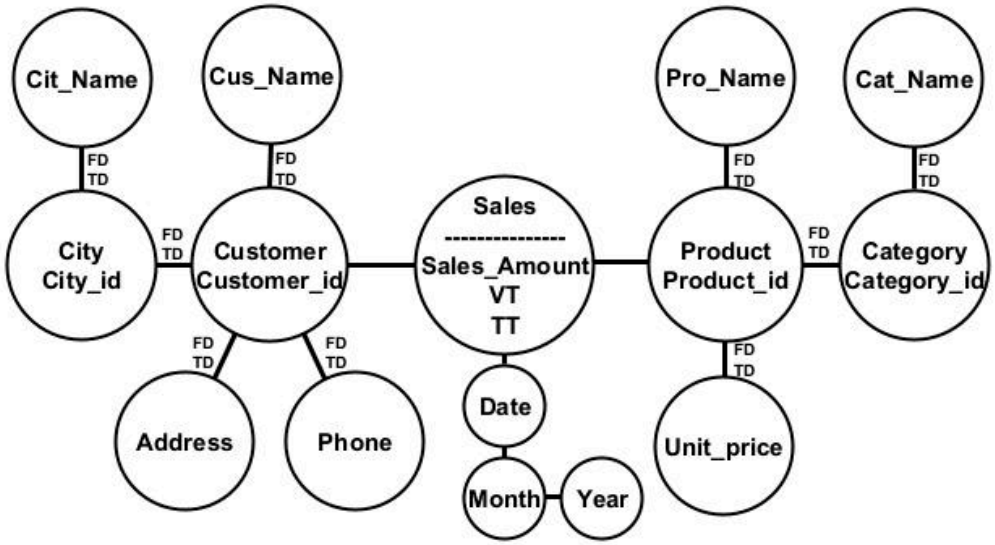



FIGURE 3.11 – Schéma logique de GTDW

## 3.7 Conclusion

Dans ce chapitre, nous avons présenté un cadre formel et une représentation en graphes du modèle GAMM. Notre approche propose une solution, tant au niveau conceptuel que logique, pour résoudre les problèmes liés à l'évolution des modèles multidimensionnels classiques. Le modèle GAMM repose sur le principe d'un schéma évolutif en multi-version et sur un entrepôt de données stocké dans une base de données orientée graphe unique et globale (modèle physique), ce qui permet de se dispenser des contraintes des bases de données relationnelles.

Le modèle GAMM s'appuie sur un méta-modèle afin de faciliter la gestion et l'identification des différentes versions générées par les évolutions. De plus, un étiquetage temporel a été appliqué aux instances de données ainsi qu'aux relations entre ces instances ; ceci garantit la réalisation d'analyses cohérentes et de résultats concordants.

En adoptant GAMM, il est alors possible de bénéficier d'une approche agile pour l'évolution des modèles multidimensionnels, tout en maintenant la cohérence et la qualité des analyses de données.

Le formalisme a été illustré à l'aide d'exemples graphiques dans le but de faciliter la compréhension du modèle GAMM. Des règles de transformation ont été proposées pour passer d'un modèle multidimensionnel classique au modèle GAMM.

Dans le chapitre suivant (voir chapitre 4), nous approfondirons les différents processus que nous avons développés pour permettre l'évolution du schéma et des contenus dans le modèle GAMM. Ces processus permettront d'ajouter, de modifier ou de supprimer des dimensions, des niveaux, des attributs et des relations dans le modèle, tout en préservant l'intégrité des données et en assurant une transition fluide entre les dif-

férentes versions. Nous aborderons également les techniques de gestion des versions et les mécanismes de suivi des changements dans le modèle GAMM.

CHAPITRE 4

# Évolution de schémas ou de données dans le modèle GAMM

## Sommaire

## 4.1 Introduction

Ce chapitre se focalise sur l'évolution temporelle des schémas et des données dans le modèle GAMM, mettant en évidence sa flexibilité pour répondre aux nouveaux besoins d'analyse. Nous examinerons les fonctions et les processus mis en place pour faciliter ces évolutions, et illustrerons ces concepts à travers plusieurs exemples.

Dans la première partie, nous examinerons comment le schéma du modèle GAMM peut évoluer pour répondre à de nouveaux besoins d'analyse et pour gérer de nouvelles sources de données requises. Cette évolution du schéma est réalisée à travers un processus multi-versions, où chaque version représente l'état du schéma pendant une période de temps.

Nous présenterons les opérateurs d'évolution de schéma dans GAMM. Ces opérateurs permettent l'ajout ou la suppression de faits, de mesures, de dimensions, de niveaux ou d'attributs, ainsi que l'ajout ou la suppression de relations entre les entités. En fonction de l'évolution, nous pouvons mettre à jour le schéma de manière itérative pour répondre aux nouveaux besoins d'analyse. Cela garantit que le schéma évolue de manière adaptée tout en préservant la cohérence et l'intégrité des données.

Dans la seconde partie de ce chapitre, nous détaillerons les différents processus d'évolution des données. Ces processus assurent l'étiquetage temporel des nœuds ainsi que des relations, conformément aux principes des bases de données temporelles.

Des mécanismes appropriés sont mis en place pour gérer ces processus d'évolution des données tout en maintenant l'intégrité et la cohérence de l'entrepôt de données. Différentes fonctions ont été définies pour gérer l'évolution temporelle des instances de données, telles que le chargement initial, le rafraîchissement et la modification des données, permettant ainsi une évolution temporelle, incrémentale et sans redondance des données, tout en conservant l'historique des évolutions.

## 4.2 Évolution de schéma dans GAMM

Le schéma du modèle GAMM est conçu pour évoluer afin de répondre à de nouveaux besoins d'analyse et pour gérer les nouvelles sources de données requises pour ces besoins. À chaque évolution, une nouvelle version du schéma est créée avec un ST correspondant à la date de création. Cette date est également attribuée au ET de la version précédente pour déterminer la période de validité de cette version. Ainsi, les évolutions et les modifications de schéma dans GAMM sont gérées comme un processus multi-versions, où chaque version représente l'état du schéma durant une période de temps T.

Il convient de noter que dans certains cas, l'ajout ou la suppression d'une entité ne nécessite pas nécessairement de modifier le schéma. Par exemple, l'ajout d'une mesure générée à partir de mesures existantes ou l'ajout d'un niveau de hiérarchie qui

ne modifie pas le degré de granularité des mesures et qui correspondrait à toutes les mesures existantes dans le fait. Dans de tels cas, les évolutions peuvent être effectués sans changer de version. Dans notre étude, nous considérons la création d'une nouvelle version à chaque évolution, car c'est le cas le plus général, tandis que le maintien d'une version existante est considéré comme un cas particulier.

### 4.2.1 Fonctions d'évolution de schéma dans GAMM

Nous détaillons ici les fonctions d'évolution que nous avons établies, lesquelles permettent la modification de schéma dans le modèle GAMM. Ces fonctions comprennent l'ajout de faits, de mesures, de dimensions, de niveaux ou d'attributs, la suppression d'entités, ainsi que la l'ajout ou la suppression de relations entre les entités. Les évolutions de schéma dans le modèle GAMM sont effectuées selon la fonction suivante :

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

où :

*GAMM($t_{n+1}$)* : représente le schéma après l'évolution.

*GAMM($t_n$)* : représente le schéma avant l'évolution.

*Evolution_Operation($t_n$)* : représente l'évolution à appliquer à une version donnée en fonction du type de cette évolution.

La fonction *Evolution_Operation* englobe les actions nécessaires pour mettre en œuvre une évolution spécifique dans GAMM. Cette fonction prend en entrée le schéma au temps $t_n$ et applique les changements nécessaires pour le transformer selon le schéma désiré après l'évolution.

En appliquant la fonction *Evolution_Operation* appropriée au schéma GAMM($t_n$), on obtient le schéma résultant GAMM($t_{n+1}$), qui représente la version mise à jour du schéma après l'évolution.

Les détails spécifiques et les actions effectuées par la fonction *Evolution_Operation* dépendent du type d'évolution en cours d'exécution. Il est possible d'avoir plusieurs modifications simultanées dans le schéma, telles que l'ajout d'une mesure accompagnée de la suppression d'une dimension. Dans ce cas, les différentes fonctions d'évolution appropriées seront exécutées simultanément pour mettre à jour le schéma selon les modifications souhaitées.

Chaque fonction d'évolution sera responsable des actions spécifiques requises pour la modification correspondante. Les fonctions d'évolution peuvent être conçues de manière à prendre en compte les dépendances entre les modifications et à s'assurer que les changements sont effectués de manière cohérente.

L'exécution simultanée des fonctions d'évolution permet de mettre à jour le schéma de manière globale et cohérente, en prenant en compte toutes les modifications souhaitées. Cela garantit que les différentes évolutions sont appliquées de manière synchronisée, afin d'obtenir le schéma final souhaité à l'issue de ces modifications.

Il est important de planifier et de gérer attentivement les modifications simultanées pour s'assurer de la cohérence et de l'intégrité du schéma après les évolutions. Les opérateurs d'évolution sont définis dans le TABLEAU 4.1 :

TABLE 4.1 – Opérateurs d'évolution de schéma

| **Évolution** | **Opérateur d'évolution** | **Description** |
|---|---|---|
| Ajouter un attribut | *Add_attribute(A1,D1)*, *Add_attribute(A1,L1)* | Ajouter l'attribut *A1* à la dimension *D1* ou au niveau *L1* |
| Ajouter une mesure | *Add_measure(M1,F1)* | Ajouter la mesure *M1* au fait *F1* |
| Ajouter une dimension | *Add_dimension(D1,F1, «A»)* | Ajouter la dimension *D1* au fait *F1* avec les attributs «*A*» |
| Ajouter un niveau | *Add_level(L1,D1,«A»)*, *Add_level(L1,L2,«A»)* | Ajouter le niveau *L1* à la dimension *D1* ou au niveau *L2* avec les attributs «*A*» |
| Ajouter un fait | *Add_fact(F1,«D»,«M»)* | Ajouter le fait *F1* aux dimensions «*D*» avec les mesures «*M*» |
| Supprimer une entité | *Delete_entity(e)* | Supprimer une entité *E (*fait, mesure, dimension, niveau ou attribut) |
| Créer une relation | *Add_link(e1,e2)* | Créer un lien entre l'entité *E1* et l'entité *E2* |
| Supprimer une relation | *Delete_link(e1,e2)* | Supprimer une relation entre l'entité *E1* et l'entité *E2* |

Les opérateurs d'évolution décrits dans le TABLEAU 4.1 sont utilisées pour effectuer des modifications sur le schéma dans le modèle GAMM. Ils permettent de modifier la structure et les relations entre les éléments d'un schéma pour répondre aux besoins d'analyse et de visualisation des données.

Pour la fonction *Delete_entity*, il est en effet possible d'envisager plusieurs cas en fonction du type d'entité à supprimer et des objectifs de l'utilisateur. Par exemple, pour supprimer un niveau de hiérarchie, l'utilisateur peut avoir le choix de soit supprimer les niveaux de hiérarchie de granularité inférieure, soit les maintenir dans la hiérarchie.

Dans la suite, nous allons réutiliser notre exemple de motivation pour proposer

les modifications de schéma décrites précédemment (voir sous-section 4.2), qui correspondent à trois nouveaux besoins d'analyse. Nous allons illustrer pour chacun de ces besoins l'évolution du schéma ainsi que les fonctions d'évolution appropriées pour chaque opération.

En suivant cette approche, nous pourrons mettre à jour le schéma de manière itérative en ajoutant, modifiant ou supprimant des entités selon les besoins spécifiques de chaque analyse. Cela garantira que le schéma évolue de manière adaptée pour répondre aux nouveaux besoins d'analyse tout en préservant la cohérence et l'intégrité des données.

### 4.2.2 Exemple 1 : Ajouter un nouveau niveau Country à la dimension Customer.

L'ajout d'un niveau COUNTRY au niveau CITY de la dimension CUSTOMER, représenté par le schéma de la figure 4.1, crée une nouvelle hiérarchie sur l'axe d'analyse (Customer, City, Country), répondant ainsi à un nouveau besoin d'analyse géographique plus agrégé à l'aide du niveau COUNTRY.

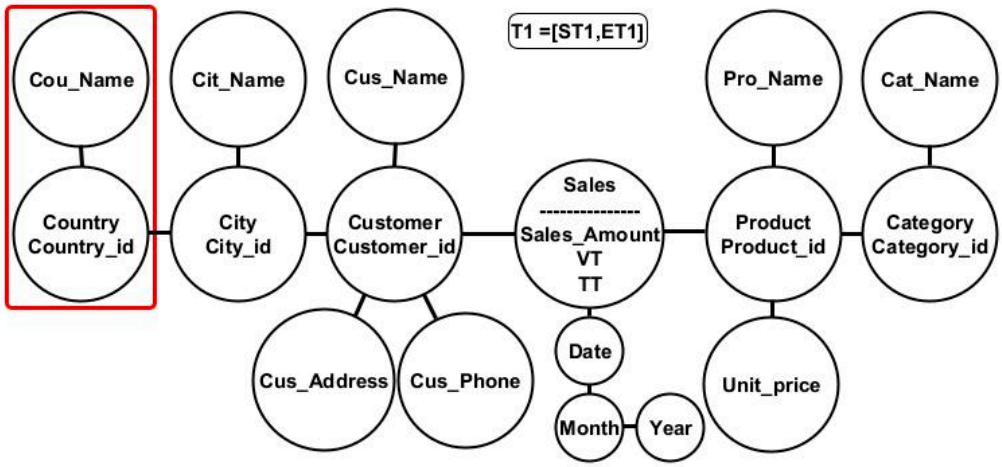



FIGURE 4.1 – Schéma logique de GAMM($t_1$)

L'ajout de cette nouvelle hiérarchie modifie le schéma du modèle en créant la nouvelle version sans nécessiter de modification du niveau factuel de SALES, ni des mesures associées à SALES puisque le niveau ajouté ne représente pas un niveau de base et ne correspond donc pas au niveau de granularité le plus fin de la dimension CUSTOMER.

La fonction d'évolution correspondant à cette opération est représentée comme suit :

$$GAMM(t_1) = GAMM(t_0) + Add\_level(\text{COUNTRY}, \text{CITY}, [\textit{COU\_NAME}]).$$

*Add_level(*Country,City*, [Cou_Name])* représente l'opérateur d'évolution permettant de rajouter un niveau Country ayant l'attribut «Cou_Name» à la dimension Customer.

### 4.2.3 Exemple 2 : Ajouter une nouvelle dimension Supplier.

L'ajout d'une nouvelle dimension Supplier avec les attributs Sup_Name, Address et Phone, représentée par le schéma de la figure 4.2, offre un nouvel axe d'analyse, répondant ainsi à un nouveau besoin d'analyse à l'aide de la dimension ajoutée.

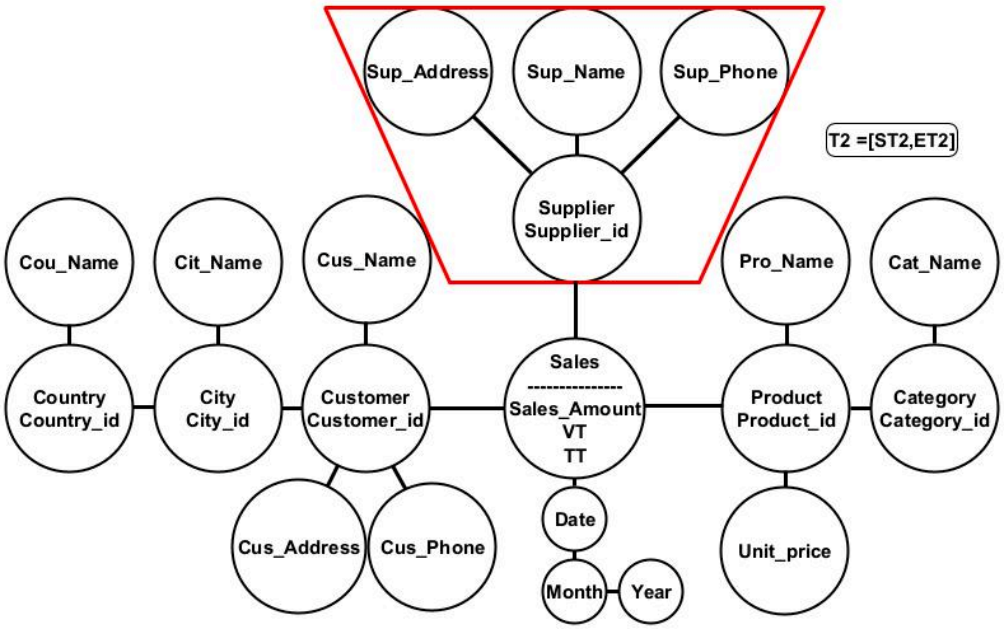



Figure 4.2 – Schéma logique de GAMM($t_2$)

Ce nouvel axe d'analyse, créé par l'ajout de la dimension Supplier, augmente le niveau de granularité de la mesure présente dans le fait Sales et modifie le schéma du modèle en créant la nouvelle version. Par conséquent, de nouvelles instances de ces mesures, ainsi que du fait correspondant, seront créées pour répondre à ce nouveau niveau de détail. Les anciennes instances seront conservées à des fins d'historisation.

La fonction d'évolution correspondant à cette opération est représentée comme suit :

$$GAMM(t_2) = GAMM(t_1) + Add\_dimension(\text{Supplier}, \text{Sales},$$

$$[Sup\_Name, Sup\_Address, Sup\_Phone]).$$

*Add_dimension(*Supplier,Sales,*Sup_Name,Sup_Address,Sup_Phone)* représente l'opérateur d'évolution permettant de rajouter la dimension Supplier ayant les attributs «Sup_Name, Sup_Address, Sup_Phone» au fait Sales.

### 4.2.4 Exemple 3 : Supprimer la dimension Customer avec ses niveaux de hiérarchie et ajouter la mesure Quantity.

L'ajout de cette nouvelle mesure, représentée par la figure 4.3, répond à un nouveau besoin d'analyse, à savoir la possibilité de déterminer la quantité des ventes en fonction des dimension PRODUCT et SUPPLIER.

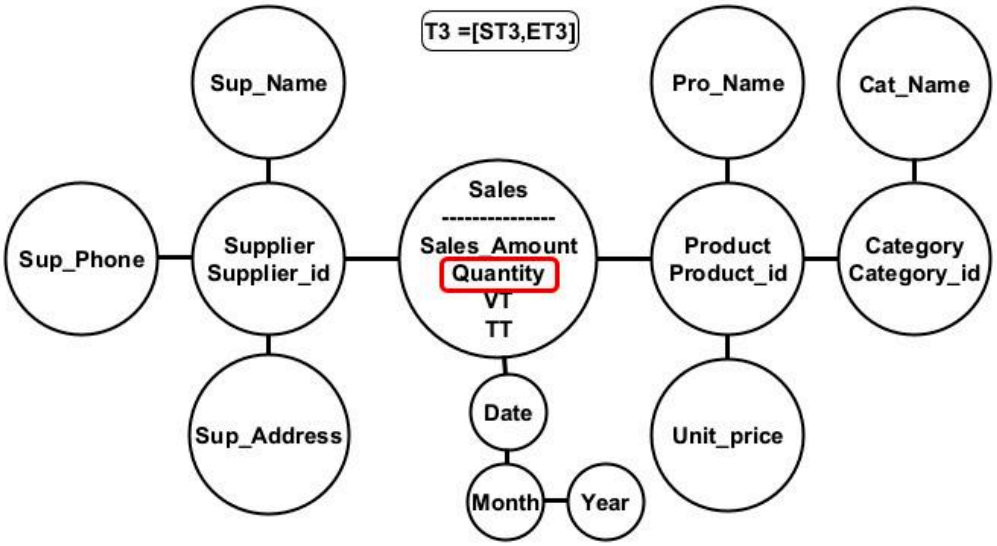



FIGURE 4.3 – Schéma logique de GAMM($t_3$)

Cette mesure n'est représentée que par les dimensions PRODUCT et SUPPLIER, ce qui implique la suppression de la dimension CUSTOMER du schéma. Cette opération réduit le niveau de granularité des mesures associées au fait SALES. Par conséquent, de nouvelles instances de ces mesures seront créées pour répondre à ce nouveau niveau de détail.

La fonction d'évolution correspondant à cette opération est représentée comme suit :

$$GAMM(t_3) = GAMM(t_2) + Delete\_entity(\text{CUSTOMER})$$

$$+Add\_measure(\text{QUANTITY}, \text{SALES}).$$

*Delete_entity(CUSTOMER)* représente l'opérateur d'évolution permettant de supprimer la dimension CUSTOMER ainsi que ses niveaux de hiérarchie tandis que *Add_measure(QUANTITY,SALES)* représente l'opérateur d'évolution permettant de rajouter la mesure QUANTITY au fait SALES.

Les évolutions dans GAMM sont caractérisées par la création d'une nouvelle version de schéma pour chaque modification du schéma. $GAMM(t_0)$ représente le modèle initial ; $GAMM(t_1)$ représente la première évolution, qui correspond à l'ajout d'un niveau de hiérarchie ; $GAMM(t_2)$ représente la deuxième évolution, qui correspond à l'ajout

d'une dimension ; et $GAMM(t_3)$ représente la dernière évolution, qui correspond à l'ajout d'une mesure et à la suppression d'une dimension.

## 4.3 Évolution des données dans GAMM

Dans un entrepôt de données, l'évolution des données se produit à travers plusieurs processus, notamment le chargement initial des données, le rafraîchissement des données, ainsi que la modification ou la suppression des données (voir sous-section 2.2.1.6). Voici les différentes opérations recensées pour ces processus :

1. *Chargement initial des données* : C'est la première étape de la construction de l'entrepôt de données. Elle consiste à extraire, transformer et charger les données provenant de différentes sources hétérogènes (bases de données opérationnelles, fichiers, etc.) dans l'entrepôt de données. Ce chargement initial se fait généralement pour l'alimenter l'entrepôt à l'issue de la construction.
2. *Rafraîchissement des faits* : Les données factuelles dans un entrepôt de données représentent des mesures numériques ou des agrégations, telles que les ventes, les revenus, les quantités, etc. Le rafraîchissement des faits consiste à mettre à jour ces mesures avec de nouvelles données provenant des sources opérationnelles. Selon la fréquence de rafraîchissement souhaitée, cela peut être effectué quotidiennement, hebdomadairement ou à tout autre intervalle de temps défini.
3. *Rafraîchissement des dimensions* : Les dimensions dans un entrepôt de données représentent les attributs des données décrivant les faits à observer, tels que le temps, la géographie, les produits, etc. Le rafraîchissement des dimensions est nécessaire lorsque les informations de référence associées à ces dimensions changent. Par exemple, l'ajout d'un nouveau client ou d'une nouvelle catégorie de produit. Les dimensions sont généralement rafraîchies moins fréquemment que les faits.
4. *Modification des faits* : Les faits dans un entrepôt de données peuvent également être sujets à des modifications. Cela peut se produire lorsque des erreurs sont détectées dans les sources opérationnelles ou de nouvelles occurrences des faits arrivent.
5. *Modification des dimensions* : Les dimensions d'un entrepôt de données peuvent être sujettes à des modifications. Cela peut se produire lorsque les données de la dimension changent, comme les attributs ou les liens hiérarchiques, ou encore plus rarement lorsque des erreurs sont détectées dans les sources opérationnelles.
6. *Suppression de données* : Les données dans un entrepôt de données peuvent également être sujettes à des suppressions. Cela peut se produire lorsque des données obsolètes doivent être supprimées de l'entrepôt.

Les modifications/suppressions de données dans un entrepôt doivent être conformes à une politique préalablement adoptée par l'entreprise. Il est essentiel de mettre en place des mécanismes appropriés pour gérer ces modifications/suppressions tout en maintenant l'intégrité et la cohérence de l'entrepôt de données. Dans le modèle GAMM,

aucune fonction de suppression n'a été définie en raison de la conservation de toutes les données à des fins d'historisation. Cependant, les périodes de validité des relations hiérarchiques peuvent être paramétrées en utilisant l'étiquetage temporel de type intervalle (FD et TD).

Pour gérer ces processus, différentes méthodes et outils peuvent être utilisés, tels que les processus ETL, les scripts de mise à jour automatisés, les outils de gestion de données, etc. L'objectif principal est de maintenir l'entrepôt de données à jour avec les dernières informations disponibles et de garantir la qualité et l'intégrité des données pour les analyses ultérieures.

Dans ce qui suit, nous détaillons les fonctions d'évolution des données que nous avons établies, lesquelles permettent d'évoluer les instances dans le modèle GAMM. Ces fonctions comprennent le chargement initial, le rafraîchissement ainsi que la modification des données.

### 4.3.1 Fonctions d'évolution des données dans GAMM

Les instances de données dans GAMM évoluent de manière incrémentale et sans redondance. Toutes les instances de données des schémas précédents sont disponibles pour consultation. Les fonctions d'évolution des données sont définies dans le tableau 4.2 comme suit :

Les fonctions présentées dans le tableau 4.2 sont utilisées dans le contexte de l'évolution des données. Elles permettent d'effectuer des opérations de chargement initial, de rafraîchissement et de modification des données, tout en préservant l'historique de ces évolutions. L'étiquetage temporel de type horodatage ou de type intervalle sera appliqué respectivement aux nœuds et aux relations, conformément au principe des bases de données temporelles ainsi qu'au formalisme du modèle GAMM présenté dans le chapitre 3.

#### 4.3.1.1 La fonction « *Load_data()* »

La fonction « *Load_data* » est utilisée pour effectuer le chargement initial des données à partir des sources de données dans le modèle GAMM. Cette opération consiste à extraire les données des sources, à les transformer si nécessaire et à les charger dans l'entrepôt de données.

Lors du chargement initial des données, un étiquetage temporel est appliqué aux nœuds représentant les faits ainsi qu'aux relations entre les nœuds de dimensions. Plus précisément :

1. Les nœuds représentant les faits sont étiquetés avec un horodatage ponctuel de type VT et TT. L'attribut VT prend la date de validité réelle de l'instance à partir des données sources, tandis que TT prend la date de chargement des données dans l'entrepôt de données. Cela permet d'attribuer une valeur temporelle à chaque instance de fait, indiquant quand elle est valide et quand elle a été

TABLE 4.2 – Fonctions d'évolution des données

| **Évolution** | **Fonction d'évolution** | **Description** |
|---|---|---|
| Chargement initial des données | *Load_data(Data_sources)* | Chargement initial des données à partir des sources de données. Un horodatage ponctuel de type VT et TT sera appliqué aux nœuds représentant les faits, tandis qu'un étiquetage temporel de type intervalle FD et TD sera appliqué aux relations entre les nœuds de dimensions. |
| Rafraîchissement des dimensions | *Refresh_dimension( Data_sources)* | Chargement des nouvelles entités de type dimension (nœuds/relations) à partir des sources de données. Un étiquetage temporel de type intervalle FD et TD sera appliqué aux relations entre les nœuds de dimensions concernés. |
| Rafraîchissement des faits | *Refresh_fact(Data_sources)* | Chargement des nouveaux nœuds de type faits à partir des sources de données et création des liaisons avec les nœuds de dimensions correspondants. Un horodatage ponctuel de type VT et TT sera appliqué aux nœuds concernés. |
| Modification des dimensions | *Change_dimension*($\{D, A\}$), *Change_dimension*($\{L, A\}$), *Change_dimension*($\{D, L\}$), *Change_dimension*($\{L, L\}$) | Modification d'un attribut de description ou d'un lien hiérarchique dans les dimensions. Création d'un nouveau nœud d'attribut et liaison avec le nœud approprié (dimension/niveau) en cas de changement d'attribut, ou création d'un nouveau lien vers le niveau approprié en cas de changement de relation hiérarchique. Un étiquetage temporel de type intervalle FD et TD sera appliqué aux relations entre les nœuds de dimensions, et le TD de l'ancien lien sera mis à jour. |
| Modification des faits | *Change_fact*($\{F\}$) | Modification des nœuds de type faits en cas d'erreur dans les sources de données. L'horodatage ponctuel de type VT et TT des nœuds en question sera maintenu. |

chargée dans l'entrepôt de données.

2. Les relations entre les nœuds de dimensions sont étiquetées avec un étiquetage temporel de type intervalle représenté par les attributs FD et TD. L'attribut FD prend la date de début de la relation à partir des données sources, tandis que TD prend la date du 31/12/9999. Ces étiquettes indiquent la période de validité de chaque relation entre les valeurs des dimensions ( La date 31/12/9999 signifie que la relation est considérée comme toujours valide et qu'elle n'a pas encore expiré ).

#### 4.3.1.2 La fonction « *Refresh_dimension()* »

La fonction d'évolution « *Refresh_dimension* » est utilisée pour charger les nouvelles entités de type dimension à partir des sources de données dans le modèle GAMM. Cela permet d'ajouter de nouvelles instances de dimensions ou de nouvelles relations entre les nœuds de dimensions existants.

Lors du chargement de ces nouvelles entités de dimension, un étiquetage temporel de type intervalle, représenté par les attributs FD et TD, est appliqué aux relations entre les nœuds de dimensions concernés. De la même manière que pour le chargement initial des données, l'attribut FD prend la date de début de la relation à partir des données sources, tandis que l'attribut TD prend la date du 31/12/9999. Cela permet de définir la période de validité de ces relations, indiquant quand elles sont actives et quand elles ont été chargée dans l'entrepôt de données.

#### 4.3.1.3 La fonction « *Refresh_fact()* »

La fonction d'évolution « *Refresh_fact* » est utilisée pour charger les nouveaux nœuds de type faits à partir des sources de données et les relier aux nœuds de dimensions correspondants dans le modèle GAMM. Cette opération permet d'ajouter de nouvelles instances de faits à l'entrepôt de données.

Lors du chargement de ces nouveaux nœuds de faits, un étiquetage temporel de type horodatage ponctuel est appliqué aux nœuds concernés, en utilisant les attributs VT et TT. De la même manière que pour le chargement initial des données, l'attribut VT prend la date de validité réelle de l'instance à partir des données sources, tandis que TT prend la date de chargement des données dans l'entrepôt de données. Cela permet d'attribuer une valeur temporelle à chaque nouvelle instance de fait, indiquant quand elle est valide et quand elle a été modifiée.

De plus, lors du chargement, les nouveaux nœuds de faits sont reliés aux nœuds de dimensions correspondants. Ces relations sont établies en fonction des clés de correspondance entre les données de fait et les données de dimensions (En général, le rafraîchissement des faits est toujours précédé par celui des dimensions).

#### 4.3.1.4 La fonction « *Change_dimension()* »

La fonction d'évolution « *Change_dimension* » est utilisée pour effectuer des modifications sur les dimensions dans le modèle GAMM. Cette évolution peut être appliquée dans différents scénarios, tels que le changement d'un attribut de description ou d'un lien hiérarchique.

Lors de cette évolution, plusieurs cas sont envisagés en fonction du type de modification :

1. « *Change_dimension($\{D, A\}$)* » : Si un attribut de description change dans une dimension, la fonction d'évolution crée un nouveau nœud représentant ce nouvel attribut et le relie au nœud de dimension approprié. Les relations entre les nœuds de dimension sont mises à jour en appliquant un étiquetage temporel de type intervalle FD et TD pour déterminer la période de validité de ces relations. L'attribut FD prend la date de début de la relation à partir des données sources, tandis que l'attribut TD prend la date du 31/12/9999. De plus, la date de début de cette nouvelle relation sera attribuée au paramètre TD de l'ancien lien pour déterminer la fin de sa validité, Cela permet de définir la période de validité de de chacune des ces relations.

2. « *Change_dimension($\{D, L\}$)* » : Si un lien hiérarchique change entre un noued de type dimension et un nœud de type niveau de hiérarchie, la fonction d'évolution crée un nouveau lien vers le niveau approprié dans la hiérarchie. Comme précédemment, un étiquetage temporel de type intervalle FD et TD a cette nouvelle relation. L'attribut FD prend la date de début de la relation à partir des données sources, tandis que l'attribut TD prend la date du 31/12/9999. De plus, la date de début de cette nouvelle relation sera attribuée au paramètre TD de l'ancien lien hiérarchique pour déterminer la fin de sa validité, Cela permet de définir la période de validité de de chacune des ces relations.

3. « *Change_dimension($\{L, A\}$)* » : De la même manière que pour le cas d'un changement d'attribut d'un nœud dimension ; si un attribut de description change pour un niveau de hiérarchie, la fonction d'évolution crée un nouveau nœud représentant ce nouvel attribut et le relie au nœud de dimension approprié. Les relations entre les nœuds de dimension sont mises à jour en appliquant un étiquetage temporel de type intervalle FD et TD pour déterminer la période de validité de ces relations. l'attribut FD prend la date de début de la relation à partir des données sources, tandis que l'attribut TD prend la date du 31/12/9999. De plus, la date de début de cette nouvelle relation sera attribuée au paramètre TD de l'ancien lien pour déterminer la fin de sa validité, Cela permet de définir la période de validité de chacune des ces relations.

4. « *Change_dimension($\{L, L\}$)* » : De la même manière que pour le cas d'un changement d'un lien hiérarchique entre un nœud de type dimension et un nœud de type niveau de hiérarchie, si un lien hiérarchique change entre deux

nœuds de type niveau de hiérarchie, la fonction d'évolution crée un nouveau lien vers le niveau approprié dans la hiérarchie. Un étiquetage temporel de type intervalle FD et TD a cette nouvelle relation. l'attribut FD prend la date de début de la relation à partir des données sources, tandis que l'attribut TD prend la date du 31/12/9999. De plus, la date de début de cette nouvelle relation sera attribuée au paramètre TD de l'ancien lien hiérarchique pour déterminer la fin de sa validité, Cela permet de définir la période de validité de chacune des ces relations.

#### 4.3.1.5 La fonction « *Change_fact()* »

La fonction d'évolution « *Change_fact* » est utilisée pour effectuer des modifications sur les nœuds de type faits dans le modèle GAMM en cas d'erreurs détectées dans les données sources. Cette évolution permet de corriger les valeurs erronées ou incohérentes des nœuds de faits existants.

Lors de cette évolution, lorsqu'il s'agit de corriger des erreurs chargées précédemment à partir des données sources, l'horodatage ponctuel de type VT et TT des nœuds en question est maintenu. Cela signifie que les horodatages initiaux des nœuds de faits restent inchangés, même après la modification des valeurs.

Ces modifications peuvent être basées sur des vérifications supplémentaires des données, des corrections manuelles ou des mises à jour automatiques à partir de sources de données fiables. En maintenant l'horodatage ponctuel des nœuds de faits, le modèle GAMM conserve l'information temporelle initiale sur ces nœuds.

Les fonctions d'évolution de données du modèle GAMM ont été conçues pour assurer une évolution harmonieuse des instances de données. L'étiquetage temporel appliqué aux nœuds et aux relations permet de gérer la temporalité des données, garantissant ainsi une représentation cohérente et historique dans l'entrepôt. Cette approche permet d'effectuer des requêtes OLAP et des analyses précises et concordantes, offrant ainsi une base solide pour les prises de décision.

Dans ce qui suit, nous allons réutiliser notre exemple de motivation pour proposer les évolution de données décrites précédemment (voir sous-section 3.2.2).

### 4.3.2 Exemple 1 : Évolution d'un niveau de hiérarchie d'une instance de la dimension Customer

Nous examinons les instances de la dimension Customer présentées dans le tableau 3.5. Ce tableau indique que la cliente Mary Saveley est associée à deux villes différentes pendant deux périodes de temps distinctes. En effet, la cliente résidait à Paris depuis le 01/01/2020 et a déménagé à Lyon le 01/08/2020.

Ce changement se traduit par une évolution au niveau du lien hiérarchique (Customer, City), qui peut être concrétisée dans le modèle GAMM par la fonction d'évolution de données *Change_dimension(Customer, City)*. Nous représentons cette évolution des instances de la dimension Customer en utilisant le formalisme du modèle GAMM, comme illustré dans la figure 4.4. Les paramètres FD et TD sont utilisés pour identifier les durées de vie des relations hiérarchiques ainsi que les relations avec les attributs.

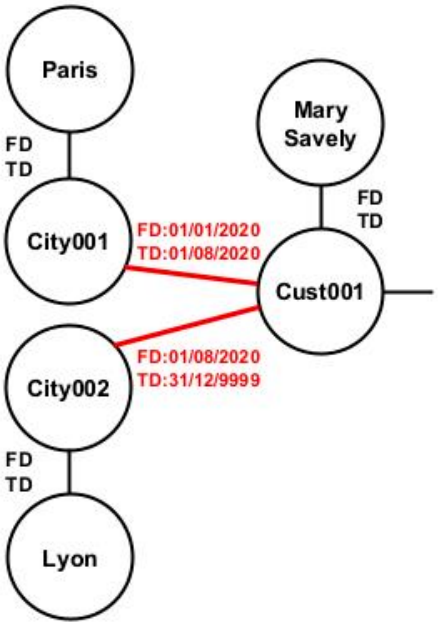



Figure 4.4 – Exemple de changement d'une instance de données de la dimension Customer

Le schéma montre que la relation entre le client Mary Saveley, qui a l'identifiant Cust001, et le niveau Paris, qui a l'identifiant City001, est valide pendant l'intervalle de temps du 01/01/2020 au 01/08/2020 ; tandis que la relation entre ce même client et le niveau Lyon, qui a l'identifiant City002, est valable pendant l'intervalle de temps allant du 01/08/2020 et demeure toujours valide.

### 4.3.3 Exemple 2 : Modification de niveau de hiérarchie d'une instance de la dimension Product

Nous réutilisons les instances de la dimension Customer qui sont présentées dans le tableau 3.5. Ce tableau indique que le produit Mozzarella est associé à deux catégories différentes pendant deux périodes distinctes. En effet, le produit était initialement classé dans la catégorie Fresh depuis le 01/01/2020, mais il a été reclassé dans la catégorie Dairy à partir 01/04/2020.

Ce changement se traduit par une évolution au niveau de la relation hiérarchique (PRODUCT, CATEGORY), qui peut être réalisée dans le modèle GAMM par la fonction de données *Change_dimension(PRODUCT, CATEGORY)*. Nous représentons cette évolution des instances de la dimension PRODUCT en utilisant le formalisme du modèle GAMM, comme illustré dans la figure 4.5.

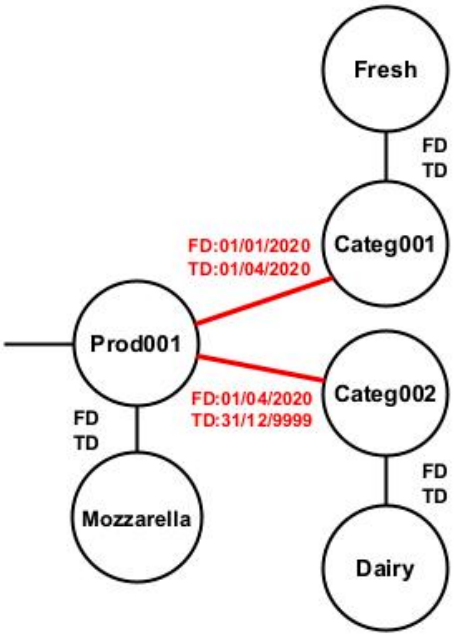



FIGURE 4.5 – Exemple de changement d'une instance de données de la dimension PRODUCT

Ce schéma illustre que la relation hiérarchique entre le produit MOZZARELLA, identifié par PROD001, et le niveau FRESH, identifié par CATEG001, est valide pendant la période de temps allant du 01/01/2020 au 01/04/2020, tandis que la relation entre ce même produit et le niveau DAIRY, identifié par CATEG002, est valide à partir du 01/04/2020 jusqu'à aujourd'hui.

Par conséquent, le schéma global des instances de notre exemple de motivation d'évolution de données (voir section 3.2.2) est représenté en utilisant le formalisme de GAMM dans la figure 4.6.

L'utilisation des graphes facilite l'incorporation de la temporalité dans les relations entre les instances, car les liens portent également des informations, tout comme les nœuds. Dans notre cas, les liens entre les nœuds CUSTOMER et CITY contiennent les attributs FD et TD, ce qui permet de déterminer les durées de vie des relations entre ces instances. De même, les liens entre les nœuds PRODUIT et CATÉGORIE suivent la

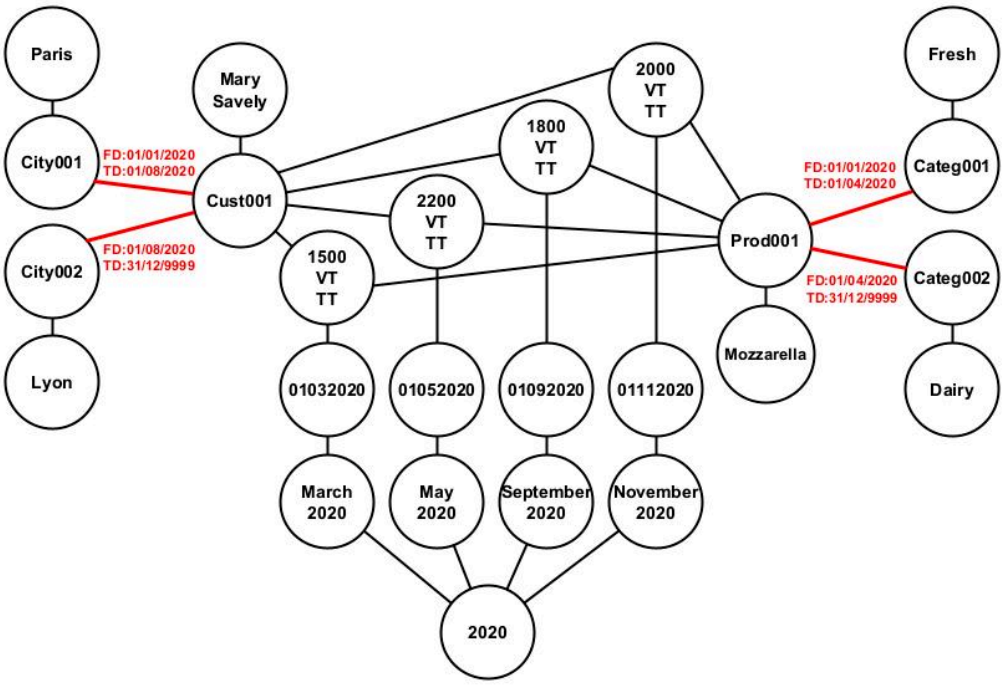



FIGURE 4.6 – Exemple de changement d'instances de données dans le modèle GAMM même logique.

## 4.4 Conclusion

Ce chapitre a mis en évidence l'évolution temporelle des schémas et des données dans le modèle GAMM et démontre ainsi sa flexibilité pour répondre aux nouveaux besoins d'analyse et gérer les nouvelles sources de données. Les évolutions de schéma sont gérées comme un processus multi-versions, où chaque version représente l'état du schéma pendant une période de temps.

Le schéma évolue en ajoutant, modifiant ou supprimant des entités (faits, mesures, dimensions, niveaux ou attributs). Chaque évolution est réalisée à l'aide d'opérateurs d'évolution qui appliquent les changements nécessaires pour transformer le schéma en une version désirée. Ces opérateurs prennent en compte les dépendances entre les modifications pour assurer la cohérence du schéma.

L'entreposage des données dans une base de données orientée graphe de type NoSQL permet la création de nouvelles entités uniquement au niveau du méta-modèle, facilitant ainsi le chargement incrémental des données correspondant aux nouvelles entités créées à l'aide des processus ETL.

Les exemples d'évolution de schéma présentés dans ce chapitre ont illustré différentes opérations possibles, telles que l'ajout de niveaux, de dimensions et de mesures, ainsi que la suppression d'entités. Ces exemples ont démontré la capacité du modèle GAMM à s'adapter aux changements et à répondre aux nouveaux besoins d'analyse.

L'évolution des instances de données dans GAMM est également abordée, avec des processus tels que le chargement initial des données, le rafraîchissement des faits et des dimensions, ainsi que la modification des faits. Des exemples d'illustration ont été présentés.

En conclusion, le modèle GAMM permet une évolution flexible des schémas et des données pour répondre aux nouveaux besoins d'analyse. Les fonctions d'évolution et les processus de chargement et de rafraîchissement des données garantissent la cohérence et l'intégrité du schéma et des données tout au long de leur évolution.

CHAPITRE 5

# Interrogation des versions de schémas et des données dans GAMM

## Sommaire

## 5.1 Introduction

L'interrogation des versions de schémas et des données constitue un aspect crucial de la gestion des entrepôts de données évolutifs en multi-versions. Dans ce chapitre, nous abordons les différents aspects liés à cette interrogation, notamment l'extraction du schéma pour les différentes versions existantes dans le modèle GAMM, l'interrogation des instances de données à partir des versions de schéma, ainsi que celle des données à évolution temporelle.

Tout d'abord, nous examinons l'extraction de schéma pour les différentes versions dans le modèle GAMM. Cette opération consiste à identifier et représenter les structures et les relations entre les entités pour chaque version spécifique. Elle permet de comprendre la composition des faits et des dimensions dans chaque version, ainsi que les éventuelles modifications de schéma au fil du temps. L'extraction de schéma facilite ainsi la compréhension de l'évolution des données et permet de formuler des requêtes adaptées à chaque version spécifique.

Ensuite, nous abordons l'interrogation des instances de données à partir des versions de schéma. Cela implique de spécifier les versions de schéma à partir desquelles les données doivent être extraites. Grâce à des requêtes explicites, nous pouvons sélectionner les données correspondant à une version spécifique en utilisant les paramètres temporels appropriés. De plus, les requêtes implicites nous permettent d'interroger les données sans restrictions sur les versions. Cela nous permet d'obtenir des résultats précis et pertinents en fonction de la version de schéma souhaitée.

Enfin, nous nous intéressons à l'interrogation des données à évolution temporelle. Cette approche permet d'analyser les changements et les tendances au fil du temps en effectuant des requêtes qui tiennent compte des périodes de validité des données. En spécifiant des intervalles de temps, nous pouvons obtenir des informations sur l'évolution des données et les comparer entre différentes versions de schéma.

Ces différents aspects d'interrogation des versions de schémas et des données sont essentiels pour exploiter pleinement les entrepôts de données évolutifs en multi-versions et pour obtenir des *insights* pertinents sur l'évolution des données au fil du temps.

## 5.2 Identification des versions de schéma

L'approche GAMM propose une méthodologie flexible pour gérer l'évolution des schémas et des données dans les entrepôts de données. Cette approche se base sur un modèle évolutif en multi-versions, où chaque version de schéma est valide pendant une période de temps spécifique.

Plus précisément, nous utilisons un méta-modèle qui permet d'extraire le schéma valide parmi les différentes versions enregistrées. Chaque schéma est associé à une période de validité T=[ST,ET], et l'utilisateur a la possibilité de visualiser les différentes versions du schéma de l'entrepôt de données à tout moment.

Dans notre étude, nous avons développé l'algorithme 1 qui permet d'identifier le schéma correspondant à partir du méta-modèle mentionné précédemment (voir section 3.4). Cet algorithme facilite la visualisation des schémas passés et présents, ce qui permet à l'utilisateur de repérer les analyses possibles à n'importe quel moment dans l'entrepôt de données.

**Algorithm 1:** Récupération d'un schéma dans GAMM à l'instant t

**Entrée:** instant t
**Sortie :** Schéma GAMM(t)

$version \leftarrow get\_version(t)$;
$F \leftarrow get\_fact(version)$;
**foreach** $f$ dans $F$ **do**
  $M_f \leftarrow get\_measure(f, version)$;
  $M.update(M_f)$;
  $F.MAssoc[F, M].create\_association[f, M_f]$;
  $D_f \leftarrow get\_dimension(f, version)$;
  $D.update(D_f)$;
  $FAssoc[F, D].create\_association[f, D_f]$;
**end**
**foreach** $d$ dans $D$ **do**
  $A_d \leftarrow get\_attribute(d, version)$;
  $A.update(A_d)$;
  $D.DAssoc_a[D, A].create\_association[d, A_d]$;
  **if** has_hierarchy(d,version) **then**
    $H_d \leftarrow get\_hierarchy(d, version)$;
    $H.update(H_d)$;
  **end**
**end**
**return** F, D, FAssoc[F, D], H, version

L'algorithme prend en entrée un instant $t$ qui sera compris entre un Starting_Time (ST) et un Ending_Time (ET) d'une version spécifique du schéma. Il utilise principalement quatre fonctions principales (get_version, get_fact, get_dimension et gets_hierarchy) pour identifier les différentes entités constituant une version de schéma. Il utilise également d'autre fonctions secondaires pour mettre à jours les différentes entités ainsi que les fonctions d'association.

*get_version*($t$) : permet de retourner un objet de type **Version** correspondant à un instant précis dans le temps $t$. L'objet **Version** contient l'identifiant de la version ainsi que les informations ST (temps de début) et ET (temps de fin) de cette version spécifique.

*get_fact*($version$) : permet de retourner un objet de type **F** représentant l'en-

semble des faits correspondant à la version spécifiée. Les mesures associées à chaque fait $f$ sont identifiées et liées grâce aux fonctions $get\_measure(f, version)$, $M.update(M_f)$ et $FAssoc[F, D].create\_association[f, D_f]$.

$get\_dimension(f, version)$ : permet de retourner un objet de type **D** représentant l'ensemble des dimensions associées au fait $f$ et correspondant à la version spécifiée. L'ensemble global des dimensions est mis à jour à l'aide de la fonction $D.update(D_f)$, tandis que la fonction d'association des dimensions $D$ au fait $f$ est mise à jour à l'aide de la fonction $FAssoc[F, D].create\_association[f, D_f]$. De plus, les attributs de description associés à chaque dimension $d$ sont identifiés et liés grâce aux fonctions $A.update(A_d)$ et $D.DAssoc_a[D, A].create\_association[d, A_d]$.

$get\_hierarchy(d, version)$ : permet de retourner un objet de type **H** représentant l'ensemble des hiérarchies associées à la dimension $d$ et correspondant à la version spécifiée. la fonction permet d'identifier les niveaux de hiérarchies, leurs attributs de description ainsi que les liens d'agrégation entre ces niveaux. L'ensemble global des hiérarchies est mis à jour à l'aide de la fonction $H.update(H_d)$.

L'algorithme 1 a pour fonction de retourner plusieurs objets : l'ensemble des faits $F$, l'ensemble des dimensions $D$, la fonction d'association entre les faits et les dimensions $FAssoc[F, D]$, ainsi que l'ensemble des hiérarchies $H$, qui constituent la version spécifiée associée à l'instant $t$. Il renvoie également l'objet $version$, qui permet de déterminer le ST et le ET de cette version. Ces paramètres temporels seront utilisés ultérieurement pour formuler des requêtes sur les versions, comme décrit dans la section 5.3.

Grâce à cet algorithme, l'utilisateur est en mesure de visualiser le schéma de l'entrepôt de données à un instant précis et d'identifier les paramètres temporels associés à cette version spécifique. Il pourra récupérer les entités du schéma ainsi que leurs associations correspondantes, conformément à la définition 1 du formalisme du modèle GAMM décrit dans la section 3.5. L'algorithme offre une flexibilité pour analyser les différentes versions du schéma, ce qui permet d'obtenir une compréhension approfondie de l'évolution du schéma dans le modèle GAMM.

## 5.3 Interrogation des données à partir des versions de schéma

Dans le modèle GAMM, chaque version de schéma (SV) est valide pendant une période de temps spécifique, avec un temps de début (ST) et un temps de fin (ET). Les versions de schéma évoluent de manière séquentielle, et le chargement et le rafraîchissement des données se font toujours sur la dernière version active.

Un horodatage ponctuel, représenté par le temps de transaction (TT) et le temps de validité (VT), est appliqué aux nœuds de type fait lors du chargement des données. Le temps de transaction (TT) correspond au temps de chargement des données. Ainsi, chaque valeur du paramètre temporel TT sera comprise entre le temps de début (ST)

et le temps de fin (ET) de la version de schéma active. Cela permet d'assurer que les données sont enregistrées dans la version de schéma appropriée.

Les valeurs temporelles ST et ET, correspondant au temps de début et au temps de fin de chaque version, ainsi que le temps de transaction TT des nœuds de type fait, sont utilisées pour paramétrer les requêtes. Les requêtes peuvent être des requêtes simples sur une seule version de schéma ou des requêtes croisées sur plusieurs versions de schéma.

Nous présentons ci-dessous un exemple d'instance de données dans un entrepôt de données évolutif en multi-versions, basé sur le modèle entité-relation (voir sous-section 5.3.1). Par la suite, nous reprenons le même exemple pour illustrer des requêtes simples et des requêtes croisées dans le modèle GAMM (voir sous-section 5.3.2).

### 5.3.1 Exemple d'un entrepôt de données multi-versions basés sur le modèle entité-relation

Dans la plupart des travaux de recherche portant sur l'évolution des modèles multi-dimensionnels classiques basés sur le modèle entité-relation Ahmed et al. (2021), l'approche généralement reconnue consiste à créer une nouvelle table de faits pour chaque version, en particulier en cas de changement du niveau de granularité des mesures. Nous proposons ici un exemple d'instances de faits basé sur l'approche relationnelle précitée, correspondant aux trois versions du schéma dans notre exemple actuel. Les tableaux 5.1, 5.2 et 5.3 montrent des extraits des versions de la table de faits Sales, qui affichent les instances de faits en fonction des différentes évolutions du schéma.

Table 5.1 – Table de faits version 1

| **Sale_id** | **Customer_id** | **Product_id** | **Orderdate** | **Sales_Amount** |
|---|---|---|---|---|
| S1001 | Cus001 | Pro001 | 15062020 | 1800 |

Table 5.2 – Table de faits version 2

| Sale_id | Customer_id | Product_id | Supplier_id | Orderdate | Sales_Amount |
|---|---|---|---|---|---|
| S2001 | Cus001 | Pro001 | Sup001 | 12072020 | 1200 |
| S2002 | Cus001 | Pro001 | Sup002 | 25072020 | 1500 |

Table 5.3 – Table de faits version 3

| **Sale_id** | **Product_id** | **Supplier_id** | **Orderdate** | **Sales_Amount** |
|---|---|---|---|---|
| S3001 | Pro001 | Sup001 | 08082020 | 2000 |
| S3002 | Pro001 | Sup002 | 22082020 | 2500 |

Effectivement, dans l'approche présentée, les requêtes sur chaque version sont effectuées séparément sur la table des faits correspondant à cette version. Cela signifie que pour chaque version du schéma, une requête distincte doit être formulée pour extraire les données spécifiques à cette version.

Quant aux requêtes croisées, elles nécessitent l'union de plusieurs requêtes sur différentes versions de la table des faits. Cela devient complexe car il faut aligner le niveau de granularité des résultats entre les différentes versions. Cela peut être particulièrement difficile lorsque de nombreuses versions sont impliquées.

L'alignement du niveau de granularité implique de s'assurer que les attributs et les dimensions utilisés dans les requêtes ont des correspondances appropriées entre les différentes versions. Il peut également nécessiter des opérations de regroupement, d'agrégation ou d'ajustement des données pour obtenir des résultats cohérents et comparables entre les versions.

La complexité de cette approche croît avec le nombre de versions et la complexité des schémas évolutifs. Plus il y a de versions et de différences entre les schémas, plus il devient difficile de formuler des requêtes croisées et d'obtenir des résultats cohérents.

C'est l'un des défis auxquels sont confrontés les entrepôts de données évolutifs en multi-versions, en particulier dans le cadre d'approches basées sur des tables de faits distinctes pour chaque version. Cela a conduit au développement de modèles alternatifs, tels que le modèle GAMM, qui propose une approche basée sur les graphes pour traiter plus facilement les requêtes sur les versions de schéma et les requêtes croisées.

### 5.3.2 Requêtes sur versions de schéma dans GAMM

En utilisant notre approche GAMM basée sur les graphes, nous représentons dans la Figure 5.1 les mêmes instances de données que celles utilisées dans l'exemple précédent. Ces instances illustrent l'évolution du schéma et des données dans un entrepôt de données évolutif en multi-versions.

Le nœud $TT_1$ représentent le montant des ventes entre le client Cus001 et le produit Pro001 dans la version valable pour la période $T_1 = [01062020, 30062020]$. Ces nœuds indiquent les valeurs spécifiques de cette instance pour cette période.

De même, les nœuds $TT_2$ représentent le montant des ventes entre le client Cus001, le produit Pro001, et les fournisseurs Sup001 et Sup002 dans la version valable pour la période $T_2 = [01072020, 31072020]$. Ils représentent les valeurs spécifiques de cette instance pour cette période.

Enfin, les nœuds $TT_3$ représentent le montant des ventes entre le produit Pro001 et les fournisseurs Sup001 et Sup002 dans la version valable pour la période $T_3 =$

[01082020, 31082020]. Encore une fois, ces nœuds indiquent les valeurs spécifiques de cette instance pour cette période.

Ainsi, en utilisant notre approche basée sur les graphes, nous sommes en mesure de représenter les différentes versions du schéma et d'associer les instances de données correspondantes à chaque version. Cette représentation graphique permet de visualiser clairement l'évolution des données dans un entrepôt de données évolutif en multi-versions.

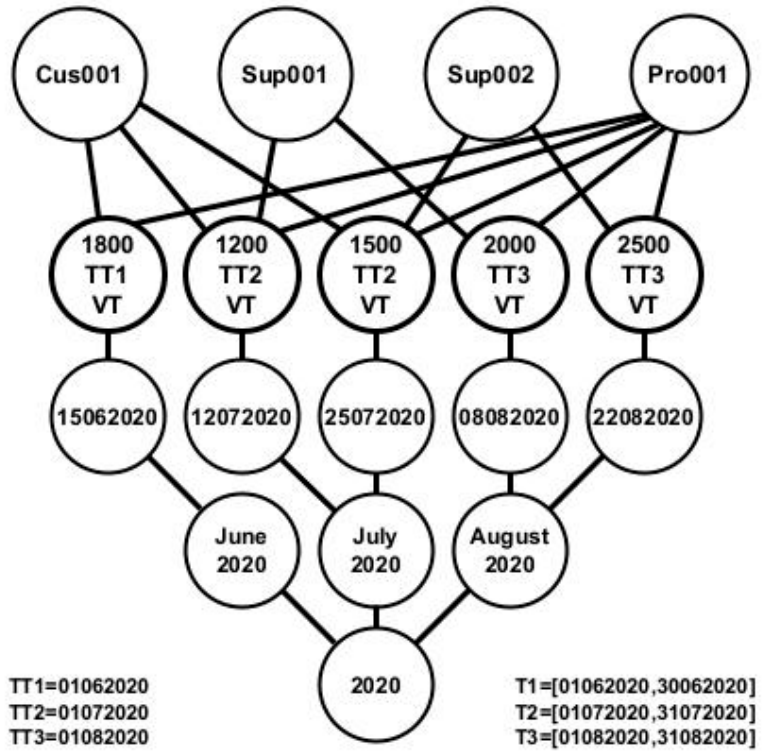



FIGURE 5.1 – Exemple d'instances de GAMM

GAMM offre une flexibilité considérable lors de la création de requêtes explicites et implicites, ce qui est illustré dans les exemples suivants. Les requêtes sont rédigées en utilisant le langage de requête de graphes (Cypher), qui est spécifique à la base de données orientée graphes Neo4j.

Cypher est un langage de requêtes performant et expressif qui permet de formuler des requêtes graphiques de manière intuitive. Il utilise des motifs de graphe pour spécifier les relations et les conditions de recherche, facilitant ainsi l'interrogation des données dans un environnement de base de données de graphes.

Le modèle GAMM permet d'élaborer des requêtes explicites qui définissent des critères de recherche sur des versions de schéma précises. Il prend également en charge

les requêtes implicites, où les informations sont extraites des données sans qu'il y ait un critère de recherche spécifique sur les versions de schéma.

#### 5.3.2.1 Requêtes explicites

Les requêtes explicites dans le modèle GAMM permettent d'extraire des données d'une version spécifique ou d'un ensemble de versions sur la base du paramètre temporel TT à l'aide de la clause $ST_n <= TT < ET_n$ ; où $T_n = [ST_n, ET_n]$ est la période de validité de la version $V_n$ ou d'un ensemble de versions. En utilisant cette clause dans une requête explicite, nous pouvons spécifier une condition sur la valeur de TT pour extraire les données correspondantes à une version spécifique ou à un ensemble de versions en utilisant les valeurs appropriées de $ST_n$ et $ET_n$.

Pour une requêtes simple sur une seule version de schéma $SV_n$, $ST_n$ représente la date de début et $ET_n$ représente la date de fin de la période de validité de la version spécifiée. Ainsi, $T_n = [ST_n, ET_n]$ représente la période de validité de la version de schéma $SV_n$. de même, pour une requêtes croisée sur plusieurs versions à évolution séquentielle, $ST_n$ représente la date de début de la première et $ET_n$ représente la date de fin de la dernière version.

Cette clause permet de définir avec précision la plage de validité temporelle des versions sur laquelle vous souhaitez effectuer la recherche, ce qui permet d'extraire les données pertinentes pour l'analyse ou la manipulation à effectuer. Nous présentons ci-après, deux requêtes respectivement simple et croisée pour illustrer les requêtes explicites dans le modèle GAMM.

Requête 1 : Affichage des montants des ventes par mois d'un client.

```
MATCH (m:month)<-[:date_month]-(:date)<-[:sale_date]-(s:sales)-[sale_cust
↪  ]->(c:customer)
WHERE ST2 <= s.TT < ET2
RETURN c.customer_id, m.month, sum(s.sales_amt)
ORDER BY m.month;
```

Table 5.4 – Résultats de la requête 1

| Customer_id | Month | Sales_Amount |
|---|---|---|
| Cus001 | July | 2700 |

La requête 1 nous permet d'obtenir les montants des ventes du client Cus001 pour chaque mois. Selon le paramètre TT, les résultats ne seront disponibles qu'à partir de la version 2, même si le même client possède des instances dans la version 1. Cela signifie que seules les données de la version 2 seront prises en compte pour cette requête.

Dans le language Cypher, la commande "MATCH" est utilisée pour spécifier les motifs de graphe que vous souhaitez rechercher dans la base de données. Aussi, la clause "RETURN $value_1$, ..., $value_n$, AGGREGATE_FUNCTION(attribute)" permet d'aggréger le résultat en fonction des valeurs $value_1$, ..., $value_n$. Nous utilisons cette clause pour spécifier les colonnes ou les attributs que nous souhaitons inclure dans la sortie de la requête, ainsi que les fonctions d'agrégation à appliquer à ces attributs si nécessaire.

En utilisant cette clause, nous pouvons sélectionner les valeurs que nous souhaitons afficher dans les résultats de notre requête, en spécifiant les colonnes ou attributs pertinents. Nous pouvons également appliquer des fonctions d'agrégation telles que SUM, COUNT, AVG, etc., à un attribut particulier pour obtenir des résultats agrégés.

Cela nous permet de personnaliser la sortie de notre requête en fonction de nos besoins spécifiques et de regrouper les résultats en fonction des colonnes ou attributs souhaités. Cette fonctionnalité est particulièrement utile dans un contexte de requête OLAP, où nous cherchons à effectuer des analyses multidimensionnelles et à agréger les données selon différents critères.

Requête 2 : Affichage des montants des ventes par mois des fournisseurs.

```
MATCH (m:month)<-[:date_month]-(:date)<-[:sale_date]-(s:sales)-[sale_sup]
↪  ->(s:supplier)
WHERE ST2 <= s.TT < ET3
RETURN s.supplier_id, m.month, sum(s.sales_amt)
ORDER BY m.month;
```

TABLE 5.5 – Résultats de la requête 2

| Supplier_id | Month | Sales_Amount |
|---|---|---|
| Sup001 | July | 1200 |
| Sup002 | July | 1500 |
| Sup001 | August | 2000 |
| Sup002 | August | 2500 |

Dans la requête 2, nous avons une requête croisée qui interroge simultanément la version 2 et la version 3. Cette requête vise à obtenir les montants des ventes des fournisseurs SUP001 et SUP002 pour chaque mois.

Le paramètre TT est utilisé pour filtrer les résultats en fonction de la période de validité des versions. La clause « ST2 <= s.TT < ST3 » spécifie que la valeur de TT doit être supérieure ou égale à la date de début de la version 2 (ST2) et strictement inférieure à la date de début de la version 3 (ST3). Cela permet d'inclure les données des deux versions dans les résultats de la requête.

#### 5.3.2.2 Requêtes implicites

Les requêtes implicites dans le modèle GAMM offrent la possibilité de parcourir toutes les instances liées aux entités spécifiées dans la requête, sans restriction de version. Cette caractéristique est rendue possible grâce à la capacité des bases de données orientées graphes à naviguer à travers les relations entre les entités.

Ce type de requêtes présente un avantage majeur : les requêtes croisées peuvent être formulées de manière simple et uniforme, indépendamment du nombre de versions de données. Elle permettent d'exploiter pleinement la puissance des bases de données de graphes en éliminant la nécessité de spécifier des contraintes de version dans la requête ou d'identifier les versions à interroger comme c'est le cas pour le modèle entité-relation. Les requêtes implicites dans le modèle.

Nous appelons requêtes croisées dans un système en multiversion celles qui ne sont pas capables d'extraire des données à partir de plusieurs versions et d'unifier le résultat.

Requête 3 : Affichage des montants des ventes d'un client par mois sur plusieurs versions.

```
MATCH (m:month)<-[:date_month]-(:date)<-[:sale_date]-(s:sales)-[sale_cust
↪  ]->(c:customer)
RETURN c.customer_id, m.month, sum(s.sales_amt)
ORDER BY m.month;
```

TABLE 5.6 – Résultats de la requête 3

| Customer_id | Month | Sales_Amount |
|---|---|---|
| Cus001 | June | 1800 |
| Cus001 | July | 2700 |

La requête 3 est essentiellement une version simplifiée de la requête 1, où le paramètre TT n'est pas spécifié. Les résultats retournent les montants des ventes du client CUS001 pour chaque mois, sans tenir compte des versions spécifiques.

Requête 4 : Affichage des montants des ventes par mois sur plusieurs versions.

```
MATCH (m:month)<-[:date_month]-(:date)<-[:sale_date]-(s:sales)
RETURN m.month, sum(s.sales_amt)
ORDER BY m.month;
```

TABLE 5.7 – Résultats de la requête 4

| Month | Sales_Amount |
|---|---|
| June | 1800 |
| July | 2700 |
| August | 4500 |

La requête 4 affiche les montants des ventes par mois, en considérant toutes les versions disponibles. Cette requête est relativement simple, mais dans une approche relationnelle, il serait nécessaire de consulter trois versions de la table des faits pour obtenir le même résultat. L'avantage de l'approche basée sur les graphes est donc la simplicité de la requête, malgré la présence de multiples versions de données.

## 5.4 Requêtes temporelles dans GAMM

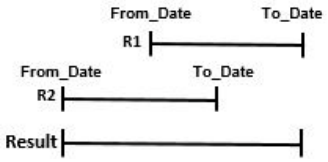
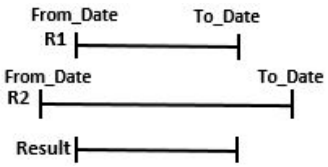
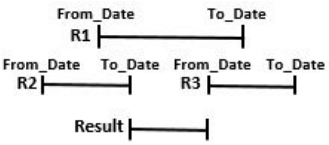
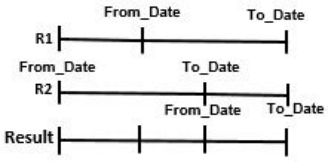
### 5.4.1 Principe des requêtes temporelles dans GAMM

Les requêtes temporelles dans le modèle GAMM offrent la possibilité de spécifier des critères de sélection afin d'obtenir des résultats précis, même en présence de l'évolution des données et des changements dans les dimensions, tels que les modifications des liens hiérarchiques. Dans la clause de filtrage, nous utilisons le paramètre temporel VT, qui représente le temps de validité des faits, pour les nœuds de type fait. De plus, nous utilisons les paramètres temporels de type intervalle de temps (FD/TD) pour les dimensions, permettant ainsi de filtrer les résultats.

En comparant le temps de validité des faits, représenté par VT, avec la période de validité des relations dans les dimensions, déterminée par les intervalles de temps T=[FD,TD], nous sommes en mesure de restreindre les résultats de la requête et d'obtenir des résultats cohérents, même en présence de changements dans les dimensions.

Le tableau 5.8 présente des exemples de cas pour les opérations temporelles dans les entrepôts de données, tels que présentés dans le chapitre 2, et illustre comment les conditions de sélection peuvent être utilisées pour filtrer les résultats en fonction du temps et de la période de validité des informations. Les conditions dans les requêtes sont exprimées en langage Cypher.

TABLE 5.8 – Opérations temporelles

| Opération temporelle | Exemple de cas | Condition dans la requête |
|---|---|---|
| Union temporelle |  | `WHERE MIN(R1.From_Date,R2.From_Date)<= sales.valid_date < MAX(R1.To_Date,R2.To_Date)` |
| Intersection temporelle |  | `WITH`<br>`  CASE WHEN R1.From_Date >= R2.From_Date THEN R1.From_Date ELSE R2.From_Date END as min_date,`<br>`  CASE WHEN R1.To_Date <= R2.To_Date THEN R1.To_Date ELSE R2.To_Date END as max_date`<br>`WHERE min_date <= sales.valid_date < max_date` |
| Différence temporelle |  | `WITH`<br>`  CASE WHEN R1.From_Date >= R2.To_Date THEN R1.From_Date ELSE R2.To_Date END as min_date,`<br>`  CASE WHEN R1.To_Date <= R3.From_Date THEN R1.To_Date ELSE R3.From_Date END as max_date`<br>`WHERE min_date <= sales.valid_date < max_date` |
| Agrégation temporelle |  | `WHERE R1.From_Date <= sales.valid_date < R1.To_Date`<br>`AND R2.From_Date <= sales.valid_date < R2.To_Date` |

Le tableau présente les différentes opérations temporelles dans les entrepôts de

données et leurs conditions correspondantes dans les requêtes comme suit :

1. Dans l'opération d'union temporelle qui permet de fusionner les intervalles de temps de deux ou plusieurs relations en un seul intervalle. La condition vérifie si la date de validité du fait Sales se trouve dans l'intervalle de temps couvert par les relations R1 et R2. Elle sélectionne les faits qui se trouvent dans au moins l'un des deux intervalles de temps.
2. Dans l'opération d'intersection temporelle qui consiste à trouver l'intersection entre les intervalles de temps de deux ou plusieurs relations. La condition détermine l'intervalle de temps d'intersection entre les relations R1 et R2, représenté par les variables *min_date* (date de début) et *max_date* (date de fin). Elle sélectionne les faits dont la date de validité se situe dans cet intervalle d'intersection.
3. Dans l'opération de différence temporelle qui consiste à trouver les faits qui sont valides dans un ou plusieurs intervalles de temps donnés mais non valides dans un autre ou plusieurs autres intervalles. La condition détermine l'intervalle de temps qui est dans la relation R1 et qui n'est pas dans les relation R2 et R3, représenté par les variables *min_date* (date de début) et *max_date* (date de fin). Elle sélectionne les faits dont la date de validité se situe dans cet intervalle de différence.
4. Dans l'opération d'agrégation temporelle qui permet d'agréger les faits qui sont valides dans plusieurs intervalles de temps. La condition spécifie que le fait « sales » doit être valide dans les deux intervalles de temps des relations R1 et R2. Elle sélectionne les faits qui satisfont ces conditions de validité simultanée.

En utilisant ces conditions dans les requêtes, les résultats sont filtrés en fonction du temps et des périodes de validité des informations, permettant ainsi d'obtenir des résultats pertinents malgré l'évolution des données et les changements dans les dimensions.

### 5.4.2 Exemples de requêtes temporelles dans GAMM

Nous présentons ci-dessous les exemples d'instances de notre exemple de motivation qui traitent de l'évolution temporelle des données (voir la sous-section 3.2.2), et leur représentation à l'aide de notre approche GAMM est indiquée dans la figure 5.1.

La figure 5.2 illustre la relation entre la cliente Mary Saveley avec l'identifiant Cust001 et le niveau Paris avec l'identifiant City001, qui est valide dans l'intervalle de temps allant du 01/01/2020 au 01/08/2020. En revanche, la relation entre la même cliente et le niveau Lyon avec l'identifiant City002 est valable à partir du 01/08/2020 et reste valide jusqu'à aujourd'hui.

De plus, la relation hiérarchique entre le produit Mozzarella avec l'identifiant Prod001 et le niveau Fresh avec l'identifiant Categ001 est valide pendant la période allant du 01/01/2020 au 01/04/2020. Ensuite, la relation entre le même produit et le niveau Dairy avec l'identifiant Categ002 est valide à partir du 01/04/2020 jusqu'à la date actuelle.

La figure 5.2 illustre les périodes de validité des relations impliquant les niveaux Ville et Catégorie, où les intersections des différentes périodes de validité créent trois intervalles de temps notés $T_1$, $T_2$ et $T_3$.

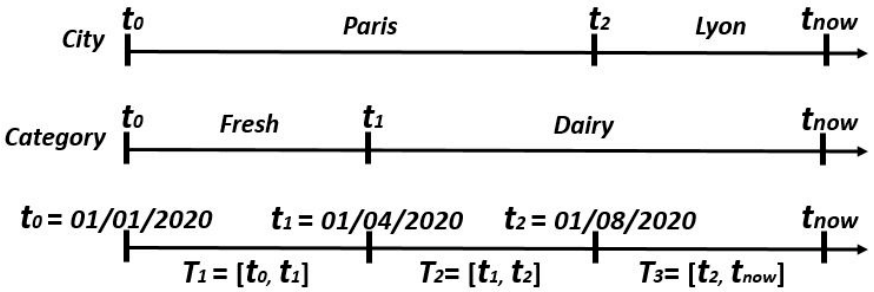



Figure 5.2 – Chronologie de l'évolution des niveaux City et Category

Nous proposons ci-après, à titre illustratif, trois requêtes utilisant l'agrégation temporelle pour les niveaux suivants : (i) le niveau City, (ii) le niveau Category, et (iii) les niveaux City et Category conjointement. Les requêtes sont rédigées dans le langage de requête Cypher.

Requête 5 : Affichage des montants des ventes par client et par ville.

```
MATCH (cn:city_name)<-[]-(:city)<-[r:customer_city]-(c:customer)<-[]-(s:s⌋
↪  ales)
WHERE r.From_Date <= s.valid_date < r.To_Date
RETURN c.customer_id, cn.city_name, SUM(s.sales_amount)
```

Table 5.9 – Résultats de la requête 5

| Customer_ID | City_Name | Sales_Amount |
|---|---|---|
| Cust001 | Paris | 3700 |
| Cust001 | Lyon | 3800 |

La requête 5 permet d'obtenir les montants des ventes par ville pour le client Cust001. Les résultats sont présentés dans le tableau 5.9. Les montants des ventes sont agrégés sur la période de validité de la relation entre le client et la ville, telle que spécifiée dans la condition de la requête.

La requête permet ainsi d'analyser l'évolution des montants des ventes pour le client Cust001 dans différentes villes au fil du temps. Cela permet d'identifier les villes qui ont généré les ventes les plus élevées pour ce client et de comprendre les variations des performances de vente dans chaque ville.

Requête 6 : Affichage des montants des ventes par produit et par catégorie.

```
MATCH (cn:category_name)<-[]-(:category)<-[r:product_category]-(p:product
↪  )<-[]-(s:sales)
WHERE r.From_Date <= s.valid_date < r.To_Date
RETURN p.product_id, cn.category_name, SUM(s.sales_amount)
```

TABLE 5.10 – Résultats de la requête 6

| Product_ID | Category_Name | Sales_Amount |
|---|---|---|
| Prod001 | Fresh | 1500 |
| Prod001 | Dairy | 6000 |

La requête 6 permet d'obtenir les montants des ventes pour chaque catégorie du produit PROD001. Les résultats sont affichés dans le tableau 5.10. Les montants des ventes sont agrégés sur la période de validité de la relation entre le produit et la catégorie, telle que spécifiée dans la condition de la requête.

La requête permet de visualiser les performances de vente du produit PROD001 dans chaque catégorie au fil du temps et d'identifier les catégories qui ont généré les ventes les plus élevées pour ce produit.

Requête 7 : Affichage des montants des ventes par ville et par catégorie.

```
MATCH (ctn:city_name)<-[]-(:city)<-[r1:customer_city]-(c:customer)<-[]-(s
↪  :sales)-[]->(p:product)-[r2:product_category]->(:category)-[]->(cgn:c
↪  ategory_name)
WHERE r1.From_Date <= s.valid_date < r1.To_Date
AND   r2.From_Date <= s.valid_date < r2.To_Date
RETURN ctn.city_name, cgn.category_name, SUM(s.sales_amount)
```

TABLE 5.11 – Résultats de la requête 7

| City_Name | Category_Name | Sales_Amount |
|---|---|---|
| Paris | Fresh | 1500 |
| Paris | Dairy | 2200 |
| Lyon | Dairy | 3800 |

La requête 7 permet d'obtenir les montants des ventes pour les différentes villes et les différentes catégories. Les résultats sont présentés dans le tableau 5.11. Les montants des ventes sont agrégés sur la période de validité des relations entre le client et la ville, ainsi que entre le produit et la catégorie, telles que spécifiées dans les conditions de la requête.

Cette requête permet de faire une analyse croisée des ventes en considérant à la fois les villes et les catégories, ce qui permet d'obtenir des informations plus détaillées sur la performance des ventes pour différentes combinaisons de ville et de catégorie.

L'utilisation de l'étiquetage temporel de type horodatage ponctuel dans les faits et de l'étiquetage temporel de type intervalle de temps dans les relations des entités de dimensions présente plusieurs avantages pour obtenir des résultats précis dans les requêtes temporelles, même en cas de changements apportés aux instances de ces dimensions.

Tout d'abord, l'étiquetage temporel de type horodatage ponctuel permet de capturer avec précision le moment précis auquel un fait a eu lieu. Cela signifie que chaque fait est associé à une valeur temporelle spécifique, ce qui permet de filtrer les données en fonction de périodes spécifiques ou de points temporels précis.

D'autre part, l'étiquetage temporel de type intervalle de temps dans les relations des entités de dimensions permet de représenter les périodes de validité des relations entre les entités. Par exemple, dans le cas d'une relation entre un client et une ville, cette approche permet de spécifier la période pendant laquelle le client a résidé dans une certaine ville.

En utilisant ces informations temporelles dans les requêtes, il devient possible d'effectuer des opérations de filtrage, d'agrégation et de jointure basées sur des critères temporels précis. Par exemple, il est possible de récupérer les faits qui se sont produits dans une période donnée, d'agréger les valeurs des faits sur des périodes spécifiques ou de réaliser des jointures entre les entités en tenant compte de leurs périodes de validité.

La capacité des bases de données orientées graphes à utiliser des informations dans les relations entre les entités permet donc d'obtenir des résultats précis et fiables lors de l'exécution de requêtes temporelles. Cela offre une flexibilité et une précision accrues lors de l'analyse des données temporelles, ce qui peut être particulièrement bénéfique dans des domaines tels que l'analyse des tendances, la prévision et l'historique des données.

## 5.5 Conclusion

L'interrogation des versions et des données revêt une grande importance dans la gestion des entrepôts de données évolutifs en multi-versions. Dans ce chapitre, nous avons exploré différents aspects liés à cette interrogation dans le contexte du modèle GAMM.

Nous avons abordé l'extraction de schéma pour les différentes versions existantes dans un modèle GAMM, ce qui permet de comprendre la structure et l'évolution des données au fil du temps. De plus, nous avons étudié l'interrogation des instances de données à partir des versions de schéma, ce qui offre la possibilité d'extraire des informations spécifiques à une ou plusieurs versions données.

La flexibilité offerte par GAMM pour formuler des requêtes explicites et implicites. Les requêtes explicites nous permettent de spécifier des critères de recherche sur des versions précises, tandis que les requêtes implicites nous permettent de parcourir toutes les instances liées aux entités spécifiées, indépendamment de la version.

Enfin, nous avons examiné l'interrogation des données à évolution temporelle, en mettant en évidence la capacité du modèle GAMM à analyser les changements et les tendances au fil du temps en effectuant des requêtes qui prennent en compte les périodes de validité des données. En spécifiant des intervalles de temps, nous sommes en mesure d'obtenir des informations sur l'évolution des données et de les comparer entre différentes versions de schéma. Ainsi, nous avons pu observer les variations et les transformations des données au cours du temps.

Dans l'ensemble, GAMM offre une approche puissante pour interroger efficacement les entrepôts de données évolutifs en multi-versions. Cette approche facilite l'exploration des données, la compréhension de leur évolution dans le temps et la prise de décisions basée sur des informations historiques.

CHAPITRE 6

# Évolution automatique des schémas

## Sommaire

## 6.1 Introduction

Dans les chapitres précédents, nous avons présenté le modèle multidimensionnel agile à base de graphes GAMM pour la gestion des big data. Ce modèle permet l'évolution temporelle des schémas et des données dans les entrepôts de données (voir chapitre 3). Nous avons examiné les fonctions et les opérateurs qui permettent l'évolution du modèle GAMM (voir chapitre 4), ainsi que les techniques d'interrogation des schémas et des données (voir chapitre 5).

Le modèle GAMM est conçu pour être évolutif en multi-versions ; il crée une nouvelle version du schéma à chaque changement, et ce à l'aide d'opérateurs d'évolution. Cette évolutivité permet la prise en compte de nouvelles sources de données et/ou de nouveaux besoins d'analyse. Chaque version du schéma correspond à un ensemble d'instances de données stockées dans l'entrepôt de données en graphe.

Les opérateurs d'évolution du modèle GAMM comprennent l'ajout ou la suppression d'entités multidimensionnelles telles que les faits, les mesures, les dimensions, les niveaux de hiérarchie et les attributs. Ces opérations permettent d'accroître le potentiel analytique du modèle. Cependant, jusqu'à présent, les évolutions ont principalement été réalisées de manière manuelle, sur la base des décisions de l'utilisateur du modèle GAMM.

Néanmoins, il est possible que certaines interconnexions fonctionnelles ne soient pas visibles dans un entrepôt de données existant en raison de ses caractéristiques ou des évolutions pouvant intégrer de grandes quantités de données hétérogènes.

La combinaison du processus d'entreposage des données et des techniques de fouille de données offre un potentiel considérable pour améliorer le traitement des données agrégées. En effet, certaines techniques d'exploration de données peuvent être utilisées comme opérateurs d'agrégation pour découvrir des modèles cachés dans les données Bentayeb and Favre (2009). De plus, en raison de leur capacité à extraire des connaissances à partir des données, les techniques de fouille de données sont utilisées dans différentes approches pour la détection automatique de schémas multidimensionnels à partir de diverses sources de données.

Il existe plusieurs travaux de recherche dans la littérature qui ont proposé des approches automatisées ou semi-automatisées pour détecter des structures globales ou partielles et créer des schémas multidimensionnels à partir de différentes sources de données. Ces sources comprennent les bases de données relationnelles Bimonte et al. (2017); Jensen et al. (2004); Song et al. (2007), les documents XML Golfarelli et al. (2001) et les données tabulaires ou les feuilles de calcul Sanprasit et al. (2021); Yang et al. (2022). Par ailleurs, les travaux présentés dans le chapitre 2 ont principalement porté sur les approches d'évolution des modèles multidimensionnels. Ces approches permettent aux concepteurs d'intégrer de nouvelles sources de données et de prendre en compte les nouveaux besoins des utilisateurs pour enrichir les possibilités d'analyse.

Les approches mentionnées précédemment proposent soit (i) la génération initiale

d'un schéma multidimensionnel à partir de sources de données spécifiques, soit (ii) des modèles d'évolution de schémas multidimensionnels. Cependant, il n'existe pas d'approche globale permettant l'évolution automatique des schémas en plusieurs versions, impliquant ainsi simultanément plusieurs entités d'un schéma multidimensionnel dans le processus d'évolution hiérarchique pour obtenir un résultat optimal.

Dans ce chapitre, nous proposons une approche automatique d'évolution des schémas dans les entrepôts de donnée multi-versions basés sur des graphes en utilisant le modèle GAMM. Notre processus consiste à identifier de nouveaux niveaux d'hiérarchies en se basant sur des clusters détectés, et à créer de nouvelles versions de schémas tout en préservant les anciennes.

Notre approche est entièrement automatisée et peut être appliquée après chaque intégration de nouvelles sources de données, permettant ainsi de procéder à des traitements pour détecter les connaissances cachées. Grâce à la formalisation en graphe pondéré, où les liens ont un poids, notre approche prend simultanément en compte plusieurs dimensions et mesures lors de la détection des communautés. Cela nous permet d'obtenir une excellente qualité de partitionnement et de créer des niveaux d'hiérarchies sur plusieurs dimensions à la fois.

L'objectif de notre approche est de faciliter le processus d'évolution des schémas dans un entrepôt de données multi-versions basé sur le modèle GAMM. Elle vise à renforcer la flexibilité du modèle et à accroître automatiquement son potentiel analytique. En utilisant notre approche, l'utilisateur est assisté dans ces tâches, ce qui permet d'adapter et d'enrichir le schéma de manière plus efficace et automatisée.

## 6.2 Motivation

Pour concrétiser notre approche, nous prenons pour exemple un schéma conceptuel représenté selon le formalisme *Dimensional Fact Model* (DFM), illustré dans la FIGURE 6.1. Ce modèle multidimensionnel incarne un réseau de publications de recherches scientifiques, rassemblant des données issues de sites spécialisés tels que DBLP (*Digital Bibliography and Library Project*), ACM (*Association for Computing Machinery*) et MAG (*Microsoft Academic Graph*). Ces sources sont fusionnées en une seule base de données, formant ainsi le modèle. Les données sources sont principalement textuelles, et le modèle résultant est conçu pour satisfaire une gamme variée de requêtes OLAP.

Ce schéma se compose essentiellement de cinq dimensions : PAPER, AUTHOR, KEYWORD, VENUE et DATE, associées à quatre faits : PAPER_PUBLIED, AUTHOR_PUBLISH, KEYWORD_USED et FIELD_TREATED. Une particularité à noter est que les articles de la dimension PAPER ne sont publiés qu'une seule fois. Toutefois, afin de maintenir le niveau de granularité le plus fin des mesures présentes dans le fait PAPER_PUBLIED, cette dimension est conservée dans le schéma.

L'intérêt principal de cette représentation réside dans les liens implicites entre les

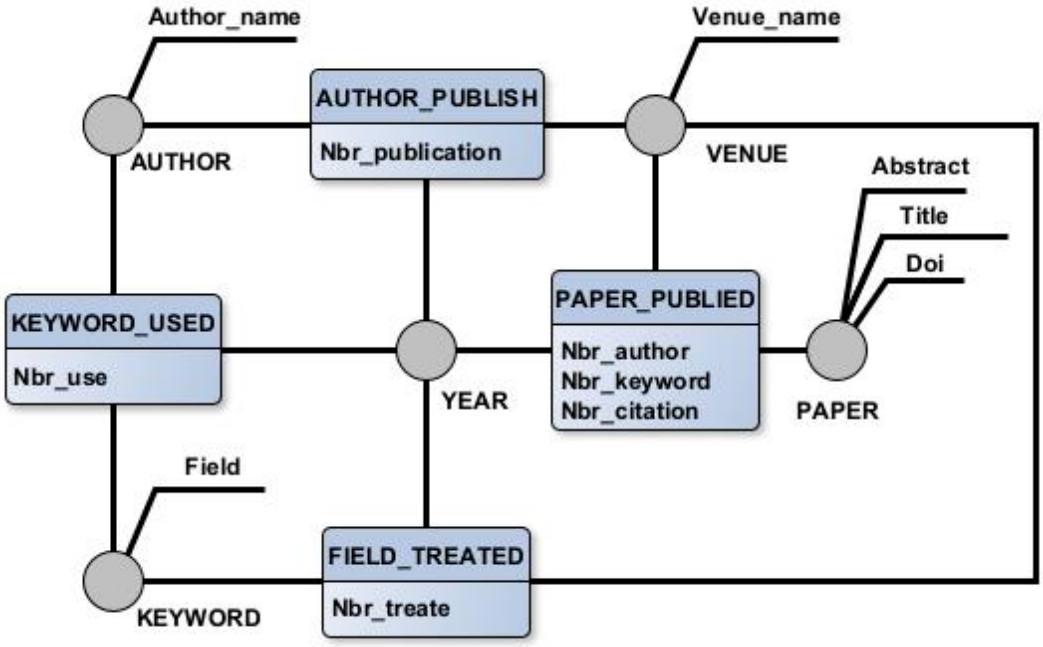



FIGURE 6.1 – Schéma d'un modèle multidimensionnel pour un réseau bibliographique de recherches scientifiques

différentes données. Par exemple, bien que nous ne nous focalisions pas uniquement sur les liens directs entre les entités, les co-auteurs partageant des publications communes, il existe des similarités entre les auteurs travaillant sur des thèmes similaires, des lieux partageant des domaines d'expertise, et ce, sur une période de temps donnée. Nous présentons ici un exemple de relations pondérées qui peuvent être déduites de cette représentation.

Exemples de relations :

— Relation 1 : Nombre de publications d'un auteur dans un lieu (conférence/journal, etc.) : Exemple (Auteur = 'Fadila BENTAYEB, Lieu = 'DAWAK', Poids = '5').

— Relation 2 : Nombre de fois qu'un mot-clé est traité dans un lieu : Exemple (Lieu = 'DAWAK', Mot-clé = 'Big Data', Poids = '27').

— Relation 3 : Nombre de publications d'un auteur au cours d'une année donnée : Exemple (Auteur = 'Fadila BENTAYEB, Année = '2015', Poids = '3').

— Relation 4 : Nombre de fois où un mot-clé a été traité au cours d'une année donnée : Exemple (Mot-clé = 'Big Data', Année = '2015', Poids = '12').

— Relation 5 : Nombre de fois qu'un mot-clé a été utilisé par un auteur : Exemple (Auteur = 'Fadila BENTAYEB', Mot-clé = 'Big Data', Poids = '12').

— Relation 6 : Nombre de publications dans un lieu au cours d'une année donnée : Exemple (Lieu = 'DAWAK', Année = '2015', Poids = '24').

Ces liens implicites apportent un supplément d'informations aux données, facilitant une exploration approfondie pour la découverte de clusters au sein des entités interconnectées. Cette détection de cluster s'avère crucial pour déceler des niveaux de hiérarchie et des structures latentes dans les données agrégées.

L'exploration de ces relations à l'aide de techniques de fouille de données, notamment le regroupement (*clustering*), permet de détecter des clusters regroupant des entités similaires. Ces regroupements peuvent être considérés comme des niveaux de hiérarchie et serviront à organiser et structurer les données de manière à faciliter des analyses approfondies et pertinentes.

En effet, les techniques d'exploration de données jouent un rôle crucial dans la détection automatique de schémas multidimensionnels à partir de diverses sources de données. Leur capacité analytique est exploitée pour extraire des connaissances à partir des données, facilitant ainsi l'identification de nouvelles dimensions, hiérarchies et mesures. Cette capacité permet à notre approche d'évolution de schéma de s'adapter dynamiquement aux nouvelles sources de données et aux évolutions des tendances dans les données agrégées.

Le modèle GAMM, avec sa structure de graphes et son concept d'évolution en multi-versions, offre la possibilité d'utiliser des techniques de partitionnement de graphes dans le processus d'évolution. Cela peut aider l'utilisateur à détecter de nouvelles structures, et à mieux comprendre les connexions entre les entités.

Dans ce contexte, nous avons développé une approche d'évolution automatique des schémas dans le modèle GAMM. Ce processus permet d'identifier de nouveaux niveaux d'hiérarchies en se basant sur les thèmes des clusters détectés (voir section 6.3). En appliquant les règles de passage, décrites dans le chapitre 3.5, au schéma de notre exemple, nous obtenons le schéma en graphe représenté dans la FIGURE 6.2.

Il convient de souligner que, dans un souci de simplification du schéma, nous avons choisi de regrouper les concepts métiers des dimensions ainsi que les descripteurs (attributs) au sein d'un même nœud. Cette configuration schématique sera utilisée tout au long de ce chapitre comme exemple pour illustrer notre approche.

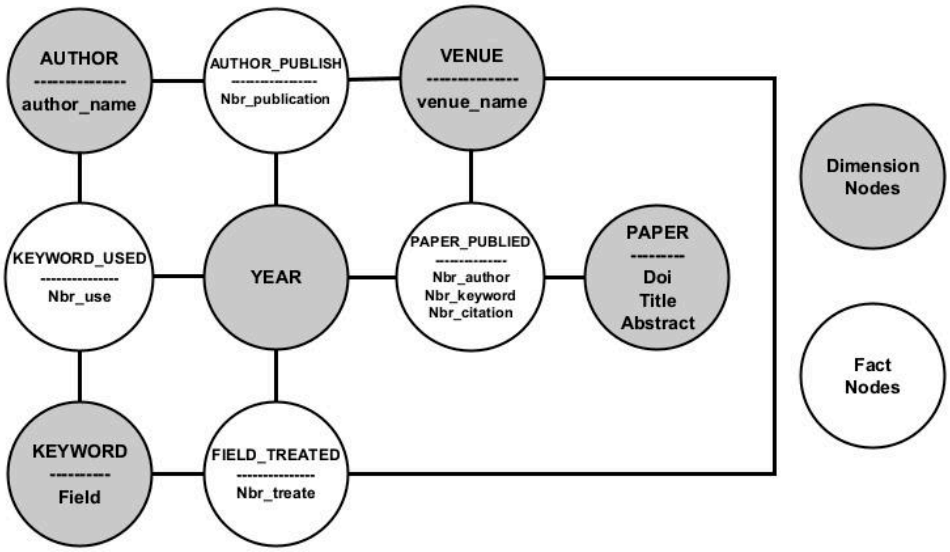



FIGURE 6.2 – Schéma logique du modèle multidimensionnel représentant un réseau bibliographique de recherches scientifiques.

## 6.3 Approche d'évolution automatique des schémas dans GAMM

La figure 6.3 offre un aperçu de l'approche d'évolution automatique des schémas au sein du modèle GAMM. Ce processus vise à identifier de nouveaux niveaux d'hiérarchies en se basant sur les thèmes des clusters détectés. À la fin de ce processus, une nouvelle version de notre entrepôt de données est créée en accord avec les fonctions d'évolution établies dans la section 4.2, tandis que l'ancienne version est conservée pour des raisons historiques.

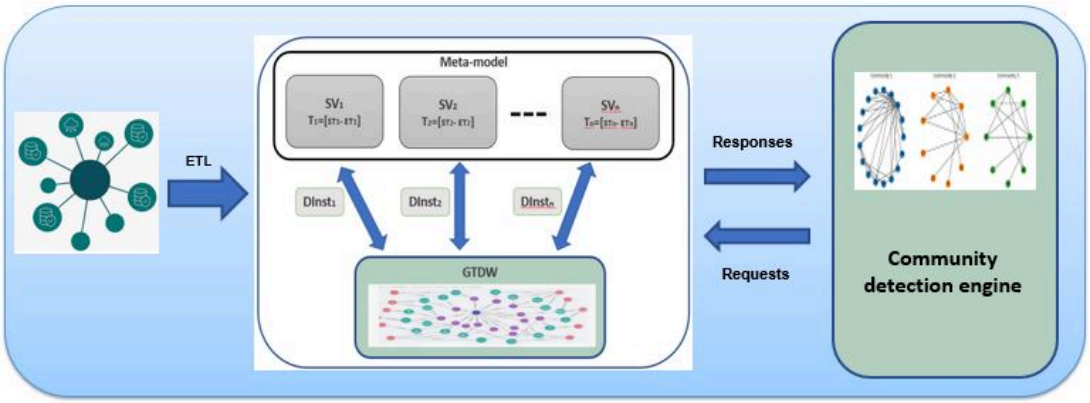



FIGURE 6.3 – Approche d'évolution automatique des schémas dans GAMM

Le processus d'évolution automatique des schémas au sein de modèle GAMM s'articule selon trois étapes principales : le chargement du graphe n-parties pondéré dont l'ensemble de sommets peut être partitionné en n sous-ensembles, la détection des nouveaux niveaux d'hiérarchies, et l'exploration des niveaux obtenus ainsi que la création de nouvelles versions de schémas de l'entrepôt de données en graphe.

La première étape : le chargement du graphe n-parties pondéré, consiste en la récupération et la transformation des données provenant de diverses sources en un graphe pondéré. Ce graphe représente de manière explicite les entités ainsi que les relations intrinsèques entre les différentes dimensions du schéma multidimensionnel. La création de ce graphe est fondamentale car elle offre la base nécessaire pour la détection des clusters et prépare le terrain pour l'évolution subséquente des schémas (voir sous-section 6.3.1).

La deuxième étape : elle est centrale dans le processus ; elle englobe la détection de clusters au sein du graphe. L'exploration de données et l'analyse des liens entre ces données sont utilisées pour identifier des regroupements d'entités partageant des similarités ou des liens significatifs. Ces clusters représentent des ensembles de données cohérents, révélant ainsi de nouveaux niveaux de hiérarchies pertinents pour le schéma multidimensionnel. Cela ajoute une dimension aux données, favorisant des analyses plus approfondies (voir sous-section 6.3.2).

Enfin, la troisième étape consiste à explorer les niveaux d'hiérarchies nouvellement identifiés et à créer de nouvelles versions de schémas. Une analyse minutieuse des caractéristiques des entités au sein de chaque cluster permet de déterminer les dimensions pertinentes pour chaque groupe. En se basant sur cette analyse, le schéma existant est adapté en intégrant les nouveaux niveaux d'hiérarchies dans les dimensions existantes. Ces ajustements adaptent le schéma aux besoins spécifiques des utilisateurs (voir sous-section 6.3.3).

En somme, l'approche d'évolution automatique des schémas dans GAMM tire profit des avantages combinés de l'exploration de données et de l'architecture en graphes pour offrir une approche itérative et flexible pour la structuration des données et la révélation de nouveaux niveaux d'hiérarchies.

### 6.3.1 Sélection du graphe n-parties pondéré

La première étape fondamentale de notre approche d'évolution automatique des schémas dans le cadre du modèle GAMM consiste à créer et à charger un graphe pondéré à n-parties. Cette démarche repose sur les principes de la théorie des graphes, où les sommets du graphe sont judicieusement répartis en $n$ ensembles indépendants.

Dans notre contexte, ces ensembles correspondent directement aux dimensions constitutives du schéma conceptuel de notre entrepôt de données, tandis que les interactions et les liens entre ces dimensions sont quantifiés sous forme de poids spécifiques sur les arêtes du graphe. La représentation schématique d'un tel graphe pondéré à n-parties est illustrée dans la figure 6.4.

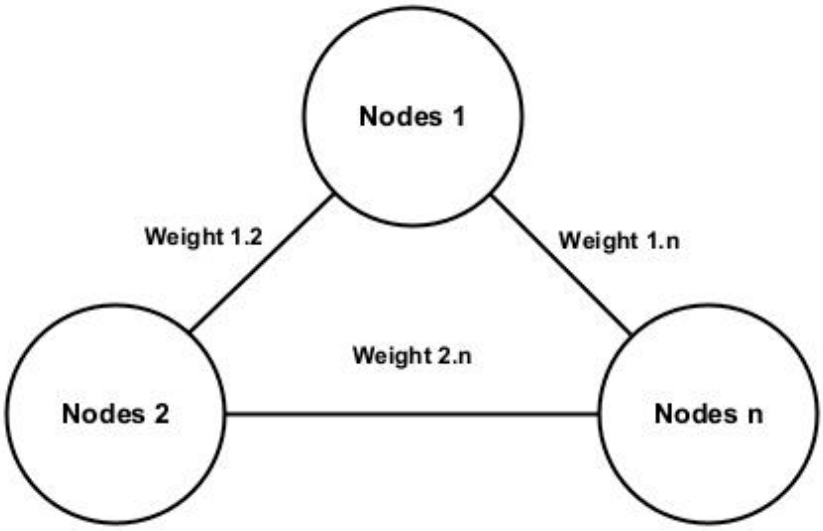



FIGURE 6.4 – Schéma logique d'un graphe pondéré n-parties

Cette phase permet de construire une vision inter-connectée et riche de l'entrepôt de données. Prenons pour exemple un réseau de chercheurs scientifiques afin d'appréhender plus clairement ce processus. Chaque dimension, telle que AUTHOR, VENUE, KEYWORD, ou DATE, se voit attribuer un ensemble indépendant de sommets dans le graphe. Les interactions entre ces dimensions, qui sont en réalité des agrégations basées sur des mesures telles que le nombre de publications d'un auteur dans un lieu donné ou la fréquence d'utilisation d'un mot-clé dans un lieu spécifique, sont symbolisées par les arêtes du graphe. Cette approche permet de détecter des relations indirectes et des similitudes sous-jacentes entre les entités, au-delà des simples liens directs entre les nœuds.

Pour illustrer cette étape, nous allons reprendre notre exemple de motivation basé sur le réseau de chercheurs scientifiques. Le graphe pondéré résultant est représenté dans la Figure 6.5. Voici comment les poids sont définis dans notre exemple :

Weight 1 : Nombre de publications d'un auteur dans une lieu (conférence/journal...etc.) ;

Weight 2 : Nombre de fois qu'un mot-clé est traité dans un lieu ;

Weight 3 : Nombre de publications d'un auteur au cours d'une année donnée ;

Weight 4 : Nombre de fois où un mot-clé a été traité au cours d'une année donnée ;

Weight 5 : Nombre de fois qu'un mot-clé a été utilisé par un auteur ;

Weight 6 : Nombre de publications dans un lieu au cours d'une année donnée.

Dans cet exemple, chaque type d'interaction reçoit des poids distincts. Par exemple, le poids 1 exprime le nombre de publications d'un auteur dans un lieu spécifique, tandis

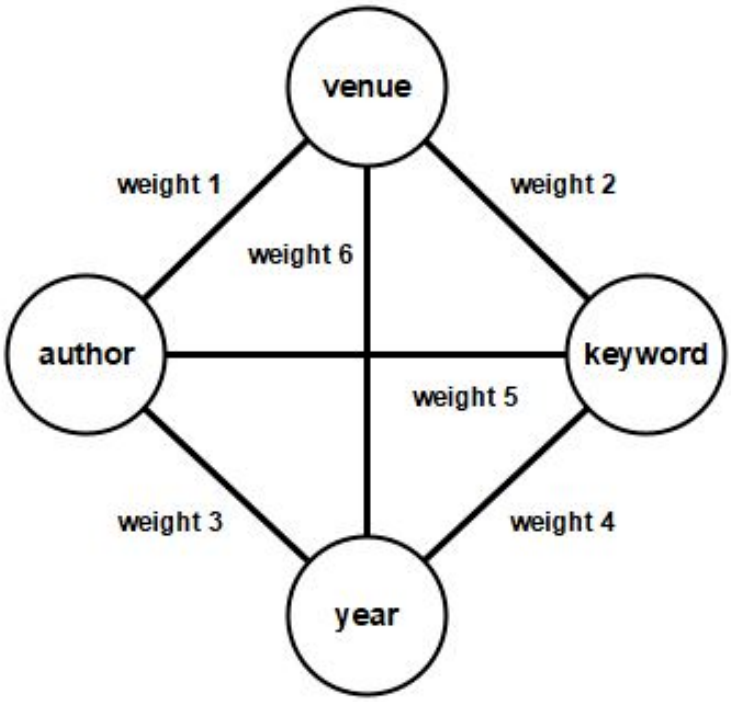



FIGURE 6.5 – Graphe pondéré (4-parties) basé sur quatre dimensions

que le poids 2 représente la fréquence d'utilisation d'un mot-clé dans un lieu donné, et ainsi de suite. Cette attribution pondérée instaure une perspective multidimensionnelle des liens implicites inter-dimensions, ouvrant la voie à des explorations approfondies et complexes des données agrégées.

Cette première étape de création et de chargement du graphe pondéré à n-parties marque une avancée significative dans notre manière d'appréhender les interactions multidimensionnelles au sein de l'entrepôt de données. Elle jette les bases nécessaires pour la détection de nouveaux niveaux de hiérarchies et pour la progression ultérieure de l'évolution des schémas.

Cette représentation offre la possibilité de classifier les instances d'une dimension selon un profil établi, caractérisés par les poids d'interaction avec les autres dimensions plutôt que par les interactions intra-dimensionnelles. Par exemple, les auteurs peuvent être regroupés dans un même cluster en fonction de leurs domaines de publications, allant au-delà de leurs collaborations directes. De plus, notre approche permet de classifier des nœuds variés tels que les auteurs, les lieux et les mots-clés dans un même cluster, facilitant ainsi une meilleure caractérisation de ces clusters .

### 6.3.2 Détection des clusters et identification des niveaux de hiérarchie

La deuxième étape de notre approche consiste à détecter les clusters dans le graphe pondéré généré lors de la première étape. Pour cela, nous avons utilisé des algorithmes de détection de communautés basés sur la mesure de modularité (voir sous-section 2.2.7).

La modularité est une mesure de la qualité du partitionnement d'un graphe en communautés. Elle est largement utilisée dans l'analyse des réseaux sociaux et a été introduite par M. E. J. Newman Newman (2006). L'idée de la modularité est d'évaluer la déviation entre le nombre d'arêtes réelles observées entre les nœuds d'une même communauté et le nombre d'arêtes attendues si les arêtes étaient réparties de manière aléatoire. Une valeur élevée de modularité indique une structure communautaire significative dans le graphe.

La formule de la modularité pour un partitionnement $c$ est donnée par :

$$Q = \frac{1}{2m} \sum_{ij} \left[ A_{ij} - \frac{k_i \cdot k_j}{2m} \right] \delta\left(c_i, c_j\right),$$

où :
$A_{ij}$ représente le poids de l'arête entre les nœuds $i$ et $j$.
$k_i$ et $k_j$ représentent la somme des poids des arêtes liées aux nœuds $i$ et $j$ respectivement.
$m$ représente la somme de tous les poids des arêtes du graphe.
$c_i$ et $c_j$ représentent les communautés des nœuds $i$ et $j$ respectivement.
$\delta$ représente le delta de Kronecker.

L'objectif est de trouver le partitionnement qui maximise la modularité, ce qui permet d'identifier les communautés les plus significatives dans le graphe. Les algorithmes de détection de communautés cherchent donc à optimiser cette fonction en explorant différentes combinaisons de partitions des nœuds du graphe (voir sous-section 2.2.7).

Dans notre approche, les communautés détectées représentent des regroupements significatifs d'entités multidimensionnelles et nous permettent de définir de nouveaux niveaux de hiérarchies dans le schéma de l'entrepôt de données.

Cependant, étant donné que nous traitons des graphes volumineux dans le contexte du big data, nous avons choisi d'utiliser la méthode de Louvain basée sur la modularité pour détecter les communautés. Cette méthode est un algorithme hiérarchique sans recouvrement qui maximise la modularité et permet un regroupement hiérarchique des communautés sous la forme d'un dendrogramme Blondel et al. (2008). L'algorithme de Louvain est connu pour sa capacité à traiter efficacement les grands réseaux en termes de temps de calcul tout en fournissant une bonne qualité de partitionnement Newman

(2016).

La méthode de Louvain se compose de deux phases itératives pour maximiser la modularité. Dans la première phase, chaque nœud du réseau est initialement assigné à sa propre communauté. Ensuite, pour chaque nœud $i$, on évalue le gain de modularité obtenu en retirant $i$ de sa communauté actuelle et en le déplaçant dans la communauté de chacun de ses voisins $j$. Le calcul du gain de modularité $\Delta Q$ se fait en utilisant la formule suivante :

$$\Delta Q = \left[\frac{\Sigma_{in} + 2k_{i,in}}{2m} - \left(\frac{\Sigma_{tot} + k_i}{2m}\right)^2\right] - \left[\frac{\Sigma_{in}}{2m} - \left(\frac{\Sigma_{tot}}{2m}\right)^2 - \left(\frac{k_i}{2m}\right)^2\right],$$

où : $\Sigma_{in}$ représente la somme des poids des liens à l'intérieur de la communauté vers laquelle le nœud $i$ se déplace.
$\Sigma_{tot}$ représente la somme des poids des liens vers les nœuds de la communauté vers laquelle le nœud $i$ se déplace.
$k_i$ représente le degré pondéré du nœud $i$.
$k_{i,in}$ représente la somme des poids des liens entre le nœud $i$ et les autres nœuds de la communauté vers laquelle le nœud $i$ se déplace.
$m$ représente la somme des poids de tous les liens du réseau.

Cette étape itérative est répétée jusqu'à ce qu'il n'y ait plus de gain de modularité significatif ou qu'une condition d'arrêt prédéfinie soit atteinte. À la fin de cette phase, nous obtenons un partitionnement initial du réseau en communautés.

La seconde phase consiste à agréger les nœuds qui appartiennent à la même communauté dans un nouveau graphe, appelé graphe agrégé. Les nœuds du graphe agrégé représentent les communautés détectées dans la première phase, et les arêtes sont pondérées par la somme des poids des liens entre les communautés correspondantes. Cette phase est ensuite répétée jusqu'à ce qu'aucun gain de modularité supplémentaire ne soit obtenu. Le résultat final est un dendrogramme qui représente la hiérarchie des communautés dans le graphe initial.

L'algorithme de Louvain offre un compromis entre l'efficacité de calcul et la qualité du partitionnement, ce qui en fait une méthode adaptée pour détecter les communautés dans les grands graphes utilisés dans notre approche d'évolution automatique des schémas.

Ensuite, une fois cette valeur calculée pour toutes les communautés auxquelles $i$ est connecté, $i$ est placé dans la communauté qui a permis la plus grande augmentation de modularité. Si aucune augmentation n'est possible, $i$ reste dans sa communauté d'origine. Ce processus est appliqué de manière répétée et séquentielle à tous les nœuds jusqu'à ce qu'aucune augmentation de modularité ne puisse se produire.

Dans la deuxième phase, l'algorithme regroupe tous les nœuds de chaque communauté et construit un nouveau réseau dont les nœuds sont les communautés de la phase précédente. Tous les liens entre plusieurs nœuds d'une même communauté et un nœud d'une communauté différente sont représentés par des arêtes pondérées entre les communautés. Une fois le nouveau réseau créé, la deuxième phase est terminée et la première phase peut être réappliquée au nouveau réseau.

Une fois qu'il n'y a plus de gain en terme de modularité, l'algorithme se termine et le partitionnement de notre graphe est fait. Le résultat sera exploité dans la troisième étape qui consiste à explorer les communautés et créer le nouveau schéma (voir sous-section 6.3.3).

### 6.3.3 Exploration des communautés et création de nouveaux schémas

La troisième étape de notre approche d'évolution automatique des schémas consiste à explorer en détail les clusters détectés et à créer de nouveaux schémas adaptés pour l'entrepôt de données. Pour ce faire, nous initions des requêtes OLAP spécifiques pour chaque cluster hétérogène, contenant des entités appartenant à plusieurs dimensions. Cette démarche vise à mieux caractériser ces regroupements d'entités et à élaborer de nouveaux niveaux de hiérarchie pour enrichir l'analyse en ligne. Cette flexibilité dans l'évolution du schéma est rendue possible grâce à l'architecture même du modèle GAMM ainsi qu'aux fonctions d'évolution que nous avons définies.

L'exploration approfondie des communautés (qui sont les clusters d'entités) permet de dégager des modèles, des tendances et des relations spécifiques à chacune d'elles. En employant des requêtes OLAP, nous agrégeons, filtrons et analysons les données propres à chaque cluster, en nous focalisant sur les dimensions et les mesures les plus pertinentes pour chaque cas. Cela nous offre une compréhension plus profonde des particularités et des caractéristiques propres à chaque groupe d'entités au sein du graphe.

Une fois que nous avons minutieusement exploré les clusters et acquis une meilleure appréhension de leurs dynamiques, nous sommes en mesure de créer de nouveaux schémas pour l'entrepôt de données. Ces nouvelles versions de schémas sont élaborées en intégrant de nouveaux niveaux de hiérarchie qui reflètent les regroupements et les relations identifiées dans les clusters. Ces ajouts enrichissent le modèle multidimensionnel existant, permettant ainsi une analyse plus approfondie et une représentation des données plus riche et précise.

L'approche d'évolution automatique des schémas que nous proposons dans le modèle GAMM se caractérise par sa nature itératif. Elle peut être répétée à chaque fois que de nouvelles sources de données sont intégrées ou en fonction des besoins spécifiques des utilisateurs. Cette itérativité permet une évolution continue du schéma pour prendre en compte les nouvelles connaissances acquises ainsi que les nouveaux besoins en matière d'analyse.

Dans notre exemple basé sur le réseau des chercheurs scientifiques, les clusters sont composés d'auteurs, de lieux (*venue*) et de mots-clés. Voici un exemple de requête pour l'étape d'exploration (la requête est exprimée en langage Cypher) :

```
WITH list_venues AS venues
UNWIND venues AS ven
WITH ven, list_keywords AS keywords
UNWIND keywords AS key
MATCH (v:venue {VENUE_ID: ven})<-[:field_treated_venue]-(ft:field_treated
↪  )-[:field_treated_keyword]->(k:keyword{KEYWORD:key})
RETURN k.KEYWORD, sum(ft.nbr_treate) AS cnt
ORDER BY cnt DESC
```

Cette requête vise à afficher la fréquence d'utilisation des mots-clés pour chaque groupe au sein d'un regroupement. Les listes « *list_venues* » et « *list_keywords* » représentent respectivement les lieux de publication et les mots-clés associés à un cluster spécifique. Grâce à cette requête, il devient possible de définir différents thèmes et d'établir des niveaux d'hiérarchies thématiques pour chaque groupe d'éléments de ces regroupements.

Le même processus d'évolution du schéma peut être répété pour les clusters obtenus lors de la première étape, afin de détecter les sous-regroupements et d'obtenir une précision encore plus fine. Par conséquent, de nouveaux niveaux d'hiérarchies seront détectés et une nouvelle version du schéma sera générée, permettant ainsi une exploration plus approfondie et une représentation complète des données.

Cette approche itérative, qui englobe l'exploration des clusters, la création de niveaux de hiérarchie thématiques et la mise à jour du schéma, favorise un enrichissement continu de l'entrepôt de données. De plus, elle facilite une analyse plus approfondie et ciblée des données scientifiques, en s'adaptant de manière continue aux nouvelles connaissances et aux besoins en matière d'analyse.

## 6.4 Conclusion

Dans ce chapitre, nous avons présenté une approche automatisée pour l'évolution des schémas dans les entrepôts de données multi-versions basés sur le modèle GAMM. Nous avons souligné que le modèle GAMM permet une évolution temporelle des schémas et des données, ce qui facilite l'intégration de nouvelles sources de données et la prise en compte de nouveaux besoins d'analyse.

Nous avons identifié le besoin d'une approche automatisée pour détecter les connaissances cachées dans les données d'un entrepôt, et pour faciliter le processus d'évolution des schémas. Nous avons souligné que les techniques de fouille de données offrent un potentiel considérable pour améliorer le traitement des données agrégées et pour dé-

tecter des schémas multidimensionnels à partir de différentes sources de données.

Nous avons examiné les travaux de recherche existants sur la détection automatisée de schémas multidimensionnels, cependant, nous avons noté qu'il n'existait pas d'approche globale permettant l'évolution automatique des schémas en plusieurs versions.

En réponse à cette question, nous avons proposé une approche automatique d'évolution des schémas basée sur le modèle GAMM. Notre approche consiste à détecter de nouveaux niveaux de hiérarchies en se basant sur des clusters détectés, et à créer de nouvelles versions de schémas tout en préservant les anciennes. Nous avons souligné que notre approche est entièrement automatisée et peut être appliquée après chaque intégration de nouvelles sources de données.

Nous avons illustré notre approche à l'aide d'un exemple de réseau de recherches scientifiques, en montrant comment les clusters détectés peuvent conduire à de nouveaux niveaux d'hiérarchies dans le schéma.

En conclusion, notre approche d'évolution automatique des schémas dans le modèle GAMM vise à renforcer la flexibilité du modèle et à accroître automatiquement son potentiel analytique. En utilisant cette approche, les concepteurs d'entrepôts de données peuvent adapter et enrichir le schéma de manière plus efficace et automatisée, ce qui facilite l'évolution et la modification des schémas dans un environnement multi-versions.

Dans le prochain chapitre, nous présenterons en détail l'implémentation de notre approche, ainsi que les résultats expérimentaux qui démontrent son efficacité et sa pertinence dans le contexte de l'évolution automatique des schémas

CHAPITRE 7

# Études de cas et Validation

## Sommaire

## 7.1 Introduction

Le chapitre présent se consacre à des études de cas du modèle GAMM que nous proposons pour la gestion évolutive des entrepôts de données en graphes agiles et leur validation. Cette section démontre la pertinence de notre approche à travers diverses études de cas ; chacune de ces dernières vise à mettre en lumière des aspects spécifiques de notre modèle et à évaluer ses performances dans des scénarios réels, tout en démontrant sa flexibilité et sa pertinence pour la gestion évolutive d'entrepôts de données en graphes.

Nous amorçons ce chapitre en explorant l'étude de cas d'évolution de schéma et de données à travers le *Star Schema Benchmark* (SSB). Celui-ci sert de toile de fond à notre démarche. Dans cette section (voir section 7.2), nous examinons en détail la structure et les caractéristiques du benchmark SSB pour démontrer l'efficacité de notre modèle GAMM. Puis, nous détaillons comment nous avons adapté le SSB à notre approche en graphes, afin d'en tirer des informations pertinentes pour notre étude.

Dans la deuxième section (voir section 7.3), nous présentons un prototype logiciel que nous avons conçu pour gérer le méta-modèle, l'entrepôt de données en graphes et les opérateurs d'évolution. Ce prototype permet la mise en œuvre et le test de notre modèle GAMM dans un environnement contrôlé. Nous expliquons les fonctionnalités de ce prototype, qui constitue un outil précieux pour évaluer et valider les différentes études de cas que nous avons entreprises.

La troisième section se penche sur une étude de validation fonctionnelle de l'évolution de schéma dans le modèle GAMM (voir section 7.4). Nous exposons, en détail, le processus d'instanciation que nous avons réalisé sur Neo4j pour plusieurs versions chronologiques, en utilisant les données du benchmark SSB. Cette section se focalise également sur l'exécution de requêtes directes et croisées sur les différentes versions du schéma. L'objectif est d'évaluer la cohérence des résultats et de démontrer l'efficacité de notre modèle pour gérer l'évolution de schéma dans un contexte temporel.

La quatrième section de ce chapitre met en avant une étude de validation fonctionnelle portant sur l'évolution temporelle des données (voir section 7.5). Pour ce faire, nous avons entrepris une étude de cas basée sur le Benchmark SSB afin de démontrer l'intérêt de notre approche dans un contexte évolutif. Au cœur de cette étude de cas se trouve le processus d'instanciation réalisé dans Neo4j en utilisant les données du SSB. En parallèle, nous avons introduit des modifications temporelles au niveau des relations d'agrégation, touchant certains niveaux d'hiérarchies et certains attributs. Les résultats obtenus démontrent la capacité de notre modèle à gérer l'évolution temporelle des données.

La cinquième section aborde une étude de performance visant à comparer l'efficacité de notre modèle dans le contexte des entrepôts de données en graphes par rapport aux approches relationnelles (voir section 7.6). Nous avons utilisé SSB comme référence et l'avons comparé à une approche relationnelle. Cette comparaison permet de mettre en évidence les différences de comportement entre ces deux approches en termes de

performance, et d'évaluer l'efficacité de notre modèle dans ce contexte spécifique.

Enfin, dans la sixième section de ce chapitre (voir section 7.7), nous explorons une étude de cas d'évolution automatique appliquée au modèle GAMM basée sur des données textuelles issues d'un réseau de recherches scientifique. Pour cela, nous avons développé un module d'évolution automatique qui vise à faciliter le processus d'évolution des schémas en détectant les clusters. Cette section illustre comment notre approche peut être mise en œuvre de manière automatisée pour détecter les évolutions cachées au sein des données et adapter le schéma en conséquence.

## 7.2 Présentation du benchmark SSB

Le *Star Schema Benchmark* (ou SSB), a été spécialement conçu pour évaluer les performances des systèmes de bases de données dans le contexte des requêtes d'entrepôts de données en étoile. Inspiré du benchmark PTC-H [1], le schéma SSB adopte une structure modifiée (voir Figure 7.1), mettant en avant la table centrale LINEORDER accompagnée de quatre tables de dimensions essentielles : Client, Part, Supplier et Date (O'Neil et al. (2009)).

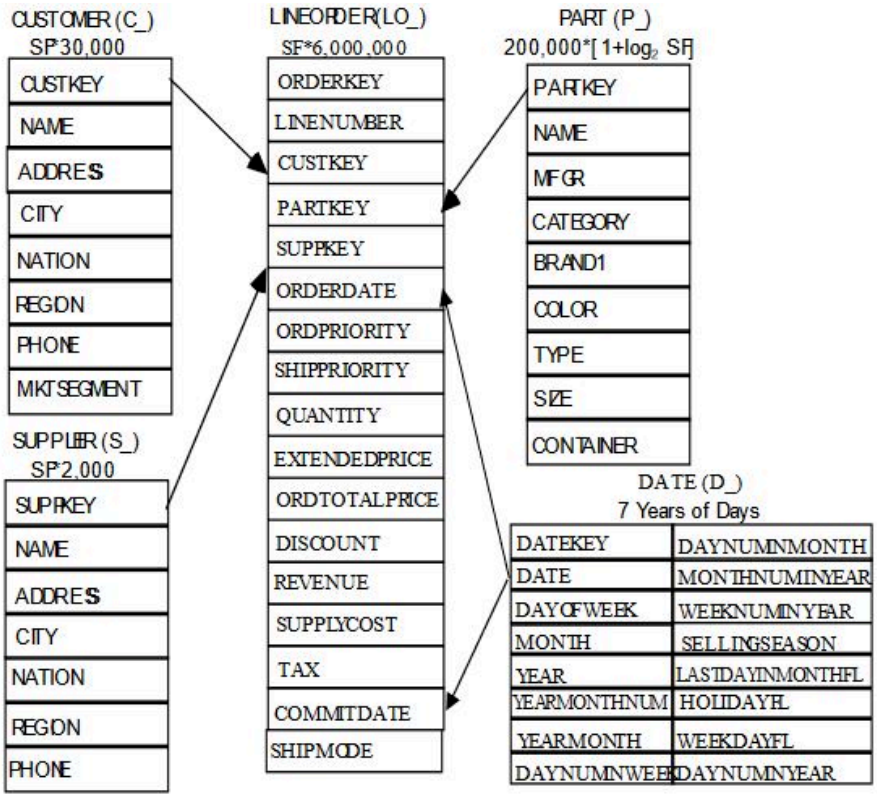



Figure 7.1 – Schéma SSB (O'Neil et al. (2009))

1. http ://www.tpc.org/tpch/

Le benchmark SSB repose sur des requêtes dérivées du TPC-H, mais avec une sélection réduite à 13 requêtes [2]. Ces requêtes, parfois adaptées, ont été choisies pour englober le plus large spectre possible du SSB , ce qui permet aux utilisateurs de mesurer les performances du sous-ensemble pondéré qu'ils prévoient d'employer (O'Neil et al. (2009)). Classées en 4 groupes selon leur complexité et le nombre de dimensions impliquées, ces requêtes visent à évaluer l'efficacité des systèmes de gestion de bases de données (SGBD) dans le traitement d'analyses complexes. La quantité totale de lignes extraites de la table des faits est déterminée par le Facteur de Filtrage total (FF) résultant des restrictions appliquées sur les dimensions.

Le Facteur de Filtrage, ou sélectivité, mesure la proportion de lignes sélectionnées par une condition de filtre dans une requête. Il permet d'estimer l'impact d'une condition de filtre sur la performance d'une requête, aidant ainsi à optimiser les plans d'exécution. Le calcul du Facteur de Filtrage se fait par la formule suivante :

*Facteur de Filtrage (FF) = (Nombre de Lignes Sélectionnées par le filtre) / (Nombre Total de Lignes)*

Si plusieurs clauses de filtrage sont présentes dans une requête, le Facteur de Filtrage combiné est obtenu en multipliant les FF individuels. Les Facteurs de Filtrage des 13 requêtes du benchmark SSB sont représentés dans le tableau suivant :

TABLE 7.1 – Facteurs de filtrage combinés des requêtes SSB (O'Neil et al. (2009))

| **Requêtes** | **FF combinés** |
|---|---|
| Q1.1 | 0.019 |
| Q1.2 | 0.00065 |
| Q1.3 | 0.000075 |
| Q2.1 | 0.008 |
| Q2.2 | 0.0016 |
| Q2.3 | 0.0002 |
| Q3.1 | 0.034 |
| Q3.2 | 0.0014 |
| Q3.3 | 0.000055 |
| Q3.4 | 0.00000076 |
| Q4.1 | 0.016 |
| Q4.2 | 0.0046 |
| Q4.3 | 0.000091 |

Le jeu de données utilisé pour le Benchmark SSB [3] est composé de 600 000 lignes pour la table PART, 120 000 pour la table CUSTOMER, 8 000 pour SUPPLIER, 2 556

2. https ://github.com/Kyligence/ssb-kylin
3. https ://jorgebarbablog.wordpress.com/2016/03/21/how-to-load-the-ssb-schema-into-an-oracle-database/

pour la table Date et 23 996 670 pour la table LINEORDER.

Voici un exemple de requête SSB (la première de chaque groupe). Pour consulter les treize requêtes, (c.f. Annexe A)

Requête Q1.1 :

```
SELECT SUM(lo_revenue) AS revenue
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
WHERE d_year = 1993
AND lo_discount BETWEEN 1 AND 3
AND lo_quantity < 25;
```

Requête Q2.1 :

```
SELECT SUM(lo_revenue) AS lo_revenue, d_year, p_brand1
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
LEFT JOIN part ON lo_partkey = p_partkey
LEFT JOINLEFT JOIN supplier ON lo_suppkey = s_suppkey
WHERE p_category = 'MFGR#12' AND s_region = 'AMERICA'
GROUP BY d_year, p_brand1
ORDER BY d_year, p_brand1;
```

Requête Q3.1 :

```
SELECT c_nation, s_nation, d_year, sum(lo_revenue) AS lo_revenue
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
LEFT JOIN customer ON lo_custkey = c_custkey
LEFT JOIN supplier ON lo_suppkey = s_suppkey
WHERE c_region = 'ASIA' AND s_region = 'ASIA'AND d_year >= 1992 AND d_year
↪  <= 1997
GROUP BY c_nation, s_nation, d_year
ORDER BY d_year ASC, lo_revenue DESC;
```

Requête Q4.1 :

```
SELECT d_year, c_nation, SUM(lo_revenue) - SUM(lo_supplycost) AS profit
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
LEFT JOIN customer ON lo_custkey = c_custkey
LEFT JOIN supplier ON lo_suppkey = s_suppkey
LEFT JOIN part ON lo_partkey = p_partkey
WHERE c_region = 'AMERICA' AND s_region = 'AMERICA' AND (p_mfgr = 'MFGR#1'
↪  OR p_mfgr = 'MFGR#2')
GROUP BY d_year, c_nation
ORDER BY d_year, c_nation;
```

## 7.3 Prototype

Pour illustrer notre modèle GAMM, nous avons développé un prototype logiciel qui gère le méta-modèle, l'entrepôt de données en graphes et les opérateurs d'évolution, conformément à l'architecture présentée dans la Figure 7.2.

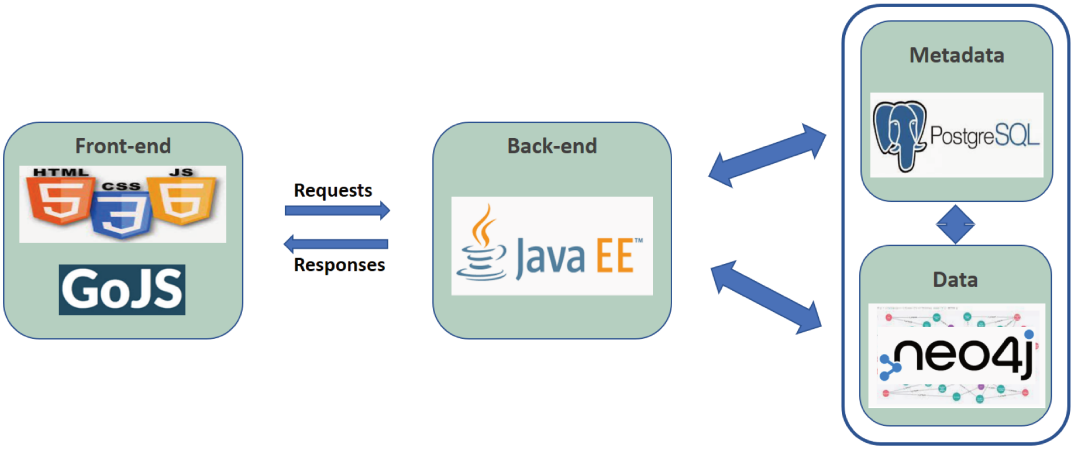



FIGURE 7.2 – Architecture du prototype du modèle GAMM

Ce prototype a été mis en œuvre selon une architecture à trois niveaux :

1. Un premier niveau, *front-end*, basé sur les technologies du web, utilise la bibliothèque GoJS pour créer et manipuler des diagrammes et des graphiques interactifs sur le web.
2. Un deuxième niveau, *back-end*, développé en JavaEE, représente le cœur du traitement. Il assure la communication entre le *front-end* et les bases de données de stockage. Les données proviennent de deux sources : PostgreSQL pour les métadonnées et Neo4j pour les données de l'entrepôt. Les connexions aux bases de données sont établies via les frameworks spécifiques de chacune des deux bases : Neo4j Driver pour Neo4j et le driver JDBC pour PostgreSQL ;
3. Un troisième niveau gère le stockage et la gestion des données et des méta-données. Ces dernières sont conservées dans une base de données relationnelle de type PostgreSQL, conformément au méta-modèle présenté dans la Figure 3.4. Cette base de données permet d'identifier et de visualiser les différentes versions de schéma existantes ainsi que les dates de début (ST) et de fin (ET) de chaque version. Quant aux données de l'entrepôt en graphes, elles sont stockées dans une base de données NoSQL orientée graphe de type Neo4j.

La Figure 7.3 illustre l'interface principale du prototype représentant la partie *front-end*.

L'interface est organisée selon trois volets. Celui de menu général pour les différentes opérations liées à la gestion de la structure et des données, un deuxième volet

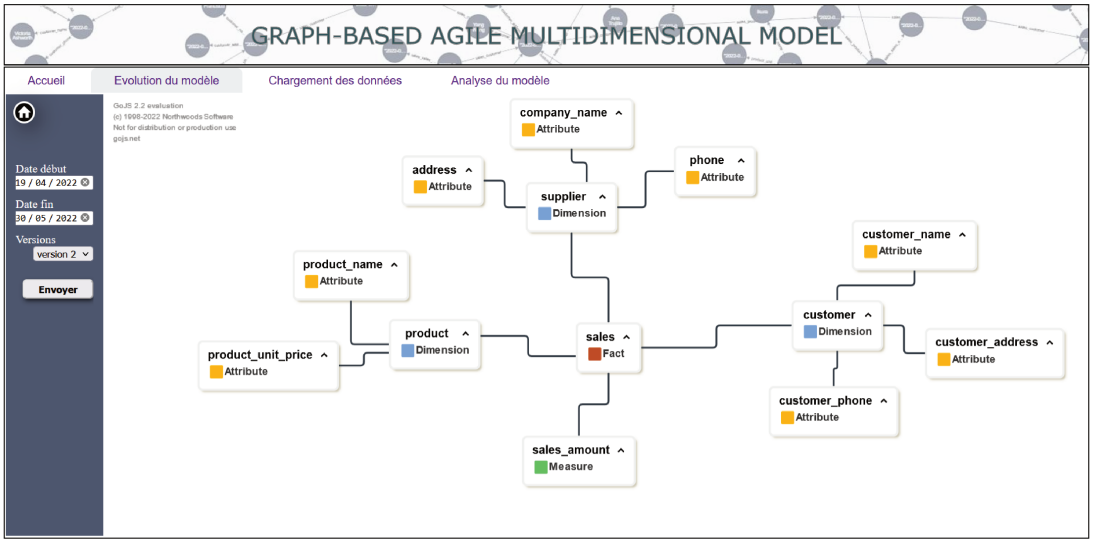



FIGURE 7.3 – Interface du prototype de GAMM

pour l'identification et la navigation à travers les versions et un dernier pour l'affichage des différents schémas des versions.

Grâce à ce prototype, les utilisateurs peuvent identifier et visualiser les différentes versions de schéma existantes, ainsi que les dates de début (ST) et de fin (ET) de chaque version, en s'appuyant sur l'algorithme 1 présenté dans la section 5.2. Ils peuvent également créer de nouvelles versions en utilisant les opérations d'évolution de schéma détaillées dans la section 4.2.

À chaque évolution, une nouvelle version du schéma est créée, avec un ST correspondant à la date de création. Cette date est également attribuée à la date de fin (ET) de la version précédente, afin de déterminer la période de validité de cette version. Ainsi, les évolutions et les modifications du schéma dans GAMM sont gérées comme un processus multi-versions, où chaque version représente l'état du schéma pendant une période de temps T.

Nous détaillons ci-après, l'environnement d'installation et de configuration de la base de données NoSQL orientée graphe Neo4j, que nous utilisons dans nos différentes études de cas dans les sections à venir.

### 7.3.1 Environnement d'Installation et de Configuration de Neo4j

Nous avons utilisé la base de données Neo4j en version 4.4.5 pour stocker notre entrepôt de données en graphes. Les données ont été préparées au format CSV, généré à partir de la base de données du benchmark SSB [4]. Voici les instructions pour la configuration et l'installation sur les plateformes Windows et Linux :

4. https ://jorgebarbablog.wordpress.com/2016/03/21/how-to-load-the-ssb-schema-into-an-oracle-database/

#### 7.3.1.1 Configuration de la mémoire RAM

Pour notre environnement avec 16 Go de RAM, nous avons ajusté les paramètres de la manière suivante :

— dbms.memory.heap.initial_size=4096m
— dbms.memory.heap.max_size=4096m
— dbms.memory.pagecache.size=9216m

#### 7.3.1.2 Création de la base de données Neo4j et chargement des données

$neo4j - admin.batimport - -database = gamm\_db - -nodes = node\_name = file\_node.csv - -relationships = relation\_name = relationship\_file.csv$;

Dans cette commande :

— *gamm_db* : représente le nom de la base de données ;
— *file_node.csv* : représente le fichier des nœuds ;
— *relationship_file.csv* : désigne le fichier des relations entre les nœuds.

Pour les environnements Linux, la commande équivalente est $neo4j-admin.import$.

Pour chaque étude de cas, nous avons préparé un projet *GitHub* détaillé qui contient plus d'informations ainsi que les liens pertinents. Vous trouverez ces liens dans les sections dédiées à chaque étude de cas.

## 7.4 Étude de cas : évolution de schéma

Dans le contexte de notre étude de validation fonctionnelle de l'évolution de schéma dans le modèle GAMM, nous avons utilisé le processus d'instanciation sur Neo4j pour plusieurs versions chronologiques à partir des données de SSB. Le schéma correspondant est illustré dans la figure 7.4. Par la suite, nous avons exécuté des requêtes directes et croisées sur les différentes versions du schéma.

Étant donné que les données du SSB couvrent la période de 1992 à 1998, nous avons adopté une approche de versionnement en plusieurs étapes :

1. La première version du schéma, couvrant les années 1992 et 1993, repose sur un fait Sales analysé à travers les dimensions Part et Customer. Les mesures associées au fait Sales sont Sales_Amount, Discount et Supplycost.

   La dimension Part est décrite par les attributs *P_Name* et *Size*, ainsi que par les niveaux de hiérarchie Category, Mfgr, Brand et Type. La dimension Customer est décrite par les attributs *C_Name*, *C_Address* et *C_Phone*, ainsi que par les niveaux de hiérarchie C_City, C_Nation, C_Region et

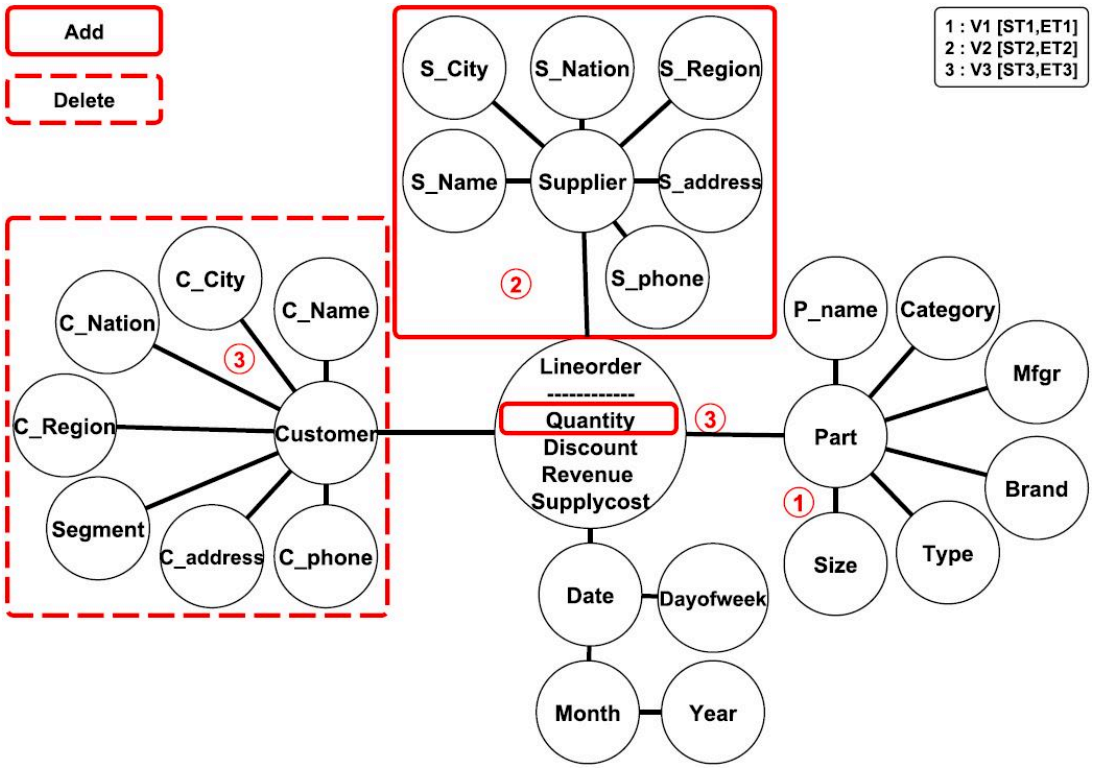



FIGURE 7.4 – Schéma logique des versions à partir des données SSB

SEGMENT. Il convient de noter qu'à des fins d'analyse approfondie, nous avons créé ces niveaux, qui étaient originellement présents sous forme d'attributs dans l'ensemble des données SSB d'origine. Cette première version est valide pour la période $T_1 = [01/01/2019, 31/12/2019]$.

2. La deuxième version du schéma, pour les années 1994 à 1996, intègre une nouvelle dimension SUPPLIER. Cette dimension est caractérisée par les attributs *S_Name*, *S_Address* et *S_Phone*, ainsi que par les niveaux de hiérarchie S_CITY, S_NATION et S_REGION. Cette deuxième version est valide pour la période $T_2 = [01/01/2020, 31/12/2020]$.

3. La troisième version du schéma, pour les années 1997 et 1998, se distingue par la suppression de la dimension CUSTOMER et l'ajout de la mesure *Quantity*. Cette version est valable pour la période $T_3 = [01/01/2021, 31/12/2021]$.

   Nous avons réalisé l'exécution des 13 requêtes du benchmark SSB en utilisant le langage de requêtes Cypher propre à la base de données Neo4j. Nous avons écrit ces requêtes soit explicitement pour interroger une version spécifique du schéma, soit implicitement pour interroger toutes les versions en fonction du schéma existant. Certaines requêtes ont été ajustées pour s'aligner sur les intervalles de temps établis lors du processus de versionnement.

À titre d'exemple, voici la première requête du SSB rédigée en Cypher. Nous avons modifié l'attribut *D_YEAR* pour qu'il corresponde à 1997 au lieu de 1994 dans le benchmark SSB, afin de correspondre à notre schéma en multi-versions (voir Figure 7.4). Cette modification est due à la création de la mesure *Quantity* à partir de la troisième version du schéma, en utilisant les données des années 1997 et 1998.

```
OPTIONAL MATCH (d:date{D_YEAR:1997})<-[r:order_date]-(l:lineorder)
WHERE l.TRANSACTION_TIME >=  DATE({year:2021,month:01})
AND 1<= l.LO_DISCOUNT <=3
AND l.LO_QUANTITY < 25
RETURN SUM(l.LO_REVENUE);
```

Tous les jeux de données, les instructions pour générer les différentes versions de notre entrepôt de données en graphe, ainsi que l'intégralité des requêtes, sont accessibles sur notre référentiel GitHub : GAMM GITHUB. Les 13 requêtes du benchmark SSB sont également présentées en version explicite et implicite dans l'annexe B du présent mémoire.

Nous démontrons le potentiel de notre approche, en particulier pour les requêtes croisées, en utilisant les trois requêtes illustratives suivantes :

**requête 1 :**

```
MATCH (d:date)<-[:order_date]-(l:lineorder)-[r:order_customer]->
↪  (c:customer)
RETURN d.D_YEAR, COUNT(DISTINCT(c.CUSTKEY)), SUM(l.LO_REVENUE)
ORDER BY d.D_YEAR;
```

TABLE 7.2 – Résultats de la requête 1

| YEAR | C CUSTOMERS | SUM(REVENUE) |
|---|---|---|
| 1992 | 79974 | 13233029288673 |
| 1993 | 79978 | 13253674263486 |
| 1994 | 79979 | 13234066608062 |
| 1995 | 79972 | 13207130575971 |
| 1996 | 79980 | 13270332561589 |

**requête 2 :**

```
MATCH (d:date)<-[:order_date]-(l:lineorder)-[r:order_supplier]->
↪  (s:supplier)
```

```
RETURN d.D_YEAR, COUNT(DISTINCT(s.SUPPKEY)), SUM(l.LO_REVENUE)
ORDER BY d.D_YEAR
```

TABLE 7.3 – Résultats de la requête 2

| YEAR | C SUPPLIERS | SUM(REVENUE) |
|---|---|---|
| 1994 | 8000 | 13234066608062 |
| 1995 | 8000 | 13207130575971 |
| 1996 | 8000 | 13270332561589 |
| 1997 | 8000 | 13216215992478 |
| 1998 | 8000 | 7763622653194 |

**requête 3 :**

```
MATCH (d:date)<-[:order_date]-(l:lineorder)-[r:order_part]-> (p :
↪  part)
RETURN d.D_YEAR, COUNT( DISTINCT(p.PARTKEY) ), sum(l.LO_REVENUE)
ORDER BY d.D_YEAR
```

TABLE 7.4 – Résultats de la requête 3

| YEAR | C PART | SUM(REVENUE) |
|---|---|---|
| 1992 | 399956 | 13233029288673 |
| 1993 | 399956 | 13253674263486 |
| 1994 | 399953 | 13234066608062 |
| 1995 | 399959 | 13207130575971 |
| 1996 | 399965 | 13270332561589 |
| 1997 | 399965 | 13216215992478 |
| 1998 | 398036 | 7763622653194 |

À la lumière des résultats obtenus, il est clair que les versions interrogées varient d'une requête à l'autre, même si la version spécifique à interroger n'a pas été explicitement spécifiée. Par exemple, la requête 1, qui montre les montants de ventes par année et le nombre de clients, a affiché des résultats entre les années 1992 et 1996. Cela correspond aux versions 1 et 2 du schéma, lesquelles incluaient la dimension CUSTOMER. De même, la requête 2, qui présente les montants de ventes par année et le nombre de fournisseurs, a donné des résultats

entre 1994 et 1998, correspondant ainsi aux versions 2 et 3 du schéma avec la dimension SUPPLIER. Enfin, la requête 3, qui affiche les montants de ventes par année et le nombre de pièces, a donné des résultats pour toutes les années. Cela est dû au fait que la dimension PART est commune aux trois versions. Cette normalisation du niveau de granularité a été réalisée en utilisant la fonction d'agrégation SUM.

## 7.5 Étude de cas : évolution temporelle des données

Dans le cadre de l'étude de validation fonctionnelle sur l'évolution temporelle des données, nous avons réalisé une étude de cas basée sur SSB pour démontrer l'efficacité de notre approche.

Pour cela, nous avons entrepris la mise en œuvre du processus d'instanciation au sein de Neo4j en utilisant les données de SSB . En parallèle, nous avons introduit diverses modifications temporelles au niveau des relations d'agrégation, touchant certains niveaux d'hiérarchies et certains attributs. Il est important de souligner que le schéma initial du SSB n'incluait pas de niveaux d'hiérarchies. Toutefois, afin de mieux répondre à nos besoins, nous les avons créés en utilisant les attributs, comme en témoigne la VOIR FIGURE 7.5.

Les modifications temporelles appliquées aux relations d'agrégation ont été exécutées selon les étapes suivantes :

(a) Pour la dimension CUSTOMER, nous avons modifié les affectations de cent clients sélectionnés au hasard aux niveaux des hiérarchies C_CITY, C_NATION et C_REGION. Ces clients ont ainsi été assignés à différents niveaux selon les intervalles de temps $T_1 = [FD_1 = 01/01/1992, TD_1 = 01/01/1994]$ et $T_1' = [FD_1 = 01/01/1994, TD_1 = 31/12/9999]$.

(b) Pour la dimension SUPPLIER, nous avons modifié les affectations de cent fournisseurs sélectionnés au hasard aux niveaux des hiérarchies C_CITY, C_NATION et C_REGION. Ces fournisseurs ont ainsi été assignés à différents niveaux selon les intervalles de temps $T_2 = [FD_2 = 01/01/1992, TD_2 = 01/01/1996]$ et $T_2' = [FD_2 = 01/01/1996, TD_2 = 31/12/9999]$.

(c) Pour la dimension PART, nous avons modifié l'attribut *Size* de cent produits choisis au hasard. Ainsi, ces produits avaient deux tailles différentes : une pour l'intervalle de temps $T_3 = [FD_3 = 01/01/1992, TD_3 = 01/01/1995]$ et une autre pour l'intervalle de temps $T_3' = [FD_3 = 01/01/1995, TD_3 = 31/12/99]$.

Ces modifications temporelles nous permettent de mettre en évidence l'impact de l'évolution temporelle des données sur les requêtes et les résultats obtenus à

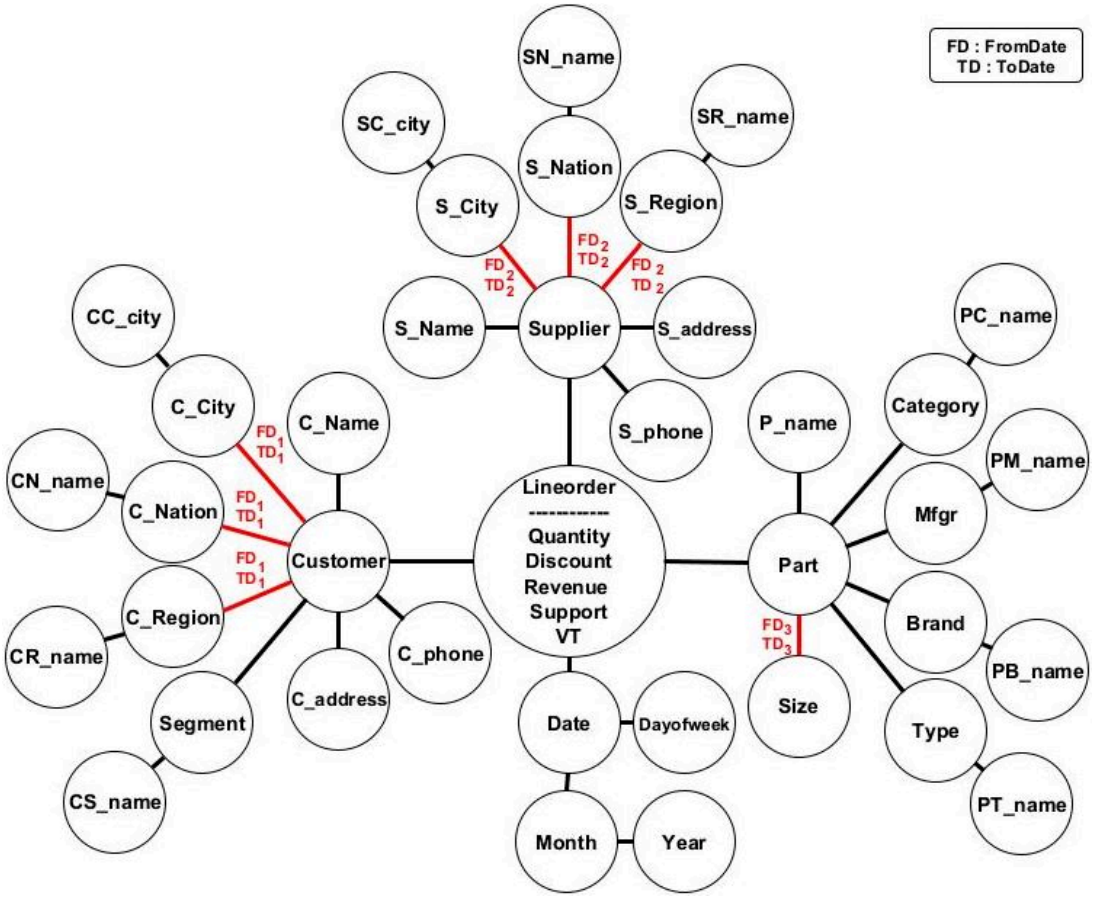



FIGURE 7.5 – Schéma logique de l'entrepôt de données temporelles en graphe basé sur le SSB

partir de notre entrepôt de données temporelles.

Les requêtes temporelles générées à partir des 13 requêtes du Benchmark SSB ont ensuite été exécutées sur notre entrepôt de données temporelles. Notons que les trois premières requêtes n'ont pas subi de modifications par rapport au SSB , car elles ne sont pas impactées par les modifications temporelles que nous avons introduites.

Pour illustrer le processus, prenons l'exemple de la requête SSB Q3.3 :

```
OPTIONAL MATCH (c:cc_city)
↪  <-[:c_c_name]-(:c_city)<-[r1:customer_city]-(:customer)<-[:orde
↪  r_customer]-(l:lineorder)-[:order_date]->(d:date),(s:sc_city)<-
↪  [:s_c_name]-(:s_city)<-[r2:supplier_city]-(s:supplier)
↪  <-[:order_supplier]-(l)
WHERE r1.From_Date<=l.valid_date<r1.To_Date
AND r2.From_Date<=l.valid_date<r2.To_Date
```

```
AND 1992<= d.D_YEAR<=1997
AND (c.c_city="united ki1" OR c.c_city="united ki5")
AND (s.s_city="united ki1" OR s.s_city = "united ki5")
RETURN c.c_city, s.s_city, d.d_year, SUM(l.lo_revenue) AS revenue
ORDER BY d.d_year ASC, revenu DESC
```

Cette requête nécessite l'agrégation des niveaux CITY des dimensions CUSTOMER et SUPPLIER. Étant donné que ces niveaux sont temporellement liés aux dimensions susmentionnées, la requête a été conditionnée en fonction des paramètres FD (From Date) et TD (To Date) de ces relations. Cette approche nous a permis d'obtenir des résultats cohérents et fidèles à l'état des données.

Nous avons utilisé les 13 requêtes du SSB dans un contexte temporel et validé l'ensemble des résultats obtenus. L'ensemble des données, les commandes pour créer l'entrepôt de données temporelles et toutes les requêtes écrites en Cypher sont accessibles sur GTDW GITHUB. Les 10 requêtes temporelles générées à partir du benchmark SSB sont également présentées dans l'annexe C de ce mémoire.

## 7.6 Étude des performances dans les entrepôts de données en graphes

Pour approfondir nos travaux de recherche et et mieux évaluer le modèle GAMM, nous avons mené une étude sur les performances des entrepôts de données en graphes . Pour ce faire, nous avons utilisé le benchmark SSB comme référence et l'avons comparé à une approche relationnelle afin de mieux saisir les différences de comportement entre ces deux approches.

Dans le contexte de cette évaluation de performances, et en raison du manque d'une base de référence pour les entrepôts de données évolutifs, nous avons généré un schéma SSB équivalent en utilisant notre approche en graphes. Ce schéma couvre toute la période de validité des données, et nous avons ensuite comparé les temps d'exécution des 13 requêtes du benchmark SSB entre les approches relationnelle et en graphe.

Les expérimentations ont été menées sous Windows 10 Professionnel, utilisant un processeur Intel(R) Core(TM) i7-10700 cadencé à 2,90 GHz et doté de 16 Go de RAM. L'approche en graphes a été implémentée via Neo4j 4.4.5, tandis que l'approche relationnelle a été mise en œuvre avec Oracle 11g. Les temps d'exécution présentés dans la FIGURE 7.7 sont la moyenne de dix exécutions pour chaque requête, réalisées avec les deux approches (des résultats similaires ont été obtenus sur une plateforme Ubuntu avec une configuration matérielle identique).

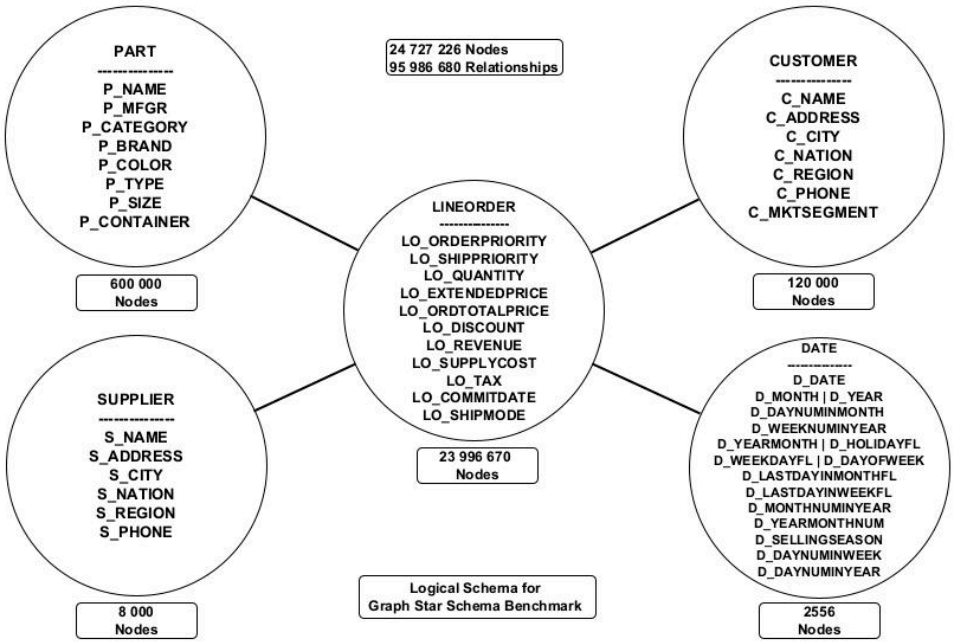



FIGURE 7.6 – Schéma du benchmark SSB en graphe

Le schéma du benchmark SSB en mode graphe est représenté dans la FIGURE 7.6. Notre démarche expérimentale a permis de constater que l'approche en graphes présentait une meilleure performance pour certaines requêtes spécifiques, notamment Q1.2, Q1.3, Q2.2, Q2.3, Q3.3 et Q3.4 (avec des gains de performance allant de 2x à 11x). Cependant, les deux approches étaient relativement proches en termes de performance pour les requêtes Q3.2 et Q4.3. Par contre, l'approche relationnelle s'est révélée plus performante pour les requêtes Q1.1, Q2.1, Q3.1, Q4.1 et Q4.2 (avec des avantages allant de 2x à 8x).

Ces variations de performances dans l'approche en graphes dépendent de facteurs tels que les filtres appliqués (FF), le nombre de dimensions et le nombre d'arêtes à parcourir. Plus spécifiquement, Neo4j s'avère très efficace lorsque le FF est bas, mais ses performances peuvent diminuer lorsque le FF et le nombre de dimensions augmentent, ce qui entraîne un parcours plus complexe de nombreuses relations pour effectuer les agrégations nécessaires. En revanche, Oracle présente une stabilité relativement constante dans les temps d'exécution des requêtes.

Cependant, il convient de souligner qu'une étude de performance plus exhaustive doit englober l'ensemble du processus d'intégration et de stockage des données dans le contexte des approches temporelles. Nous envisageons de mener de telles investigations dans nos futures recherches.

L'ensemble des données, les commandes de création de l'entrepôt de données

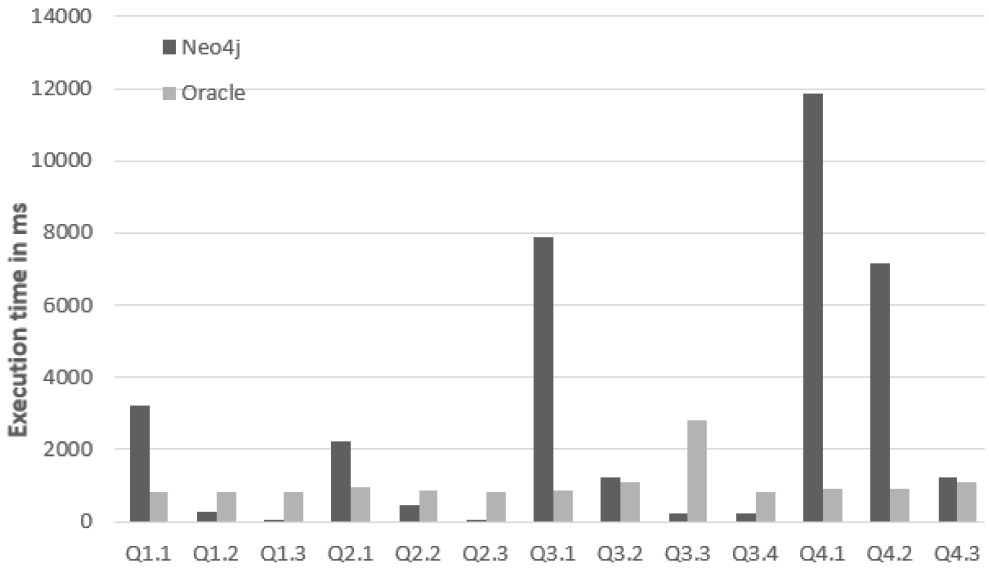



FIGURE 7.7 – Temps de réponse pour les requêtes SSB en utilisant l'approche en graphes et l'approche relationnelle

en graphes complet ainsi que toutes les requêtes sont disponibles sur GSSB GITHUB. De plus, les 13 requêtes du benchmark SSB en format Cypher sont présentées dans l'annexe D de ce mémoire.

## 7.7 Étude de cas d'évolution automatique

Afin d'illustrer notre approche, nous avons développé un module d'évolution automatique qui s'applique au modèle GAMM (voir Figure 7.8). Ce module a été implémenté en Python et vise à faciliter le processus d'évolution automatique des schémas en détectant les clusters, comme expliqué dans la section 6. Ce processus s'articule en trois principales étapes :

(a) Chargement du graphe n-parties pondéré : Cette étape consiste à importer le graphe n-parties pondéré qui représente les relations entre les entités et les attributs de l'entrepôt de données en graphe.

(b) Détection des clusters : Le module effectue une analyse de cluster sur le graphe pour identifier les groupes d'entités et d'attributs étroitement liés entre eux.

(c) Exploration des clusters et création de nouvelles versions : Une fois les clusters identifiées, le module explore les structures et les relations à l'intérieur de ces groupes. Sur cette base, de nouvelles versions des schémas pour l'entrepôt de données en graphes sont générées conformément aux fonctions

d'évolution définies dans la section 4.2. Les anciennes versions sont conservées pour des raisons d'historique.

Cette approche permet ainsi de mettre en œuvre un processus automatique de gestion de l'évolution des schémas, en prenant en compte les dynamiques et les regroupements naturels des données dans l'entrepôt de données en graphe.

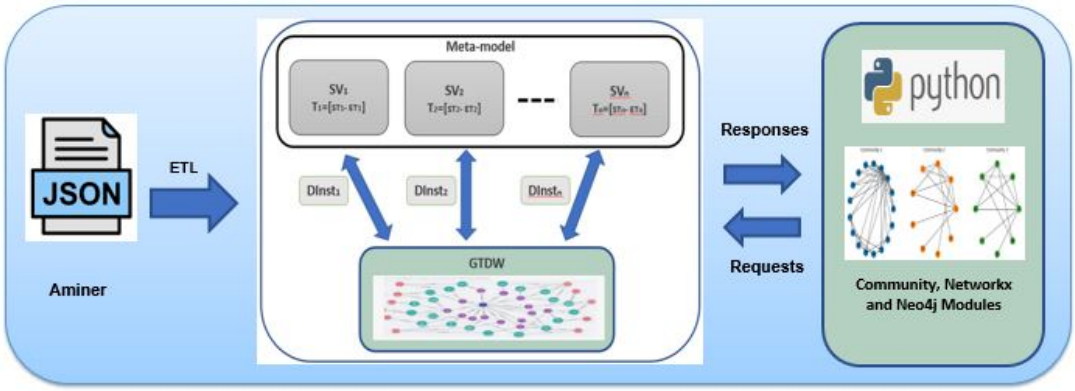



FIGURE 7.8 – Le modèle GAMM basé sur un réseau de recherches scientifiques

Pour valider notre approche, nous avons mené une étude de cas basée sur un réseau de chercheurs scientifiques en utilisant un ensemble de données extrait de sources telles que DBLP, ACM, MAG, et autres [5]. Nous avons utilisé la version 13 de cet ensemble de données, qui était valable jusqu'au 14 mai 2021 et qui comprenait 5 354 309 articles scientifiques. Chaque article est associé à des informations telles que le résumé, les auteurs, l'année de publication, le lieu, les mots-clés et le titre.

Après avoir effectué des opérations de nettoyage et exclu les articles présentant des lacunes dans les informations essentielles comme l'année, les auteurs, le lieu, etc., nous avons retenu un total de 1 135 115 articles. En utilisant cette base de données, nous avons mis en place la première version de notre entrepôt de données. Celle-ci correspond au schéma logique illustré dans la Figure 6.2. Par la suite, nous avons exécuté des requêtes OLAP sur cette première version du schéma dans un but exploratoire.

À titre d'exemple, voici une requête rédigée en Cypher qui permet d'afficher le nombre d'articles publiés en 2020, rédigés par moins de trois auteurs et cités plus de cinq fois :

```
MATCH (y:year) <-[year_paper_publied]- (pp:paper_publied)
WHERE y.YEAR=2000 AND pp.Nbr_author < 3 AND pp.Nbr_citation > 5
RETURN COUNT(pp);
```

5. https ://www.aminer.org/citation

Le premier processus d'évolution du schéma a été lancé. À des fins expérimentales, nous avons exécuté le processus de regroupement (*clustering*) sur plusieurs périodes de publications afin de comparer l'évolution temporelle des thèmes abordés. Les périodes que nous avons choisies sont les suivantes :

(a) Articles publiés entre 2000 et 2010 ;

(b) Articles publiés entre 2010 et 2021 ;

(c) Articles publiés entre 2000 et 2021 .

Il est important de noter que le nombre de clusters identifiés reste constant pour chacune des trois périodes sélectionnées. Plus précisément, nous avons trouvé 11 clusters avec des thèmes similaires, mais avec une fréquence variable d'utilisation des mots-clés au fil du temps. Le tableau 7.5 présente les résultats spécifiquement pour les articles publiés entre 2010 et 2021. Dans cette période, nous avons identifié 334 001 auteurs, 3 718 lieux et 87 933 mots-clés, pour un total de 510 661 articles.

TABLE 7.5 – Résultats du processus de clustering

| ID | Auteurs | Venues | Mots-clés | Mots-clés les plus utilisés dans chaque cluster par ordre décroissant |
|---|---|---|---|---|
| 1 | 52562 | 357 | 12516 | Computer vision, Pattern recognition, Speech recognition, Feature extraction, Algorithm, Segmentation, Image segmentation, Robustness, Support vector machine, Pixel, Contextual image classification, Computer graphics, Discriminative model, Object detection, Image processing, Facial recognition system |
| 2 | 7507 | 31 | 3014 | Remote sensing, Environmental science, Synthetic aperture radar, Satellite, Meteorology, Geology, Radar imaging, Radar, Vegetation, Radiometry, Calibration, Interferometry, Polarimetry, Geodesy, Water content, Backscatter, Inverse synthetic aperture radar, Geography, Interferometric synthetic aperture radar |
| 3 | 24651 | 499 | 8432 | Mathematics, Mathematical optimisation, Discrete mathematics, Theoretical computer science, Combinatorics, Evolutionary algorithm, Genetic algorithm, Optimisation problem, Fuzzy logic, Evolutionary computation, Graph, Heuristic, Metaheuristic, Operator, Population, Time complexity, Algebra |

| | | | | |
|---|---|---|---|---|
| 4 | 25913 | 207 | 8890 | Control theory, Robot, Simulation, Control engineering, Mobile robot, Nonlinear system, Trajectory, Robotics, Robot kinematics, Actuator, Motion planning, Robot control, Kinematics, Humanoid robot, Control system, Torque, Linear system, Human–robot interaction, Adaptive control, Motion control |
| 5 | 53186 | 671 | 11413 | Computer science, Artificial intelligence, Data mining, Information retrieval, Machine learning, Natural language processing, Cloud computing, Scalability, Data science, Database, Cluster analysis, Social network, Semantics, Big data, Ontology, Artificial neural network, Data modelling, Data set |
| 6 | 42090 | 385 | 5663 | Computer network, Distributed computing, Real-time computing, Communication channel, Wireless, Wireless sensor network, Wireless network, Interference, Network packet, Telecommunications, Quality of service, Scheduling (computing), MIMO, Key distribution in wireless sensor networks |
| 7 | 36307 | 309 | 7746 | Engineering, Electronic engineering, Embedded system, Parallel computing, Electrical engineering, CMOS, Field-programmable gate array, Computer hardware, Voltage, Computer architecture, Multicore processor, Chip, Materials science, Electronic circuit, Real-time computing, Speedup |

| | | | | |
|---|---|---|---|---|
| 8 | 13002 | 215 | 2854 | Computer security, Encryption, Internet privacy, Cryptography, Authentication, Access control, Malware, Information privacy, Public-key cryptography, Android, Cloud computing security, Computer security model, Password, Hash function, Cryptographic protocol, Network security, Vulnerability, Adversary |
| 9 | 53994 | 715 | 18680 | Human–computer interaction, Multimedia, Knowledge management, Psychology, Business, Usability, Visualisation, Mobile device, Sociology, Perception, User interface, Social media, Information system, Software, Gesture, Public relations, Cognitive psychology, Cognition, Social psychology, Health care |
| 10 | 14522 | 262 | 2812 | Software engineering, Software, Programming language, Software development, Systems engineering, Software system, Reliability engineering, Source code, Software construction, Java, Software quality, Unified Modeling Language, Model checking, Empirical research, Correctness, Test case, Formal verification |
| 11 | 10267 | 67 | 5913 | Biology, Computational biology, Genetics, Bioinformatics, Genome, Chemistry, Genomics, Gene, Molecular sequence annotation, Protein structure, Molecular dynamics, Biochemistry, Computational chemistry, Cancer, Biological system, DNA sequencing, Medicine, Systems biology, Drug discovery |

L'exploration des clusters obtenus lors de la troisième étape du processus d'évolution automatique décrit dans la sous-section 6.3.3 permet de thématiser aisément les clusters en fonction des mots-clés associés à chacune d'entre elles. Ces mots-clés sont présentés dans l'ordre décroissant de leur fréquence d'utilisation. Bien que nous ne montrions que les mots-clés les plus fréquents au sein de chaque cluster, la grande majorité de ces mots-clés convergent vers des thèmes similaires au sein d'un même cluster.

Pour illustrer davantage ce point, prenons l'exemple du premier cluster qui est axé sur le domaine du traitement du signal et de l'image. Le deuxième cluster concerne principalement le domaine de la technologie des satellites et des radars, ainsi que les domaines connexes. Le troisième cluster est centré sur le domaine des mathématiques, et ainsi de suite.

Un exemple concret est celui de la conférence « *International Conference on Big Data Analytics and Knowledge Discovery* » (DAWAK), qui a été classée dans le cinquième groupe par le biais de ce processus. Ce classement est en parfait accord avec le domaine de spécialisation réel de cette conférence, à savoir la science des données et les systèmes d'information. De nombreux autres exemples ont été vérifiés et ont montré une correspondance solide dans tous les cas.

À la suite de ce processus, trois nouveaux niveaux d'hiérarchies peuvent être créés, respectivement pour les dimensions Auteur, Mot-clé et Lieu. Une nouvelle version de notre schéma d'entrepôt de données sera ensuite générée conformément au schéma présenté dans la figure 7.9. Pendant ce temps, l'ancien schéma sera conservé à des fins d'analyses sur les données historiques.

Nous avons également étendu le processus de clustering aux clusters détectés. Cela nous a permis de découvrir de nouveaux clusters représentant des niveaux de granularité plus fins que les niveaux précédemment établis. Le processus d'évolution automatique peut être répété à chaque intégration de nouvelles sources de données. Cela nous permet de détecter continuellement des connaissances cachées et des relations plus spécifiques au fil du temps. Cette approche itérative garantit que notre entrepôt de données en graphes évolue de manière dynamique pour refléter les changements et les découvertes au fur et à mesure qu'elles se produisent dans les données sous-jacentes.

Il convient de noter qu'au cours de cette étude, nous avons tenté d'utiliser les algorithmes de partitionnement de graphes basés sur la modularité, à savoir *Girvan-Newman* et *Fast Greedy*, dans le processus d'évolution automatique. Cependant, ces algorithmes ont montré des limites en termes d'évolutivité, car les processus n'ont pas pu être menés à bien pour les ensembles de données utilisés en raison du nombre considérable de nœuds à traiter. En revanche, l'algorithme de *Louvain* a démontré de meilleures performances en termes d'évolutivité et a pu être utilisé avec succès dans notre processus d'évolution automatique.

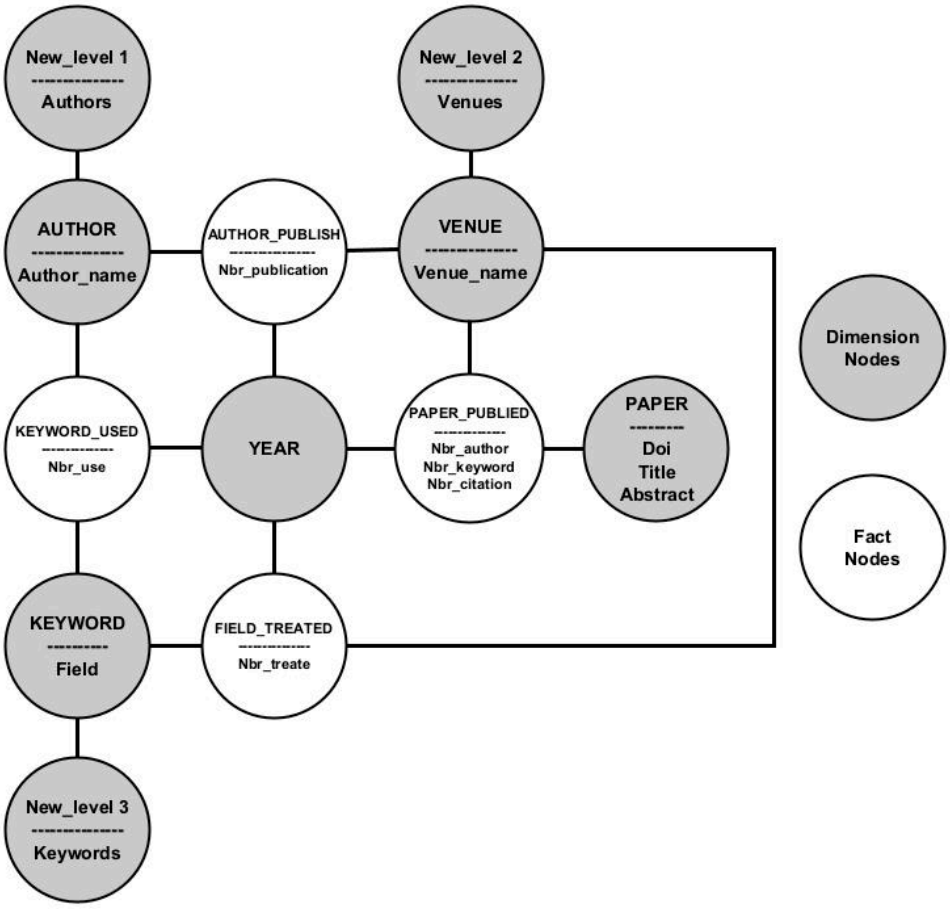



FIGURE 7.9 – Schéma logique du modèle multidimensionnel représentant un réseau bibliographique de recherches scientifiques après un processus d'évolution

## 7.8 Conclusion

Ce chapitre de validation et d'études de cas nous a permis de mettre en lumière la robustesse et la pertinence du modèle GAMM que nous avons développé pour la gestion évolutive d'entrepôts de données en graphes. À travers différentes études de cas, nous avons examiné en profondeur les capacités de notre approche dans des scénarios réels, démontrant ainsi sa flexibilité et son utilité dans des situations variées.

L'utilisation du benchmark SSB comme toile de fond pour notre étude de cas d'évolution de schéma et de données a permis de montrer comment notre modèle peut être appliqué à des données réelles et à des besoins spécifiques. Nous avons réussi à mettre en œuvre le processus d'instanciation dans Neo4j en utilisant les données du SSB, à exécuter des requêtes sur différentes versions du schéma et à démontrer l'efficacité de notre modèle pour gérer l'évolution temporelle des données.

De plus, grâce à notre prototype logiciel, nous avons pu mettre en œuvre les opérateurs d'évolution et expérimenter l'évolution automatique du schéma en détectant les clusters au sein des données. Cette fonctionnalité illustre la capacité de notre modèle à détecter des connaissances cachées et à s'adapter automatiquement à l'évolution des données.

L'étude comparative des performances entre les entrepôts de données en graphes et les approches relationnelles a également révélé les avantages et les limites de chaque approche. Notre modèle GAMM a montré son efficacité pour certaines requêtes spécifiques, soulignant ainsi son potentiel dans le contexte des entrepôts de données en graphes.

En conclusion, ce chapitre a démontré que le modèle GAMM est capable de gérer avec succès des scénarios complexes d'évolution de schéma et de données. Les études de cas et les expériences menées ont mis en évidence sa flexibilité, sa pertinence et son potentiel pour la gestion adaptative et évolutive des entrepôts de données en graphes.

CHAPITRE 8

# CONCLUSION

## 8.1 Bilan et contributions

La révolution du Big Data a considérablement modifié la gestion et l'analyse de l'information dans les entreprises. Soutenue par des avancées technologiques constantes, particulièrement l'importance croissante de l'analyse avancée ; cette transformation voit l'adoption croissante des technologies telles que la BI, l'apprentissage automatique et l'intelligence artificielle. Ces tendances devraient stimuler la demande en matière d'analyse des Big Data à l'avenir, incitant les entreprises à revoir leurs systèmes d'information et à évoluer vers de nouvelles architectures pour exploiter pleinement les vastes volumes de données essentiels à leurs activités.

Cependant, face à la diversification croissante des sources de données et à l'essor du Big Data, les modèles multidimensionnels classiques ont montré leurs limites. L'évolution des schémas et des données dans ces modèles est souvent complexe, chronophage et parfois impossible.

Au cours de nos recherches dans le cadre de cette thèse, plusieurs problématiques ont émergé, nécessitant des solutions innovantes pour faire face aux défis posés par le Big Data, tels que l'évolution structurelle des modèles multidimensionnels, l'évolution temporelle des données et l'automatisation de l'évolution du schéma.

En effet, les modèles de données multidimensionnelles traditionnels ont montré leurs limites face à la diversification des sources de données. La question clé est de savoir comment adapter ces modèles pour répondre de manière agile aux besoins changeants d'analyse dans un contexte de Big Data. De plus, avec la démultiplication constante des données, il est essentiel de contrôler la validité des données et les relations hiérarchiques au fil du temps pour garantir des analyses cohérentes. Enfin, l'automatisation joue un rôle essentiel pour s'adapter rapidement aux nouvelles sources de données et aux tendances émergentes.

L'objectif principal de cette thèse est de proposer un modèle multidimensionnel agile capable d'intégrer de nouvelles sources de données et de s'adapter aux besoins d'analyse changeants. Trois grandes contributions ont été apportées.

La première contribution de cette thèse s'articule autour de l'évolution des modèles multidimensionnels, en mettant l'accent sur les schémas. L'objectif fondamental est de résoudre le problème persistant d'inflexibilité présent dans les modèles multidimensionnels traditionnels. Pour remédier à cette limitation, nous avons proposé un nouveau modèle pour les schémas dimensionnels agiles que nous avons intitulé *Graph-based Agile Multidimensional Model* (GAMM), permettant l'évolution de schéma sous forme de versions multiples. Chaque version de schéma est caractérisée par un intervalle de temps, correspondant à des instances de données extraites de l'entrepôt de données temporelles basé sur les graphes. Cette approche est étayée par le stockage des données dans une base de données orientée graphe de type NoSQL, qui offre une flexibilité accrue en termes d'évolution.

D'un point de vue conceptuel, notre approche s'étend au-delà de la modélisation multidimensionnelle classique, en incorporant les concepts de (`fait - mesure - dimension - hiérarchie - niveau`). Pour prendre en compte l'aspect chronologique de l'évolution des schémas, nous avons proposé un cadre formel détaillé. Un méta-modèle a également été élaboré pour identifier et gérer les différentes versions du schéma, incluant des opérateurs pour l'évolution des schémas ainsi que des règles de passage d'un modèle multidimensionnel classique au modèle GAMM. De plus, nous avons facilité la navigation à travers différentes versions de données par le biais de requêtes simples et croisées explicites et implicites dans un cadre de processus OLAP.

Notre deuxième contribution se concentre sur l'évolution temporelle des données au sein d'un entrepôt de données, dans le but de représenter de manière précise le contexte d'analyse. Dans le prolongement de notre travail, nous avons introduit la notion de la temporalité des données dans le modèle GAMM. L'objectif principal de cette contribution est de résoudre la problématique de la gestion temporelle des données et des relations hiérarchiques, dans le but d'assurer une gestion adéquate des changements d'instances et de garantir leur pertinence dans le temps.

Cette notion de temporalité a été appliquée à la fois dans le formalisme du modèle GAMM et au niveau des règles de passage. Pour ce faire, nous avons attribué des étiquettes temporelles aux nœuds de type « fait » ainsi qu'aux relations au sein des instances des dimensions. Cette stratégie facilite l'identification de la période de validité de ces relations, garantissant ainsi la cohérence des analyses et la concordance des résultats en fonction des changements intervenus. De plus, nous avons présenté des requêtes temporelles, notamment pour l'union, la jointure, la différence et l'agrégation, afin de prendre en compte l'évolution temporelle des données dans les analyses.

Notre troisième contribution se concentre sur l'automatisation de l'évolution des schémas dans les entrepôts de données multi-versions. Nous tirons parti de la combinaison entre le modèle en graphes et les techniques de fouilles de données, ouvrant ainsi de vastes opportunités pour améliorer le traitement des données agrégées. Ces techniques ont été exploitées pour détecter automatiquement des schémas multidimensionnels à partir des données d'un schéma existant, facilitant ainsi l'adaptation dynamique de notre modèle aux nouvelles sources et aux tendances changeantes dans les données agrégées.

Dans notre approche, nous avons automatisé le processus d'évolution des schémas pour améliorer l'efficacité et la précision de la gestion de l'évolution des schémas. L'approche repose sur l'utilisation d'algorithmes de partitionnement de graphes, notamment ceux basés sur la modularité, pour identifier de nouveaux niveaux de hiérarchies correspondant aux clusters détectés dans le graphe.

Cette automatisation présente plusieurs avantages. Elle permet de gérer efficacement l'évolution des schémas en réponse aux nouvelles données tout en détectant de manière proactive des connaissances cachées et des modèles émergents. De plus, elle facilite la création simultanée de niveaux de hiérarchies sur plusieurs dimensions, enrichissant ainsi la structure des données orientées décision. Grâce à la création de ces nouveaux axes, l'analyse multidimensionnelle devient encore plus pertinente, fournissant des informations plus profondes et utiles pour la prise de décision.

Pour terminer, nous avons validé nos contributions à travers plusieurs études de cas et des tests de performances, démontrant l'efficacité et la pertinence du modèle GAMM pour gérer l'évolution des entrepôts de données multidimensionnels dans un contexte de Big Data.

Ainsi nos travaux développés dans le cadre de cette thèse proposent une approche novatrice pour l'évolution des entrepôts de données multidimensionnels dans un environnement de Big Data. Notre modèle agile, basé sur des graphes, permet d'intégrer de nouvelles sources de données, de s'adapter aux besoins d'analyse changeants, et de préserver l'historique. Il offre ainsi une solution élégante pour la flexibilité, l'évolution, et la cohérence des schémas et des données dans un environnement en constante évolution.

## 8.2 Perspectives

Outre les résultats obtenus dans les différentes étapes de nos travaux de recherche, nous avons également identifié des pistes que nous jugeons intéressantes à étudier, et qui s'inscrivent parfaitement dans le cadre de notre projet. Ces perspectives visent à renforcer les résultats de nos travaux de recherche et à explorer de nouveaux horizons.

Le modèle GAMM permet une évolution de schéma en multi-versions, où les niveaux de granularité peuvent varier d'une version de schéma à une autre en raison de l'ajout ou de la suppression de dimensions. Ces versions de schéma peuvent être interrogées à travers des requêtes explicites, qui définissent des critères de recherche spécifiques pour des versions de schéma données, ainsi que par le biais de requêtes implicites, où les informations sont extraites des données sans qu'il soit nécessaire de définir un critère de recherche particulier pour les versions de schéma. Sachant que les versions de schéma n'ont pas forcément les mêmes entités multidimensionnelles, le modèle GAMM permet à ce stade des interrogations croisées sur divers versions.

Dans le travail présenté, l'alignement du niveau de granularité des résultats est effectué manuellement dans les requêtes à l'aide de fonctions d'agrégation. Une de nos perspectives consiste à examiner différentes techniques d'alignement du niveau de granularité des résultats des requêtes OLAP et à proposer des mécanismes automatiques pour faciliter l'interrogation des données sur plusieurs versions, en particulier pour les requêtes croisées complexes.

GAMM offre également la capacité d'intégrer de nouvelles sources de données et de s'adapter aux besoins changeants en matière d'analyse. Cette flexibilité présente une solution élégante pour maintenir la cohérence des schémas et des données dans un environnement en constante évolution. Cependant, il est essentiel de noter que certaines connaissances significatives peuvent ne pas être observable immédiatement, notamment en raison de la multiplication des sources de données. Ces connaissances pourraient se formaliser sous forme de nouvelles entités multidimensionnelles (mesures, dimensions, ou autres), ou alors de présenter de nouvelles tendances ou modèles qui peuvent émerger à partir des données.

En conséquence, nous envisageons de recourir à d'autres techniques d'apprentissage automatique et de fouille de données, notamment l'analyse sémantique, pour assister les utilisateurs du modèle GAMM dans la détection automatique de ces nouvelles connaissances révélées au cours des différents processus d'évolution. Celles-ci ont le potentiel d'apporter plus de capacité analytique à partir de l'entrepôt de données.

Une autre perspective consiste à explorer des techniques d'optimisation avancées pour les requêtes OLAP, dans le but de fournir à l'utilisateur du modèle

GAMM des conditions optimales en termes de fonctionnement et de performances.

En effet, au cours de notre étude sur les performances (voir section 7.6), nous avons constaté que celles-ci peuvent se dégrader au fur et à mesure que le facteur de filtrage (FF) et le nombre de dimensions augmentent. Cela est dû à la nécessité de parcourir un grand nombre de relations pour effectuer les agrégations requises, ce qui peut être contraignant dans un environnement de Big Data, et en tenant compte de l'aspect évolutif du modèle GAMM.

Pour cela, l'interrogation des données et le temps d'exécution des requêtes dans le modèle GAMM représentent un défi majeur qui mérite une étude approfondie. Notre objectif est d'accélérer le traitement des données multidimensionnelles, que ce soit pour les requêtes croisées ou les requêtes temporelles.

Par ailleurs, dans le prolongement de notre travail, nous avons réalisé un ensemble d'initiatives complémentaires visant à enrichir davantage notre modèle, ainsi qu'à faciliter son utilisation.

Tout d'abord, nous prévoyons de réaliser une évaluation approfondie du modèle GAMM par une étude de performances sur l'ensemble des processus d'intégration, en mettant particulièrement l'accent sur le chargement des données, l'évolution des schémas ainsi que le processus d'analyse en ligne (OLAP). Cette évaluation approfondie nous permettra d'acquérir une meilleure compréhension des aspects à améliorer dans le modèle.

Bien que notre modèle GAMM permette de traiter des données textuelles, il est important de renforcer sa capacité d'exploiter plus profondément ce type de données, et ce en appliquant des techniques d'analyse sémantique, notamment sur les structure de graphes.

Nous avons également l'intention de développer une version globale améliorée de notre prototype, en y intégrant toutes les fonctionnalités telles que la gestion des schémas, la gestion des données, les évolutions manuelles et automatiques, ainsi que l'interrogation et l'analyse des données. Cette démarche vise à faciliter l'adoption du modèle GAMM par plus d'acteurs, chercheurs et autres.

Enfin, nous envisageons d'approfondir la gestion des versions au sein du modèle GAMM en intégrant davantage de fonctionnalités pour suivre l'évolution des schémas et des données. Cela pourrait inclure la mise en place de mécanismes plus robustes pour gérer les migrations de schéma et les changements de granularité.

# Bibliographie

W. Ahmed and E. Zimányi. Querying multiversion data warehouses. In T. Morzy, P. Valduriez, and L. Bellatreche, editors, New Trends in Databases and Information Systems – ADBIS 2015 Short Papers and Workshops, BigDap, DCSA, GID, MEBIS, OAIS, SW4CH, WISARD, Poitiers, France, September 8–11, 2015. Proceedings, volume 539 of Communications in Computer and Information Science, pages 346–357. Springer, 2015. doi : 10.1007/978-3-319-23201-0_36. URL https://doi.org/10.1007/978-3-319-23201-0_36.

W. Ahmed, E. Zimányi, and R. Wrembel. A logical model for multiversion data warehouses. In L. Bellatreche and M. K. Mohania, editors, 16th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2014), Germany, Lecture Notes in Computer Science. Springer, 2014. doi : 10.1007/978-3-319-10160-6_3.

W. Ahmed, E. Zimányi, and R. Wrembel. Temporal data warehouses : Logical models and querying. In E. Zimányi, S. Vansummeren, and T. Calders, editors, Actes des 11es journées francophones sur les Entrepôts de Données et l'Analyse en Ligne, volume B-11 of RNTI, pages 33–48, 2015. URL http://editions-rnti.fr/?inprocid=1002125.

W. Ahmed, E. Zimányi, A. A. Vaisman, and R. Wrembel. A temporal multidimensional model and OLAP operators. Int. J. Data Warehous. Min., 16(4) : 112–143, 2020. doi : 10.4018/IJDWM.2020100107.

W. Ahmed, E. Zimányi, A. A. Vaisman, and R. Wrembel. Schema evolution in multiversion data warehouses. Int. J. Data Warehous. Min., 17(4) :1–28, 2021. doi : 10.4018/IJDWM.2021100101.

E. M. Airoldi, D. M. Blei, S. E. Fienberg, and E. P. Xing. Mixed membership stochastic blockmodels. Journal of Machine Learning Research, 9(65) :1981–2014, 2008. URL http://jmlr.org/papers/v9/airoldi08a.html.

J. Akoka, I. Comyn-Wattiau, C. du Mouza, and N. Prat. Mapping multidimensional schemas to property graph models. In I. Reinhartz-Berger and S. W. Sadiq, editors, Advances in Conceptual Modeling – ER 2021 Workshops CoMoNoS, EmpER, CMLS, St. John's, NL, Canada, October 18–21, 2021, Proceedings, volume 13012 of Lecture Notes in Computer

Science, pages 3–14. Springer, 2021. doi : 10.1007/978-3-030-88358-4_1. URL https://doi.org/10.1007/978-3-030-88358-4_1.

J. F. Allen. Maintaining knowledge about temporal intervals. Commun. ACM, 26(11) :832–843, 1983. doi : 10.1145/182.358434. URL https://doi.org/10.1145/182.358434.

B. Bebel, J. Eder, C. Koncilia, T. Morzy, and R. Wrembel. Creation and management of versions in multiversion data warehouse. In H. Haddad, A. Omicini, R. L. Wainwright, and L. M. Liebrock, editors, Proceedings of the 2004 Symposium on Applied Computing (SAC), Cyprus. ACM, 2004. doi : 10.1145/967900.968049.

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

R. Benhissen, F. Bentayeb, and O. Boussaid. GAMM : un modèle multidimensionnel agile à base de graphes pour des entrepôts multi-versions. In S. Bimonte and J. Aligon, editors, Business Intelligence & Big Data, Actes de la conférence EDA 2022, Clermont-Ferrand, France, 27 et 28 octobre 2022, volume B-18 of RNTI, pages 29–46. Editions RNTI, 2022. URL http://editions-rnti.fr/?inprocid=1002781.

R. Benhissen, F. Bentayeb, and O. Boussaid. GAMM : graph-based agile multidimensional model. In E. Gallinucci and L. Golab, editors, Proceedings of the 25th International Workshop on Design, Optimization, Languages and Analytical Processing of Big Data (DOLAP) co-located with (EDBT/ICDT), Ioannina, Greece, March 28, 2023, CEUR Workshop Proceedings, pages 23–32. CEUR-WS.org, 2023a. URL https://ceur-ws.org/Vol-3369/paper2.pdf.

R. Benhissen, F. Bentayeb, and O. Boussaïd. Temporal multidimensional model for evolving graph-based data warehouses. In O. Gusikhin, S. Hammoudi, and A. Cuzzocrea, editors, Proceedings of the 12th International Conference on Data Science, Technology and Applications, DATA 2023, Rome, Italy, July 11-13, 2023, pages 40–51. SCITEPRESS, 2023b. doi : 10.5220/0012080400003541. URL https://doi.org/10.5220/0012080400003541.

E. Benitez-Guerrero, C. Collet, and M. Adiba. The WHES approach to data warehouse evolution. e-Gnosis Num.002, 2004.

F. Bentayeb. Entrepôts et analyse en ligne de données complexes centrés utilisateur : un nouveau défi. (User-centric complex data warehouses and on-line analysis). 2011. URL https://tel.archives-ouvertes.fr/tel-00752126.

F. Bentayeb and C. Favre. RoK : Roll-up with the k-means clustering method for recommending OLAP queries. In Proceedings of DEXA 2009, Linz, Austria, August 31–September 4, 2009, 2009. URL https://doi.org/10.1007/978-3-642-03573-9_43.

S. Bimonte, L. Sautot, L. Journaux, and B. Faivre. Multidimensional model design using data mining : A rapid prototyping methodology. Int. J. Data Warehous. Min., 13(1) :1–35, 2017. doi : 10.4018/IJDWM.2017010101. URL https://doi.org/10.4018/IJDWM.2017010101.

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

D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. J. Mach. Learn. Res., 3(null) :993–1022, mar 2003. ISSN 1532-4435.

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

V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of community hierarchies in large networks. CoRR, abs/0803.0476, 2008. URL http://arxiv.org/abs/0803.0476.

M. H. Böhlen, A. Dignös, J. Gamper, and C. S. Jensen. Temporal data management - an overview. In E. Zimányi, editor, Business Intelligence and Big Data - 7th European Summer School, eBISS 2017, Bruxelles, Belgium, July 2-7, 2017, Tutorial Lectures, volume 324 of Lecture Notes in Business Information Processing, pages 51–83. Springer, 2017. doi : 10.1007/978-3-319-96655-7_3. URL https://doi.org/10.1007/978-3-319-96655-7_3.

A. Campos, J. Mozzino, and A. Vaisman. Towards temporal graph databases. arXiv preprint. arXiv :1604.08568, 2016.

A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. Physical Review E, 70(6), dec 2004. doi : 10.1103/physreve.70.066111. URL https://doi.org/10.1103%2Fphysreve.70.066111.

C. Date. Time and Relational Theory : Temporal Databases in the Relational Model and SQL : a Chris Date Master Class. O'Reilly Media, Incorporated, 2015. ISBN 9781491917763. URL https://books.google.fr/books?id=X-6lzQEACAAJ.

A. Debrouvier, E. Parodi, M. Perazzo, V. Soliani, and A. A. Vaisman. A model and query language for temporal graph databases. The VLDB Journal, 30 : 825–858, 2021.

M. Dorigo and T. Stützle. Ant colony optimization. MIT Press, 2004. URL dblpcomputersciencebibliography,https://dblp.org.

S. Faisal and M. Sarwar. Handling slowly changing dimensions in data warehouses. J. Syst. Softw., 94 :151–160, 2014. doi : 10.1016/j.jss.2014.03.072. URL https://doi.org/10.1016/j.jss.2014.03.072.

S. Faisal, M. Sarwar, K. Shahzad, S. Sarwar, S. W. Jaffry, and M. M. Yousaf. Temporal and evolving data warehouse design. Sci. Program., 2017 : 7392349 :1–7392349 :18, 2017. doi : 10.1155/2017/7392349. URL https://doi.org/10.1155/2017/7392349.

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

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

D. Fernandes and J. Bernardino. Graph databases comparison : Allegrograph, arangodb, infinitegraph, neo4j, and orientdb. In J. Bernardino and C. Quix, editors, Proceedings of the 7th International Conference on Data Science, Technology and Applications, DATA 2018, Porto, Portugal, July 26-28, 2018, pages 373–380. SciTePress, 2018. doi : 10.5220/0006910203730380. URL https://doi.org/10.5220/0006910203730380.

C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. In Proceedings of the 19th Design Automation Conference, DAC '82, page 175–181. IEEE Press, 1982. ISBN 0897910206.

M. Fiedler. Algebraic connectivity of graphs. Czechoslovak Mathematical Journal, 23 :298–305, 1973. URL https://api.semanticscholar.org/CorpusID:117770486.

L. R. Ford and D. R. Fulkerson. Flows in Networks. Princeton University Press, 1962. ISBN 9780691625393. URL http://www.jstor.org/stable/j.ctt183q0b4.

S. Fortunato and D. Hric. Community detection in networks : A user guide. Physics Reports, 659 :1–44, 2016. ISSN 0370-1573. doi : https://doi.org/10.1016/j.physrep.2016.09.002. URL https://www.sciencedirect.com/science/article/pii/S0370157316302964. Community detection in networks : A user guide.

J. Gantz and E. Reinsel. Extracting value from chaos. IDC's Digital Universe Study, sponsored by EMC, 2011.

G. Garani, G. K. Adam, and D. Ventzas. Temporal data warehouse logical modelling. Int. J. Data Min. Model. Manag., 8(2) :144–159, 2016. doi : 10.1504/IJDMMM.2016.077156.

M. Golfarelli and S. Rizzi. Managing late measurements in data warehouses. Int. J. Data Warehous. Min., 3(4) :51–67, 2007. doi : 10.4018/jdwm.2007100103. URL https://doi.org/10.4018/jdwm.2007100103.

M. Golfarelli and S. Rizzi. Temporal data warehousing : Approaches and techniques. In D. Taniar and L. Chen, editors, Integrations of Data Warehousing,

Data Mining and Database Technologies – Innovative Approaches, pages 1–18. Information Science Reference, 2011. doi : 10.4018/978-1-60960-537-7.ch001.

M. Golfarelli and S. Rizzi. From star schemas to big data : 20+ years of data warehouse research. In S. Flesca, S. Greco, E. Masciari, and D. Saccà, editors, A Comprehensive Guide Through the Italian Database Research Over the Last 25 Years, Studies in Big Data. Springer International Publishing, 2018. doi : 10.1007/978-3-319-61893-7_6.

M. Golfarelli, D. Maio, and S. Rizzi. Conceptual design of data warehouses from E/R schema. In Thirty-First Annual Hawaii International Conference on System Sciences, Kohala Coast, Hawaii, USA, January 6–9, 1998, pages 334–343. IEEE Computer Society, 1998. doi : 10.1109/HICSS.1998.649228. URL https://doi.org/10.1109/HICSS.1998.649228.

M. Golfarelli, S. Rizzi, and B. Vrdoljak. Data warehouse design from XML sources. In J. Hammer, editor, Proceedings of the 4th ACM International Workshop on Data Warehousing and OLAP (DOLAP 2001), Atlanta, Georgia, USA, November 9, 2001, pages 40–47. ACM, 2001. doi : 10.1145/512236.512242. URL https://doi.org/10.1145/512236.512242.

T. Heath and C. Bizer. Linked data : Evolving the web into a global data space. Springer Nature, 2022.

P. Holland, K. B. Laskey, and S. Leinhardt. Stochastic blockmodels : First steps. Social Networks, 5 :109–137, 1983. URL https://api.semanticscholar.org/CorpusID:34098453.

H. Hultgren. Introductory guide to data vault modeling. Genesee Academy, 2012a.

H. Hultgren. Data vault modelling guide–introductory guide to data vault modelling. G.A., 2012b.

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

W. Inmon. Building the Data Warehouse. Wiley, 2005. ISBN 9780471774235. URL https://books.google.fr/books?id=QFKTmh5IFS4C.

W. Inmon. The comparison of anchor and star schema from a query performance perspective. 2013.

W. H. Inmon. Building the Data Warehouse. John Wiley & Sons, Inc., USA, 1992. ISBN 0471569607.

C. S. Jensen and R. T. Snodgrass. Transaction time. In L. Liu and M. T. Özsu, editors, Encyclopedia of Database Systems, Second Edition. Springer,

2018a. doi : 10.1007/978-1-4614-8265-9_1064. URL https://doi.org/10.1007/978-1-4614-8265-9_1064.

C. S. Jensen and R. T. Snodgrass. Valid time. In L. Liu and M. T. Özsu, editors, Encyclopedia of Database Systems, Second Edition. Springer, 2018b. doi : 10.1007/978-1-4614-8265-9_1066. URL https://doi.org/10.1007/978-1-4614-8265-9_1066.

M. R. Jensen, T. Holmgren, and T. B. Pedersen. Discovering multidimensional structure in relational data. In Y. Kambayashi, M. K. Mohania, and W. Wöß, editors, Data Warehousing and Knowledge Discovery, 6th International Conference, DaWaK 2004, Zaragoza, Spain, September 1-3, 2004, Proceedings, volume 3181 of Lecture Notes in Computer Science, pages 138–148. Springer, 2004. doi : 10.1007/978-3-540-30076-2_14. URL https://doi.org/10.1007/978-3-540-30076-2_14.

D. R. Karger and C. Stein. A new approach to the minimum cut problem. J. ACM, 43(4) :601–640, jul 1996. ISSN 0004-5411. doi : 10.1145/234533.234534. URL https://doi.org/10.1145/234533.234534.

G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing, 20(1) : 359–392, 1998. doi : 10.1137/S1064827595287997. URL https://doi.org/10.1137/S1064827595287997.

K. Kaur and R. Rani. Modeling and querying data in nosql databases. In X. Hu, T. Y. Lin, V. V. Raghavan, B. W. Wah, R. Baeza-Yates, G. C. Fox, C. Shahabi, M. Smith, Q. Yang, R. Ghani, W. Fan, R. Lempel, and R. Nambiar, editors, 2013 IEEE International Conference on Big Data (IEEE BigData 2013), 6-9 October 2013, Santa Clara, CA, USA, pages 1–7. IEEE Computer Society, 2013. doi : 10.1109/BigData.2013.6691765. URL https://doi.org/10.1109/BigData.2013.6691765.

B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. The Bell System Technical Journal, 49(2) :291–307, 1970. doi : 10.1002/j.1538-7305.1970.tb01770.x.

R. Kimball. The Data Warehouse Toolkit : Practical Techniques for Building Dimensional Data Warehouses. John Wiley & Sons, Inc., USA, 1996. ISBN 0471153370.

R. Kimball and M. Ross. The Data Warehouse Toolkit : The Definitive Guide to Dimensional Modeling. Wiley, Indianapolis, IN, USA, third edition, 2013. ISBN 978-1-118-53080-1. URL https://www.safaribooksonline.com/library/view/the-data-warehouse/9781118530801/.

S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. Science (New York, N.Y.), 220 :671–80, 06 1983. doi : 10.1126/science.220.4598.671.

K. G. Kulkarni and J. Michels. Temporal features in SQL : 2011. SIGMOD Rec., 41(3) :34–43, 2012. doi : 10.1145/2380776.2380786. URL https://doi.org/10.1145/2380776.2380786.

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

N. A. Lorentzos. Time period. In L. Liu and M. T. Özsu, editors, Encyclopedia of Database Systems, Second Edition. Springer, 2018. doi : 10.1007/978-1-4614-8265-9_1425. URL https://doi.org/10.1007/978-1-4614-8265-9_1425.

J.-N. Mazón, J. Pardillo, and J. Trujillo. Applying transformations to model driven data warehouses. In A. M. Tjoa and J. Trujillo, editors, Data Warehousing and Knowledge Discovery, pages 13–22, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. ISBN 978-3-540-37737-5.

A. O. Mendelzon and A. A. Vaisman. Temporal queries in OLAP. In A. E. Abbadi, M. L. Brodie, S. Chakravarthy, U. Dayal, N. Kamel, G. Schlageter, and K. Whang, editors, VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt, pages 242–253. Morgan Kaufmann, 2000. URL http://www.vldb.org/conf/2000/P242.pdf.

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

T. Morzy and R. Wrembel. On querying versions of multiversion data warehouse. In I. Song and K. C. Davis, editors, DOLAP 2004, ACM Seventh International Workshop on Data Warehousing and OLAP, Washington, DC, USA, November 12-13, 2004, Proceedings, pages 92–101. ACM, 2004. doi : 10.1145/1031763.1031781. URL https://doi.org/10.1145/1031763.1031781.

R. NĚMEC and F. ZAPLETAL. 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, 2014.

M. E. Newman. Community detection in networks : Modularity optimization and maximum likelihood are equivalent. arXiv preprint. arXiv :1606.02319, 2016.

M. E. J. Newman. Modularity and community structure in networks. Proceedings of the National Academy of Sciences, 103(23) :8577–8582, June 2006. URL https://doi.org/10.1073%2Fpnas.0601602103.

M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. Phys. Rev. E, 69 :026113, Feb 2004. doi : 10.1103/PhysRevE.69.026113. URL https://link.aps.org/doi/10.1103/PhysRevE.69.026113.

A. Ng, M. Jordan, and Y. Weiss. On spectral clustering : Analysis and an algorithm. In T. Dietterich, S. Becker, and Z. Ghahramani, editors, Advances in Neural Information Processing Systems, volume 14. MIT Press, 2001. URL https://proceedings.neurips.cc/paper_files/paper/2001/file/801272ee79cfde7fa5960571fee36b9b-Paper.pdf.

R. Němec. The comparison of anchor and star schema from a query performance perspective. International Journal of Computer and Information Engineering, 6(11) :1421–1425, 2012. ISSN eISSN : 1307-6892. URL https://publications.waset.org/vol/71.

R. Němec and F. Zapletal. 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, 2014.

P. E. O'Neil, E. J. O'Neil, and X. Chen. The star schema benchmark (SSB), 2009.

T. Phungtua-Eng and S. Chittayasothorn. Slowly changing dimension handling in data warehouses using temporal database features. In N. T. Nguyen, F. L. Gaol, T. Hong, and B. Trawinski, editors, 11th Asian Conference on Intelligent Information and Database Systems (ACIIDS 2019), Indonesia. Springer, 2019. doi : 10.1007/978-3-030-14799-0_58.

J. Pokorný. Graph databases : Their power and limitations. In K. Saeed and W. Homenda, editors, Computer Information Systems and Industrial Management - 14th IFIP TC 8 International Conference, CISIM 2015, Warsaw, Poland, September 24-26, 2015. Proceedings, volume 9339 of Lecture Notes in Computer Science, pages 58–69. Springer, 2015. doi : 10.1007/978-3-319-24369-6_5. URL https://doi.org/10.1007/978-3-319-24369-6_5.

P. Poscic, I. Babic, and D. Jaksic. Temporal functionalities in modern database management systems and data warehouses. In 41st International Convention on Information and Communication Technology, Electronics and Microelectronics. IEEE, 2018. doi : 10.23919/MIPRO.2018.8400287. URL https://doi.org/10.23919/MIPRO.2018.8400287.

A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. SIAM Journal on Matrix Analysis and Applications, 11(3) :430–452, 1990. doi : 10.1137/0611030. URL https://doi.org/10.1137/0611030.

U. N. Raghavan, R. Albert, and S. Kumara. Near linear time algorithm to detect community structures in large-scale networks. Phys. Rev. E, 76 :036106, Sep 2007. doi : 10.1103/PhysRevE.76.036106. URL https://link.aps.org/doi/10.1103/PhysRevE.76.036106.

F. Ravat, O. Teste, and G. Zurfluh. A multiversion-based multidimensional model. In A. M. Tjoa and J. Trujillo, editors, 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, pages 65–74. Springer, 2006. doi : 10.1007/11823728_7. URL https://doi.org/10.1007/11823728_7.

O. Regardt, L. Rönnbäck, M. Bergholtz, P. Johannesson, and P. Wohed. Anchor modeling. In International Conference on Conceptual Modeling, pages 234–250. Springer, 2009.

S. Rizzi and M. Golfarelli. X-time : Schema versioning and cross-version querying in data warehouses. In R. Chirkova, A. Dogac, M. T. Özsu, and T. K. Sellis, editors, Proceedings of the 23rd International Conference on Data Engineering (ICDE 2007), Istanbul, Turkey, April 15–20, 2007, pages 1471–1472. IEEE Computer Society, 2007. doi : 10.1109/ICDE.2007.369038.

I. Robinson, J. Webber, and E. Eifrem. Graph Databases. O'Reilly, Beijing, 2 edition, 2015. ISBN 978-1-4919-3089-2. URL https://www.safaribooksonline.com/library/view/graph-databases-2nd/9781491930885/.

L. Rönnbäck and H. Hultgren. Comparing anchor modeling with data vault modeling. Modeling for the Modern Data Warehouse, White Paper, 2013.

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

N. Sanprasit, K. Jampachaisri, T. Titijaroonroj, and K. Kesorn. Intelligent approach to automated star-schema construction using a knowledge base. Expert Syst. Appl., 182 :115226, 2021. doi : 10.1016/j.eswa.2021.115226. URL https://doi.org/10.1016/j.eswa.2021.115226.

K. Saroha and A. Gosain. Bi-temporal schema versioning in bi-temporal data warehouse. CSI Transactions on ICT, 3 :135–142, 2015.

J. Shi and J. Malik. Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(8) :888–905, 2000. doi : 10.1109/34.868688.

D. Solodovnikova. Data warehouse evolution framework. In S. D. Kuznetsov, A. Fomichev, B. Novikov, and D. Shaporenkov, editors, Proceedings of the SYRCoDIS 2007 Colloquium on Databases and Information Systems, Moscow, Russia, May 31 - June 1, 2007. CEUR-WS.org, 2007. URL http://ceur-ws.org/Vol-256/submission_4.pdf.

I. Song, R. Khare, and B. Dai. SAMSTAR : a semi-automated lexical method for generating star schemas from an entity-relationship diagram. In I. Song and T. B. Pedersen, editors, DOLAP 2007, ACM 10th International

Workshop on Data Warehousing and OLAP, Lisbon, Portugal, November 9, 2007, Proceedings, pages 9–16. ACM, 2007. doi : 10.1145/1317331.1317334. URL https://doi.org/10.1145/1317331.1317334.

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

N. G. Trillos, D. Slepcev, J. von Brecht, T. Laurent, and X. Bresson. Consistency of cheeger and ratio graph cuts. arXiv preprint arXiv :1411.6590, 2014.

A. Vaisman and E. Zimányi. Temporal and Multiversion Data Warehouses, pages 373–436. Springer Berlin Heidelberg, Berlin, Heidelberg, 2022. ISBN 978-3-662-65167-4. doi : 10.1007/978-3-662-65167-4_11. URL https://doi.org/10.1007/978-3-662-65167-4_11.

R. Wrembel and B. Bebel. Metadata management in a multiversion data warehouse. In R. Meersman, Z. Tari, M. Hacid, J. Mylopoulos, B. Pernici, Ö. Babaoglu, H. Jacobsen, J. P. Loyall, M. Kifer, and S. Spaccapietra, editors, On the Move to Meaningful Internet Systems 2005 : CoopIS, DOA, and ODBASE, OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31 - November 4, 2005, Proceedings, Part II, volume 3761 of Lecture Notes in Computer Science, pages 1347–1364. Springer, 2005. doi : 10.1007/11575801_26. URL https://doi.org/10.1007/11575801_26.

R. Wrembel and T. Morzy. Managing and querying versions of multiversion data warehouse. In Advances in Database Technology (EDBT 2006), volume 3896. Springer, 2006. doi : 10.1007/11687238_73.

J. Xie, M. Chen, and B. K. Szymanski. Labelrankt : incremental community detection in dynamic networks via label propagation, 2013. URL https://doi.org/10.1145/2489247.2489249.

J. Yang and J. Leskovec. Overlapping community detection at scale : A nonnegative matrix factorization approach. pages 587–596, 02 2013. doi : 10.1145/2433396.2433471.

Y. Yang, F. Abdelhédi, J. Darmont, F. Ravat, and O. Teste. Automatic machine learning-based OLAP measure detection for tabular data. In R. Wrembel, J. Gamper, G. Kotsis, A. M. Tjoa, and I. Khalil, editors, Big Data Analytics and Knowledge Discovery - 24th International Conference, DaWaK 2022, Vienna, Austria, August 22-24, 2022, Proceedings, volume 13428 of Lecture Notes in Computer Science, pages 173–188. Springer, 2022. doi : 10.1007/978-3-031-12670-3_15. URL https://doi.org/10.1007/978-3-031-12670-3_15.

ANNEXE A

# Les treize requêtes du SSB en SQL

Q1 - Requêtes impliquant une seule dimension dans la clause de filtrage (Date).

Requête Q1.1 :

```
SELECT SUM(lo_revenue) AS revenue
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
WHERE d_year = 1993
AND lo_discount BETWEEN 1 AND 3
AND lo_quantity < 25;
```

Requête Q1.2 :

```
SELECT SUM(lo_revenue) AS revenue
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
WHERE d_yearmonthnum = 199401
AND lo_discount BETWEEN 4 AND 6
AND lo_quantity BETWEEN 26 AND 35;
```

Requête Q1.3 :

```
SELECT SUM(lo_revenue) AS revenue
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
WHERE d_weeknuminyear = 6 and d_year = 1994
AND lo_discount BETWEEN 5 AND 7
AND lo_quantity BETWEEN 26 AND 35;
```

Q2 - Requêtes impliquant deux dimensions dans la clause de filtrage (Part et Supplier).

Requête Q2.1 :

```
SELECT SUM(lo_revenue) AS lo_revenue, d_year, p_brand1
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
LEFT JOIN part ON lo_partkey = p_partkey
LEFT JOINLEFT JOIN supplier ON lo_suppkey = s_suppkey
WHERE p_category = 'MFGR#12' AND s_region = 'AMERICA'
GROUP BY d_year, p_brand1
ORDER BY d_year, p_brand1;
```

Requête Q2.2 :

```
SELECT SUM(lo_revenue) AS lo_revenue, d_year, p_brand1
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
LEFT JOIN part ON lo_partkey = p_partkey
LEFT JOIN supplier ON lo_suppkey = s_suppkey
WHERE p_brand1 BETWEEN 'MFGR#2221' AND 'MFGR#2228' AND s_region =
↪  'ASIA'
GROUP BY d_year, p_brand1
ORDER BY d_year, p_brand1;
```

Requête Q2.3 :

```
SELECT SUM(lo_revenue) AS lo_revenue, d_year, p_brand1
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
LEFT JOIN part ON lo_partkey = p_partkey
LEFT JOIN supplier ON lo_suppkey = s_suppkey
WHERE p_brand1 = 'MFGR#2239' AND s_region = 'EUROPE'
GROUP BY d_year, p_brand1
ORDER BY d_year, p_brand1;
```

Q3 - Requêtes impliquant trois dimensions dans la clause de filtrage (Customer, Supplier et Date).

Requête Q3.1 :

```
SELECT c_nation, s_nation, d_year, sum(lo_revenue) AS lo_revenue
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
LEFT JOIN customer ON lo_custkey = c_custkey
LEFT JOIN supplier ON lo_suppkey = s_suppkey
WHERE c_region = 'ASIA' AND s_region = 'ASIA'AND d_year >= 1992 AND
↪  d_year <= 1997
GROUP BY c_nation, s_nation, d_year
ORDER BY d_year ASC, lo_revenue DESC;
```

Requête Q3.2 :

```
SELECT c_city, s_city, d_year, sum(lo_revenue) as lo_revenue
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
LEFT JOIN customer ON lo_custkey = c_custkey
LEFT JOIN supplier ON lo_suppkey = s_suppkey
WHERE c_nation = 'UNITED STATES' AND s_nation = 'UNITED STATES'
AND d_year >= 1992 AND d_year <= 1997
GROUP BY c_city, s_city, d_year
ORDER BY d_year ASC, lo_revenue DESC;
```

Requête Q3.3 :

```
SELECT c_city, s_city, d_year, SUM(lo_revenue) AS lo_revenue
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
LEFT JOIN customer ON lo_custkey = c_custkey
LEFT JOIN supplier ON lo_suppkey = s_suppkey
WHERE (c_city='UNITED KI1' OR c_city='UNITED KI5')
AND (s_city='UNITED KI1' OR s_city='UNITED KI5')
AND d_year >= 1992 AND d_year <= 1997
GROUP BY c_city, s_city, d_year
GROUP BY by d_year asc, lo_revenue desc;
```

Requête Q3.4 :

```
SELECT c_city, s_city, d_year, SUM(lo_revenue) AS lo_revenue
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
LEFT JOIN customer ON lo_custkey = c_custkey
LEFT JOIN supplier ON lo_suppkey = s_suppkey
WHERE (c_city='UNITED KI1' OR c_city='UNITED KI5') and
↪  (s_city='UNITED KI1' OR s_city='UNITED KI5') AND d_yearmonth =
↪  'Dec1997'
GROUPE BY c_city, s_city, d_year
ORDER BY d_year ASC, lo_revenue DESC;
```

Q4 - Requêtes impliquant quatre dimensions dans la clause de filtrage (Customer, Supplier, Part et Date).

Requête Q4.1 :

```
SELECT d_year, c_nation, SUM(lo_revenue) - SUM(lo_supplycost) AS
↪  profit
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
LEFT JOIN customer ON lo_custkey = c_custkey
LEFT JOIN supplier ON lo_suppkey = s_suppkey
```

```
LEFT JOIN part ON lo_partkey = p_partkey
WHERE c_region = 'AMERICA' AND s_region = 'AMERICA' AND (p_mfgr =
↪  'MFGR#1' OR p_mfgr = 'MFGR#2')
GROUP BY d_year, c_nation
ORDER BY d_year, c_nation;
```

Requête Q4.2 :

```
SELECT d_year, s_nation, p_category, SUM(lo_revenue) -
↪  SUM(lo_supplycost) AS profit
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
LEFT JOIN customer ON lo_custkey = c_custkey
LEFT JOIN supplier ON lo_suppkey = s_suppkey
LEFT JOIN part ON lo_partkey = p_partkey
WHERE c_region = 'AMERICA'AND s_region = 'AMERICA'
AND (d_year = 1997 OR d_year = 1998)
AND (p_mfgr = 'MFGR#1' OR p_mfgr = 'MFGR#2')
GROUP BY d_year, s_nation, p_category
ORDER BY d_year, s_nation, p_category;
```

Requête Q4.3 :

```
SELECT d_year, s_city, p_brand1, SUM(lo_revenue) - SUM(lo_supplycost)
↪  AS profit
FROM lineorder
LEFT JOIN date_table ON lo_orderdate = d_datekey
LEFT JOIN customer ON lo_custkey = c_custkey
LEFT JOIN supplier ON lo_suppkey = s_suppkey
LEFT JOIN part ON lo_partkey = p_partkey
WHERE c_region = 'AMERICA' AND s_nation = 'UNITED STATES'
AND (d_year = 1997 OR d_year = 1998)
AND p_category = 'MFGR#14'
GROUP BY d_year, s_city, p_brand1
ORDER BY d_year, s_city, p_brand1;
```

ANNEXE B

# Les treize requêtes SSB dans GAMM

Voici les treize requêtes SSB formulées en langage Cypher en versions explicites et implicites, que nous avons utilisées dans le modèle GAMM.

Requête Q1.1 :

Version implicite (la requête extraira les données de la version 3 puisque la mesure QUANTITÉ n'a été ajoutée que dans la version 3)

```
optional match (d:date{D_YEAR:1997})<-[r:order_date]-(l:lineorder)
where 1<= l.LO_DISCOUNT <=3
and l.LO_QUANTITY < 25
return sum(l.LO_REVENUE);
```

Version explicite (version 3)

```
optional match (d:date{D_YEAR:1997})<-[r:order_date]-(l:lineorder)
where date({year:2021,month:01}) <= l.TRANSACTION_TIME
and 1<= l.LO_DISCOUNT <=3
and l.LO_QUANTITY < 25
return sum(l.LO_REVENUE);
```

Requête Q1.2 :

Version implicite (la requête extraira les données de la version 3)

```
optional match
↪  (d:date{D_YEARMONTHNUM:199701})<-[:order_date]-(l:lineorder)
where 4 <= l.LO_DISCOUNT <=6
and 26 <= l.LO_QUANTITY <= 35
return sum(l.LO_REVENUE);
```

Version explicite (version 3)

```
optional match (d:date{D_YEARMONTHNUM:199701})<-[:order_date]-(l:li
↪  neorder)
where date({year:2021,month:01}) <= l.TRANSACTION_TIME <=
↪  date({year:2021,month:12})
and 4 <= l.LO_DISCOUNT <=6
and 26 <= l.LO_QUANTITY <= 35
return sum(l.LO_REVENUE);
```

Requête Q1.3 :

Version implicite (la requête extraira les données de la version 3)

```
optional match (d:date{D_YEAR:1997,D_WEEKNUMINYEAR:6})<-[r:order_da
↪  te]-(l:lineorder)
where 5<= l.LO_DISCOUNT <=7
and 26<= l.LO_QUANTITY <= 35
return sum(l.LO_REVENUE);
```

Version explicite (version 3)

```
optionalmatch(d:date{D_YEAR:1997,D_WEEKNUMINYEAR:6})<-[r:order_date
↪  ]-(l:lineorder)
where date({year:2021,month:01}) <= l.TRANSACTION_TIME <=
↪  date({year:2021,month:12})
and 5<= l.LO_DISCOUNT <=7
and 26<= l.LO_QUANTITY <= 35
return sum(l.LO_REVENUE);
```

Requête Q2.1 :

Version implicite (la requête extraira les données des versions 2 et 3)

```
optional match (pb:p_brand)<-[:part_brand]-(p:part)<-[:order_part]-
↪  (l:lineorder)-[:order_supplier]->(s:supplier)-[:supplier_region
↪  ]->(sr:s_region),(d:date)<-[:order_date]-(l)
where sr.S_REGION = "AMERICA"
return sum(l.LO_REVENUE),d.D_YEAR,pb.P_BRAND
ORDER BY d.D_YEAR, pb.P_BRAND
```

Version explicite (version 2)

```
optional match (pb:p_brand)<-[:part_brand]-(p:part)<-[:order_part]-
↪  (l:lineorder)-[:order_supplier]->(s:supplier)-[:supplier_region
↪  ]->(sr:s_region),(d:date)<-[:order_date]-(l)
where date({year:2020,month:01}) <= l.TRANSACTION_TIME <
↪  date({year:2021,month:1})
and sr.S_REGION = "AMERICA"
return sum(l.LO_REVENUE),d.D_YEAR,pb.P_BRAND
ORDER BY d.D_YEAR, pb.P_BRAND
```

Requête Q2.2 :

Version implicite (la requête extraira les données des versions 2 et 3)

```
optional match (pb:p_brand)<-[:part_brand]-(p:part)<-[:order_part]-
↪  (l:lineorder)-[:order_supplier]->(s:supplier)-[:supplier_region
↪  ]->(sr:s_region),(d:date)<-[:order_date]-(l)
where (pb.P_BRAND >= "MFGR#2221" and pb.P_BRAND <= "MFGR#2228")
and sr.S_REGION = "ASIA"
return sum(l.LO_REVENUE),d.D_YEAR,pb.P_BRAND
ORDER BY d.D_YEAR, pb.P_BRAND
```

Version explicite (version 3)

```
optional match (pb:p_brand)<-[:part_brand]-(p:part)<-[:order_part]-
↪  (l:lineorder)-[:order_supplier]->(s:supplier)-[:supplier_region
↪  ]->(sr:s_region),(d:date)<-[:order_date]-(l)
where date({year:2021,month:01}) <= l.TRANSACTION_TIME <
↪  date({year:2022,month:01})
and (pb.P_BRAND >= "MFGR#2221" and pb.P_BRAND <= "MFGR#2228")
and sr.S_REGION = "ASIA"
return sum(l.LO_REVENUE),d.D_YEAR,pb.P_BRAND
ORDER BY d.D_YEAR, pb.P_BRAND
```

Requête Q2.3 :

Version implicite (la requête extraira les données des versions 2 et 3)

```
optional match (pb:p_brand)<-[:part_brand]-(p:part)<-[:order_part]-
↪  (l:lineorder)-[:order_supplier]->(s:supplier)-[:supplier_region
↪  ]->(sr:s_region),(d:date)<-[:order_date]-(l)
where pb.P_BRAND = "MFGR#2239"
and sr.S_REGION = "EUROPE"
return sum(l.LO_REVENUE),d.D_YEAR,pb.P_BRAND
ORDER BY d.D_YEAR, pb.P_BRAND
```

Version explicite (version 2)

```
optional match (pb:p_brand)<-[:part_brand]-(p:part)<-[:order_part]-
↪  (l:lineorder)-[:order_supplier]->(s:supplier)-[:supplier_region
↪  ]->(sr:s_region),(d:date)<-[:order_date]-(l)
where date({year:2020,month:01}) <= l.TRANSACTION_TIME <
↪  date({year:2021,month:01})
and pb.P_BRAND = "MFGR#2239"
and sr.S_REGION = "EUROPE"
return sum(l.LO_REVENUE),d.D_YEAR,pb.P_BRAND
ORDER BY d.D_YEAR, pb.P_BRAND
```

Requête Q3.1 :

Version implicite (la requête extraira les données des versions 1 et 2)

```
optional match
↪  (cr:c_region)<-[:customer_region]-(c:customer)<-[:order_custome
↪  r]-(l:lineorder)-[:order_date]->(d:date),(sr:s_region)<-[:suppl
↪  ier_region]-(s:supplier)<-[:order_supplier]-(l),(cn:c_nation)<-
↪  [:customer_nation]-(c),(sn:s_nation)<-[:supplier_nation]-(s)
where 1994<= d.D_YEAR <=1995
and cr.C_REGION = "ASIA"
and sr.S_REGION = "ASIA"
return cn.C_NATION, sn.S_NATION, d.D_YEAR, sum(l.LO_REVENUE) as
↪  revenu
ORDER BY d.D_YEAR ASC, revenu DESC
```

Version explicite (version 2)

```
optional match
↪  (cr:c_region)<-[:customer_region]-(c:customer)<-[:order_custome
↪  r]-(l:lineorder)-[:order_date]->(d:date),(sr:s_region)<-[:suppl
↪  ier_region]-(s:supplier)<-[:order_supplier]-(l),(cn:c_nation)<-
↪  [:customer_nation]-(c),(sn:s_nation)<-[:supplier_nation]-(s)
where date({year:2021,month:01}) <= l.TRANSACTION_TIME <
↪  date({year:2022,month:01})
and 1994<= d.D_YEAR <=1995
and cr.C_REGION = "ASIA"
and sr.S_REGION = "ASIA"
return cn.C_NATION, sn.S_NATION, d.D_YEAR, sum(l.LO_REVENUE) as
↪  revenu
ORDER BY d.D_YEAR ASC, revenu DESC
```

Requête Q3.2 :

Version implicite (la requête extraira les données des versions 1 et 2)

```
match (cn:c_nation)<-[:customer_nation]-(c:customer)<-[:order_custo
↪  mer]-(l:lineorder)-[:order_date]->(d:date),(sn:s_nation)<-[:sup
↪  plier_nation]-(s:supplier)<-[:order_supplier]-(l),(cc:c_city)<-
↪  [:customer_city]-(c),(sc:s_city)<-[:supplier_city]-(s)
where 1994<= d.D_YEAR <=1995
and cn.C_NATION = "UNITED STATES"
and sn.S_NATION = "UNITED STATES"
return cc.C_CITY, sc.S_CITY, d.D_YEAR, sum(l.LO_REVENUE) as revenu
ORDER BY d.D_YEAR ASC, revenu DESC
```

Version explicite (version 2)

```
match (cn:c_nation)<-[:customer_nation]-(c:customer)<-[:order_custo
↪  mer]-(l:lineorder)-[:order_date]->(d:date),(sn:s_nation)<-[:sup
↪  plier_nation]-(s:supplier)<-[:order_supplier]-(l),(cc:c_city)<-
↪  [:customer_city]-(c),(sc:s_city)<-[:supplier_city]-(s)
where date({year:2019,month:01}) <= l.TRANSACTION_TIME <
↪  date({year:2022,month:01})
and 1994<= d.D_YEAR <=1995
and cn.C_NATION = "UNITED STATES"
and sn.S_NATION = "UNITED STATES"
return cc.C_CITY, sc.S_CITY, d.D_YEAR, sum(l.LO_REVENUE) as revenu
ORDER BY d.D_YEAR ASC, revenu DESC
```

Requête Q3.3 :

Version implicite (la requête extraira les données des versions 1 et 2)

```
match (cc:c_city)<-[:customer_city]-(c:customer)<-[:order_customer]
↪  -(l:lineorder)-[:order_date]->(d:date),(sc:s_city)<-[:supplier_
↪  city]-(s:supplier)<-[:order_supplier]-(l)
where 1994<= d.D_YEAR <=1995
and (cc.C_CITY = "UNITED KI1" or cc.C_CITY = "UNITED KI5")
and (sc.S_CITY = "UNITED KI1" or sc.S_CITY = "UNITED KI5")
return cc.C_CITY, sc.S_CITY, d.D_YEAR, sum(l.LO_REVENUE) as revenu
ORDER BY d.D_YEAR ASC, revenu DESC
```

Version explicite (version 2)

```
optional match (cc:c_city)<-[:customer_city]-(c:customer)<-[:order_
↪  customer]-(l:lineorder)-[:order_date]->(d:date),(sc:s_city)<-[:
↪  supplier_city]-(s:supplier)<-[:order_supplier]-(l)
where date({year:2020,month:01}) <= l.TRANSACTION_TIME <
↪  date({year:2021,month:01})
and 1994<= d.D_YEAR <=1995
and (cc.C_CITY = "UNITED KI1" or cc.C_CITY = "UNITED KI5")
and (sc.S_CITY = "UNITED KI1" or sc.S_CITY = "UNITED KI5")
return cc.C_CITY, sc.S_CITY, d.D_YEAR, sum(l.LO_REVENUE) as revenu
ORDER BY d.D_YEAR ASC, revenu DESC
```

Requête Q3.4 :

Version implicite (la requête extraira les données des versions 1 et 2)

```
optional match (cc:c_city)<-[:customer_city]-(c:customer)<-[:order_
↪  customer]-(l:lineorder)-[:order_date]->(d:date),(sc:s_city)<-[:
↪  supplier_city]-(s:supplier)<-[:order_supplier]-(l)
where (cc.C_CITY = "UNITED KI1" or cc.C_CITY = "UNITED KI5")
```

```
and (sc.S_CITY = "UNITED KI1" or sc.S_CITY = "UNITED KI5")
and d.D_YEARMONTH = "Dec1995"
return cc.C_CITY, sc.S_CITY, d.D_YEAR, sum(l.LO_REVENUE) as revenu
ORDER BY d.D_YEAR ASC, revenu DESC
```

Version explicite (version 2)

```
optional match (cc:c_city)<-[:customer_city]-(c:customer)<-[:order_
↪  customer]-(l:lineorder)-[:order_date]->(d:date),(sc:s_city)<-[:
↪  supplier_city]-(s:supplier)<-[:order_supplier]-(l)
where date({year:2020,month:01}) <= l.TRANSACTION_TIME <=
↪  date({year:2021,month:01})
and (cc.C_CITY = "UNITED KI1" or cc.C_CITY = "UNITED KI5")
and (sc.S_CITY = "UNITED KI1" or sc.S_CITY = "UNITED KI5")
and d.D_YEARMONTH = "Dec1995"
return cc.C_CITY, sc.S_CITY, d.D_YEAR, sum(l.LO_REVENUE) as revenu
ORDER BY d.D_YEAR ASC, revenu DESC
```

Requête Q4.1 :

Version implicite (la requête extraira les données de la version 2)

```
optional match (cr:c_region)<-[:customer_region]-(c:customer)<-[:or
↪  der_customer]-(l:lineorder)-[:order_part]->(p:part)-[:part_mfgr
↪  ]->(pm:p_mfgr),
(sr:s_region)<-[:supplier_region]-(s:supplier)<-[:order_supplier]-(
↪  l)-[:order_date]->(d:date),(cn:c_nation)<-[:customer_nation]-(c)
where cr.C_REGION = "AMERICA"
and (pm.P_MFGR = "MFGR#1" or pm.P_MFGR = "MFGR#2")
and sr.S_REGION = "AMERICA"
return d.D_YEAR, cn.C_NATION,(sum(l.LO_REVENUE) -
↪  sum(l.LO_SUPPLYCOST)) as supplycost
ORDER BY d.D_YEAR, cn.C_NATION;
```

Version explicite (version 2)

```
optional match (cr:c_region)<-[:customer_region]-(c:customer)<-[:or
↪  der_customer]-(l:lineorder)-[:order_part]->(p:part)-[:part_mfgr
↪  ]->(pm:p_mfgr),
(sr:s_region)<-[:supplier_region]-(s:supplier)<-[:order_supplier]-(
↪  l)-[:order_date]->(d:date),(cn:c_nation)<-[:customer_nation]-(c)
where date({year:2020,month:01}) <= l.TRANSACTION_TIME <
↪  date({year:2021,month:01})
and cr.C_REGION = "AMERICA"
and (pm.P_MFGR = "MFGR#1" or pm.P_MFGR = "MFGR#2")
and sr.S_REGION = "AMERICA"
return d.D_YEAR, cn.C_NATION, (sum(l.LO_REVENUE) -
↪  sum(l.LO_SUPPLYCOST)) as supplycost
ORDER BY d.D_YEAR, cn.C_NATION;
```

Requête Q4.2 :

Version implicite (la requête extraira les données de la version 2)

```
optional match (cr:c_region)<-[:customer_region]-(c:customer)<-[:or
↪  der_customer]-(l:lineorder)-[:order_part]->(p:part)-[:part_mfgr
↪  ]->(pm:p_mfgr),
(sr:s_region)<-[:supplier_region]-(s:supplier)<-[:order_supplier]-(
↪  l)-[:order_date]->(d:date),
(cn:c_nation)<-[:customer_nation]-(c),(p)-[:part_category]->(pc:p_c
↪  ategory)
where cr.C_REGION = "AMERICA"
and (d.D_YEAR = 1994 or d.D_YEAR = 1995)
and (pm.P_MFGR = "MFGR#1" or pm.P_MFGR = "MFGR#2")
and sr.S_REGION = "AMERICA"
return d.D_YEAR, cn.C_NATION, pc.P_CATEGORY, (sum(l.LO_REVENUE) -
↪  sum(l.LO_SUPPLYCOST)) as supplycost
ORDER BY d.D_YEAR, cn.C_NATION, pc.P_CATEGORY;
```

Version explicite (version 2)

```
optional match (cr:c_region)<-[:customer_region]-(c:customer)<-[:or
↪  der_customer]-(l:lineorder)-[:order_part]->(p:part)-[:part_mfgr
↪  ]->(pm:p_mfgr),
(sr:s_region)<-[:supplier_region]-(s:supplier)<-[:order_supplier]-(
↪  l)-[:order_date]->(d:date),
(cn:c_nation)<-[:customer_nation]-(c),(p)-[:part_category]->(pc:p_c
↪  ategory)
where date({year:2020,month:01}) <= l.TRANSACTION_TIME <
↪  date({year:2020,month:01})
and cr.C_REGION = "AMERICA"
and (d.D_YEAR = 1994 or d.D_YEAR = 1995)
and (pm.P_MFGR = "MFGR#1" or pm.P_MFGR = "MFGR#2")
and sr.S_REGION = "AMERICA"
return d.D_YEAR, cn.C_NATION, pc.P_CATEGORY, (sum(l.LO_REVENUE) -
↪  sum(l.LO_SUPPLYCOST)) as supplycost
ORDER BY d.D_YEAR, cn.C_NATION, pc.P_CATEGORY;
```

Requête Q4.3 :

Version implicite (la requête extraira les données de la version 2)

```
optional match (cr:c_region)<-[:customer_region]-(c:customer)<-[:or
↪  der_customer]-(l:lineorder)-[:order_part]->(p:part)-[:part_cate
↪  gory]->(pc:p_category),
(sn:s_nation)<-[:supplier_nation]-(s:supplier)<-[:order_supplier]-(
↪  l)-[:order_date]->(d:date),(p)-[:part_brand]->(pb:p_brand),(s)-
↪  [:supplier_city]->(sc:s_city)
```

```
where date({year:2019,month:01}) <= l.TRANSACTION_TIME <=
↪  date({year:2021,month:12})
and cr.C_REGION = "AMERICA"
and pc.P_CATEGORY = "MFGR#14"
and sn.S_NATION = "UNITED STATES"
return d.D_YEAR, sc.S_CITY, pb.P_BRAND,(sum(l.LO_REVENUE) -
↪  sum(l.LO_SUPPLYCOST)) as supplycost
ORDER BY d.D_YEAR, sc.S_CITY, pb.P_BRAND;
```

Version explicite (version 2)

```
optional match (cr:c_region)<-[:customer_region]-(c:customer)<-[:or
↪  der_customer]-(l:lineorder)-[:order_part]->(p:part)-[:part_cate
↪  gory]->(pc:p_category),
(sn:s_nation)<-[:supplier_nation]-(s:supplier)<-[:order_supplier]-(
↪  l)-[:order_date]->(d:date),(p)-[:part_brand]->(pb:p_brand),(s)-
↪  [:supplier_city]->(sc:s_city)
where date({year:2020,month:01}) <= l.TRANSACTION_TIME <
↪  date({year:2021,month:01})
and cr.C_REGION = "AMERICA"
and pc.P_BRAND = "MFGR#14"
and sn.S_NATION = "UNITED STATES"
return d.D_YEAR, sc.S_CITY, pb.P_BRAND,(sum(l.LO_REVENUE) -
↪  sum(l.LO_SUPPLYCOST)) as supplycost
ORDER BY d.D_YEAR, sc.S_CITY, pb.P_BRAND;
```

ANNEXE C

# Les dix requêtes temporelles du SSB dans GAMM

Nous présentons ci-dessous dix requêtes temporelles issues du benchmark SSB, formulées en langage Cypher. Les requêtes Q1.1, Q1.2 et Q1.3 ne sont pas concernées par les changements temporels.

Requête Q2.1 :

```
optional match (pbn:p_brand_name)<-[:p_brand_name]-(pb:p_brand)<-[:
↪  part_brand]-(p:part)<-[:order_part]-(l:lineorder)-[:order_suppl
↪  ier]->(s:supplier)-[r:supplier_region]->(sr:s_region)-[:s_regio
↪  n_name]->(srn:s_region_name),(d:date)<-[:order_date]-(l)
where r.From_Date <= l.ORDERDATE < r.To_Date
and srn.S_REGION_NAME = "AMERICA"
return sum(l.LO_REVENUE),d.D_YEAR,pbn.P_BRAND_NAME
ORDER BY d.D_YEAR,pbn.P_BRAND_NAME;
```

Requête Q2.2 :

```
optional match (pbn:p_brand_name)<-[:p_brand_name]-(pb:p_brand)<-[:
↪  part_brand]-(p:part)<-[:order_part]-(l:lineorder)-[:order_suppl
↪  ier]->(s:supplier)-[r:supplier_region]->(sr:s_region)-[:s_regio
↪  n_name]->(srn:s_region_name),(d:date)<-[:order_date]-(l)
where r.From_Date <= l.ORDERDATE < r.To_Date
and (pbn.P_BRAND_NAME >= "MFGR#2221" and pbn.P_BRAND_NAME <=
↪  "MFGR#2228")
and srn.S_REGION_NAME = "ASIA"
return sum(l.LO_REVENUE),d.D_YEAR,pbn.P_BRAND_NAME
ORDER BY d.D_YEAR,pbn.P_BRAND_NAME;
```

Requête Q2.3 :

```
optional match (pbn:p_brand_name)<-[:p_brand_name]-(pb:p_brand)<-[:
↪  part_brand]-(p:part)<-[:order_part]-(l:lineorder)-[:order_suppl
↪  ier]->(s:supplier)-[r:supplier_region]->(sr:s_region)-[:s_regio
↪  n_name]->(srn:s_region_name),(d:date)<-[:order_date]-(l)
where r.From_Date <= l.ORDERDATE < r.To_Date
and pbn.P_BRAND_NAME = "MFGR#2239"
and srn.S_REGION_NAME = "EUROPE"
return sum(l.LO_REVENUE),d.D_YEAR,pbn.P_BRAND_NAME
ORDER BY d.D_YEAR,pbn.P_BRAND_NAME;
```

Requête Q3.1 :

```
optional match (crn:c_region_name)<-[:c_region_name]-(c_region)<-[r
↪  1:customer_region]-(c:customer)<-[:order_customer]-(l:lineorder
↪  )-[:order_date]->(d:date),(srn:s_region_name)<-[:s_region_name]
↪  -(sr:s_region)<-[r2:supplier_region]-(s:supplier)<-[:order_supp
↪  lier]-(l),(cnn:c_nation_name)<-[:c_nation_name]-(c_nation)<-[r3
↪  :customer_nation]-(c),(snn:s_nation_name)<-[:s_nation_name]-(sn
↪  :s_nation)<-[r4:supplier_nation]-(s)
where r1.From_Date <= l.ORDERDATE < r1.To_Date
and r2.From_Date <= l.ORDERDATE < r2.To_Date
and 1992<= d.D_YEAR <=1997
and crn.C_REGION_NAME = "ASIA"
and srn.S_REGION_NAME = "ASIA"
return cnn.C_NATION_NAME, snn.S_NATION_NAME, d.D_YEAR,
↪  sum(l.LO_REVENUE) as revenu
ORDER BY d.D_YEAR ASC, revenu DESC;
```

Requête Q3.2 :

```
optional match (cnn:c_nation_name)<-[:c_nation_name]-(c_nation)<-[r
↪  1:customer_nation]-(c:customer)<-[:order_customer]-(l:lineorder
↪  )-[:order_date]->(d:date),(snn:s_nation_name)<-[:s_nation_name]
↪  -(sn:s_nation)<-[r2:supplier_nation]-(s:supplier)<-[:order_supp
↪  lier]-(l),(ccn:c_city_name)<-[:c_city_name]-(c_city)<-[r3:custo
↪  mer_city]-(c),(scn:s_city_name)<-[:s_city_name]-(sc:s_city)<-[r
↪  4:supplier_city]-(s)
where r1.From_Date <= l.ORDERDATE < r1.To_Date
and r2.From_Date <= l.ORDERDATE < r2.To_Date
and 1992<= d.D_YEAR <=1997
and cnn.C_NATION_NAME = "UNITED STATES"
and snn.S_NATION_NAME = "UNITED STATES"
return ccn.C_CITY_NAME, scn.S_CITY_NAME, d.D_YEAR, sum(l.LO_REVENUE)
↪  as revenu
ORDER BY d.D_YEAR ASC, revenu DESC;
```

Requête Q3.3 :

```
optional match (ccn:c_city_name)<-[:c_city_name]-(c_city)<-[r1:cust
↪  omer_city]-(c:customer)<-[:order_customer]-(l:lineorder)-[:orde
↪  r_supplier]->(s:supplier)-[r2:supplier_city]->(sc:s_city)-[:s_c
↪  ity_name]->(scn:s_city_name),(l)-[:order_date]->(d:date)
where r1.From_Date <= l.ORDERDATE < r1.To_Date
and r2.From_Date <= l.ORDERDATE < r2.To_Date
and 1992<= d.D_YEAR <=1997
and (ccn.C_CITY_NAME = "UNITED KI1" or ccn.C_CITY_NAME = "UNITED
↪  KI5")
and (scn.S_CITY_NAME = "UNITED KI1" or scn.S_CITY_NAME = "UNITED
↪  KI5")
return ccn.C_CITY_NAME, scn.S_CITY_NAME, d.D_YEAR, sum(l.LO_REVENUE)
↪  as revenu
ORDER BY d.D_YEAR ASC, revenu DESC;
```

Requête Q3.4 :

```
optional match (ccn:c_city_name)<-[:c_city_name]-(c_city)<-[r1:cust
↪  omer_city]-(c:customer)<-[:order_customer]-(l:lineorder)-[:orde
↪  r_supplier]->(s:supplier)-[r2:supplier_city]->(sc:s_city)-[:s_c
↪  ity_name]->(scn:s_city_name),(l)-[:order_date]->(d:date)
where r1.From_Date <= l.ORDERDATE < r1.To_Date
and r2.From_Date <= l.ORDERDATE < r2.To_Date
and d.D_YEARMONTH = "Dec1997"
and (ccn.C_CITY_NAME = "UNITED KI1" or ccn.C_CITY_NAME = "UNITED
↪  KI5")
and (scn.S_CITY_NAME = "UNITED KI1" or scn.S_CITY_NAME = "UNITED
↪  KI5")
return ccn.C_CITY_NAME, scn.S_CITY_NAME, d.D_YEAR, sum(l.LO_REVENUE)
↪  as revenu
ORDER BY d.D_YEAR ASC, revenu DESC;
```

Requête Q4.1 :

```
optional match (crn:c_region_name)<-[:c_region_name]-(c_region)<-[r
↪  1:customer_region]-(c:customer)<-[:order_customer]-(l:lineorder
↪  )-[:order_supplier]->(s:supplier)-[r2:supplier_region]->(sr:s_r
↪  egion)-[:s_region_name]->(srn:s_region_name),
↪  (pmn:p_mfgr_name)<-[:p_mfgr_name]-(pm:p_mfgr)<-[r3:part_mfgr]-(
↪  p:part)<-[:order_part]-(l)-[:order_date]->(d:date),
↪  (cnn:c_nation_name)<-[:c_nation_name]-(c_nation)<-[r4:customer_
↪  nation]-(c)
where r1.From_Date <= l.ORDERDATE < r1.To_Date
and r2.From_Date <= l.ORDERDATE < r2.To_Date
and r3.From_Date <= l.ORDERDATE < r3.To_Date
and crn.C_REGION_NAME = "AMERICA"
```

```
and (pmn.P_MFGR_NAME = "MFGR#1" or pmn.P_MFGR_NAME = "MFGR#2")
and srn.S_REGION_NAME = "AMERICA"
return d.D_YEAR, cnn.C_NATION_NAME, (sum(l.LO_REVENUE) -
↪  sum(l.LO_SUPPLYCOST)) as supplycost
ORDER BY d.D_YEAR, cnn.C_NATION_NAME;
```

Requête Q4.2 :

```
optional match (crn:c_region_name)<-[:c_region_name]-(c_region)<-[r
↪  1:customer_region]-(c:customer)<-[:order_customer]-(l:lineorder
↪  )-[:order_supplier]->(s:supplier)-[r2:supplier_region]->(sr:s_r
↪  egion)-[:s_region_name]->(srn:s_region_name),
↪  (pmn:p_mfgr_name)<-[:p_mfgr_name]-(pm:p_mfgr)<-[r3:part_mfgr]-(
↪  p:part)<-[:order_part]-(l)-[:order_date]->(d:date),
↪  (snn:s_nation_name)<-[:s_nation_name]-(sn:s_nation)<-[r4:suppli
↪  er_nation]-(s),
↪  (pcn:p_category_name)<-[:p_category_name]-(p_category)<-[r5:par
↪  t_category]-(p)
where r1.From_Date <= l.ORDERDATE < r1.To_Date
and r2.From_Date <= l.ORDERDATE < r2.To_Date
and r3.From_Date <= l.ORDERDATE < r3.To_Date
and (d.D_YEAR = 1997 or d.D_YEAR = 1998)
and crn.C_REGION_NAME = "AMERICA"
and (pmn.P_MFGR_NAME = "MFGR#1" or pmn.P_MFGR_NAME = "MFGR#2")
and srn.S_REGION_NAME = "AMERICA"
return d.D_YEAR, snn.S_NATION_NAME,pcn.P_CATEGORY_NAME,
↪  (sum(l.LO_REVENUE) - sum(l.LO_SUPPLYCOST)) as supplycost
ORDER BY d.D_YEAR, snn.S_NATION_NAME,pcn.P_CATEGORY_NAME;
```

Requête Q4.3 :

```
optional match (crn:c_region_name)<-[:c_region_name]-(c_region)<-[r
↪  1:customer_region]-(c:customer)<-[:order_customer]-(l:lineorder
↪  )-[:order_supplier]->(s:supplier)-[r2:supplier_nation]->(sn:s_n
↪  ation)-[:s_nation_name]->(snn:s_nation_name),
↪  (pcn:p_category_name)<-[:p_category_name]-(p_category)<-[r3:par
↪  t_category]-(p:part)<-[:order_part]-(l)-[:order_date]->(d:date)
↪  ,
↪  (ccn:c_city_name)<-[:c_city_name]-(c_city)<-[r4:customer_city]-
↪  (c),
(pbn:p_brand_name)<-[:p_brand_name]-(pb:p_brand)<-[:part_brand]-(p)
where r1.From_Date <= l.ORDERDATE < r1.To_Date
and r2.From_Date <= l.ORDERDATE < r2.To_Date
and r3.From_Date <= l.ORDERDATE < r3.To_Date
and (d.D_YEAR = 1997 or d.D_YEAR = 1998)
and crn.C_REGION_NAME = "AMERICA"
and pcn.P_CATEGORY_NAME = "MFGR#14"
```

```
and snn.S_NATION_NAME = "UNITED STATES"
return d.D_YEAR, ccn.C_CITY_NAME,pbn.P_BRAND_NAME,
↪  (sum(l.LO_REVENUE) - sum(l.LO_SUPPLYCOST)) as supplycost
ORDER BY d.D_YEAR, ccn.C_CITY_NAME,pbn.P_BRAND_NAME;
```

ANNEXE D

# Les treize requêtes du SSB en Cypher

Requête Q1.1 :

```
optional match (d:date{D_YEAR:1993})<-[r:order_date]-(l:lineorder)
where 1<= l.LO_DISCOUNT <=3
and l.LO_QUANTITY < 25
return sum(l.LO_REVENUE);
```

Requête Q1.2 :

```
optional match (d:date{D_YEARMONTHNUM:199401})<-[r:order_date]-(l:
↪  lineorder)
where 4<= l.LO_DISCOUNT <=6
and 26<= l.LO_QUANTITY <= 35
return sum(l.LO_REVENUE);
```

Requête Q1.3 :

```
optional match (d:date{D_YEAR:1994,D_WEEKNUMINYEAR:6})<-[r:order_da
↪  te]-(l:lineorder)
where 5<= l.LO_DISCOUNT <=7
and 26<= l.LO_QUANTITY <= 35
return sum(l.LO_REVENUE);
```

Requête Q2.1 :

```
optional match (p:part)<-[:order_part]-(l:lineorder)-[:order_suppli
↪  er]->(s:supplier),(d:date)<-[:order_date]-(l)
where p.P_CATEGORY= "MFGR#12"
and s.S_REGION = "AMERICA"
return sum(l.LO_REVENUE),d.D_YEAR, p.P_BRAND1
ORDER BY d.D_YEAR, p.P_BRAND;
```

Requête Q2.2 :

```
optional match (p:part)<-[:order_part]-(l:lineorder)-[:order_suppli
↪  er]->(s:supplier),(d:date)<-[:order_date]-(l)
where (p.P_BRAND >= "MFGR#2221" and p.P_BRAND <= "MFGR#2228")
and s.S_REGION = "ASIA"
return sum(l.LO_REVENUE),d.D_YEAR, p.P_BRAND
ORDER BY d.D_YEAR, p.P_BRAND;
```

Requête Q2.3 :

```
optional match (p:part)<-[:order_part]-(l:lineorder)-[:order_suppli
↪  er]->(s:supplier),(d:date)<-[:order_date]-(l)
where p.P_BRAND1 = "MFGR#2239"
and s.S_REGION = "EUROPE"
return sum(l.LO_REVENUE),d.D_YEAR, p.P_BRAND
ORDER BY d.D_YEAR, p.P_BRAND;
```

Requête Q3.1 :

```
optional match (c:customer)<-[:order_customer]-(l:lineorder)-[:orde
↪  r_date]->(d:date),(s:supplier)<-[:order_supplier]-(l)
where 1992<= d.D_YEAR <=1997
and c.C_REGION = "ASIA"
and s.S_REGION = "ASIA"
return c.C_NATION, s.S_NATION, d.D_YEAR, sum(l.LO_REVENUE) as revenu
ORDER BY d.D_YEAR ASC, revenu DESC;
```

Requête Q3.2 :

```
optional match (c:customer)<-[:order_customer]-(l:lineorder)-[:orde
↪  r_date]->(d:date),(s:supplier)<-[:order_supplier]-(l)
where 1992<= d.D_YEAR <=1997
and c.C_NATION = "UNITED STATES"
and s.S_NATION = "UNITED STATES"
return c.C_CITY, s.S_CITY, d.D_YEAR, sum(l.LO_REVENUE) as revenu
ORDER BY d.D_YEAR ASC, revenu DESC;
```

Requête Q3.3 :

```
optional match (c:customer)<-[:order_customer]-(l:lineorder)-[:orde
↪  r_date]->(d:date),(s:supplier)<-[:order_supplier]-(l)
where 1992<= d.D_YEAR <=1997
and (c.C_CITY = "UNITED KI1" or c.C_CITY = "UNITED KI5")
and (s.S_CITY = "UNITED KI1" or s.S_CITY = "UNITED KI5")
return c.C_CITY, s.S_CITY, d.D_YEAR, sum(l.LO_REVENUE) as revenu
ORDER BY d.D_YEAR ASC, revenu DESC;
```

Requête Q3.4 :

```
optional match (c:customer)<-[:order_customer]-(l:lineorder)-[:orde
↪  r_date]->(d:date),(s:supplier)<-[:order_supplier]-(l)
where (c.C_CITY = "UNITED KI1" or c.C_CITY = "UNITED KI5")
and (s.S_CITY = "UNITED KI1" or s.S_CITY = "UNITED KI5")
and d.D_YEARMONTH = "Dec1997"
return c.C_CITY, s.S_CITY, d.D_YEAR, sum(l.LO_REVENUE) as revenu
ORDER BY d.D_YEAR ASC, revenu DESC;
```

Requête Q4.1 :

```
optional match (c:customer)<-[:order_customer]-(l:lineorder)-[:orde
↪  r_part]->(p:part),(d:date)<-[:order_date]-(l)-[:order_supplier]
↪  ->(s:supplier)
where c.C_REGION = "AMERICA"
and (p.P_MFGR = "MFGR#1" or p.P_MFGR = "MFGR#2")
and s.S_REGION = "AMERICA"
return d.D_YEAR, c.C_NATION, (sum(l.LO_REVENUE) -
↪  sum(l.LO_SUPPLYCOST)) as supplycost
ORDER BY d.D_YEAR, c.C_NATION;
```

Requête Q4.2 :

```
optional match (c:customer)<-[:order_customer]-(l:lineorder)-[:orde
↪  r_part]->(p:part),(d:date)<-[:order_date]-(l)-[:order_supplier]
↪  ->(s:supplier)
where c.C_REGION = "AMERICA"
and (d.D_YEAR = 1997 or d.D_YEAR = 1998)
and (p.P_MFGR = "MFGR#1" or p.P_MFGR = "MFGR#2")
and s.S_REGION = "AMERICA"
return d.D_YEAR, s.S_NATION, p.P_CATEGORY,(sum(l.LO_REVENUE) -
↪  sum(l.LO_SUPPLYCOST)) as profit
ORDER BY d.D_YEAR, s.S_NATION, p.P_CATEGORY;
```

Requête Q4.3 :

```
optional match (c:customer)<-[:order_customer]-(l:lineorder)-[:orde
↪  r_part]->(p:part),(d:date)<-[:order_date]-(l)-[:order_supplier]
↪  ->(s:supplier)
where (d.D_YEAR = 1997 or d.D_YEAR = 1998)
and c.C_REGION = "AMERICA"
and p.P_CATEGORY = "MFGR#14"
and s.S_NATION = "UNITED STATES"
return d.D_YEAR, s.S_CITY, p.P_BRAND,(sum(l.LO_REVENUE) -
↪  sum(l.LO_SUPPLYCOST)) as supplycost
ORDER BY d.D_YEAR, s.S_CITY, p.P_BRAND;
```