

PREMIÈRE PARTIE

I

PARTIE 1

CHAPITRE

1

CADRE INSTITUTIONNEL DE L'ÉTUDE

Le cursus en Master 2, est sanctionné par la rédaction et la soutenance d'un mémoire de fin de formation. Afin de valider notre diplôme, il est nécessaire d'effectuer un stage, au cours duquel nos acquis théoriques seront réinvestis pour solutionner des problèmes métiers. À cet effet, ce chapitre présente le cadre institutionnel de notre stage et du sujet qui nous a été confié. Après une présentation de la structure d'accueil et de la problématique, nous allons exposé le cahier de charges ainsi que la méthode de travail adoptée.

1.1 CADRE CONTEXTUEL

1.1.1 PRÉSENTATION DE L'ORGANISME D'ACCUEIL

Sanlam Group

Fondée en 1918 en tant que compagnie d'assurance vie, Sanlam (South African National LifeAssurance Company Limited) est aujourd'hui un groupe leader des services financiers diversifiés, basé en Afrique du Sud. Il déploie ses activités à travers

l'ensemble du continent africain ainsi qu'en Malaisie, aux Etats-Unis, en France, en Suisse, en Inde et en Australie. Groupe financier de référence coté à la bourse de Johannesburg, Sanlam propose une offre de solutions financières complètes et personnalisées dans tous les segments du marché, à travers ses 5 pôles d'activités : Sanlam Personal Finance, Sanlam Pan-Africa, Sanlam Investments, Sanlam Corporate et Santam (South African National Trust and Assurance Company Limited). C'est l'un des plus grands groupes d'assurance ayant une couverture internationale,

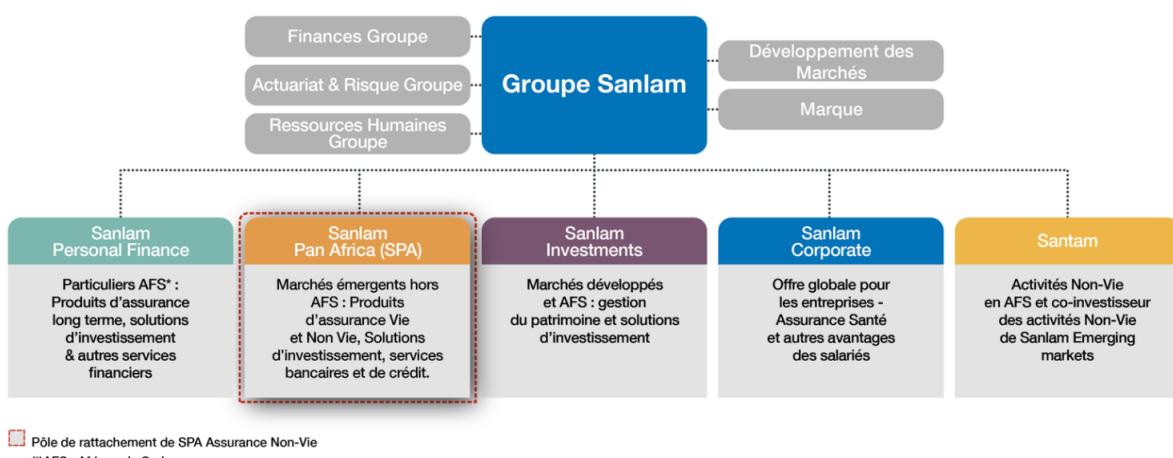


FIGURE 1.1 – Organigramme de Sanlam Group

dans le monde, en terme de présence, avec une présence directe et indirecte dans 44 pays, à l'exception de l'Afrique du Sud. Sanlam Group, fournit plus précisément des solutions et produits financiers aux particuliers et aux entreprises à savoir :

- planification financière et conseil ;
- assurance (Incendies Accidents et Risques Divers, Vie, spécialiste et réassurance) ;
- gestion de patrimoine, intermédiaire boursier ;
- retraite, gestions de fonds, santé ;
- asset management.

À Sanlam Group, on développe une vision portée sur la création de valeur pour le client, placé au cœur de toute stratégie de développement. Par ailleurs, il ambitionne de consolider ses positions sur le segment des solutions d'investissement sur les marchés développés. Sa vision stratégique est de créer de la valeur durable pour toutes les parties prenantes. Elle se décline sur 3 axes principaux :

- pour l'Afrique du Sud : être leader dans la gestion de patrimoine, le management et la protection ;
- pour l'Afrique, l'Inde, la Malaisie et le Liban : être un Groupe de services financiers panafricain de premier plan avec une présence significative en Inde et en Malaisie ;
- pour les marchés développés : se positionner sur la niche de gestion du patrimoine et de placements sur des marchés développés spécifiques.

Au sein du groupe notre stage, s'est déroulé au pôle Sanlam Pan-Africa.

Sanlam Pan-Africa

Sanlam Pan-Africa (SPA) est le pôle d'activités de Sanlam Group chargé de la gestion des services financiers sur les marchés émergents (hors Afrique du Sud). SPA assure ainsi le développement et le déploiement d'une gamme diversifiée de produits d'assurance Vie et Non-Vie, ainsi que des solutions d'investissement, des services bancaires et de crédit à la consommation, la bancassurance, la gestion d'actifs et les produits d'assurances Non-Vie spécialisées. L'Afrique est aujourd'hui une composante fondamentale de la vision de Sanlam Group, d'où la mission stratégique fondamentale dévolue à SPA : ériger un groupe de services financiers panafricain de premier plan. Disposant de la première empreinte panafricaine qui lui garantit un rayonnement continental unique, SPA se positionne notamment en tant que partenaire privilégié des multinationales et autres réseaux de distribution.

Au sein de SPA, nous avons effectué notre stage à la Digital Factory de Saham Maroc, afin de bénéficier d'un suivi adéquat.

Digital Factory

La Digital Factory de Saham Assurance Maroc, est un espace où des experts en digital travaillent et réfléchissent sur l'accélération de la transformation digitale et l'élaboration de solutions pour une expérience client optimale. La Digital Factory se veut *customer centric* et proactive pour livrer des produits efficaces et utiles pour les métiers en un temps beaucoup plus court. Pour cela, en plus des compétences recrutées et de leurs expériences, elle mise sur une approche organisationnelle du travail : l'agilité à travers la méthode Scrum. Grâce à la Digital Factory, Saham Assurance Maroc innove sur des produits qu'elle lance au fur et à mesure. Elle comprend plusieurs équipes constituées aussi bien de développeur,d'UI/UX designer,

de data scientist, data engineer, data architect, de coach agile...

En raison des conditions sanitaires, notre stage s'est effectué physiquement à Colina Participations (qui est la *holding* du groupe Sanlam détenant les parts dans les différentes filiales en Afrique hors Maroc et Afrique du Sud) mais virtuellement (à distance) à la Digital Factory. Nous avons été suivis et encadrés dans un environnement convivial et agile. Ce stage nous a permis d'être beaucoup plus autonomes et de pouvoir transformer les besoins métiers ainsi que les contraintes business en réalisations techniques.

1.1.2 QUALITÉ DES DONNÉES EN ASSURANCE

L'assurance est un secteur qui a pour principale mission de fournir une prestation lors de la survenance d'un événement incertain et aléatoire souvent appelé «*risque*». La prestation, généralement financière, peut être destinée à un individu, une association ou une entreprise, en échange de la perception d'une cotisation ou prime. L'assurance se définit aussi comme étant une opération par laquelle l'assuré transfère ses risques à l'assureur en contrepartie du paiement d'une prime.

Bien qu'étant un secteur majeur dans les activités économiques d'un pays, l'assurance se distingue de la majorité de ces derniers par l'inversion du cycle de production. En effet, il est impossible aux compagnies de savoir avec certitude, combien la prestation qu'elles vendent leur coûtera, la prime étant payée par le client avant que la prestation (indemnisation en cas de sinistre) n'ait été fournie par l'assureur. Ainsi, pour fixer le montant de sa prime ou calculer les provisions, l'assureur ne peut se baser que sur des études statistiques lui permettant de se faire une idée de combien lui coûtera sa prestation en analysant par exemple le taux de sinistralité et le montant moyen des sinistres des années passées. Cela ne donne pas pour autant la certitude qu'il n'aura pas à faire face à de plus gros coûts (en cas d'augmentation du taux de sinistralité dans l'année en cours, par exemple, ou de survenance d'une catastrophe naturelle imprévisible – tremblement de terre, tempête, épidémie mondiale etc.). Le secteur se trouve pour ainsi dire dans un environnement incertain et malgré tout concurrentiel. La fiabilité des systèmes d'informations et la qualité des données doivent alors constituer un objectif permanent.

Ce sujet, est donc au cœur du modèle d'affaires des assureurs. Il en ressort que la parfaite maîtrise des systèmes d'informations et des dispositifs de sécurités sont

des leviers stratégiques voire vitale pour maintenir une position de leader dans le domaine. Elle permet notamment de :

- se distinguer en terme de tarification, grâce à une segmentation plus efficiente avec la possibilité de créer de nouvelles offres et de mieux comprendre le marché ;
- se distinguer en terme de gestion des risques, par une optimisation des couvertures et des provisions,
- mettre en place de meilleurs systèmes de détection des fraudes.

L'enjeu est crucial à tout niveau : que ce soit pour une bonne appréhension des risques, pour mener les études actuarielles, réaliser les tarifications, évaluer les provisions ou fiabiliser les modèles, etc.

Les organismes assureurs sont donc, sensibles aux gains de productivité espérés qui pourront se traduire dans la compétition avec les autres acteurs du marché. De ce fait, les défauts de qualité des données peuvent constituer un frein à la compétitivité du groupe face aux concurrents et s'avérer être coûteux pour plusieurs raisons. Tout d'abord, ils rendent plus difficiles l'ensemble des travaux de production puisqu'ils complexifient les traitements. De plus, des données de mauvaises qualités sont susceptibles de conduire à une dégradation ou à l'allongement des travaux et des analyses qui en résultent. Enfin, elles impactent la qualité des services offert aux clients. En effet, selon une étude du Massachusetts Institute of Technology (MIT, 2017) [1], la non-qualité occasionne une perte d'argent estimée entre 15% et 25% du chiffre d'affaire total de la plupart des entreprises. Environ, 20 ans plutôt cette perte s'évaluait entre 5% et 10% du revenu des entreprises[2]. De même, dans [3], l'institut Gartner estime que plus de 25% des données critiques des plus grandes entreprises mondiales sont erronées et précise également qu'un tel problème n'a pas une conséquence informatique mais plutôt une conséquence commerciale, se chiffrant en millions de devise monétaire. Cette perte représentant environ un quart des revenus, s'explique par les mauvais choix stratégiques opérés à partir d'informations erronées, mais aussi par le temps perdu par les services informatiques à traiter ces données inexactes : contrôles, corrections et maintenances. Ramener au secteur de l'assurance, une non-qualité des données peut nuire aux décisions prises s'agissant aussi bien des exigences réglementaires que des choix stratégiques de l'entreprise (mauvaise interprétation de la situation actuelle par exemple) [4]. Prendre une décision à partir de mauvaises informations nuit à l'entreprise, à ses clients ou ses partenaires.

La définition d'une bonne stratégie de gouvernance des données est un sujet qui prend de l'importance dans les entreprises et les administrations. Il urge alors de se pencher plus sérieusement sur cette problématique tout en prenant en compte le contexte technologique. Nous touchons là au cœur de l'activité des compagnies d'assurances. Dans un environnement économico-financier sinistré, il est impensable que les données de l'assureur ne soient pas à la hauteur des attentes. Une défaillance sur ce volet-là a de lourdes conséquences. Point d'ancrage de toute stratégie de gouvernance des données, la mise en place d'un projet de qualité des données requiert quelques interrogations : qu'est ce que la qualité des données et comment la mesurer ? Quelles sont les possibilités qu'offrent l'outil Apache Griffin ? Comment corrigé les problèmes de qualité des données détectés ?

1.2 MANAGEMENT DU PROJET

1.2.1 CAHIER DE CHARGES

Le présent projet s'inscrit dans une perspective plus globale de gouvernance des données (*data governance*). En effet, les données prennent une place de plus en plus importante dans les prises de décisions des entreprises. Mais comment réunir les conditions nécessaires à une exploitation saine de ces données ? La gouvernance des données a donc pour rôle de s'assurer de la qualité et de la sécurité des données au sein d'une organisation. Pour cela, elle détermine un ensemble de processus, rôles, règles, normes et métriques permettant d'assurer une utilisation efficace et efficiente des informations, dans le but d'aider les entreprises à atteindre leurs objectifs.

Elle inscrit l'entreprise dans une démarche de procédures et de responsabilités garantissant la qualité et la sécurité des données. Une stratégie claire, est fondamentale pour toute organisation traitant les *big data*. La qualité des données est à l'origine de la plupart des activités de gouvernance des données. À la Digital Factory de Saham Maroc, le besoin s'est fait ressentir de mettre en place un outil permettant d'industrialiser la gestion de la qualité des données. Cet outil viendra remplacer les tests manuels (SQL) et les explorations effectués sans industrialisation.

Objectif général

L'objectif générale est de détecter à l'aide d'Apache Griffin les problèmes de qualité et de corriger dans le cadre du projet socle de données de la Digital Factory. Il est

de ce fait subdiviser en deux volets : le volet outil et le volet qualité de données.

> **Volet outil : Objectifs spécifiques** Il s'agit pour ce volet de :

1. prendre en main l'outil d'audit de la qualité des données Apache Griffin ;
2. établir la connexion avec les différentes bases de données de l'équipe socle de données ;
3. faire une analyse des perspectives qu'offre cet outil.

> **Volet correctif : Objectifs spécifiques** Il s'agira ici de se servir de l'outil pour identifier les différents problèmes de qualités de données et le cas échéant proposer des mesures de redressement. Plus précisément, il faudra faire :

1. une revue des différentes données du socle et détecter les incohérences par rapport aux différentes données / extractions utilisées par le métier ;
2. une priorisation des champs à mettre en qualité en urgence ;
3. une implémentation d'algorithmes de redressement des données quand cela est possible ;
4. éventuellement une injection d'open data pour une mise en qualité des données prioritaires.

1.2.2 MÉTHODE DE TRAVAIL AGILE

La méthode Agile est une méthodologie de gestion de projet. Il s'agit d'une organisation de travail en cycles courts, permettant aux équipes de développement de gérer un produit de manière souple, adaptative et itérative. Son but est d'améliorer leur process et réduire leur taux d'échec. Pour cela, elle place le client au cœur du projet et s'adapte tout le long du projet. De plus, au lieu de planifier le projet de A à Z dès le départ, ce qui laisse peu de place aux imprévus, des objectifs courts sont fixés, par exemple à deux ou trois semaines. Le projet est divisé en sous-projets et l'on ne passe au suivant que lorsque le précédent est réglé.

Le principal avantage est la flexibilité, la possibilité de s'adapter en fonction des nouvelles exigences du client ou des évolutions du marché. Cela permet aussi un meilleur contrôle des coûts, puisqu'un point budget peut être fait à chaque étape. Les effets positifs se font enfin ressentir sur la motivation : les collaborateurs voient les tâches avancer, au lieu de passer plusieurs mois d'affilé sur un gros dossier qui

semble stagner et de redouter la date limite finale.

Il existe en réalité plusieurs méthodes qui ont toutes un point commun : elles décourent toutes du Manifeste Agile. Scrum est aujourd’hui l’approche Agile la plus répandue, il s’agit plus précisément d’un cadre méthodologique plutôt que d’une méthode. Elle est d’ailleurs celle implantée par la Digital Factory. De plus, Scrum est une pratique Agile élémentaire qui permet également une mise à l’échelle, autrement dit le déploiement progressif de l’agilité à l’échelle de l’entreprise. Scrum est constitué d’une définition des rôles, de réunions et d’artefacts [5] [6].

Rôles

1. **Le «*Product Owner*» ou «*PO*» :** il porte la vision du produit à réaliser, il s’agit donc généralement d’un expert métier. Il travaille en collaboration directe avec l’équipe de développement et a notamment la charge de remplir le «*Product Backlog*» et de déterminer la priorité des «*user stories*» (phrase simple, rédigée dans un langage courant, qui permet de décrire avec suffisamment de précision le contenu d’une fonctionnalité) à réaliser.
2. **Le «*Scrum Master*» ou «*SM*» :** Il s’agit d’un membre à part entière de l’équipe de projet, et il doit maîtriser Scrum car il est chargé de s’assurer que la méthodologie est correctement appliquée. Il ne faut surtout pas le confondre avec un chef de projet. Il facilite le dialogue et le travail entre les différents intervenants, de façon à ce que l’équipe soit pleinement productive. Le rôle de «*Scrum Master*» change régulièrement au sein de l’équipe projet.
3. **L’équipe de développement :** généralement composée de 4 à 6 personnes, elle est chargée de transformer les besoins exprimés par le «*Product Owner*» sous la forme de «*user stories*» en fonctionnalités réelles, opérationnelles et utilisables. L’équipe est généralement composée de plusieurs profils, en fonction des besoins.

Réunions

1. **Planification du *Sprint*** (*Sprint* = itération) : au cours de cette réunion, l’équipe de développement sélectionne les éléments prioritaires du «*Product Backlog*» (liste ordonnancée des exigences fonctionnelles et non fonctionnelles du projet) qu’elle pense pouvoir réaliser au cours du *Sprint* (en accord avec le «*Product Owner*»).
2. **Revue de *Sprint*** : au cours de cette réunion qui a lieu à la fin du *Sprint*,

l'équipe de développement présente les fonctionnalités terminées au cours du sprint et recueille les feedbacks du «*Product Owner*» et des utilisateurs finaux. C'est également le moment d'anticiper le périmètre des prochains *Sprints* et d'ajuster au besoin la planification de *release* (nombre de *Sprints* restants).

3. **Rétrospective de *Sprint*** : la rétrospective qui a généralement lieu après la revue de sprint est l'occasion de s'améliorer (productivité, qualité, efficacité, conditions de travail, etc) à la lueur du "vécu" sur le *Sprint* écoulé (principe d'amélioration continue).
4. **Mêlée quotidienne** : il s'agit d'une réunion de synchronisation de l'équipe de développement qui se fait debout (elle est aussi appelée «*stand up meeting*») en 15 minutes maximum au cours de laquelle chacun répond principalement à 3 questions : «*Qu'est ce que j'ai terminé depuis la dernière mêlée ? Qu'est ce que j'aurai terminé d'ici la prochaine mêlée ? Quels obstacles me retardent ?*».

Artefacts

1. **Le *Sprint*** : il s'agit d'une période pendant laquelle un travail spécifique doit être mené à bien avant de faire l'objet d'une révision.
2. **Le *Product Backlog*** : il s'agit d'une liste hiérarchisée des exigences initiales du client concernant le produit à réaliser.
3. **Le *Sprint Backlog*** : c'est le plan détaillé de la réalisation de l'objectif du *Sprint*, défini lors de la réunion de planification du *Sprint*.
4. **Le *Task Board*** : outil central du *Sprint scrum*, ce tableau de bord du projet permet de suivre en temps réel la progression de la réalisation des différentes tâches. Il est composé de trois colonnes comportant les tâches à faire, les tâches en cours et les tâches terminées.
5. **Le *Burndown Chart*** : il s'agit d'un graphique simple permettant de visualiser le degré d'avancement de chacune des tâches. Il est généralement mis à jour lors de la réunion quotidienne.

CHAPITRE

2

CADRE THÉORIQUE ET MÉTHODOLOGIQUE DE L'ÉTUDE

Pour mener à bien la présente étude, il est nécessaire d'effectuer un point théorique afin de mieux la situer dans le contexte et de comprendre les réflexions théoriques sous jacentes. Ainsi, ce chapitre fait un résumé de l'existant en matière de cadre d'analyse de la qualité des données et amorce la compréhension de l'outil de qualité Apache Griffin . Suite à une clarification conceptuelle, nous ferons la lumière sur les dimensions de qualité de données tout en abordant la question des mesures correctives.

2.1 CLARIFICATION CONCEPTUELLE

2.1.1 BIG DATA

Le *big data* (données massives), fait désormais partie du quotidien de toutes les entreprises. Bien qu'omniprésent dans notre quotidien, il constitue un phénomène nouveau et parfois difficile à définir. Le terme *big data* a été popularisé par John Mashey, informaticien chez Silicon Graphics dans les années 1990 [7]. Ce dernier,

faisait référence, aux bases de données trop grandes et complexes pour être étudiées avec les méthodes statistiques traditionnelles – et, par extension, à tous les nouveaux outils d'analyse de ces données. En 2001, Douglas Laney a analysé cette nouvelle tendance à travers une liste très simple de trois « *V* », ensuite élargie à cinq « *V* » [7] [8] :

- le volume : pour désigner la grande quantité de données ou d'informations contenues dans ces bases de données ;
- la vélocité : également appelée vitesse, correspond à la rapidité à laquelle les données sont générées, collectées et circulent pour transmission et analyse ;
- la variété : pour désigner la multiplicité des types de données disponibles, autrement dit les différences de natures, formats et structures⁽¹⁾ ;
- la valeur : fait référence à la capacité de ces données à générer du profit ; chaque donnée devant apporter une valeur ajoutée à l'entreprise ;
- la véracité : qui permet de garantir la qualité et la fiabilité des données.

Ces cinq « *V* » permettent donc de décrire et de caractériser les *big data*. Nous nous intéressons dans cette étude à la dernière caractéristique. En effet, pour pouvoir tirer de la valeur des données, la qualité est la condition préalable à l'analyse et à l'utilisation du *big data*. Au vu de la masse des données, il y a beaucoup de choses à affiner, pour atteindre cette qualité. D'où la nécessité de prendre des mesures de précaution pour minimiser les biais liés au manque de fiabilité des données. Les méthodes permettant d'améliorer et de garantir la qualité des *big data* sont essentielles pour prendre des décisions commerciales précises, efficaces et fiables. Mais qu'est-ce qu'une donnée de qualité ?

2.1.2 QUALITÉ DES DONNÉES

Définir la qualité des données n'est pas une opération aisée. On est bien souvent tenter de plutôt définir la non-qualité [9]. En effet, il est plus facile dans ce cas d'identifier ce qu'on ne souhaite pas avoir dans nos données. Afin de mieux saisir cette notion, nous définirons d'abord ce qu'on entend par qualité, information et donnée avant de revenir sur la définition de la qualité des données en elle-même. À cet effet, d'après [10], la qualité pourrait se définir comme le degré auquel un ensemble de caractéristiques inhérentes à un objet répond aux exigences. On parle également de conformité aux exigences. Toujours selon [10], l'exigence se définit

(1). Données structurées, semi-structurées ou non structurées

comme un besoin ou une attente énoncée ; généralement implicite ou obligatoire.

Dans [9], on retiendra que les données « *sont des faits et des statistiques qui peuvent être quantifiées, mesurées, comptées, et stockées* » et que l'information quant à elle « *est un ensemble de données organisées selon une ontologie qui définit les relations entre certains sujets* ». La qualité des données pourrait donc se définir simplement comme le degré auquel un ensemble de caractéristiques inhérentes aux données répond aux exigences [10]. Mais plus loin, Wang et Strong(1996) cité dans [11], définissent la qualité comme l'aptitude à l'emploi et proposent que le jugement de la qualité des données dépende des consommateurs de données. L'objectif est quand même, d'avoir à disposition des données exemptes d'erreurs, d'incohérences, de redondances, de formatage médiocre et d'autres problèmes susceptibles d'empêcher une utilisation aisée [12]. Ces deux définitions font ressortir deux aspects très important qui se réfèrent également dans la littérature. En effet, tous les auteurs s'accordent sur le fait que la qualité dépend non seulement de ses caractéristiques propres, mais aussi de l'environnement dans lequel ces données sont utilisées [11]. En d'autres termes, la qualité des données dépend autant de son utilisation que de son état.

Cette perception met ainsi en exergue le caractère subjectif de cette notion en raison des diverses finalités de son utilisation. Aussi peut-on lire dans [13], que :

- pour le consommateur par exemple, des données de qualité sont des données :
 - qui sont aptes à être utilisées ;
 - qui répondent à ses attentes ou les dépassent et ;
 - qui satisfont aux exigences de leur utilisation prévue ;
- il en est de même, pour l'entreprise qui de façon spécifique, s'inscrit dans un cadre opérationnel, décisionnels et commerciale ;
- par contre, du point de vue des normes, la qualité des données est mesurée par :
 - le degré auquel un ensemble de caractéristiques (dimensions de qualité) répond aux exigences ainsi que ;
 - l'utilité, la précision et l'exactitude des données pour leur utilisation.

Cette diversité de point de vue, se justifie par le fait qu'avec l'avènement du *big data*, contrairement au passé, les utilisateurs de données ne sont pas nécessairement les producteurs de ces données ; renforçant de ce fait l'absence d'une définition unique.

Abondant dans le même sens, sur la définition de la qualité des données, le NISS (National Institute of Statistical Sciences) [14] identifie sept(7) principes clés permettant de saisir sa **quitessence**. Ainsi, ils énoncent que les données peuvent être vues comme un produit et leur qualité dépend de multiples facteurs. En principe, la qualité peut être mesurée et améliorée.

Cette clarification conceptuelle montre sans équivoque que la qualité des données est **concept** à dimensions multiples. Chacune faisant référence à un aspect spécifique. Elles décrivent des caractéristiques qui peuvent être mesurées ou évaluées par rapport à des attentes prédéfinies [15]. Le caractère mesurable des dimensions s'avère nécessaire pour leur quantification dans la pratique. On parle alors de métrique. Une métrique de qualité des données selon [16], est par conséquent une fonction qui à une dimension de qualité associe une valeur numérique, ce qui permet d'interpréter sa réalisation. Une telle métrique peut être calculée à différents niveaux d'agrégation : au niveau des valeurs, des colonnes ou des attributs, des tuples ou des enregistrements, des tables ou des relations, ainsi qu'au niveau de la base de données. À la faveur de cette clarification conceptuelle, on pourrait alors se demander comment mesurer ou évaluer la qualité de nos données ? Quelles sont les dimensions qui existent et quels aspects permettent-elles de capter ?

2.2 REVUE DE LITTÉRATURE

La qualité des données est bien souvent à tort réduite à une mesure d'exactitude : par exemple, le nom de la ville "Abidjan" mal orthographié en "Abdjan" serait le seul type de problème rencontré. En effet, on considère qu'une donnée est de mauvaise qualité si des fautes de frappe sont présentes ou si des valeurs erronées s'y trouvent. **Mais cela ne se résume pas qu'à cela : la qualité des données ne se limite pas qu'à l'exactitude.** D'autres dimensions non moins importantes sont nécessaires pour pleinement la caractériser. Une étude de la qualité des données ne saurait alors se faire sans avoir clairement identifier des dimensions cibles. Il s'agira ici de faire d'entrée, une revue des différents événements qui peuvent entraver la qualité des données, pour ensuite déboucher sur quelques dimensions présentées dans la littérature et enfin les pistes de correctifs proposées.

2.2.1 DÉFIS DE LA QUALITÉ DES *BIG DATA*

Le *big data*, loin de ce qu'il pourrait laisser **imaginé**, n'est évidemment pas qu'une simple question de taille. L'extraction et le traitement de données de haute qualité, massives, variables et compliquées devient de nos jours une préoccupation très urgente. En effet, l'ère de l'information moderne produit des tonnes de données. Environ 1,7 mégaoctets de données ont été générées chaque seconde, par chaque individu tout au long de l'année 2020 [17]. Avec l'avènement des téléphones intelligents et de l'internet, des quantités phénoménales de données sont créées. À mesure que le volume et la variété des données augmentent, il devient plus difficile de contrôler chaque entrée afin de s'assurer de leur bonne qualité : les défis à relever sont énormes. Cai et Zhu [11], citent quelques défis auxquels le *big data* est confronté de nos jours en terme de qualité :

- la diversité des sources de données, qui entraîne une abondance de types et de structures complexes, augmente la difficulté de leur intégration ;
- un volume énorme de données, qui rend difficile l'appréciation de la qualité des données dans un délai raisonnable ;
- avec les données qui arrivent en temps réel *streaming* ou l'internet des objets, il devient plus difficile d'évaluer la qualité. Et lorsqu'elle ne peut être évaluée, les données ne sont plus fiables, ce qui induit des décisions imprécises ;
- l'inexistence de normes unifiées et approuvées de qualité des données ;
- de plus, la recherche sur la qualité des *big data* est relativement récente.

Voilà donc, les défis qu'une entreprise à la recherche d'un cadre méthodologique d'analyse de la qualité de ses *big data* doit relever. En effet, aucune entreprise ne saurait aspirer à une réelle expansion sans intégrer le *big data* et une véritable stratégie de gouvernance des données.

2.2.2 SOURCES DE NON-QUALITÉ

Les problèmes de qualité de données surviennent lorsque les exigences qualité ne sont pas satisfaites. Mais surtout, ces problèmes sont dûs à plusieurs facteurs ou processus. En effet, comme on peut le lire dans [18], [19],[9], [20] ; la littérature énonce quelques sources de non-qualité couramment rencontrées :

- l'entrée des données par l'homme ;
- la dégradation de la donnée dans les chaînes et processus de traitement : troncatures, caractères mal interprétés, erreurs lors des conversions, rejets non

traités, absence de contrôles sur le format, passage d'un système d'encodage à un autre,... ;

- la corruption volontaire ou intentionnelle des données à des fins malhonnêtes ;
- des données non actualisées qui deviennent une source d'inexactitude avec le temps ;
- des défauts de conception qui en laissant subsister une ambiguïté sémantique peuvent amener à des erreurs de valorisation et également d'interprétation de la donnée par la suite ;
- des définitions d'attributs pas suffisamment bien structurées ou normalisées lors de la modélisation conceptuelle des données ainsi qu'un schéma non valide et/ou un manque de contraintes d'intégrité et de procédure pour maintenir la cohérence des données ;
- l'intégration de données provenant de sources externes et faisant l'objet de contradiction ou d'incohérences avec celles locales.

Combinée avec les caractéristiques majeures du *big data*, la complexité de l'organisation d'une entreprise et la multiplicité des chaînes de traitement augmentent le risque de tous ces facteurs laissant ainsi surgir de nouveaux défis. Dans ses conditions, plus l'anomalie ou la non-qualité est détectée tôt, suivie et corrigée à la source, plus l'ensemble du patrimoine global de données sera fiable. D'où la nécessité d'un outil d'audit de la qualité des données. Mais avant de penser à l'outil, il serait judicieux de se pencher sur les aspects de la qualité qu'on souhaite abordé. C'est pour cela qu'il est important d'analyser les dimensions de qualité existantes, largement utilisées pour évaluer la qualité des données, afin de déterminer dans quelles mesures elles sont applicables au *big data*.

2.2.3 ÉVALUATION DE LA QUALITÉ DES DONNÉES

Pour garantir des données d'une certaine qualité aux utilisateurs, et des décisions pertinentes, chaque organisation se doit de définir les dimensions qu'elle utilisera dans son processus. Ben Salem [19], précise que chaque organisme doit créer ses propres définitions opérationnelles en fonction de ses objectifs et priorités, de sorte à déterminer des indicateurs pour chacune des dimensions choisies, et vérifier par des mesures régulières leur évolution dans le temps. Les données doivent avoir la qualité nécessaire pour supporter le type d'utilisation prévue. Pour les données structurées, la littérature **proposent** différentes dimensions.

Dimensions de la qualité des données

Bien que relativement récent, les travaux sur les dimensions de la qualité des données ont mis en exergue un ensemble de dimensions. Plusieurs auteurs se sont prêtés à cet exercice de revue des différentes dimensions utilisées pour attester de la qualité d'un ensemble de données. Notre étude porte principalement sur l'utilisation d'Apache Griffin, comme outil d'audit de la qualité des données. Ce dernier, s'inspire de la définition que donne la Data Administration Management Association of United Kingdom (DAMA UK) [15], qui identifie six principales dimensions pour l'évaluation de la qualité des données [15], [16] :

1. **l'exhaustivité ou la complétude (*completeness*)** : qui mesure l'absence de valeurs manquantes (chaînes de caractères nulles ou vides, données numériques manquantes). Il est à noter que le nombre de valeurs manquantes peut être calculé de différentes manières, soit en ne prenant en compte que les vraies valeurs manquantes (i.e. null), soit les valeurs par défaut ou une entrée textuelle mentionnant "NaN" (c'est-à-dire, *Not a Number*) ;
2. **l'unicité (*uniqueness*)** : cette dimension permet l'analyse des valeurs uniques. L'unicité est l'inverse d'une évaluation des doublons. Ces deux aspects représentent les faces d'une même pièce ;
3. **l'actualité (*timeliness*)** : décrit le degré de fraîcheur des données pour la tâche à accomplir et est étroitement liée aux notions de fréquence de mise à jour des données et de volatilité (vitesse à laquelle les données deviennent non pertinentes). Une autre définition indique que l'actualité peut être interprétée comme la probabilité qu'un attribut soit toujours à jour [16]. Elle se mesure en détectant le taux de valeurs obsolètes dans la base de données par rapport à une date prédéfinie ou en analysant la latence entre les différentes entrées ;
4. **la validité (*validity*)** : une donnée est jugée valide si elle est conforme aux exigences de sa définition : format (email, date), type (numérique, réel), plage (intervalle de variation). Il s'agit d'une comparaison entre les données et les métadonnées ou la documentation ;
5. **l'exactitude (*accuracy*)** : cette dimension évalue la mesure dans laquelle un système d'information décrit correctement ou se rapproche du monde réel qu'il est censé modéliser. Elle se mesure en détectant le taux de valeurs correctes ou incorrectes dans la base de données au regard d'une source de données définie comme référence : "source.colonne" : "address", "reference.colonne" : "address" ;

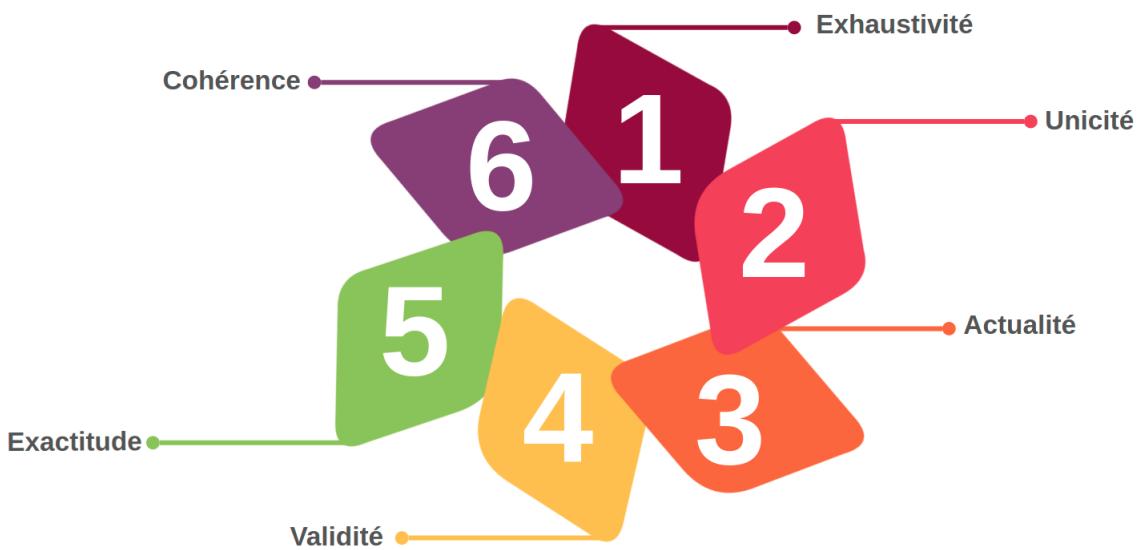


FIGURE 2.1 – Dimensions de qualité de données selon la DAMA UK

6. **la cohérence (*consistency*)** : selon Batini et Scannapieco cité dans [16] , la cohérence capture la violation des règles sémantiques définies sur les données, où les éléments peuvent être des tuples de tables relationnelles ou des enregistrements dans un fichier. Les contraintes d'intégrité de la théorie relationnelle sont un exemple de telles règles. Elle se mesure donc par rapport à l'ensemble des contraintes en détectant les données qui ne les satisfont pas.

Ces dimensions sont parmi les plus citées. Dans une tentative de synthèse, [21] fait remarquer en 2004 que quel que soit le domaine d'application, les mesures de qualité de données les plus fréquemment mentionnées, sont l'exactitude, la complétude, l'actualité et la cohérence. Mais nous retiendrons les six dimensions sus mentionnées.

Dimensions de la qualité des données à l'ère du *big data*

De nos jours, les *big data* imposent de nouveaux défis liés à leurs principales caractéristiques : volume, vélocité et variété. En particulier, afin d'aborder les questions de volume et de vélocité, il est nécessaire de repenser les méthodes d'évaluation pour exploiter les scénarios de calcul parallèle et pour réduire l'espace de calcul. Mais aussi, il revient de se demander si les dimensions existantes sont toujours d'actualité ou nécessite des ajustements. Ainsi, pour évaluer la qualité des *big data* , Cai et Zhu [11], proposent une **hierarchie** de dimensions à **deux niveaux** :

> la **disponibilité** de la donnée caractérisée par :

- l'accessibilité : existe t-il des facilités d'accès aux données ? ;
- l'actualité ;
- l'autorisation : a-t-on le droit d'utiliser ces données ? a-t-on les accès nécessaires ? ;

> l'**utilisabilité** de la donnée caractérisée par :

- l'existence d'une documentation ;
- la crédibilité : qui concerne la fiabilité de la source de données, la normalisation des données et le moment où les données sont produites ;
- l'existence de métadonnées ;

> la **fiabilité** de la donnée caractérisée par quatre des six dimensions identifiées par [15] :

- l'exactitude ;
- l'intégrité ou validité [15] ;
- la cohérence ;
- la complétude ;
- l'auditabilité : l'auditabilité signifie que les auditeurs peuvent évaluer équitablement l'exactitude et l'intégrité des données dans des limites rationnelles de temps et de main-d'œuvre au cours de la phase d'utilisation des données ;

> la **pertinence** de la donnée caractérisée par :

- l'aptitude de la donnée à satisfaire son utilisateur ;

> la **qualité de la présentation** caractérisée par :

- la lisibilité : est ce que la donnée est bien définie suivant une terminologie connue et courante ? ;
- la structuration : la structure fait référence à la difficulté à transformer les données semi-structurées ou non structurées en données structurées.

Plusieurs autres auteurs, ont également proposé des cadres méthodologiques d'analyse de la qualité des *big data*. Ramasamy et Chowdhury [22], en font un résumé. En plus de l'accessibilité [11], la lisibilité [11], la crédibilité et la confiance [22] (la crédibilité [11]), la cohésion [22] (la cohérence [15], [11]) ainsi que la confidentialité [22] (l'autorisation [11]), ils font ressortir quatre autres dimensions jugées importantes :

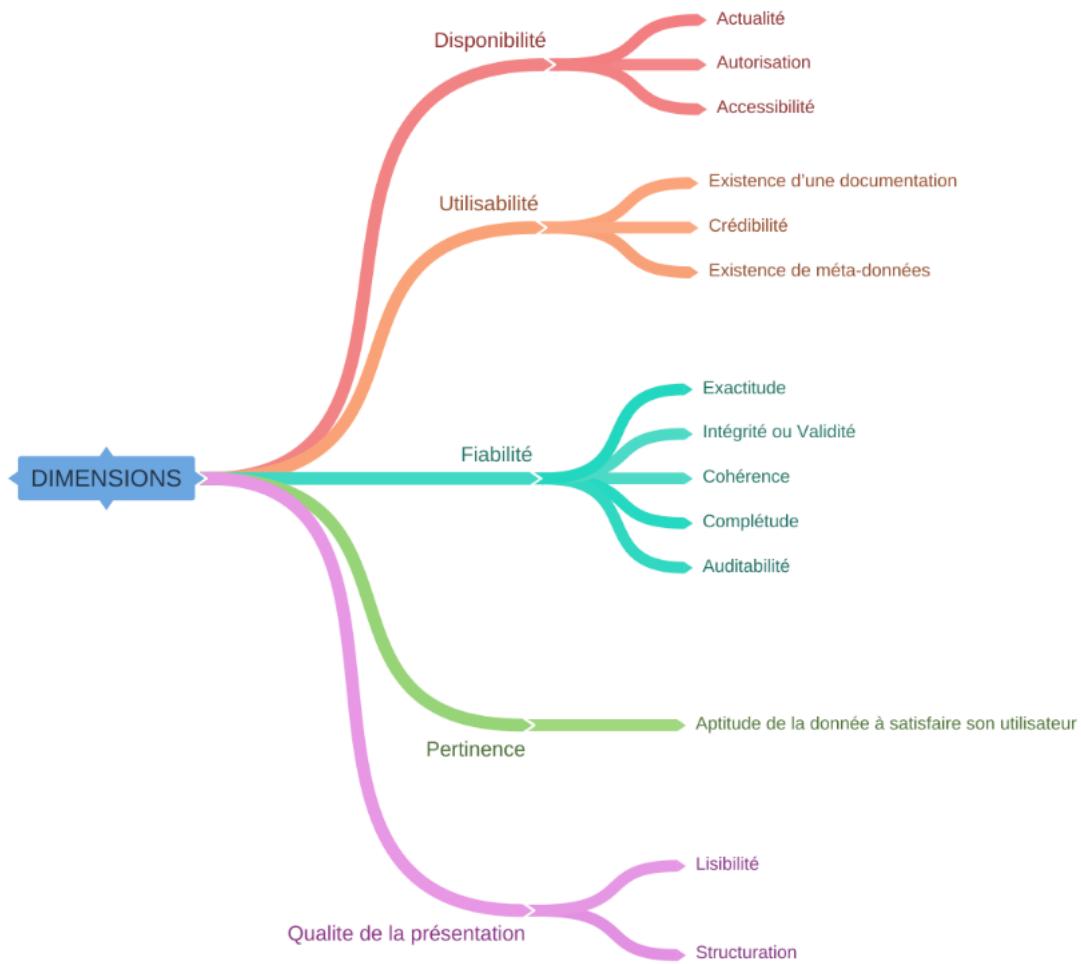


FIGURE 2.2 – Hierarchisation des dimensions selon Cai et Zhu

- le pédigrée ou la lignée : cette dimension permet de connaître la source des données afin de pouvoir corriger les éventuelles incohérences directement là bas ;
- la capacité d'analyse en temps réel ;
- la redondance : elle fait référence à la capacité à représenter le monde réel sans répétition des informations ;
- le volume : cette dimension permet d'analyser le volume des données **extraits**.

Au regard des dimensions énumérées, on constate que plusieurs d'entre elles intègrent le fait que les *big data* ne sont pas forcément produite en entreprise mais pour la plupart proviennent de sources externes et que cela a un impact sur la qualité. Ensuite,

on peut distinguer des dimensions qui sont quantifiables et d'autres ayant un aspect plus qualitatif. Toutefois, les dimensions retenues pour les données structurées demeurent quand même applicables dans le contexte des *big data*.

2.2.4 ANOMALIES ET MESURES CORRECTIVES

DIFFÉRENTS TYPES D'ANOMALIES

Les données jouent un rôle cruciale pour prendre les bonnes décisions à propos de la direction future de l'entreprise, et comme le dit l'adage «*à données inexactes, résultats erronés* ». En ce qui concerne la qualité, il apparaît sans équivoque qu'elle est multidimensionnelle aussi bien par rapport à sa définition que sur son évaluation, mais également cela voudrait dire que les données sont affectées par plusieurs types **d'imperfections**. Les anomalies dans les données sont très variées et sont causées par plusieurs facteurs. Dans [23], on retrouve cinq (5) des problèmes les plus courants en matière de qualité des données. De son côté [19], met en évidence huit (8) classes d'anomalies dans les données regroupant aussi bien les résultats de [23] que ceux de Todoran [24]. Essentiellement nous avons :

- **l'incohérence des formats, la codification des données, la multiplicité des langues et des unités de mesures** : on désigne ici l'existence de formatages différents pour la même donnée ; par exemple la date aux formats JJ/MM/AAAA et MM/JJ/AAAA, le format des données de type numérique doit être clairement précisé. Comme pour le formatage, les différences de langue, d'alphabet ou d'unités de mesure peuvent générer des erreurs (caractères spéciaux, fahrenheit ou celsius) ;
- **les valeurs erronées, les valeurs aberrantes (communément appelées *outliers*)** : ce type d'anomalie est le plus souvent rencontré. Un *outlier* est une valeur qui est anormale par rapport au comportement des autres valeurs. Une valeur erronée désigne une valeur différente de la vraie valeur. Ce type d'anomalie est parfois difficile à détecter surtout si le formatage reste acceptable ;
- **les données incomplètes** : elles se manifestent par l'absence d'informations pertinentes (ou non), laissant place à des champs partiellement remplis ou laissés blancs. Todoran [24], fait remarquer que les données incomplètes représentent un type d'imperfection qui affecte surtout les bases de données volumineuses ;
- **la duplication des données** : les données dupliquées sont un problème que

toutes les entreprises devront traiter. Ben Salem [19], parle au sens plus large de redondance, y ajoutant ainsi la notion de similarité entre les lignes. Son évaluation nécessite un calcul de similarité entre les données ;

- **les données imprécises, les valeurs par défaut** : une donnée imprécise dépeint la situation dans laquelle on a des difficultés à identifier la vraie valeur de la donnée : par exemple la température par défaut peut être mise à zéro (0) mais aussi zéro peut être la valeur réelle ;
- **l'obsolescence des données** : deux aspects ici sont concernés, l'actualité de la données vis à vis d'une référence et l'ancienneté de la donnée ;
- **les dépendances** : il s'agit des dépendances entre attributs qui sont violées ou qui doivent être matérialiser.

MESURES CORRECTIVES (EN COURS DE REDACTION)

DEUXIÈME PARTIE

II

PARTIE 2

CHAPITRE

3

INSTALLATION ET PRISE EN MAIN D'APACHE GRIFFIN

La qualité est un critère clé pour de nombreuses applications utilisant des données comme l'apprentissage automatique. Cependant, il n'existe pas encore de standard sur la manière dont on pourrait **caractériser** une "bonne" donnée, ce qui influence grandement la conception des outils. Apache Griffin, apparaît comme un dispositif, qui veut redonner confiance aux décideurs qui craignent que des données mal contrôlées puissent avoir un impact négatif sur leurs affaires. Dans ce **chapitre** nous allons de prime abord présenter **l'outil**. Suivra une description de son installation ainsi que des processus ayant **permis** sa prise en main et enfin nous **termineront** par une ouverture sur ses limites et perspectives.

3.1 PRÉSENTATION DE APACHE GRIFFIN

La qualité est un critère clé pour de nombreuses applications utilisant des données comme l'apprentissage automatique. Cependant, il n'existe pas encore de standard sur la manière dont on pourrait caractériser une "bonne" donnée, ce qui influence grandement la conception des outils. Apache Griffin, apparaît comme un dispositif, qui veut redonner confiance aux décideurs qui craignent que des données mal

contrôlées puissent avoir un impact négatif sur leurs affaires.

3.1.1 APERÇU DE L'OUTIL

Apache Griffin, est une solution open source de qualité des données pour le *big data*. Il est décrit par ses concepteurs comme étant une plateforme offrant des services de qualité des données. De ce fait, il fournit un cadre complet d'analyse, permettant de réaliser diverses tâches telles que la définition d'un modèle de qualité des données, le calcul de diverses métriques, l'automatisation de la validation des données ainsi qu'une visualisation unifiée des métriques. Grâce aux infrastructures composant son écosystème, Apache Griffin permet de gérer des données venant aussi bien en temps réel (*streaming*) que par lot (*batch*), tout en relevant les défis en matière de qualité dans les applications *big data*. Face à l'absence de consensus sur la qualité, Apache Griffin propose un standard sans pour autant être restictif.

Conçu dans les locaux d'eBay en Mars 2016, puis confié à la fondation Apache le 7 Décembre de la même année, Apache Griffin venait satisfaire un certain nombre de besoin :

- l'absence d'outil permettant de suivre la qualité depuis de multiples sources de données jusqu'aux applications cibles, tout en tenant compte de leurs historiques ;
- l'absence d'un système unifié pour évaluer la qualité des données en *streaming* et offrant la possibilité de monitorer les différentes métriques ;
- l'absence d'une plateforme exposant une *Application Programming Interface* (API), permettant aux développeurs d'implémenter leur propre interface utilisateur.

3.1.2 FONCTIONNALITÉS D'APACHE GRIFFIN

Afin de satisfaire ces différents besoins, nombreuses sont les fonctionnalités proposées par Apache Griffin. Telle que présenté dans la documentation, nous avons :

- **la définition de mesures** : exactitude, exhaustivité ou complétude, actualité, unicité, validité, cohérence ; il s'agit des différentes dimensions de la qualité des données qui peuvent être évalué ;
- **la surveillance des anomalies** : l'établissement d'attentes spécifiques pour

déetecter les données qui anormales, tout en offrant la possibilité de les télécharger ;

- **les alertes en cas de détection d'anomalies** : signaler les problèmes de qualité des données par e-mail ou sur la plateforme ;
- **la surveillance visuelle** : grâce à son interface web, on peut monitorer l'état de la qualité des données ;
- **le temps réel** : l'inspection de la qualité des données peut être effectuée en temps réel, et les problèmes peuvent être **détecté** à temps ;
- **la scalabilité** : il peut être utilisé pour la vérification des données provenant de plusieurs entrepôts de données. De plus, il fonctionne dans un environnement pouvant gérer de grands **volume** de données (1,2 Pétatoctet soit 1000 Téraoctet, cas d'eBay) ;
- **le libre-service** : Griffin fournit une interface utilisateur simple et facile à utiliser qui peut gérer les actifs de données et les règles de qualité des données ; en même temps, les utilisateurs peuvent voir les résultats de qualité des données et personnaliser le contenu de l'affichage par le biais du panneau de contrôle.

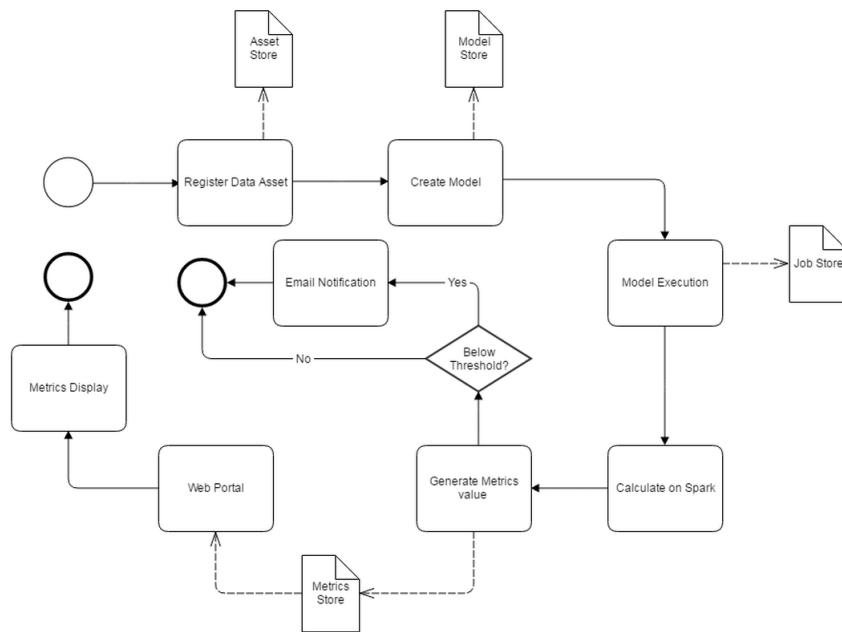
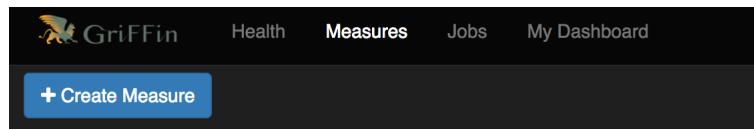


FIGURE 3.1 – Fonctionnalités d'Apache Griffin

3.1.3 ARCHITECTURE D'APACHE GRIFFIN

L'outil est constitué de trois (3) principaux modules : UI, Service, Measure. Le module UI pour *User Interface*, est développé avec le *framework* javascript Angular JS. Il s'agit du module qui gère l'interface graphique d'Apache Griffin. Il permet la visualisation des métriques, leurs définitions et ordonnancement de façon graphique. Tout ceci est possible grâce aux communications avec l'API Service via le protocole *HyperText Transfer Protocol* (HTTP).



(a) UI de Apache Griffin

Target Fields	Map To	Source Fields
default.demo_tgt.id	=	default.demo_src.id

Accuracy Calculation Formula as Below:

$$\text{Accuracy Rate(%)} = \frac{\text{Total Count of Matched records between } \text{default.demo_tgt} \text{ and } \text{default.demo_src} \text{ fields}}{\text{Total Count of records in } \text{default.demo_src}} \times 100\%$$

(b) Mapping pour la dimension Accuracy

FIGURE 3.2 – Création d'une mesure sur l'UI de Apache Griffin

Le module Service, représente donc le web service d'Apache Griffin. Développé en Java avec le *framework* Spring Boot, il expose une API RESTful très riche. Elle offre bien plus de possibilités qu'une utilisation via l'interface graphique, mais dans ce cas nécessite de faire recourt à un client tel que postman pour les tests. Le calcul des différentes métriques est fait par le module Measure, qui quant à lui est écrit en scala.

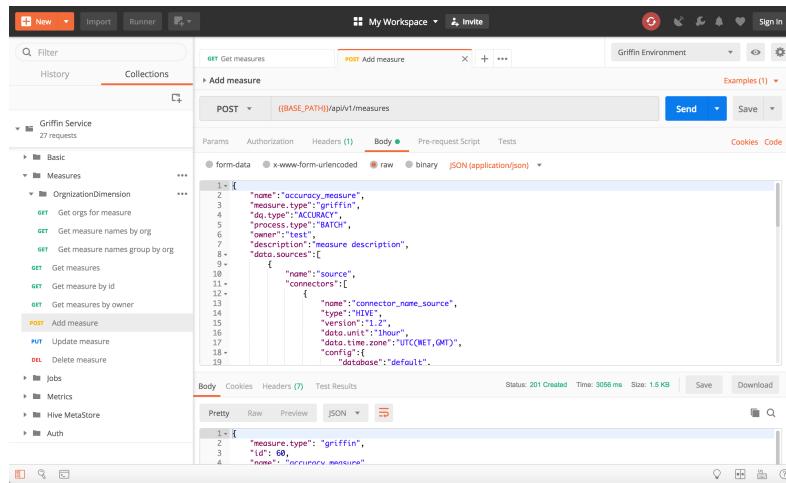


FIGURE 3.3 – Crédation d'une mesure via Postman sur Apache Griffin

Une vue méso de l'architecture d'Apache Griffin met en évidence trois (3) principales couches : Analyze, Measure et Define chacune jouant un rôle fondamentale dans le fonctionnement de Griffin. D'abord **on** a la couche Define, qui est responsable de la définition des dimensions de qualité. **Ensuite** vient la couche Measure qui permet le calcul des différentes métriques. Enfin, la couche Analyze quant à elle s'occupe de la sauvegarde et de l'affichage des résultats.

Suivant cette architecture, on a un flux de travail se présentant comme suit :

1. **D'abord** enregistrer la source ou les sources de données ;
2. **Ensuite** configurer le modèle de mesure, qui peut être défini à partir des dimensions de qualité de données ;
3. Puis planifier la tâche à soumettre périodiquement au cluster spark et vérifier les données régulièrement ;
4. Enfin visualiser les indicateurs sur l'interface du portail et analyser le rendu : la carte thermique et le tableau de bord peuvent être **utilisé** à cet effet.

Au niveau micro, Apache Griffin est un outil qui dépend et utilise plusieurs autres

Apache Griffin Architecture

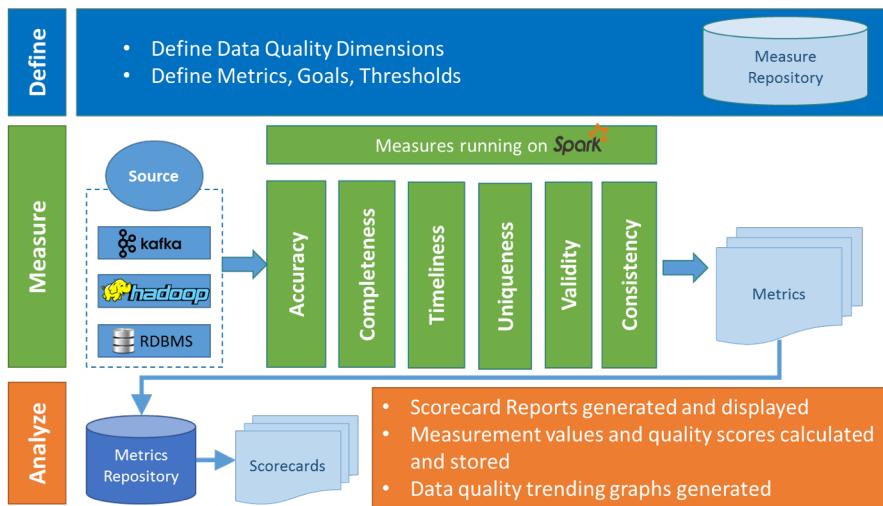


FIGURE 3.4 – Architecture de Apache Griffin

applications *big data* pour l'exécution de ses tâches. Nul besoin de précisez qu'on ne saurait parler d'application *big data* sans parler d'Apache Hadoop et d'Apache Spark en premier lieu. Ainsi on y trouve les applications suivantes :

- **Apache Hadoop** (version 2.6.0 ou plus) : c'est un *framework* de stockage et de traitement distribué de grands ensemble de données sur des clusters d'ordinateur. Il sert principalement de source de données *batch* et pour le stockage des métriques. Il joue le rôle de socle ;
- **Apache Spark** (version 2.2.1 ou plus) : c'est un moteur de calcul distribué et de traitement rapide des données dédié au *big data*. Il permet le calcul des métriques pour les données venant par *batch* ou en temps réel. Le calcul est effectué sur Spark pour un gain maximale de performance ;
- **Apache Hive** (version 2.x) : il s'agit d'un entrepôt de données qui sert de métastore et également de source de données pour Griffin ;
- **PostgreSQL** (version 10.4 ou plus) ou **MySQL** (version 8.0.11 ou plus) : ce sont des Systèmes de Gestion de Bases de Données Relationnelles qui servent ici au stockage des différentes configurations en termes de dimensions de qualité et ainsi que les détails concernant l'exécution (model store,asset store,job store) ;
- **Livy** : permet de soumettre de façon programmatique des jobs spark depuis des applications web/mobiles. Il s'agit d'une API RESTful ;
- **Elasticsearch** (version 5.0 ou plus) : quant à lui permet également de stocker

les résultats (métriques) issues de l'analyse de la qualité des données ;

- **Kafka** : Kafka est utilisé principalement pour la mise en place de flux de donnée en temps réel.

Griffin intègre également les dépendances suivantes :

- **JDK** pour Java Développement Kit ;
- **Scala** ;
- **Maven** qui est un outil de gestion de projet, utilisé pour empaqueter le projet griffin en fichier *.jar* ;

3.2 INSTALLATION ET CONNEXION D'APACHE GRIFFIN À POSTGRESQL

3.2.1 INSTALLATION DE L'OUTIL

Apache Griffin, offre l'avantage d'avoir une documentation bien rédigée et plutôt détaillée, disponible sur [github](#). Pour son installation, deux modes sont proposés :

- une installation en utilisant des conteneurs docker **ou**
- une installation à partir de zéro des différentes applications de son **écosystème**.

Le deuxième mode est privilégié lorsqu'on souhaite configurer les composantes de Griffin et dimensionner l'architecture suivant des exigences particulières. Tandis que, l'installation via docker est recommandée si on n'envisage que tester les fonctionnalités de **Griffin**; ce qui est notre cas ici présent. Pour **chacun** de ces modes, un processus différent est proposé suivant qu'on souhaite analyser des données en *streaming* ou en *batch*. L'objectif visé dans un premier temps en installant Apache Griffin, est de réussir à le connecter à PostgreSQL. Nous avons donc opté pour une installation de type *batch*. Ce choix, est motivé par le fait que l'audit de la qualité des données se fera de façon périodique sur des extractions.

Pour l'installation et la réalisation des tests , nous avons **utiliser** notre propre ordinateur portable. Il s'agit d'un ordinateur portable ayant les caractéristiques **suivante** :

- Mémoire RAM : 16 Gigaoctet ;
- Processeur : Intel Core i7-8565U CPU 1.80GHz × 8 ;

- Mémoire disque : 1 Téraoctet en HDD (*Hard Disk Drive*) et 300 Gigaoctet en SSD (*Solid State Drive*) ;
- Carte Graphique dédiée : NVIDIA Corporation GP107M [GeForce GTX 1050 Mobile] ;
- Système d’exploitation : Ubuntu 20.04.3 LTS 64-bit installé en dual-boot.

L’installation via docker se déroule en deux principales étapes, conformément à [25]. Dans un premier temps une préparation de l’environnement de travail est requise. Cette étape est réalisée en trois (3) points essentiels :

1. L’installation de docker et docker compose ;
2. L’augmentation des limites de la mémoire virtuelle en vue de l’utilisation d’Elasticsearch ;
3. La récupération des images docker nécessaire pour l’installation.

Docker est une plateforme de conteneurisation, qui permet la création et l’utilisation de conteneurs Linux. Un conteneur est comme une machine virtuelle très légère et modulaire et est donc une très bonne alternative à ces dernières. En outre, les conteneurs offrent une grande **flexibilité** : on peut les créer, déployer, copier et déplacer d’un environnement à un autre, ce qui permet d’optimiser les applications pour le cloud [26]. Un conteneur docker est une instance exécutable d’une image docker. Une image quant à elle est un fichier immuable, qui constitue essentiellement une capture instantanée d’un conteneur [27]. Il s’agit d’un ensemble de processus logiciels léger et indépendant, regroupant tous les fichiers nécessaires à l’exécution des processus : code, runtime, outils système, bibliothèque et paramètres. Docker-Compose ou Compose tout simplement d’après [28], est quant à lui un outil de docker qui permet de définir et d’exécuter des applications multi-conteneurs. Grâce aux configurations écrites dans un fichier YAML (*Yet Another Markup Language*), il permet de créer un ensemble de services (conteneurs, **réseau**, volumes), de les démarrer et arrêter tous en même temps.

Les images docker **necessaire** pour le bon fonctionnement de Griffin en mode *batch* sont :

- **apachegriffin/griffin_spark2** : Cette image contient les logiciels **suivant** : MySQL, PostgreSQL, Apache Hadoop, Apache Hive, Apache Spark, Apache Livy ainsi que les modules Service et Measure d’Apache Griffin. De même, on y trouve également quelques données de test. Elle fonctionne comme un *cluster* spark à un seul noeud, fournissant le moteur spark et le service Apache Griffin.

- **apache.griffin/elasticsearch** : Cette image est une surcouche de l'image officielle d'Elasticsearch. Elle a été **configuré** pour permettre la réception de requêtes CORS (*Cross-Origin Resource Sharing* : partage de ressources entre origines multiples) ainsi que la sauvegarde des métriques.

En plus de ces images, nous avons utiliser :

- **postgres** : Il s'agit de la dernière version de l'image de PostgreSQL. Elle servira pour les tests de connexion.
- **docker.elastic.co/elasticsearch/elasticsearch :7.13.1** : Cette image est utilisée en lieu et place de l'image apache.griffin/elasticsearch. Ce changement a été fait pour faciliter l'utilisation de Kibana pour la visualisation.
- **docker.elastic.co/kibana/kibana :7.13.1** : Kibana est une plateforme open source d'analyse et de visualisation conçue pour fonctionner avec Elasticsearch.

Après la préparation de l'environnement, l'étape qui suit est la création des différents conteneurs. À cette étape, [25] fournit un fichier docker-compose-batch.yml. Ce fichier définit deux principaux services : griffin et es (elasticsearch). Ce sont des conteneurs instanciés à partir des images proposées par la documentation. Il spécifie également le mappage des ports pour chaque **conteneurs**, ainsi que le nom du conteneur, celui de la machine virtuelle de même que le liens entre les conteneurs. À cette **étape** nous avons **défini** une nouvelle configuration, afin de mieux organiser notre architecture. En effet, cette configuration tient compte des nouvelles images ajoutées. En plus des services précédemment cités nous avons ajouté un **conteneurs** Kibana configuré et lié selon les spécifications de [29] ainsi qu'un conteneur postgres, tout en prenant soin de substituer l'image d'Elasticsearch par une version plus récente. Un service réseau a été ajouté. Il permet la création d'un sous réseau de type *bridge*. Cette configuration, permet d'attribuer de façon statique des **adresses IP** (*Internet Protocole*) aux différents conteneurs. Un dernier service de stockage persistant avec la machine hôte fut également ajouté. Avec un tel stockage, les fichiers de configuration des jobs griffin sont **sauvegardé**, même en cas de perte ou de suppression des conteneurs.

Suite à l'installation de Griffin, il est alors possible d'utiliser ses différentes fonctionnalités. On peut donc avoir accès à son interface graphique, exécuter des requêtes par l'entremise de postman pour la réalisation des tâches de qualité de données ou simplement utiliser en ligne de commande via le conteneur griffin.

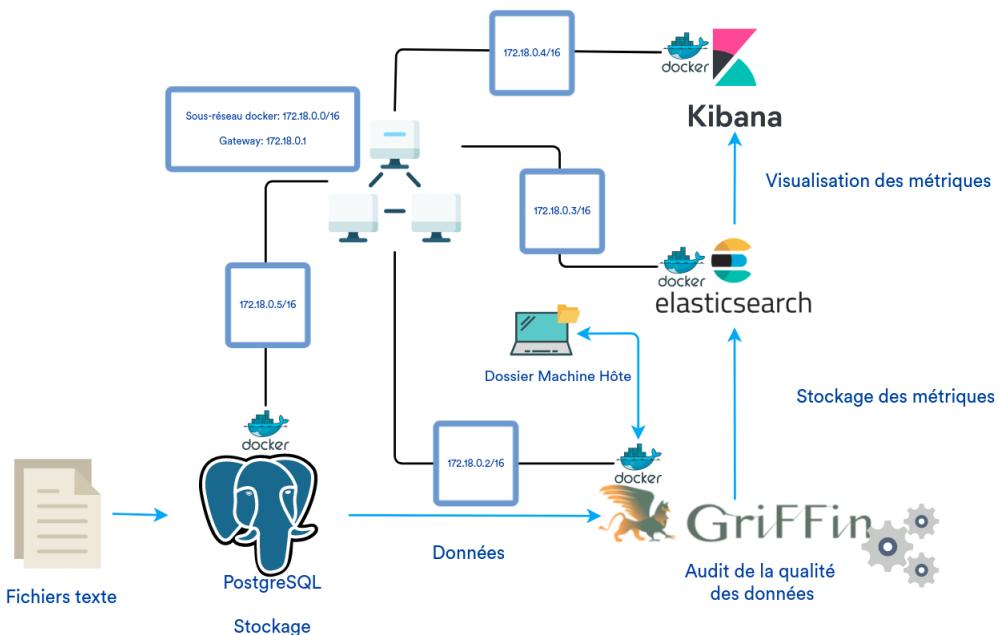


FIGURE 3.5 – Architecture mise en place pour tester Apache Griffin

3.2.2 CONNEXION

L'**évaluation** de la qualité des données, nécessite bien évidemment des données. À cet effet, Griffin permet de configurer l'accès à différentes sources de données. Notamment, en mode *batch*, Griffin peut auditer la qualité des données situées dans des tables Hive, dans des fichiers AVRO, des fichiers plats (Parquet, CSV, TSV, ORC) et des données provenant de sources possédant des pilotes JDBC (*Java Database Connectivity*) comme Oracle, MySQL, PostgreSQL... C'est notamment cette dernière faculté qui est utilisé pour réaliser la connexion à PostgreSQL. Comme énoncé précédemment, plusieurs modalités d'accès sont disponible pour l'utilisation de Griffin. Pour une manipulation plus conviviale, l'interface web est privilégiée. Toutefois, elle présente quelques limites. En effet, actuellement l'interface web ne prends en charge que quelques dimensions (Accuracy, Profiling) ainsi que l'accès aux tables Hive. Cette limitation se présente au niveau du module Service et de ce fait impacte également l'utilisation de postman. Par contre, l'utilisation directe en ligne de commande grâce à la soumission d'une tâche spark, est conforme aux fonctionnalités décrites dans la documentation. Elle rends néanmoins la tâche moins aisée. Ce mode de fonctionnement passe directement par le module Measure. Il va s'en dire que c'est par ce moyen que les tests ont été effectués.

Dans le vocabulaire de **Griffin** une mesure est la matérialisation d'une dimension de qualité de données. Il s'agit de l'écriture de la requête caractérisant les attentes ou spécificités de la dimension **considéré**. Elle contient les règles à **évaluées**. L'exécution de cette mesure est **appelé** un job et les informations numériques qui sont produites sont qualifiées de métriques. L'utilisation en ligne de commande nécessite l'accès au *bash* (*bourne-again shell* interpréteur en ligne de commande de type script) du conteneur griffin. Elle requiert également la configuration manuelle des mesures au moyen d'un fichier d'extension JSON (*JavaScript Object Notation*). Ce fichier contient les informations décrivant la source de données, éventuellement les accès (identifiant et mot de passe) ainsi que **la ou** les règles de qualité des données qui **seront** évaluées, comment les **évaluer** et où stocker les résultats. Une description **détaillé** du fichier de configuration est fournie dans la section suivante. Retenons pour l'instant que pour réaliser la connexion à une source de type JDBC les informations suivantes sont **nécessaire** :

- le type de connexion : JDBC ;
- la base de données : demo ;
- la table ciblé : test_auto ;
- l'url de connexion : jdbc :postgresql ://172.18.0.5 :5432/demo ;
- le nom d'utilisateur pour la connexion : aurel ;
- le mot de passe : 123456789 ;
- le pilote JDBC : org.postgresql.Driver .

Néanmoins, bien que le code source de Measure ait prévu une *class* gouvernant les sources de type JDBC, la connexion avec une base de données PostgreSQL n'était pas nativement prise en charge. En effet, cela était dû à l'absence du pilote de connexion dans le *Classpath*⁽¹⁾ du projet. Pour y **rémedier** nous avons donc modifier le code source du module Measure. Cette modification passe par l'ajout aux dépendances du projet dans le fichier *Pom.xml* du pilote de connexion à une base de données PostgreSQL. Puis au niveau des fichiers de test, il fallait adapter le code, afin qu'à la phase de *build*⁽²⁾ du projet, le pilote une fois téléchargée soit convenablement ajoutée au *Classpath*. Cette phase conduit également à la création d'un nouveau .jar du module Measure. Il est à noter que la version de griffin disponible dans l'image proposée est la 0.2.0 alors que la dernière version officiellement disponible à la date de rédaction est la version 0.6.0. La modification du code source a donc été opérée sur la version 0.6.0 . Une fois la nouvelle version obtenue, nous

(1). Chemin d'accès au répertoire où se trouvent les classes et les packages Java

(2). Phase de compilation d'un projet conduisant à la création d'un fichier .jar

avons entrepris de construire une nouvelle image de griffin en prenant comme image de base apachegriffin/griffin_spark2 :0.3.0. C'est notamment cette nouvelle image qui fut utiliser dans notre fichier docker-compose. Elle fut baptiser griffin :0.6.2. Et c'est suivant ce processus [30] que nous avons pu avoir une image permettant d'interconnecter griffin et PostgreSQL.

3.3 TEST DES FONCTIONNALITÉS D'APACHE GRIFFIN

3.3.1 TÂCHES À ACCOMPLIR

Réussir à connecter Apache Griffin avec PostgreSQL n'est qu'une étape intermédiaire, ayant pour but de faciliter l'évaluation de la qualité de données venant de cette source. À cet effet, une série de *tokens* de qualité des données a été émise pour tester les fonctionnalités d'Apache Griffin. Essentiellement, la tâche consistait à utiliser Griffin pour :

- Token 1: Déetecter les valeurs nulles ;
- Token 2: Déetecter les valeurs aberrantes ;
- Token 3: Définir la plage des valeurs ;
- Token 4: Définir les modalités de chaque colonne ;
- Token 5: Définir des expressions régulières pour certaine colonne (mail, tél, Cin, Date..) ;
- Token 6: Fournir des statistiques descriptives (min, max, ...);
- Token 7: Compter les doublons ;
- Token 8: Valider les éventuelles relations entre les tables (clé primaire, clé étrangère, ...);
- Token 9: Vérifier que les codes des valeurs de structures sur les tables transactionnelles existent au niveau des tables de structure ;
- Token 10: Implémenter des règles de qualité (Ex : somme des primes par garantie = prime nette totale) ;
- Token 11: Évaluer l'exactitude entre 2 tables ;
- Token 12: Déetecter les caractères spéciaux

3.3.2 PRÉSENTATION DES FICHIERS DE CONFIGURATION DES JOBS

La validation de ces différentes attentes, passe par l'écriture des mesures y afférente. Le module Measure a besoin de deux fichiers pour configurer l'exécution du job spark

(*env.json*) et celle des mesures. Le fichier *env.json* configure notamment trois grands points. On y retrouve les paramètres pour l'exécution sur spark aussi bien en cas de données en *batch* qu'en *streaming*. Ensuite, la configuration pour la persistance des métriques : affichage en console, sauvegarde sur HDFS ou sur Elasticsearch, ou encore une destination personnalisée. Le dernier point concerne les points de contrôle pour les données en *streaming*.

Le fichier destiné à la définition des mesures contient quant à lui principalement les paramètres suivants :

name : ce paramètre permet de nommer le job qui sera exécuté ;

process.type : deux valeurs sont admises ici : "BATCH" ou "STREAMING". Comme on peut le remarquer, cette variable ajustée en fonction du type de données ;

data.sources : il s'agit d'une liste contenant les différentes sources de données. On peut se connecter directement à plusieurs tables en même temps. La configuration de la source de données requiert la définition du type de connexion (JDBC,HIVE, AVRO,CUSTOM) et un paramétrage des accès en fonction du type **choisis** ;

evaluate.rule : il s'agit d'un attribut contenant une liste de règles à évaluer. Une règle est définie en précisant le langage utilisé (**dsl.type**). Deux types de langages sont supportés par Griffin. On a essentiellement *spark-sql* et *griffin-dsl*. Ce dernier, est une surcouche du *spark-sql* facilitant l'écriture des requêtes. Il faut également préciser la dimension de qualité des données (profiling, accuracy, completeness, uniqueness, timeliness) qui est **évaluer** (**dq.type**), la règle (**rules**) en **elle** même ainsi que les modalités de sorties (**out**) ;

sinks : permet de lister les différentes sorties pour la persistance des métriques.

Une présentation plus détaillée est disponible en annexe. Dans cette sous-section, par contre nous mettrons l'accent sur les langages de qualité des données de Griffin et sur les dimensions implémentées. En effet, les règles de qualité des données de Griffin sont **éditer** soit à l'aide de *griffin-dsl* ou du *spark-sql*. Tout deux ont pour syntaxe de base le SQL. Le *griffin-dsl* est une surcouche du *spark-sql*, mais beaucoup moins verbeux que ce dernier. Il suffit de préciser quelques mots clés en fonction de la dimension choisie pour établir la règle voulue. Elle est ensuite traduite en *spark-sql* puis exécuter sur Apache Spark. Le *spark-sql* est issu du module Spark SQL d'Apache Spark permettant de travailler avec des données structurées. Il permet de faire des requêtes sur Spark mais en utilisant le langage SQL, et cela peu importe

la structure des données.

Griffin offre également un certain formalisme en terme de qualité des données, dans la mesure où il implémente déjà en son sein plusieurs dimensions. Elles sont définies pour l'utilisation de *griffin-dsl*. Au nombre de ces dimensions nous avons :

Accuracy : on cherche à obtenir le nombre de correspondances entre une donnée source et une donnée cible, conformément à la dimension exactitude. La définition de la règle, décrit la relation de mappage entre les deux sources de données.

Par exemple on a : `source.id = target.id and source.name = target.name ;`

Profiling : cette dimension permet de faire un profilage des données. L'écriture de la règle peut se faire au moyen du langage SQL (*spark-sql*) ou en écriture simplifiée (*griffin-dsl*) comme `source.id.count(), source.age.max() group by source.cntry ;`

Distinctness : il s'agit de trouver les données qui sont en double (dimension unicité). Pour ce faire, il suffit de préciser la ou les colonne(s) ciblées séparées d'une virgule : `name, age ;`

Completeness : on vérifie ici si les valeurs d'une ou plusieurs colonnes sont nulles (dimension exhaustivité). La règle s'écrit juste en précisant la ou les colonnes dont on veut évaluer la completeness ;

Timeliness : cette implémentation mesure la latence de chaque élément et donne des statistiques de latence. Elle est disponible uniquement pour les données en *streaming*.

L'usage de *spark-sql* peut être vu comme un mode expert ou avancé. Toutefois, la version 0.6.0 d'Apache Griffin n'offre que l'implémentation de l'Accuracy, du Profiling et de la Completeness pour le *griffin-dsl*. Pour l'implémentation de nos règles de qualité nous avons fait une combinaison de *spark-sql* et de *griffin-dsl*.

3.3.3 VALIDATION DES TOKENS

Pour valider les tokens, nous avons téléchargé et utilisé une base de données test. Cette base modélise l'activité d'un aéroport. On y trouve huit (8) tables. Les dimensions retenues pour l'évaluation de la qualité des données sont les suivantes : la complétude, la validité, la cohérence, l'exactitude et l'unité. Les paragraphes suivant présentent les tokens et les dimensions concernées ainsi que le procédé d'évaluation.

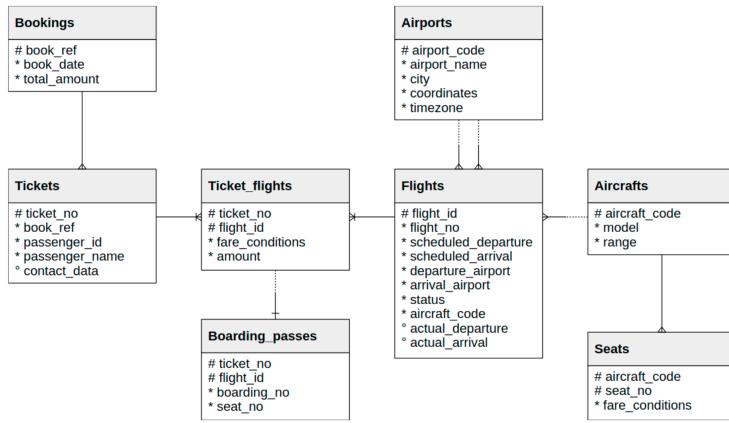


FIGURE 3.6 – Modèle des données de test

La détection des valeurs nulles relève de la complétude (Token 1). En effet, Griffin au moyen de la dimension Completeness renvoie la répartition valeurs **nulle**, complète et effectif total. La détection des valeurs aberrantes (Token 2), concourt à l'identification des valeurs qui ne sont pas conformes aux définitions des colonnes en terme de valeurs attendues ou licites. Pour faire le **test** nous avons vérifier l'existence de valeurs négatives pour la variable `total_amounts` de la table `Bookings`. On s'intéresse ainsi à un aspect de la validité des colonnes. Il en est de même lorsqu'on tente d'identifier la plage des valeurs pour les colonnes numériques, les modalités de chaque colonne **qualitatives** ou même quelques statistiques descriptives (Tokens 3, 4 et 6). La détection des caractères spéciaux dans les chaînes de caractères (Token 12) ainsi que la définition d'expressions régulières pour valider le formattage de certaines colonnes (Token 5) sont également des tâches entrant dans la vérification de la validité des données. Pour la **validation** de ces différents *token*, nous avons eu recours cette fois à la dimension Profiling de Griffin. En fonction de la complexité du résultat souhaité un usage alternatif de *griffin-dsl* et de *spark-sql* a été fait.

La dimension de cohérence est couverte par les règles suivantes : valider **d'éventuelles** relations entre les tables (Token 8 : tables `Tickets` et `Ticket_flights`), vérifier que les codes des valeurs de structures sur les tables transactionnelles existent au niveau des tables de structure et implémenter des règles de qualité telle que la somme des primes par garantie **soient** égale à la prime nette totale (Token 9 : `Bookings` (`total_amount` et `Ticket_flights` : `amount`)). Afin d'évaluer cette **dimension** nous avons fait usage de la dimension Profiling de Griffin. Le comptage des doublons (Token 7), fut également fait au moyen de la dimension Profiling avec du *spark-sql*. Elle fait ressortir le nombre de valeur distinct ainsi que le nombre total de valeur

présente. Enfin la dimension d'exactitude est matérialisée ici par l'évaluation de l'exactitude entre deux (2) tables. Naturellement, l'usage de la dimension Accuracy de Griffin s'est fait pour valider le Token 11.

```
1 {
2   "name": "batch_prof",
3   "process.type": "batch",
4   "data.sources": [
5     {
6       "name": "src",
7       "baseline": true,
8     "connector": {
9       "type": "jdbc",
10      "config": {
11        "database": "demo",
12        "tablename": "bookings.seats",
13        "url": "jdbc:postgresql://172.18.0.5:5432/demo",
14        "user": "aurel",
15        "password": "123456789",
16        "driver": "org.postgresql.Driver"
17      }
18    }
19  ],
20 ],
21
22 "evaluate.rule": {
23   "rules": [
24     {
25       "dsl.type": "griffin-dsl",
26       "dq.type": "profiling",
27       "out.dataframe.name": "prof",
28       "rule": "src.aircraft\_\_code.count() AS aircraft\_\_code\_\_count",
29       "out": [
30         {
31           "type": "metric",
32           "name": "prof"
33         }
34       ]
35     }
36   ],
37 },
38 "sinks": ["CONSOLE", "HDFS", "ELASTICSEARCH"]
39 }
```

Listing 3.1 – Exemple fichier .json : Profiling Seats

Suite à la validation des tokens, une base de données fictives sur l'assurance automobile nous a été confiée, pour l'évaluation de la qualité. Cette évaluation s'est faite sans instructions particulières. Les données sur l'assurance automobile n'étaient constitué que d'une seule table. On y trouve : le **numero** de police, les **dâtes** de début et de fin de police, la fréquence de paiement (annuelle ou mensuelle), la

langue du contrat, la profession, le type de territoire (urbain, rural ou semi-urbain), l'utilisation faite du véhicule, la présence d'alarme, la marque du véhicule, le genre du conducteur, son **age**, la durée de son permis de conduire, l'année de mise en circulation du véhicule, le nombre de sinistre, l'exposition aux sinistres ainsi que les divers coûts liés aux garanties et au contrat d'assurance. Nous avons produit les **métriques** suivantes, telle que représenté sur Kibana :



FIGURE 3.7 – Aperçu des métriques calculées

3.4 LIMITES ET PERSPECTIVES D'APACHE GRIFFIN

La documentation d'Apache Griffin fournit à cet effet une feuille de route sur les prochaines fonctionnalités à **développées**. D'après cette feuille de route des fonctionnalités actuellement disponible sont :

La détection des données via Hive metastore

La gestion des mesures : En effectuant des opérations sur l'interface utilisateur, on ne peut que créer, supprimer et mettre à jour que trois types de mesures dont : Accuracy, Profiling, Publish Metrics. Cependant, en faisant des appels par l'API, l'utilisateur dispose d'un panel plus varié de mesures (accuracy, profiling, timeliness, uniqueness ou distinctness, completeness, publish metrics). Toutefois l'absence de connecteur de type JDBC nous a empêché de tester ces fonctionnalités.

La gestion de l'exécution des mesures (jobs) : L'utilisateur peut créer, supprimer et planifier des jobs pour les données par lot.

Le calcul des mesures sur Spark

La visualisation des métriques

Bien sûr à cela s'ajoute les fonctionnalités identifiées précédemment. Toutefois la mesure de type Uniqueness présente quelques dysfonctionnements et la Publish Metrics n'est pas documentée, faute de quoi nous n'avons pas pu la testée. Pour cette dernière nous soupçonnons une obsolescence de la fonctionnalité. Car même dans le code source elle s'est avérée absente. Notons également que la documentation sur github bien que détaillée, n'est pas tenue à jour. Et celle sur la page officielle d'Apache Griffin est insuffisamment fournie.

À court-terme, Apache Griffin envisage supporté plus de source de données. Notamment un élargissement de la connexion aux bases de données relationnelles, ainsi qu'une connexion à Elasticsearch. De plus, Griffin prévoit prendre en charge de manière native d'autres dimensions de qualité de données comme la cohérence et la validité. Pour l'instant il est laissé à la charge de l'utilisateur la définition des attentes vis-à-vis de ces dimensions. Une nouvelle fonctionnalité est également ciblée : il s'agit de la détection d'anomalie en analysant les métriques calculées par Elasticsearch.

Notons également que dans le cadre de cette étude nous avons utilisé la version 0.6.0 d'Apache Griffin, mais durant cette exploration de nouvelles fonctionnalités et des améliorations de celles existantes ont été publiés sur github. Il s'agit de la réécriture et de la redéfinition des différentes dimensions. Ainsi la nouvelle version s'appuiera probablement sur les dimensions suivantes : l'Accuracy, la Completeness, la Duplication et le Profiling. En plus de ces quatre dimensions Griffin intègre une nouvelle : la conformité du schéma. Cette mesure évalue si oui ou non les données respectent le type qu'on leur attribut. Il est question là également de l'évaluation de la validité des données. Ces modifications dans les mesures se traduisent aussi par une réécriture de la configuration du fichier .json ainsi que du formatage des résultats. En terme de connectivité une classe pour la connexion à Elasticsearch a également été ajoutée. Ce qui annonce de belles perspectives pour le développement de l'outil.

En somme Griffin, de par sa constitution et sa philosophie est une application idéalement conçu pour auditer de façon flexible et scalable la qualité des données. Toute-

fois, il nécessite un temps considérable pour son installation et sa configuration afin de s'intégrer dans l'architecture de l'entreprise. Une fois ce cap passé, la configuration des fichiers et la programmation des jobs faites, l'industrialisation de la qualité des données se fait toute seule et de manière quasi autonome. Aussi les limites au niveau de l'API, amènent l'utilisateur à faire soit une concession au niveau de l'interface utilisateur et dans ce cas travailler en ligne de commande ou soit modifier le code source pour l'adapter à ses attentes. La planification des jobs dans le premier cas peut être opéré à l'aide d'Apache Airflow.