Modeling an Enterprise Data Warehouse: Case of the Star Schema, the Normalized Model and the Data Vault

Keita, Moussa

April 2022

Modéliser un Data Warehouse d'Entreprise

Cas du Schéma en Etoile, du Modèle Normalisé et du Data Vault

Source : lebigdata.fr

Moussa KEITA *

Expert/Formateur

Big Data–Data Science

Consultant Data à EDF, Société Générale, Caisse des Dépôts

Paris

Document version 1.0

(Avril 2022)

***Contact Email : keitam09@ymail.com**

SOMMAIRE

1 INTRODUCTION

1.1 Le Data Warehouse d'Entreprise

Le Data Warehouse est un système de base de données conçu pour collecter et centraliser, de manière continue, des données provenant de multiples sources. Depuis quelques années, les Data Warehouses sont devenus des outils incontournables au sein des entreprises car ils offrent une approche unifiée pour la représentation des données.

Le concept Data Warehouse (entrepôt de données) a été proposé pour la première fois vers la fin des années 1980 par Paul Murphy et Barry Devlin, deux employés d'IBM. Mais c'est suite aux travaux de Bill Inmon dans le courant des années 1990 que le concept a connu un véritable développement. Inmon définit le Data Warehouse comme une collection de données orientées sujet, intégrées, non volatiles et historisées, organisées servant de support au processus d'aide à la décision. Partant de cette définition, les données stockées dans un Data Warehouse possèdent les caractéristiques suivantes :

- *Orientées sujet* : les données sont organisées par thématique, facilitant ainsi l'analyse de n'importe quel secteur particulier de l'entreprise.
- *Intégrées:* les données sont organisées et uniformisées de sorte à garantir une cohérence interne.
- *Non volatiles :* les données déjà stockées dans l'entrepôt ne sont plus modifiables.
- *Historisées* : contrairement aux systèmes transactionnels standards où seules les données les plus récentes sont stockées, dans un Data warehouse, les données sont continuellement empilées. Ce qui facilite leur audit et leur traçage.

Attention toutefois à ne pas confondre un Data Warehouse avec un **Data Mart**. Ce dernier est un magasin conçu pour stocker des données préalablement traitées, transformées et regroupées après l'application d'un certain nombre de règles métiers. Un Data Warehouse peut couvrir plusieurs sujets métiers alors qu'en général, un Data Mart vise à répondre à un besoin spécifique. D'une manière générale, le Data Mart est construit à partir d'un sous-ensemble de données extraites d'un Data Warehouse.

Toutefois de nombreux Data Marts sont construits en collectant directement les données auprès des système sources sans passer par le Data Warehouse.

Un Data Warehouse ne devrait pas non plus être confondu avec une Base de Données Opérationnelle ODS (**Operational Data Store**). En effet, contrairement à une ODS, le Data Warehouse permet d'exploiter à la fois les données courantes mais aussi les données historiques pour répondre aux besoins métiers.

Par ailleurs, le Data Warehouse ne devrait pas être confondu avec le **Data Lake** (lac de données). En effet, bien qu'il existe quelques similarités entre le Data Warehouse et le Data Lake, des différences majeures entre les deux systèmes de stockage. Tout comme un Data warehouse, un Data Lake permet stocker tous les types de données (structurées ou non) et en conserver toute l'historique. Mais à la différence d'un Data Warehouse, un Data Lake ne permet pas nécessairement d'établir une cohérence interne dans l'organisation et l'exploitation des données. Dans un Data Lake, les données brutes sont stockées dans leurs formats originaux sans aucun traitement. Le Data Lake est fréquemment utilisé dans le contexte Big Data comme solution de stockage rapide pour répondre à la problématique de volumétrie croissante. Evoqué pour la première fois, en 2010, par James Dixon, CTO de Pentaho, l'approche Data Lake est une solution de stockage sans prétraitement des données et sans détermination à l'avance des usages qui pourront en découler. Ce qui n'est pas nécessairement le cas pour un Data Warehouse qui est plutôt un entrepôt de données brutes ou semi transformées dont la finalité d'usage est plus ou moins bien connue.

1.2 La modélisation d'un Data warehouse

D'entrée de jeu, rappelons qu'une donnée est toujours la représentation d'un phénomène informationnel, c'est-à-dire la traduction d'un *fait* et de ses multiples *dimensions contextuelles*. Les faits peuvent être par exemples : les ventes réalisées par un site de commerce en ligne, les vols assurés par une compagnie aérienne, le chiffre d'affaire d'une entreprise multinationale, etc. Tous ces faits peuvent être analysés suivant différents contextes (dimensions). Les dimensions représentent les angles sous lesquels un fait peut être analysé. Elles représentent le qui, quand, où). Par exemples : les faits de ventes peuvent être analysés par catégories de clients, par catégories de produits, par zones géographiques, par périodes, etc.

Dans un système d'information, qu'il s'agisse d'une Base de Données Opérationnelle, d'un Data Warehouse ou d'un Data Mart, les faits et leurs dimensions sont toujours collectées et stockées dans des structures bien définies : tables, objets, documents, etc. Le format « **table** » étant encore la structure la plus utilisée, le travail de modélisation se limite généralement à définir la structure et l'organisation des tables de sorte à mieux représenter la relation entre le fait et ses dimensions mais aussi à faciliter l'accès et l'utilisation des données.

Dans ce document, nous allons proposer un aperçu général sur trois approches de modélisation d'un Data Warehouse, à savoir la modélisation en étoile, la modélisation en troisième forme normale (3NF) et la modélisation Data Vault (Voûte de données).

1.3 Exemple d'illustration

Pour mieux illustrer chaque approche de modélisation, nous nous basons sur un exemple illustratif inspiré des activités d'une Plateforme de livraisons de courses à domicile. L'activité de livraisons de courses à domicile a connu un véritable développement au cours dernières années. En France, par exemple, bien que le marché de livraison de course à domicile reste encore dominé par les grandes enseignes comme Leclerc, Carrefour, Auchan, Intermarché, des acteurs plus spécialisés, pour la plupart des startups, connaissent un grand succès, notamment suite à la crise de la Covid-19 : Gorillas, Getir, Cajoo, Dija, Flink.

Le principe d'une livraison de course à domicile est relativement simple. Un client passe une commande sur l'application ou le site de la plateforme de livraison. Lorsque la commande est validée, un préparateur de commande se charge de préparer le colis et le met à disposition d'un Coursier. Ce dernier se charge, à son tour, de livrer la commande chez le client en utilisant un moyen de transport approprié : Caddie, Vélo, Scooter, voiture, etc.

Supposons qu'un client passe une commande sur l'application de la plateforme Gorillas. Supposons ensuite que cette transaction soit matérialisée par les informations suivantes :

Jeu de données 1 (JDD1)


```
{
"id_cmd"              :  "kjx204051",
"date_reception"      :  "2022-03-19 16:17",
"statut_paiement"     :  "OK",
"id_client"           :  "c896514",
"nom_client"          :  "Kagi",
"prenom_client"       :  "Robert",
"num_tel_client"      :  "923794061",
"adresse_client"      :  "65 rue des Cévennes 75035 Paris",
"id_produit"          :  "mr73492647",
"desc_produit"        :  "Champagne",
"quantite"            :  1
"prix_unitaire"       :  50
"montant"             :  50
"id_preparateur"      :  "ic5363738"
"nom_preparateur"     :  "Sylvain"
"prenom_preparateur"  :  "Olivier"
"num_tel_preparateur" :  "0937935464"
"adresse_preparateur" :  "12 Danton 75025Paris ",
"id_entrepot"         :  "cz2539",
"num_tel_entrepot"    :  "974526624",
"adresse_entrepot"    :  "83 rue lecourbe 75035 Paris ",
"id_coursier"         :  "cz8527",
"nom_coursier"        :  "Kumar",
"prenom_coursier"     :  "Lahoub",
"num_tel_coursier"    :  "903647749",
"adresse_coursier "   :  "25 Etienne Dolet 93500 Landry",
"id_vehicule"         :  "hjk635383",
"desc_vehicule"       :  "Vélo électrique californien",
"date_depart_coursier" :  "2022-03-19 16:30",
"date_livraison"      :  "2022-03-19 16:40",
"statut_livraison"    :  "OK"
}
```

Maintenant, supposons que la Direction du Système d'Information de la plate Gorillas nous sollicite, en tant qu'Expert Data, pour qu'on lui propose quelques approches de modélisation de cette transaction et d'en ressortir les avantages et les inconvénients de chacune.

Pour répondre au besoin de la DSI de Gorillas, nous proposons trois méthodes de construction du Data Warehouse : le schéma en étoile, le modèle en troisième forme normale (3NF) et le modèle Data Vault. Dans les pages qui suivent, l'objectif, pour nous, est de présenter les principales caractéristiques de chacun des modèles ainsi que leur méthode de conception.

Remarque importante : Le cas utilisé dans ce document est un exemple que nous avons délibérément simplifié pour privilégier l'aspect pédagogique. Il ne s'agit donc pas d'un exemple qui puisse refléter toute la complexité d'un Data warehouse. Il s'agit simplement d'un exemple illustratif qui montre les propriétés de chaque modèle.

2 LE MODELE EN ETOILE

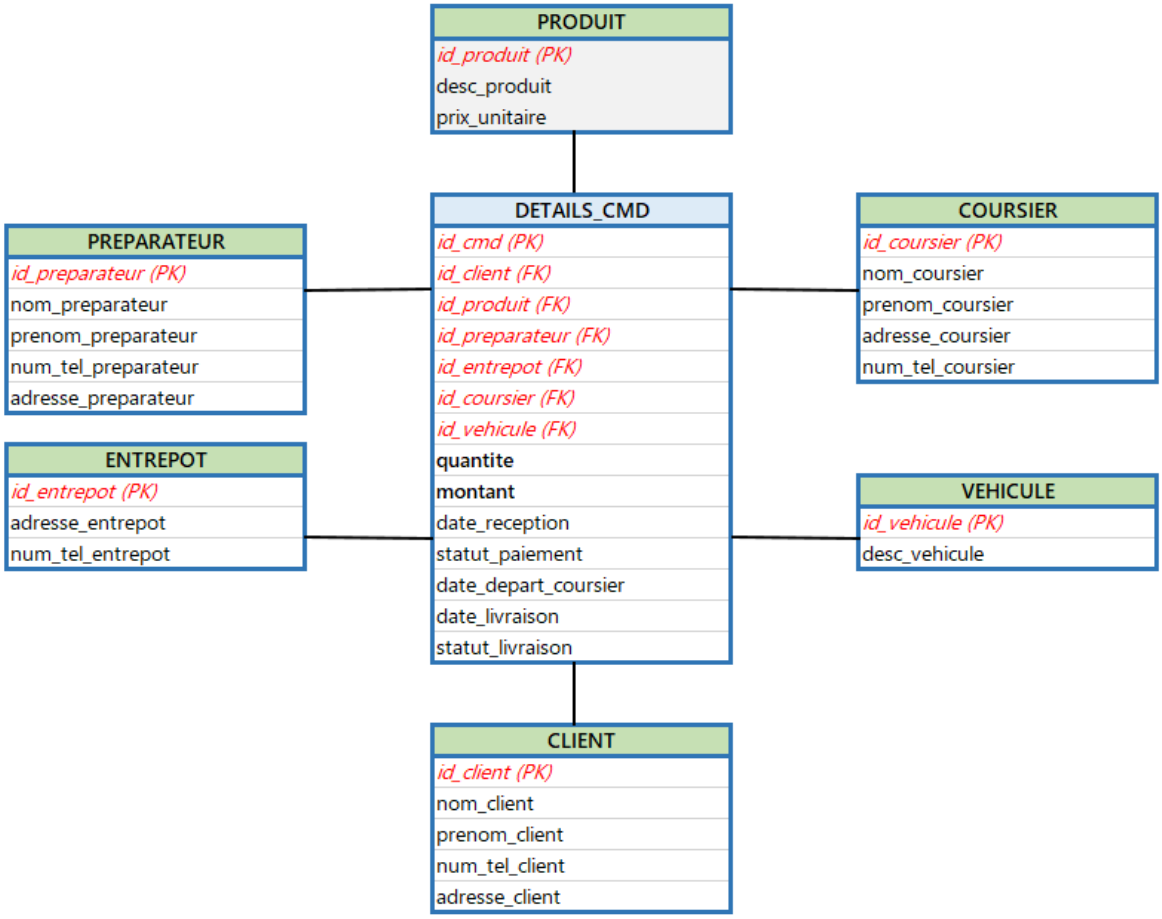
2.1 Généralités

La modélisation en étoile (encore appelé modélisation dimensionnelle) est une approche dans laquelle les informations relatives à un fait et ses dimensions sont reparties dans des tables distinctes appelées « *table de faits* » et « tables de dimension ». La table de faits est le centre du modèle autour duquel gravitent les tables dimensions, d'où l'appellation de schéma en étoile. Dans le modèle en étoile, chaque table de dimension est reliée à la table de faits grâce à une clé étrangère.

2.2 Cas d'illustration

Partant de l'exemple illustratif précédemment présenté, nous pouvons modéliser la transaction selon le schéma en étoile ci-dessous.

Figure 1 : Modèle en étoile



Ici, le modèle en étoile nous permet de décomposer la transaction en une table de faits **DETAILS_CMD** qui est la table centrale autour de laquelle gravitent six tables de dimensions construites à partir des entités suivantes : CLIENT, PRODUIT, PREPARATEUR_CMD, COURSIER, ENTREPOT et VEHICULE.

2.2.1 Structure de la table de faits

De façon standard, une table de faits est une table centrale qui contient deux types d'informations : la mesure et les clés. Elle peut, dans certains cas, contenir des attributs spécifiquement choisis.

2.2.1.1 La mesure

La table de faits sert principalement à exposer les *mesures* c'est-à-dire les informations d'intérêt lesquelles portent l'analyse. Dans notre exemple, les mesures sont représentées par les colonnes ***quantite*** et ***montant***.

Une mesure est généralement de type numérique. Elle peut être :

- *additive* : lorsque la mesure peut être agrégée selon n'importe quelle dimension. Par exemple: quantité commandée, montant de la commande, etc.
- *semi-additive* : lorsque la mesure peut être agrégée seulement selon certaines dimensions bien précises. Par exemple le solde d'un compte bancaire peut être agrégé selon les clients, mais pas dans le temps.
- *non-additive* : lorsque la mesure est de type proportion ou ratios.

2.2.1.2 Les clés primaires et étrangères :

En plus de la mesure, une table de faits contient également des clés primaires (*PK*) et des clés étrangères (*FK*). Les clés primaires (*primary_key*) servent d'identifiants uniques pour une ligne de fait. Ici, il s'agit de la colonne ***num_cmd***. Les clés étrangères (*foreign_key*) sont des clés qui permettent de faire le lien entre un fait et ses dimensions spécifiques. Dans le schéma ci-dessus, les clés ***id_client***, ***id_produit***, ***id_preparateur***, ***id_entrepot***, ***id_coursier***, et ***id_vehicule*** représentent les clés étrangères. Elles permettent de relier la table de faits à l'ensemble de ses dimensions retenues. Remarquons que les clés qui sont considérées comme étrangères (*FK*) dans la table de

faits deviennent des clés primaires (*PK*) dans leur table de dimension respective (voir figure 1).

2.2.1.3 Les attributs non dimensionnels

Dans un modèle idéal, la table de faits ne devrait contenir que des clés (primaires et étrangères) et des colonnes de mesure. Elle n'est pas censée contenir des attributs d'autres natures. Mais il y a toujours des exceptions à cette règle. Dans la pratique, la table de faits peut contenir d'autres attributs. Ici, nous qualifions ces attributs de *non dimensionnels.* Dans notre exemple, nous avons considéré les colonnes ***date_reception***, ***statut_paiement***, ***date_depart_coursier***, ***date_livraison*** et ***statut_livraison*** comme des attributs non dimensionnels. Bien entendu, il s'agit d'une simplification que nous avons expressément faite pour des fins de pédagogie. En réalité, toutes ces colonnes comportent une certaine dimensionnalité faisant qu'elles puissent être considérées comme des dimensions explicites du fait étudié (nous reviendrons plus tard sur les caractéristiques d'une table de dimension). Ici, nous avons simplement décidé de ne pas tirer de tables de dimension à partir de ces colonnes. Un choix qui peut sembler fortement critiquable au prime abord. C'est d'ailleurs pourquoi, dans la pratique le choix des attributs non dimensionnels doit être toujours guidé par des considérations opérationnelles tenant compte de l'expression de besoin métier. Il n'est pas nécessaire d'expliciter systématiquement une dimension à partir d'une colonne surtout lorsque cette dimension n'aura aucune pertinence métier et n'apportera pas d'amélioration significative dans la performance des requêtes. Par exemple, vouloir expliciter une dimension par rapport à la colonne ***date_depart_coursier*** ne semble avoir aucune pertinence métier réelle. En revanche, en considérant la colonne ***date_reception*** ou la colonne ***date_livraison*** il peut sembler pertinent de tirer une dimension temporelle[1]. De même, il est également possible de construire une dimension pertinente à partir de la colonne ***statut_livraison***. Tous ces

[1] Dans ce cas, pour créer une dimension **TEMPS** pour la date de réception commande, il aurait fallu d'abord générer un ***id_date_reception (FK)*** dans la table **DETAILS_CMD** et créer une table **TEMPS** dont la clé primaire (PK) est ***id_date_reception*** et dont les attributs sont : ***annee***, ***mois***, ***jour***, ***heure***, ***minute***. (Voir le format des dates dans l'exemple illustratif.). Une telle dimension aurait en effet permis d'analyser finement les commandes selon les dates de réception. La même logique peut s'appliquer également pour la date de livraison.

éléments démontrent que la frontière entre un *attribut dimensionnel* et un *attribut non dimensionnel* est extrêmement ténue.

Cependant malgré toutes ces réserves concernant la sélection des attributs non dimensionnels, la présence de tels attributs dans la table de faits a un certain nombre d'avantages. Par exemple, grâce aux attributs non dimensionnels, il est possible d'exécuter des requêtes simples sans avoir besoin de faire des jointures avec d'autres tables (ex : requêtes de comptages, etc). Les attributs non dimensionnels permettent également de calculer des indicateurs simples avec des requêtes moins lourdes. Par exemples, les colonnes ***date_reception***, ***date_depart_coursier***, ***date_livraion*** prises ici comme des attributs non dimensionnels peuvent être utilisées pour calculer des indicateurs comme la durée de préparation de commande, la durée des courses, les délais de livraisons, etc…).

2.2.2 Structure des tables de dimension

Une dimension est un axe d'analyse suivant lequel on peut considérer une mesure. Une table de dimension contient deux types d'informations : les clés primaires qui jouent le rôle d'identificateurs uniques et les attributs qui servent à décrire les contextes spécifiques associés à la dimension. Dans notre exemple de modélisation, nous avons retenues six dimensions: CLIENT, PRODUIT, PREPARATEUR, ENTREPOT, COURSIER et VEHICULE.

Dans une modélisation en étoile, les dimensions sont construites selon des niveaux de granularités supérieurs bien définis. La notion de dimension renvoie toujours une notion d'hiérarchie de regroupement par rapport au fait analysé. En effet, il y a toujours une notion de relation N à 1 entre une ligne de fait et sa dimension, c'est-à-dire que plusieurs lignes de faits peuvent se rapporter à une valeur unique de dimension. Par exemples, plusieurs lignes de commandes peuvent être associée à un même *id_client*. Alors que l'inverse n'est pas nécessairement vrai. Ainsi dès lors que la relation entre les lignes de commandes (faits) et les identifiants des clients peut être matérialisée par une relation de type *num_cmd (N)* ←→ *(1) id_client*, il devient possible de tirer une dimension CLIENT.

Dans le processus de modélisation en étoile, le choix des dimensions reste une étape cruciale car celles-ci doivent non seulement être pertinentes d'un point de vue métier

mais elles doivent également permettre d'améliorer la performance des requêtes lors de l'accès et l'utilisation des données.

Dans un schéma en étoile traditionnel, les tables de dimensions sont autonomes et généralement indépendantes. Les liens entre les tables dimensions s'établissent nécessairement par l'intermédiaire de la table de fait. C'est la raison pour laquelle toutes les tables de dimension pointent vers la table de faits. Ce qui fait le schéma en étoile. Par ailleurs, les tables de dimension ne sont pas nécessairement des tables normalisées, c'est-à-dire qu'elles peuvent comporter des dépendances partielles et des dépendances transitives. La construction de la table de dimension peut aller de soi. Une table de dimension peut se retrouver normalisée de facto. On constate d'ailleurs que dans le modèle que nous avons présenté, toutes les tables de dimension sont des tables normalisées de facto (nous reviendrons plus en détails sur ces notions lors de la présentation du modèle normalisée). Mais rappelons tout de même qu'en pratique et dans la plupart des modèles en étoile, les tables de dimension sont des tables dénormalisées.

Signalons par ailleurs que les tables de dimension comportent généralement une forte redondance des données. Par exemple dans notre modèle, si nous avions construit une dimension ADRESSE, un cas typique de redondance aurait apparu du fait que plusieurs villes appartiennent à un même département, plusieurs départements appartiennent à une même région et les régions appartiennent à un même pays. La redondance des valeurs des attributs de niveau hiérarchique supérieur permet d'accélérer l'exécution des requêtes impliquant cette table de dimension. Par exemple, en lançant une requête sur la table hypothétique ADRESSE, le moteur d'exécution n'aura plus à récupérer et à dispatcher les valeurs de pays devant les valeurs de régions et à dispatcher les valeurs de régions devant les valeurs des départements et ainsi de suite. Ce qui permet de minimiser le coût et le temps d'exécution de la requête. C'est pour cette raison que la redondance des données est parfois considérée comme un avantage du schéma en étoile.

2.3 Avantages et inconvénients du modèle en étoile

Le modèle en étoile présente de nombreuses forces mais il comporte également quelques faiblesses. Nous présentons ci-dessous un certain nombre d'avantages et d'inconvénients associé au modèle.

2.3.1 Avantages

2.3.1.1 Facilité de compréhension des données

Le fait que le modèle en étoile organise les données entre une table de faits et des tables de dimension facilite considérablement l'organisation des données.

2.3.1.2 Amélioration de la performance des requêtes

Le modèle en étoile améliore la performance des requêtes d'une part grâce à la simplicité des critères de jointure entre la table des faits et les tables de dimension et d'autre part grâce à la redondance des données dans les tables de dimension. Le fait que la table des faits soit reliée à chaque table de dimension par une clé unique permet de diminuer significativement le coût de la jointure. Par ailleurs, la redondance des données au niveau des tables de dimension est aussi un facteur qui augmente significativement la vitesse d'exécution de requêtes complexes.

2.3.1.3 Facilité d'alimentation des cubes OLAP

Tout système OLAP (Online Analytical Processing) disposant d'une fonctionnalité ROLAP (Relational OLAP) est capable de se sourcer directement à partir d'un Data Warehouse conçu en schéma en étoile sans à avoir à implémenter une structure de cube.

2.3.1.4 Simplifie la logique de reporting métier

Le schéma en étoile simplifie drastiquement la logique de reporting métier. Il permet aux métiers de travailler sur des périmètres restreints de données et d'élaborer des reportings propres à telle ou telle dimension sans être obligé de parcourir l'ensemble de la base.

2.3.2 Inconvénients

2.3.2.1 Absence de garanti sur l'intégrité des données

Le schéma en étoile ne permet pas de garantir l'intégrité des données. L'arrivée d'une nouvelle donnée dans le système nécessite de mettre à jour à la fois la table de faits mais aussi l'ensemble de ses dimensions. Dès lors, tout échec qui survient lors de cette opération de mise à jour compromet la complétude et l'intégrité des données.

2.3.2.2 Manque de flexibilité

Le schéma en étoile peut manquer de flexibilité face à un changement de la logique de reporting métier. Très souvent, le modèle en étoile est mise en place en suivant une logique métier bien définie. C'est d'ailleurs sur la base de cette logique que sont définies la structure et l'organisation des tables de dimension. Dès lors tout changement majeur dans la logique de reporting métier peut nécessiter soit une refonte partielle, soit une refonte complète du système.

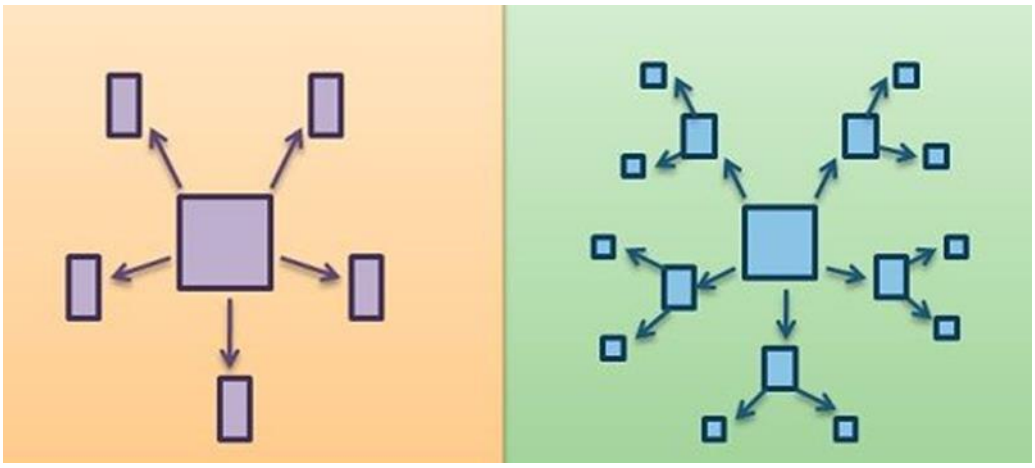
2.4 Les extensions du modèle en étoile : le schéma en flocon de neige et le schéma en constellation

2.4.1 Schéma en flocon de neige :

Le modèle en flocon de neige est un modèle obtenu en normalisant les dimensions d'un modèle en étoile. Très souvent les dimensions d'un modèle en étoile peuvent comporter, à leur tour, une ou plusieurs sous-dimensions. En explicitant ces sous-dimensions et en exposant ces sous-dimensions dans des tables dédiées par les moyens de normalisation, on obtient un modèle dit de flocon de neige (nous reviendrons plus tard sur la notion de normalisation de table).

La figure ci-dessous illustre le passage du schéma en étoile au schéma en flocon de neige.

Figure 2 : Schéma en étoile VS schéma en flocon de neige

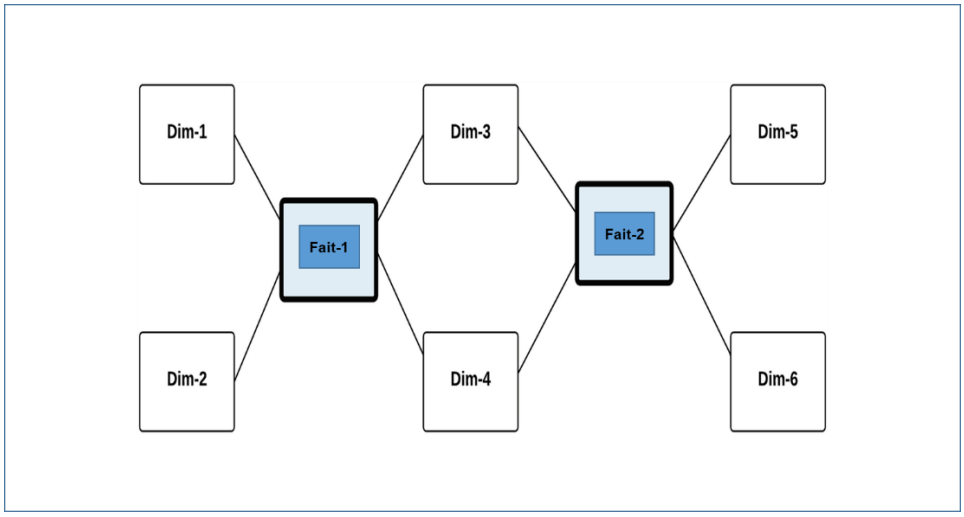


Source : https://techdifferences.com/difference-between-star-and-snowflake-schema.html

2.4.2 Schéma en constellation (modèle en galaxie) :

Le schéma en constellation (ou modèle en galaxie) est obtenu lorsque l'on juxtapose deux ou plusieurs modèles en étoile partageant des dimensions communes. Le schéma en galaxie comporte au moins deux tables de faits (voir illustration sur la figure ci-dessous).

Figure 3 : Architecture d'un modèle en galaxie à deux tables de faits



3 LE MODELE NORMALISE (3NF)

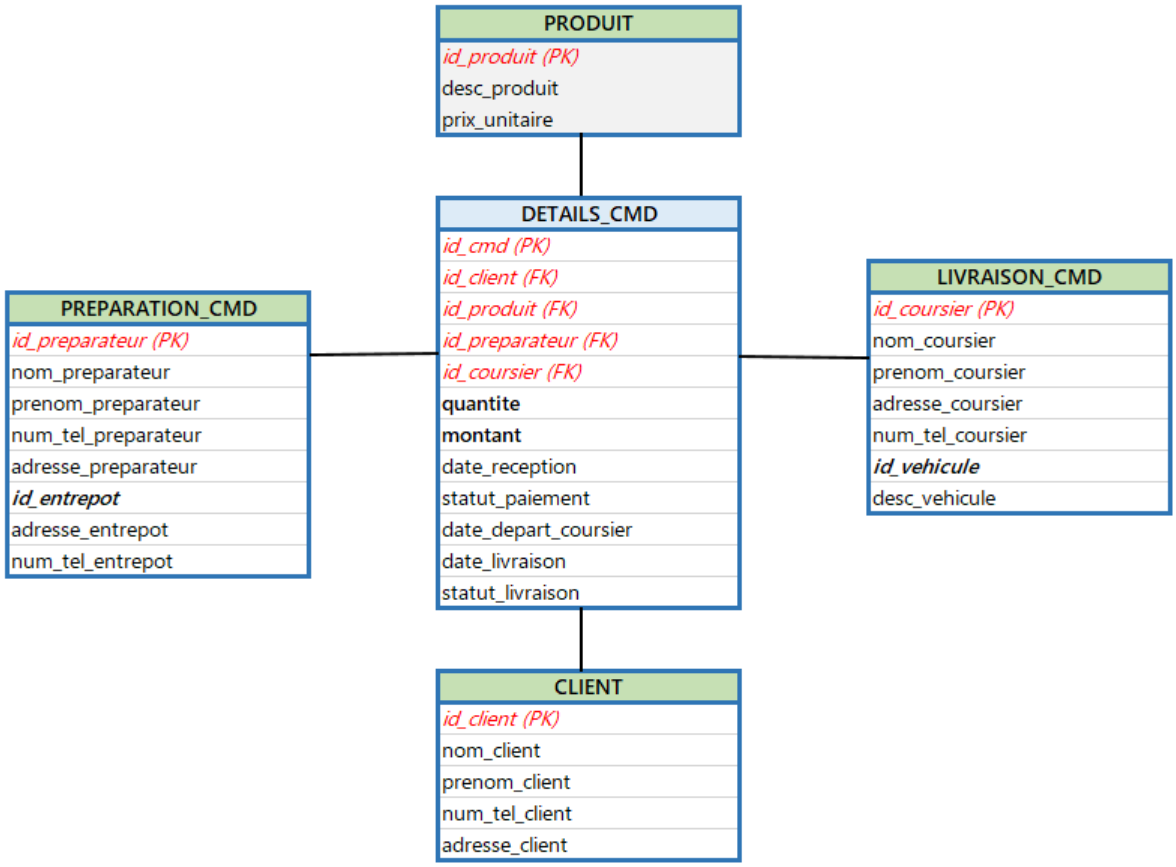
3.1 Généralités

La modélisation en troisième forme normale (3NF) est une approche de modélisation généralement choisie pour répondre à certaines limitations du schéma en étoile. Contrairement au modèle en étoile, le modèle normalisé permet d'éliminer la redondance des données, d'optimiser la structure des tables de dimension et d'assurer l'intégrité de données. Le modèle normalisé permet également d'apporter plus de flexibilité car il peut être facilement adaptable face à un changement de logique métier.

Rappelons que dans un modèle en étoile toutes les tables de dimensions pointent vers la table de faits et il n'y a aucun lien entre les tables de dimension (voir Figure 1). Dans un modèle normalisé, toutes les tables de dimension ne pointent pas obligatoirement vers la table de faits. Certaines tables de dimension peuvent pointer vers d'autres tables de dimension. Ainsi, contrairement au modèle en étoile, les relations entre les tables dimensions ne passent pas nécessairement par la table de faits. Par ailleurs, tout comme dans le modèle en flocon de neige (extension du schéma en étoile), les tables de dimension d'un modèle normalisé peuvent comporter des tables de sous-dimension.

Dans un modèle normalisé, les dimensions ainsi que les sous-dimensions sont toutes des tables en troisième forme normale. Une table est en troisième forme normale lorsqu'elle ne comporte ni dépendance partielle, ni dépendance transitive entre les attributs et les clés (nous reviendrons plus en détail sur la notion de normalisation). Mais d'ores et déjà, nous pouvons indiquer qu'une table comporte une dépendance partielle, lorsqu'un attribut ne dépend pas de la totalité des clés primaires. Et la table comporte une dépendance transitive lorsqu'un attribut dépend uniquement d'une clé qui n'est pas une clé primaire. Pour illustrer les notions de dépendance partielle et transitive, utilisons un cas fictif comme celui présenté sur la figure 4 ci-dessous.

Figure 4 : Cas fictif d'un schéma en étoile dénormalisé



Supposons qu'à la place du modèle en étoile présenté à la Figure 1, nous avions construit un autre modèle en étoile comme celui de Figure 4. On constate qu'en plus des dimensions de base CLIENT et PRODUIT, nous avons désormais deux dimensions particulières PREPARATION_CMD et LIVRAISON_CMD. La clé primaire de la table PREPARATION_CMD est ***id_preparateur***. C'est cette clé qui permet de faire le lien avec la table de faits DETAILS_CMD en tant que clé étrangère. De même la clé primaire de la table LIVRAISON_CMD est ***id_coursier*** (clé étrangère dans la table de faits).

Dans ce nouveau modèle (rappelons-le, reste un modèle très hypothétique), on constate que les clés et les attributs relatifs aux entrepôts et aux véhicules sont désormais associés respectivement aux préparateurs de commande et aux coursiers. La logique est la suivante. Les préparateurs de commandes sont toujours associés à des entrepôts. Ainsi, au lieu de construire une dimension propre aux entrepôts, les clés relatives aux entrepôts peuvent directement être portés par l'***id_preparateur*** des préparateurs de commande. Nous regroupons donc dans la même table les clés et attributs des préparateurs de commande et ceux des entrepôts. Cette opération

s'appelle la dénormalisation. Nous appliquons le même principe de dénormalisation pour construire la table de dimension LIVRAISON_CMD. En effet, les clés représentant les véhicules peuvent toujours être associés à un ***id_coursier*** lors d'une opération de livraison. De ce fait, il n'est pas nécessaire de construire une dimension propre aux véhicules indépendamment de la dimension des coursiers. C'est dans cette logique qu'a été construite la table LIVRAISON_CMD.

Le modèle hypothétique présenté sur la figure 4 est un modèle en étoile dont certaines dimensions ne sont pas normalisées. L'exercice pour nous consistera donc à construire un modèle normalisé dans lequel toutes les tables sont troisième forme normale.

3.2 Normalisation d'une table : quelques rappels

Dans un modèle normalisé, toutes les tables sont en troisième forme normale (3NF). Pour comprendre la notion de troisième forme normale, il faut d'abord comprendre les notions de première forme normale et de deuxième forme normale.

3.2.1 La première forme normale

Une table est en première forme normale lorsque :

- elle possède une ou plusieurs clés primaires
- chaque attribut contient une valeur atomique c'est-à-dire une information non divisible.

Pour illustrer la propriété de première forme normale, partons de l'exemple fictif ci-dessous. Une table qui fournit les informations sur les pilotes d'avions ainsi que les appareils sur les lesquels ils effectuent leurs vols.

id_pilote	nom_pilote	id_appareil
25	Alain	A230,A687
37	Victor	A915

Cette table possède bien une clé primaire (*id_pilote*) mais elle n'est pas en première forme normale. L'attribut *id_appareil* n'est pas atomique à cause de la non unicité de la valeur pour le pilote 25.

Pour que la table soit en première forme normale, elle doit être structurée comme suit :

id_pilote	nom_pilote	id_appareil
25	Alain	A230
25	Alain	A687
37	Victor	A915

3.2.2 La deuxième forme normale

Une table est en deuxième forme normale lorsque :

- la table est en première forme normale
- tous les attributs (non clés) dépendent de la totalité des clés primaires et non d'une partie de celles-ci.

La notion de deuxième forme est généralement évoquée pour les tables dont la clé primaire est constitué à partir de plusieurs colonnes. Par exemple pour une table de notation des élèves d'une école, la clé primaire peut être constituée à partir des colonnes *id_eleve*, *id_classe*, *id_matiere*. C'est l'association de ces trois colonnes qui garantit l'unicité d'une note. Bien entendu, il n'est pas exclu de retrouver dans la même table d'autres attributs comme le nom de l'élèves, son âge, le nom de l'enseignant, etc. Mais pour que cette table de notation soit considérée en deuxième forme normale, il faudrait que tous les attributs dépendent de la totalité des clés, c'est-à-dire l'association des trois colonnes de clé et non une partie des clés. En d'autres termes, il ne doit pas y avoir de dépendance partielle entre les attributs et la clé primaire.

Pour illustrer la notion de dépendance partielle, considérons la table de notation des élèves présentée ci-dessous.

id_eleve	*id_classe*	*id_matiere*	note	nom_eleve	enseignant
01	04	Anglais	13	Mohamed	09
02	06	Anglais	15	Jeremy	09

La clé primaire de cette table est formée par les trois colonnes *id_eleve*, *id_classe* et *id_matiere*. Car ce sont ces trois colonnes qui, prises en ensemble, permettent d'identifier une note unique. Cependant cette table n'est pas une table en deuxième

forme normale. Par exemple, la colonne *nom_eleve* ne dépend ni de *id_classe*, ni de *id_matiere*. Elle dépend uniquement de *id_eleve*. Cela traduit donc une dépendance partielle. De même, on peut admettre que l'*enseignant* dépend uniquement de *id_classe* et de *id_matiere*. Elle ne dépend pas de *id_eleve*. En d'autres termes un enseignant est toujours affecté à une classe et à une matière, mais pas à un élève spécifique de la classe. De ce point de vue, il existe une dépendance partielle entre l'attribut *enseignant* et la clé primaire de la table.

Pour que cette table soit en deuxième forme normale, il faut réorganiser la table pour éclater les informations en plusieurs tables. Pour corriger la dépendance partielle liée à l'attribut *nom_eleve*, il suffit simplement de déplacer cette colonne dans une nouvelle table pour constituer les informations spécifiques des élèves. Bien entendu, cette table sera reliée à la table notation par la clé étrangère *id_eleve*. Et pour corriger la dépendance partielle liée à l'*enseignant*, il faut déplacer l'*enseignant* dans une table dédiée. Cette table relation sera reliée à la table de notation grâce aux deux colonnes *id_classe* et *id_matiere*. Le découpage présenté ci-dessous permet de transformer la table de notation initiale en trois tables respectant chacune la deuxième forme normale. Voir le découpage ci-dessous.

NOTE_ELEVE

id_eleve	*id_classe*	*id_matiere*	note
01	04	Anglais	13
02	06	Anglais	15

ELEVE

id_eleve	nom_eleve
01	Mohamed
02	Jeremy

AFFECTATION_ENSEIGNANT

id_classe	*id_matiere*	enseignant
04	Anglais	09
06	Anglais	09

3.2.3 La troisième forme normale

Une table est en troisième forme normale lorsque :

- la table est en deuxième forme normale
- il n'y a pas de dépendance transitive, c'est-à-dire tous les attributs dépendent directement de la clé primaire (dépendance fonctionnelle directe). La dépendance entre les attributs et la clé primaire ne devrait pas passer par une clé non primaire.

Pour illustrer la notion de dépendance transitive, considérons la table ci-dessous qui fournit les informations sur les vols nationaux assurés par une compagnie aérienne.

id_pilote	nom_pilote	num_vol	ville_depart	ville_arrivee
25	Alain	AF4826	Paris	Nantes
25	Alain	AF7521	Nantes	Paris
37	Victor	AF2345	Toulouse	Paris

Par définition, cette table est en deuxième forme normale. Car comme sa clé primaire n'est constitué qu'une seule colonne (*id_pilote*), dans ce cas elle ne peut pas comporter de dépendance partielle. En revanche, cette table n'est pas en troisième forme normale car elle comporte une dépendance transitive. Par exemple, les colonnes *ville_depart* et *ville_arrivee* dépendent directement de *num_vol* qui est une clé non primaire. C'est par l'intermédiaire de *num_vol* que les attributs *ville_depart* et *ville_arrivee* dépendent de la clé primaire *id_pilote*. La table comporte donc une dépendance transitive.

Pour éliminer une dépendance transitive, il faut déplacer dans une table dédiée tous attributs qui dépendent d'une clé non primaire donnée. Dans le cas présent, les attributs *ville_depart* et *ville_arrivee* dépendent de la clé non primaire *num_vol*. Nous allons donc créer une table spécifique qui fournit les détails. Pour obtenir la forme normalisée, il faut décomposer la table initiale comme suit :

REFERENCE_VOLS

id_pilote	nom_pilote	*num_vol*
25	Alain	AF4826
25	Alain	AF7521
37	Victor	AF2345

DETAILS_VOLS

num_vol	ville_depart	ville_arrivee
AF4826	Paris	Nantes
AF7521	Nantes	Paris
AF2345	Toulouse	Paris

La mise en troisième forme normale de la table initiale permet d'aboutir à deux tables distinctes reliées par la colonne de clé non primaire initiale. La colonne *num_col* joue le rôle de clé étrangère dans la table REFERENCE_VOLS et clé primaire dans la table DETAILS_VOLS.

En résumé, dans la modélisation 3NF, tous les attributs qui ne dépendent pas des clés primaires doivent être déplacées dans des tables dédiées tout en gardant le lien avec la table d'origine à travers une clé étrangère choisie parmi les attributs qui, eux, dépendent des clés primaires. Ainsi dans le processus de mise en troisième forme normale, toute dépendance transitive identifiée doit être normalisée par la création d'une table dédiée.

3.3 Cas d'illustration

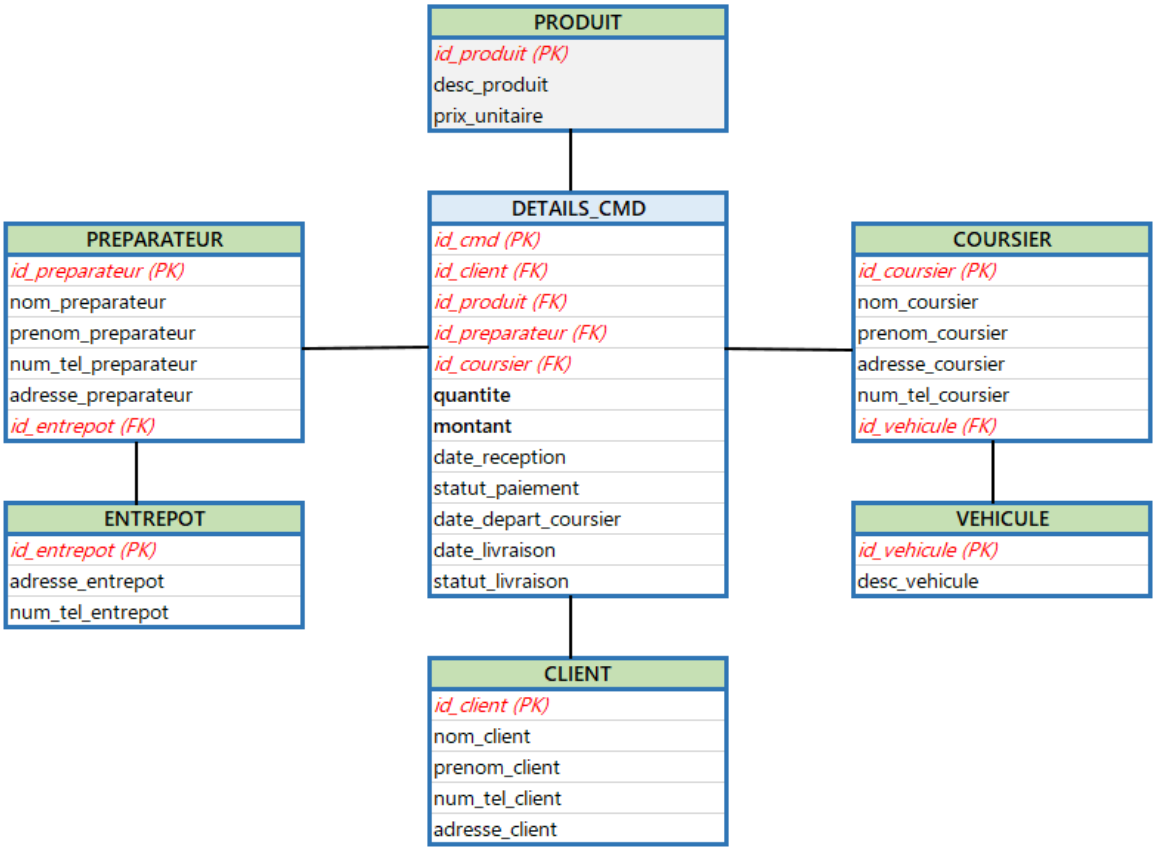
Après avoir présenté la normalisation en troisième forme, l'objectif maintenant est d'appliquer ses principes à notre cas d'illustration (voir schéma en étoile dénormalisé de la Figure 4).

Comme on peut le constater sur la Figure 4, les dimensions CLIENT et PRODUIT sont déjà normalisées. Ces dimensions respectent toutes les propriétés d'une table en troisième forme normale. Elles ne comportent ni dépendance partielle, ni transitive. A l'inverse, les tables PREPARATION_CMD et LIVRAISON_CMD sont des tables dénormalisées. Certes, elles ne comportent pas de dépendance partielle (car leur clé

primaire est constituée d'une seule colonne), mais elles comportent néanmoins une dépendance transitive. Concernant la table PREPARATION_CMD, les attributs ***adresse_entrepot*** et ***num_tel_entrepot*** dépendent directement de la clé non primaire ***id_entrepot***. Il y a donc une dépendance transitive avec la clé primaire ***id_preparateur***. De même pour la table LIVRAISON_CMD, l'attribut ***desc_vehicule*** dépend directement de la clé non primaire ***id_vehicule*** créant ainsi une dépendance transitive avec la clé primaire ***id_coursier***.

Comme le stipule le principe de normalisation en troisième forme, tous les attributs ayant une dépendance transitive avec la clé primaire doivent être déplacés dans une table dédiée. En suivant ce principe, le modèle présenté à la figure 4 peut être normalisé comme ci-dessous.

Figure 5 : Modèle en troisième forme normale (3NF)



La table initiale PREPARATION_CMD est finalement décomposée en deux dimensions distinctes PREPARATEUR et ENTREPOT reliées par la clé ***id_entrepot***. De même la table LIVRAISON_CMD est décomposée en deux dimensions COURSIER et VEHICULE reliées par la clé ***id_vehicule***.

Remarquons que cette décomposition permet d'aboutir à un modèle normalisé dont les dimensions sont les mêmes qui ont été retenues lors de la modélisation en étoile standard (Figure 1). La différence est simplement que dans le schéma en étoile toutes les dimensions pointent vers la table de faits (voir Figure 1). Ce qui n'est pas nécessairement le cas dans un modèle normalisé où les dimensions peuvent être inter-reliées (Figure 5).

3.4 Avantages et inconvénients du modèle normalisé 3NF

3.4.1 Avantages

3.4.1.1 Respect de l'intégrité des données

La réduction de la duplication des données, grâce à la suppression des dépendances transitives, réduit les anomalies d'écriture lors de la mise à jour des données.

3.4.1.2 Allègement de la structure des tables

La réduction de la redondance permet d'alléger la structure des tables tout en augmentant l'efficacité de stockage. Le parcours de l'information devient plus rapide et plus ciblé lorsque les tables sont relativement petites

3.4.1.3 Une plus grande flexibilité

Face à un changement de logique métier, le modèle normalisé peut facilement être adapté sans voir besoin d'une refonte complète du système.

3.4.2 Inconvénients

3.4.2.1 Jointure coûteuses

La normalisation aboutit parfois à un éparpillement des données dans plusieurs tables reliées entre elles par des relations en cascade. Dans une telle configuration pour reconstituée une ligne complète de données il faut passer par une requête avec de multiples jointures souvent très couteuses.

3.4.2.2 Difficulté de compréhension du schéma général des données

La normalisation aboutit parfois à une démultiplication des tables de dimensions et de sous-dimensions. Cette situation rend difficile la compréhension du schéma générale de la base et de la relation entre les différentes tables.

3.4.2.3 Charges de maintenance élevées

Lorsque le nombre de tables devient trop élevé dans la base de données, toute opération de traitement nécessite beaucoup plus de CPU, de mémoire et d'opération E/S. Ce qui réduit les performances de la base de données.

4 LE MODELE DATA VAULT

4.1 Généralités

Le modèle Data Vault est un modèle conceptuel de données permettant d'implémenter des Data Warehouses multicanaux capables de s'adapter dynamiquement à tout changement de logique métier.

Très longtemps, la modélisation de Data Warehouse d'Entreprise a été dominée par les deux approches classiques à savoir l'approche de Kimball (modèle en étoile) et celle de Inmon (modèle normalisé). C'est au cours des années 1990 que le modèle Data Vault a fait son apparition. Il a été proposé par Dan Linstedt dans l'objectif de tirer parti, au mieux, des avantages liés aux deux approches précédentes. Cependant les premières implémentations ne sont apparues qu'au cours des années 2000 suite à la publication par l'inventeur de plusieurs articles sur le sujet : « *Data Vault Series* » parus en 2002 dans le journal *The Data Administration Newsletter*. En 2013 une nouvelle spécification du modèle a été proposée pour mieux gérer les contraintes liées à l'intégration des technologies Big Data. Cette nouvelle adaptation est présentée sous le nom de Data Vault 2.0.

4.2 Architecture du Data Vault

Le modèle Data Vault est un modèle hybride entre le schéma en étoile et le modèle normalisé. Il organise les données de sorte à séparer les *informations structurelles* des *informations contextuelles*. Les informations structurelles font référence aux identifiants des objets métiers et aux relations qui lient ces objets. Les identifiants des objets métiers sont stockés dans des tables appelées **HUBs**. Et les relations entre les objets métiers sont stockées dans des tables appelées **LINKs**. Quant aux informations contextuelles, elles font référence aux attributs fonctionnels qui décrivent les objets métiers. Ces attributs fonctionnels sont stockés dans des tables appelées **SATELLITEs**. L'objectif de cette section est de présenter la structure de chaque type de table ainsi que les relations que les lient dans l'architecture générale du Data Vault.

4.2.1 Les Hubs

Dans le Data Vault, les hubs sont des entités permettant de représenter les concepts métiers d'une organisation. Ce sont des tables référentielles qui servent à identifier les objets métiers à travers des clés fonctionnelles (business keys). De ce point de vue, les hubs jouent uniquement le rôle d'identificateurs. A noter que les clés fonctionnelles sont des clés naturelles qui permettent d'identifier de manière singulière l'objet métier. Par exemples, le numéro de matricule d'un employé, le code client, la référence d'un produit, le numéro de contrat, etc… Mais pour assurer une meilleure identification des objets métiers, chaque objet métier devrait se voir attribuer la même valeur de clé fonctionnelle partout au sein de l'organisation. Par exemple, un client particulier doit être identifié avec le même code client aussi bien au sein du département commercial qu'au sein du département de Delivery.

Pour illustrer concrètement la notion de hub, prenons l'exemple du service commercial d'une entreprise qui dispose d'un fichier de données contenant l'ensemble des ventes réalisées au cours de l'année. Ce service souhaite pousser ces données dans un Data Warehouse modélisé suivant le modèle Data Vault. Quels sont les hubs qui peuvent être alimentés à partir de ces données ?

D'abord, supposons que chaque acte de vente met en jeu deux objets métiers : un objet client et un objet contrat. La clé fonctionnelle naturelle qui identifie chaque client de manière unique est le code client et la clé fonctionnelle qui identifie chaque contrat de manière unique est le numéro de contrat. Ces clés d'identification étant connues, on peut créer deux hubs distincts : un hub client et un hub contrat.

En plus des clés fonctionnelles naturelles, le hub doit disposer d'une clé primaire (pk) qui est généralement obtenue en appliquant une fonction de hashage sur la valeur concaténée des clés fonctionnelles. Par exemple, la clé fonctionnelle du hub client étant *code_client*, la clé primaire du hub sera *client_pk* dont la valeur correspond au hashage de *code_client*. De même la clé fonctionnelle du hub contrat étant le numéro contrat, la clé primaire de ce hub sera *contrat_pk* dont la valeur correspond au hashage de *numero_contrat*. Le hashage des clés fonctionnelles naturelles en une clé fonctionnelle artificielle est une opération très utile car elle permet de réduire la valeur de clé en une seule valeur surtout dans les situations où un objet métier est identifié par plusieurs clés fonctionnelles naturelles. Par exemple, si l'on prend l'exemple d'un objet de

notation des élèves d'une école. Une note spécifique ne peut être identifiée qu'en combinant plusieurs clés à savoir l'id élève, l'id classe, l'id matière. Mais en appliquant une fonction de hashage sur la valeur concaténée des clés fonctionnelles naturelles, on réduit la clé à une seule colonne. C'est pourquoi il est recommandé d'utiliser les fonctions de hashage pour générer la clés primaire d'un hub quel que soit le nombre de clés fonctionnelles naturelles en entrée.

Il est important de remarquer qu'un hub contient à la fois la valeur de clé primaire (valeurs hashée) et les valeurs des clés fonctionnelles d'origine (valeurs en clair). Ce qui ne sera pas nécessairement le cas pour les links et les satellites qui, comme nous allons le voir plus tard, ne gardent que les clés primaires hashées. Il faut toujours passer par une jointure avec les hubs concernées pour pouvoir récupérer les valeurs des clés fonctionnelles d'origine.

Par ailleurs, les colonnes représentant les clés fonctionnelles naturelles sont souvent renommées en des noms standards pour des besoins d'harmonisation entre les différentes sources de données. Pour mieux illustrer cette situation, supposons par exemple que le hub client soit alimenté à partir de trois systèmes sources différentes au sein de l'entreprise. Dans le premier système source (ex : service commerciale), les clients sont fonctionnellement identifiés *id_client* ; dans le deuxième système source (ex : Service de comptabilité), les clients sont identifiés avec *ref_client* ; et dans le troisième système (Delivery), les clients sont identifiés avec *num_client*. Sachant que l'ensemble de ces systèmes se réfèrent aux même clients, qu'ils vont alimenter le même hub client, il devient nécessaire d'harmoniser le nom de la clé fonctionnelle dans le hub afin de pouvoir accueillir l'ensemble des valeurs de clés d'identification des clés quel que soit le système. Ainsi dans le hub client la clé fonctionnelle sera nommé « *code_client* ». Dès qu'un modèle Data Vault est alimenté par plusieurs systèmes sources, il convient d'être vigilent et d'appliquer une règle de nommage des clés fonctionnelles dans le hub de façon à respecter l'harmonie dans l'identification des objets métiers.

Signalons qu'en dehors de la clé primaire (valeur hashée) et des clés fonctionnelles naturelles (éventuellement renommées), un hub ne devrait contenir aucun autre attribut spécifique à l'objet métier. Seules certaines métadonnées peuvent y être ajoutées notamment pour pouvoir retracer l'origine de l'objet métier (ex : système

source, pôle métier, département) et sa date de chargement. Toutefois, dans certains cas pour éviter les problèmes de performance liés aux clés fonctionnelles complexes, des clés de substitution (surrogate key) peuvent être ajoutée à la table. Les surrogate keys sont des clés techniques internes à la table qui n'ont aucune signification métier. Très souvent la surrogate key est simplement un numéro incrémental pour chaque ligne insérée dans le hub. C'est donc une clé interne à la table. Elle ne peut pas servir de clé de jointure entre des tables évoluant à des rythmes différents.

Enfin, remarquons également qu'un hub ne peut représenter qu'un seul objet métier. Il ne doit pas être conçu pour identifier des objets métiers de natures différentes. Par exemple, les objets métiers comme les contrats de vente et les contrats d'achats ne peuvent pas être stockés dans un même hub. De même un hub ne peut pas être utilisé pour identifier à a fois les clients et les fournisseurs. Un hub distinct doit être construit pour chacun de ces objets métiers.

4.2.2 Les Links

A la différence des hubs qui sont des entités d'identification des objets métiers, les links sont, quant à eux, des entités de mise en relation des objets métiers. En effet, nous l'avons vu précédemment, chaque hub est construit de manière autonome uniquement à partir des clés fonctionnelles de l'objet métier qu'il est censé représenté. De ce fait, la structure d'un hub ne permet pas de déterminer la relation que peut avoir un objet métier avec un autre objet métier. Ce rôle est dévolu aux links.

Un link a pour fonction d'établir la relation entre deux ou plusieurs objets métiers. En pratique, étant donné que les hubs représentent les objets métiers, pour établir une relation entre deux objets métiers il suffit de créer une relation entre leur hub en mettant en associant leurs clés fonctionnelles respectives. Les links servent de point de raccordement entre deux ou plusieurs hubs.

Pour illustrer concrètement la notion de link, reprenons l'exemple du service commercial qui souhaite pousser dans un Data Warehouse des données relatives aux ventes aux clients réalisées au cours de l'année (voir l'exemple dans section consacrée aux hubs). Nous avons supposé que l'acte de vente met en jeu deux objets métiers distincts (client et contrat) permettant ainsi de créer un hub client et un hub contrat. Mais jusque-là, rien ne nous permettait de mettre en relation le hub client et le hub

contrat afin de pouvoir déterminer par exemple quel client a souscrit à quel(s) contrat(s). Seule une table de link permet de faire une telle distinction.

Dans cet exemple, la relation entre les deux hubs sera établie avec un link *client_contrat*. La table de link est une table généralement construite sur une relation de type 1 à N, de type N à 1 ou de type N à N qui à chaque ligne d'un hub fait correspondre une ou plusieurs lignes d'un autre hub. Par exemple la table link *client_contrat* permettra de faire ressortir pour chaque *code client* l'ensemble des *numero_contrat* qui lui sont associés. Bien entendu, un link garde toute l'historique des relations entre deux objets métiers. Aucune relation n'est écrasée même si celle-ci n'est plus valable. Par exemple, le contenu de la table link *client_contrat* doit refléter l'ensemble des contrats passés et présents souscrits par un code client spécifique.

Tout comme le hub, le links ne contient aucune information descriptive. Il ne contient qu'une clé primaire (pk) et quelques clés étrangères(fk). La clé primaire traduit la relation entre les différents objets métiers. Elle est obtenue en appliquant une fonction de hashage sur la valeur concaténée de l'ensemble des clés fonctionnelles provenant de tous les hubs mis en relation. Quant aux clés étrangères (fk), elles correspondent simplement aux clés primaires des hubs mis en relation. Pour bien expliciter ces propos considérons la table de link *client_contrat* précédemment évoquée. Elle représente la table de lien entre le hub *client* et le hub *contrat*. Le *code_client* étant la clé fonctionnelle naturelle du hub client, et le *numero_contrat* la clé fonctionnelle du hub contrat, la clé primaire du link nommé *client_contrat_pk* est déterminée par hashage de la valeur concatenée de *code_client* et *numero_contrat*. Les clés étrangères seront *client_pk* et *contrat_pk* qui sont respectivement la clé primaire du hub client et la clé primaire du hub contrat. Ces deux pks ont étés toutes obtenues par hashage de la clé fonctionnelle de chaque hub, à savoir *code_client* pour hub client et *numero_contrat* pour hub contrat. Ayant toutes été obtenues par hashage, les clés primaires et étrangères d'un link sont donc des clés fonctionnelles artificielles.

Rappelons que seules les clés hashées sont gardées dans les links. Les clés fonctionnelles ne sont pas gardées. Pour retrouver les valeurs d'origine d'une clé ayant été hashée, il faut faire une jointure avec le hub correspondant pour aller récupérer la valeur de la clé fonctionnelle naturelle. C'est d'ailleurs cette particularité qui est reprochée au modèle Data Vault, car pour accéder à la valeur fonctionnelle d'une clé

d'objet il faut toujours se référer aux hubs. Cela nécessite toujours une opération de jointure.

En plus des clés d'identifications des relations, d'autres métadonnées peuvent être ajoutées au link notamment l'information sur le système source et la date de chargement. On peut également y ajouter une surrogate key qui peut servir, dans certains cas, de clé primaire de substitution. Généralement, la surrogate key est un numéro incrémental pour chaque ligne insérée. C'est une clé interne qui ne peut pas servir de clé de jointure entre des tables évoluant à des rythmes différents.

4.2.3 Les satellites

Contrairement aux hubs et links qui ne contiennent que des données structurelles (clés fonctionnelles et relations), les satellites, eux, sont les entités qui contiennent les attributs fonctionnels des objets métiers. A la différence des clés fonctionnelles, les attributs fonctionnels sont des informations qui bougent au cours du temps. Par exemples l'âge d'un client, le statut d'un contrat (actif ou résilié), statut d'un paiement (OK ou KO), etc… Le satellite empile toutes les versions des attributs d'un objet métier dans le temps sans écraser les anciennes versions.

Un satellite est généralement conçu pour stocker tout ou une partie des attributs fonctionnels d'un objet métier. Supposons, par exemple, qu'on dispose d'un objet métier spécifique pour représenter les vendeurs (voir l'exemple des contrats de ventes précédemment présenté). D'abord, nous aurons nécessairement un *hub vendeur* qui sera construit à partir d'une clé fonctionnelle *id_vendeur*. Ensuite, nous aurons un *satellite vendeur* qui contiendra tous les attributs du vendeur ayant engagé le contrat de vente. Ces attributs peuvent inclure des attributs biographiques (nom, prénom, âge, situation familiale, etc.…) et des attributs de localisation (adresse, ville, département, région, etc.). Le modélisateur peut alors décider soit de stocker tous les attributs dans un seul satellite *vendeur* soit de rassembler les attributs par groupe et créer un satellite pour chaque groupe d'attribut. Par exemples, les attributs biographiques peuvent être stockés dans un satellite *infos_vendeur*. Les informations de contact peuvent aussi être stockées dans un satellite dédié *contact_vendeur*. C'est pour cette raison que nous indiquons qu'un objet métier peut donner lieu à plusieurs satellites. Tout dépend, en

fait, de l'organisation qu'on souhaite donner aux données tout en respectant la logique métier.

Très souvent un satellite est relié à un seul hub. Et lorsqu'un satellite doit être relié à deux ou plusieurs hubs, cette liaison doit nécessairement passer par un link. La relation entre un hub et un satellite s'établit directement à travers la clé primaire (pk) du hub et la clé primaire (pk) du satellite. La clé primaire du satellite est calculée de la même manière que la clé primaire du hub, c'est-à-dire en appliquant une fonction de hashage sur la valeur concaténée des valeurs des clés fonctionnelles naturelles. Cependant, contrairement au hub où les clés fonctionnelles naturelles sont gardées en plus de la clé primaire, dans les satellites seule la clé primaire est gardée. Rappelons que la clé primaire est une valeur hashée. Pour retrouver les valeurs naturelles des clés fonctionnelles, il faut faire une jointure avec le hub en utilisant la clé primaire hashée.

Tout comme pour le hub, la relation entre un satellite et un link s'établit à travers la clé primaire (pk) du link et la clé primaire (pk) du satellite. Lorsqu'un satellite est relié à un link, la clé primaire de ce satellite est calculée de la même manière que la clé primaire du link afin de pouvoir garantir la correspondance. Rappelons que la valeur de la clé primaire du link est obtenu en appliquant une fonction de hashage sur la valeur concaténée de l'ensemble des clés fonctionnelles naturelles des hubs reliés à ce link. C'est en utilisant les mêmes colonnes fonctionnelles et en appliquant le même principe de hashage qu'on obtient la clé primaire du satellite. On remarque donc que le mode de calcul de la clé primaire d'un satellite relié directement à un hub diffère légèrement du mode de calcul de la clé primaire d'un satellite relié directement à un link.

L'une des spécificités du Data Vault est que les clés fonctionnelles naturelles ne sont pas visibles ni dans les satellites, ni dans le links. Elles sont visibles uniquement dans les hubs. En dehors des hubs, les clés hashées jouent le rôle de clés fonctionnelles artificielles.

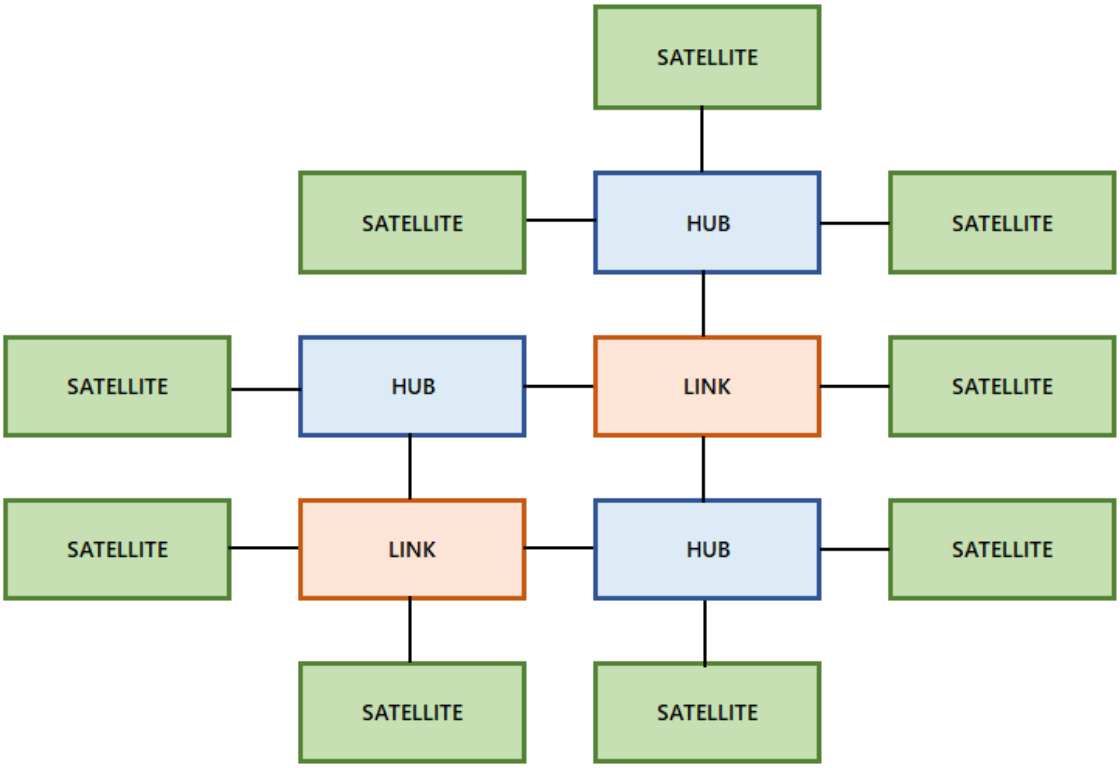
L'un des avantages des satellites, c'est qu'ils apportent une plus grande flexibilité dans le chargement des données. Par exemple, pour un même objet métier on peut créer un satellite par système source. On peut même créer un satellite selon les fréquences de chargement des données. Par exemple, pour les données concernant les soldes des comptes bancaires des clients, on peut stocker les informations sur les soldes mensuels dans un satellite dédié et les informations sur les soldes annuels dans un autre satellite.

De ce point de vue, le Data Vault offre une bien meilleure flexibilité dans la conception et le chargement des satellites que la plupart des approches de modélisation.

4.2.4 Relations entre les hubs, les links et les satellites

Les relations entre les hubs, les links et les satellites peuvent être matérialisées par l'architecture générale ci-dessous.

Figure 6 : Architecture général du modèle Data Vault



Toutes proportions gardées, le modèle Data Vault est souvent comparé à un réseau neuronal où les hubs jouent le rôle de noyaux, les links jouent le rôle de synapses et les satellites jouent le rôle de dendrites. Un hub peut être lié à plusieurs satellites mais un satellite ne peut être lié qu'à un seul hub ou un seul link.

Comme on peut le remarquer sur la figure 6, un hub est lié à au moins un satellite. Tous les hubs n'ont pas nécessairement de link. Les links permettent de relier deux ou plusieurs hubs. Dès qu'il y a une relation fonctionnelle entre deux objets métiers, cette relation est matérialisée par une table de Link.

Dans des modèles plus complexes, il arrive qu'un link soit lié à un autre link mais cette pratique pénaliserait le parallélisme au niveau du chargement des données. La création d'un second link entre les hubs concernés est recommandée.

4.3 Etapes de modélisation du Data Vault

Le processus de modélisation du Data Vault est étroitement aligné sur l'analyse métier. La modélisation du Data Vault se déroule en trois grandes étapes : 1-) Identifier les objets métiers et leurs clés fonctionnelles ; 2-) Définir les relations entre les objets métiers ; 3-) Sélectionner les attributs fonctionnels des objets métiers.

4.3.1 Identifier les objets métiers et leurs clés fonctionnelles

Les objets métiers constituent l'épine dorsale de la modélisation Data Vault. Tout travail de modélisation doit d'abord passer par une phase d'identification des objets métiers. Cette étape exige une meilleure compréhension et connaissance du processus métier. Dans une entreprise commerciale par exemple, les objets métiers sont les clients, les fournisseurs, les employés, les contrats, les produits, etc. Mais en plus de ces objets métiers standards, on peut identifier d'autres objets moins évidents comme par exemples les garanties sur les produits ou les options d'achats. C'est pourquoi, il est recommandé de mener cette étape d'identification de concert avec les acteurs métiers.

Dès qu'un objet métier est identifié, on procède par la suite à la sélection et à la validation des clés fonctionnelles (business keys). Ces clés doivent être représentatives de l'objet métier partout au sein de l'entreprise. En effet, étant donné que le Data Warehouse a une portée organisationnelle, les clés fonctionnelles choisies doivent avoir une portée à l'échelle de toute l'entreprise. Les clés fonctionnelles choisies doivent également être significatives pour les utilisateurs métiers. De ce fait, elles ne doivent pas dépendre d'un système source particulier ou varier d'un système à un autre.

Connaissant les objets métiers et leurs clés fonctionnelles, on peut procéder à la modélisation de la structure des hubs. Pour rappel, chaque objet métier donne lieu à un hub spécifique (cf. présentation des hubs plus haut dans les sections précédentes).

4.3.2 Définir les relations entre les objets métiers

La seconde étape dans le processus de modélisation du Data Vault est la définition des relations entre les objets métiers. En pratique, les objets métiers étant représentés par des hubs, définir les relations entre les objets métiers revient simplement à définir une relation entre les hubs. Cependant cette étape exige aussi une meilleure compréhension du processus métier. D'où la nécessité de travailler en étroite collaboration avec les acteurs métiers pour identifier des liens pertinents et raffinés. Par exemple, dans le pôle commercial d'une entreprise, on admettra qu'il y a un lien entre l'entité CLIENT et l'entité CONTRAT. Ce lien transparaît dans les actes de ventes matérialisés par une table VENTES. De même, il y a un lien entre l'entité FOURNISSEUR et l'entité PRODUIT ; un lien qui peut transparaître dans les opérations de gestion de stocks matérialisées par une table STOCKS.

La relation entre les hubs est toujours modélisée par une entité link. A noter qu'une relation peut associer deux ou plusieurs objets métiers. Et plus il y aura d'objets associés, plus la granularité sera fine. Dès que toutes les relations pertinentes entre les hubs ont été identifiées et clairement spécifiées, on procède maintenant à la modélisation des links (voir la structure des tables links plus haut dans les sections précédentes)

4.3.3 Sélectionner les attributs fonctionnels des objets métiers.

Les hubs et les links étant modélisés, la troisième étape du processus est la sélection des attributs fonctionnels pour constituer les satellites.

Faisons remarquer tout de suite qu'il n'est pas nécessaire de modéliser tous les satellites dans cette phase initiale de mise en place du modèle du Data vault. D'autres satellites pourront être modélisés et rajoutés durant tout le cycle de vie du modèle. Dans la phase initiale du modèle, il suffit d'identifier quelques satellites pertinents capables de répondre aux besoins métiers courants et clairement identifiés. En effet, le Data Vault étant un modèle dynamique, il est possible à tout moment d'ajouter ou de retrancher un satellite du modèle sans compromettre le modèle. A l'inverse des hubs et des links qui sont des objets figés, les satellites sont des objets volatiles. Par exemple lorsque la structure d'un satellite ne permet plus de répondre complètement à un

besoin métier (ex : besoin de rajouter des attributs supplémentaires), il suffira simplement de le décommissionner et d'initialiser un nouveau satellite qui prend en compte le besoin exprimé. L'ancien satellite gardera toujours l'historique complète des données qui ont été chargées. C'est d'ailleurs cette flexibilité qui fait toute la force du Data Vault.

4.4 Cas d'illustration

A présent nous allons appliquer les étapes de modélisation du Data Vault en partant de l'exemple d'illustration présenté en début du document. Pour rappel, il s'agit d'une transaction concernant la livraison de courses à domicile effectuée par la Plateforme Gorillas (voir l'exemple à la section 1.3 : JDD1).

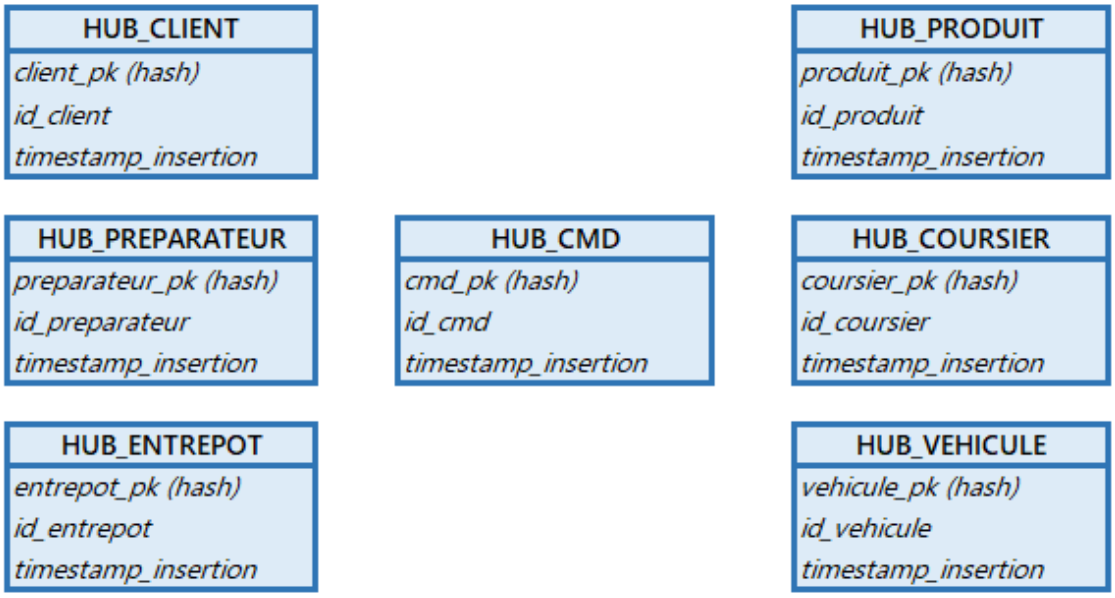
4.4.1 Modélisation des hubs

La première étape de la modélisation du Data Vault est l'identification de tous les objets métiers relatifs à un processus métier donné. C'est à partir de ces objets que nous modéliserons les hubs.

Après analyse de la transaction présentée dans le JDD1 et surtout plusieurs échanges (fictifs bien entendu) avec les responsables métiers de la plateforme Gorillas, nous avons retenu sept objets métiers. Ces objets métiers ainsi que leur(s) clé (s) fonctionnelle(s) sont matérialisés par les entités suivantes: COMMANDE (*num_cmd*), CLIENT (*id_client*), PRODUIT (*id_produit*), PREPARATEUR (*id_preparateur*), ENTREPOT (*id_entrepot*), COURSIER (*id_coursiers*) et VEHICULE (*id_vehicule*).

En se basant sur ces informations, nous pouvons modéliser les hubs suivant les structures présentées sur la figure 7 ci-dessous.

Figure 7 : Modélisation des hubs



Remarquons que chaque hub contient d'une part la clé fonctionnelle naturelle (valeur en claire) et la clé primaire dont la valeur est hashée. Par exemple, pour le hub client, la clé fonctionnelle naturelle est *id_client* mais la clé primaire est *client_pk*. Cette clé est obtenue en appliquant une fonction de hashage sur *id_client*. La génération d'une clé primaire est très important car elle permet de réduire la clé à une seule colonne dans le cas où l'objet métier serait identifié par plusieurs clés fonctionnelles naturelles.

Chaque hub contient également une colonne *timesatmp_insertion*. Il s'agit d'une métadonnée qui sert à indiquer l'instant de chargement de chaque ligne de clé dans le hub.

Une pratique courante est d'ajouter une colonne *source* pour pouvoir capter le système source de la donnée. Toutefois, l'ajout d'une métadonnée *source* dans le hub n'est pas toujours utile et n'apporte pas toujours une information pertinente. En effet, lorsqu'une clé est insérée pour la première fois dans un hub, c'est le système source qui a fourni cette ligne de donnée qui sera renseigné dans le champ *source*. Ensuite, lorsqu'un autre système source se présente avec les mêmes valeurs de clés, les informations déjà existantes ne seront pas mises à jour et aucune nouvelle ligne ne sera insérée. Car les clés des objets métiers ne sont jamais dupliquées dans un hub. C'est pourquoi, dans un Data Vault multi-sources, lorsqu'une ligne de clés est identifiée pour la première

fois dans un hub, c'est le système source qui fournit la ligne qui sera renseigné dans le champ source. Les autres systèmes sources qui présenteront la même clé ne seront pas pris en compte. L'information contenue dans le champ source dans un hub est donc une information partielle. D'ailleurs, il est fortement déconseillé d'inclure la colonne source dans toutes les requêtes impliquant les hubs et les links. La colonne source est pertinente uniquement pour les satellites. Car contrairement aux hubs et links, les satellites sont des informations contextuelles. Donc, en définitive, il n'est pas pertinent de générer un champ source ni dans les hubs, ni dans les links.

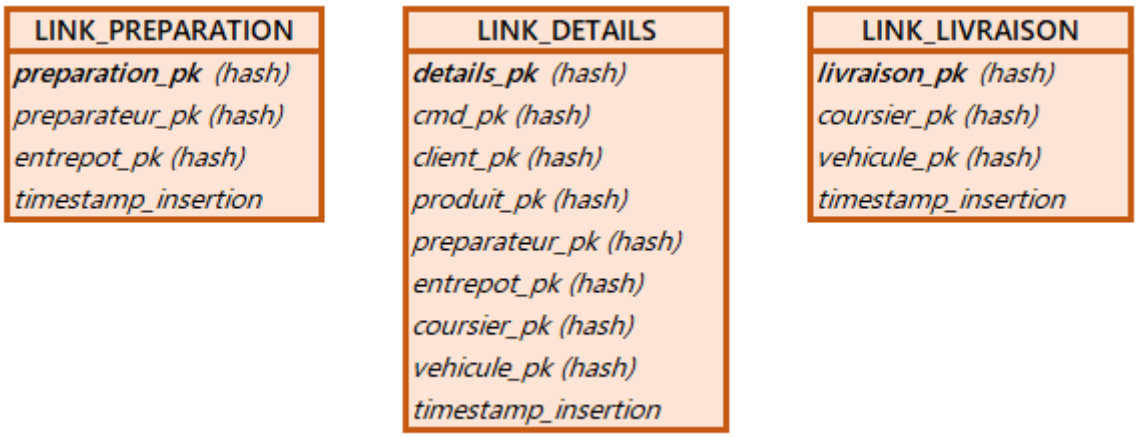
Remarquons par ailleurs qu'une bonne pratique est souvent d'ajouter à chacun des hubs une surrogate key (sk). La sk est une clé technique interne à la table qui sert de clé de substitution à la clé primaire (pk). Elle est souvent utilisée comme un incrément sur chaque ligne du hub. Mais comme il s'agit ici d'un cas très simplifié, nous avons jugé nécessaire de ne pas rajouter de sk. L'utilisation des sk n'aurait pas été très pertinente. Car, d'une part, les clés primaires des hubs sont très simples ; et d'autre part le modèle de données dans son ensemble est aussi très simple.

4.4.2 Modélisation des links

Les hubs étant dores et déjà modélisés, l'objectif maintenant est de modéliser la relation entre ces hubs à travers des tables de links. Commençons par faire remarquer que dans la phase initiale d'un modèle Data Vault, il n'est pas obligatoire de chercher à modéliser tous les links d'un coup. Le Data vault étant un modèle dynamique, de nouveaux links peuvent y être ajoutés à tout moment et lorsque les besoins métiers l'exigent. Aussi, il n'est pas nécessaire de modéliser un link s'il ne sert pas de point de raccordement vers un satellite identifié à l'avance. En clair, construire un link pour lequel il n'y a pas encore de satellite identifié n'a aucun intérêt.

Pour illustrer la modélisation des links à partir de notre cas d'illustration, nous retenons trois links. Les structures de ces trois links sont présentées sur la figure 8 ci-dessous.

Figure 8 : Modélisation des links



Compte tenu de la nature de notre cas d'illustration, le premier link que nous avons modélisé est LINK_DETAILS. En effet, nous envisageons plus tard de mettre en place un satellite qui fournit l'ensemble détails relatifs à chaque ligne de commandes. Dans cette éventualité il nous faut positionner un link qui relie l'ensemble des objets métiers qui entrent en ligne de compte pour fournir ces détails. Pour avoir tous les détails relatifs à une commande, nous avons besoin de savoir le client qui a passé la commande, le produit qui a été commandé, l'agent qui a préparé la commande et à partir de quel entrepôt, le coursier qui a livré la commande et le véhicule qu'il a utilisé. Nous avons surtout besoin de savoir le numéro de la commande, la quantité commandée, le montant total de la commande, etc. Il va donc de soi que pour construire un tel satellite, il faut mettre en jeu tous les hubs du modèle (voir plus haut les hubs modélisés). Mais nous savons déjà que lorsqu'un satellite met en relation plusieurs hubs, cette relation doit nécessairement être spécifiée sous forme d'un link. C'est pourquoi, nous choisissons de construire le LINK_DETAILS pour servir de point de raccordement entre tous les hubs.

Pour ce qui concerne la structure de LINK_DETAILS, on remarque d'abord la présence d'une clé primaire nommée *details_pk*. Pour rappel, la clé primaire d'un link est obtenue en hashant la valeur concaténée de l'ensemble des clés fonctionnelles naturelles de tous les hubs qu'il met en relation. On peut aussi remarquer qu'en plus la clé primaire (*details_pk*), le LINK_DETAILS contient aussi plusieurs autres clés (clés étrangères) qui représentent en fait les clés primaires de chacun des hubs mis en jeu. La présence des

clés étrangères dans un link permet de faire une jointure avec chacun des hubs et récupérer les valeurs en clair des clés fonctionnelles naturelles.

En plus de la clé primaire, *details_cmd_pk* et les clés étrangères (clés primaires des hubs), le link ne contient aucun autre attribut. On peut néanmoins rajouter des informations additionnelles qui ne sont pas des attributs à proprement parler. Par exemple, on ajoute une colonne de date de chargement (*timestamp_insertion*). On pouvait également ajouter une surrogate key ou une colonne indiquant le système source pour le lien ajouté. Mais comme nous l'avons déjà évoqué pour le cas des hubs, ajouter le système source dans un link n'apporte aucune information pertinente.

En plus du link LINK_DETAILS, nous avons aussi jugé utile de modéliser un LINK_PREPARATION et un LINK_LIVRAISON. Ces deux links n'ont pas forcément une grande pertinence métier, mais nous décidons quand même de les présenter notamment pour des considérations pédagogiques.

L'entité LINK_PREPARATION met en relation le hub PREPARATEUR et le hub ENTREPOT. La relation entre ces deux hubs traduit une forme d'affectation. Elle permet par exemple de savoir quel préparateur est affecté à quel entrepôt. La clé primaire du LINK_PREPARATION est *preparation_pk*. Cette clé est obtenue en hashant la valeur concaténée des clés fonctionnelles naturelles du hub PREPARATEUR et du hub ENTREPOT à savoir *id_preparateur* et *id_entrepot*. Puisque le LINK_PREPARATION relie deux hubs, il doit contenir les clés primaires de ces deux hubs. C'est pourquoi, on y retrouve les clés deux clés primaires *preparateur_pk* et *entrepot_pk* qui sont respectivement la clé primaire du hub PREPARATEUR et du hub ENTREPOT.

Pour ce qui concerne le LINK_LIVRAISON, il met en relation deux hubs : COURSIER et VEHICULE. Le principe de modélisation de ce link est le même que celui de LINK_PREPARATION. En effet, à travers ce link, nous souhaitons mettre en relation les coursiers et l'ensemble des véhicules mis à leur disposition pour effectuer les livraisons de commande. Une fois de plus, ce link ne traduit rien d'autre qu'une forme d'affectation de ressource. Il permet par exemple de savoir que tel véhicule a été une fois utilisé par tel coursier. Mais il ne donne aucun contexte. Ce rôle sera dévolu aux satellites plus tard. La clé primaire du LINK_LIVRAISON est *livraison_pk*. Cette clé est obtenue en hashant la valeur concaténée des clés fonctionnelles naturelles du hub COURSIER et du hub VEHICULE à savoir *id_coursier* et *id_vehicule*. Le LINK_LIVRAISON

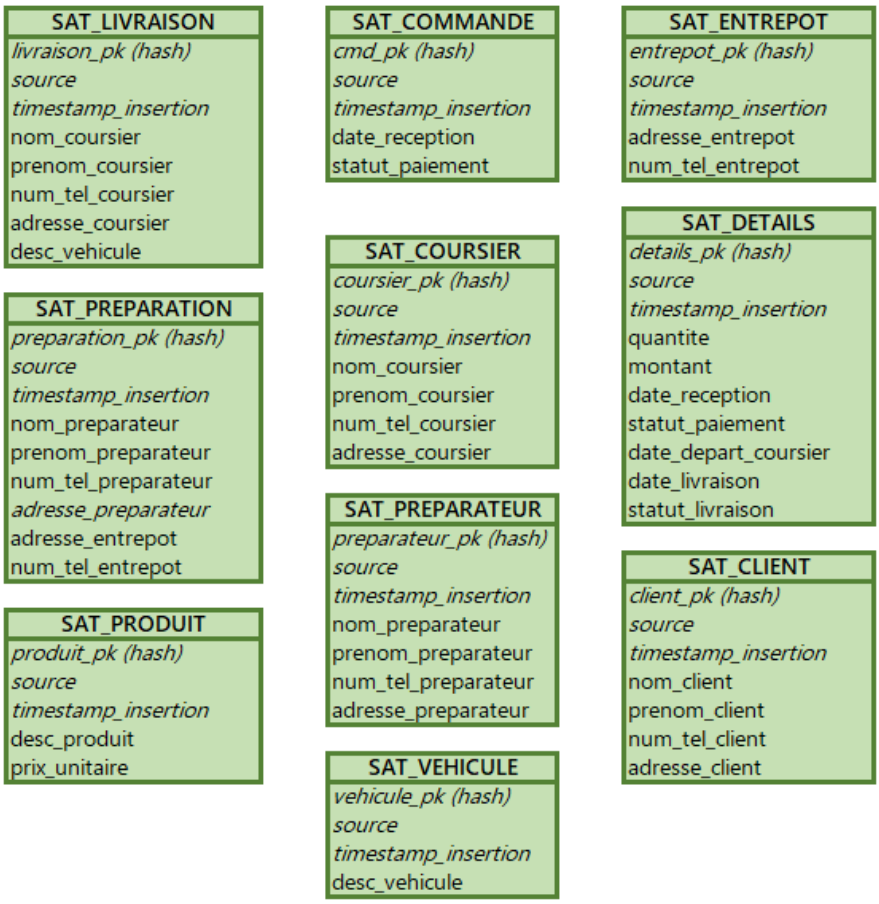
contient deux clés étrangères qui sont en fait les clés primaires des deux hubs reliés. Il s'agit notamment de *coursier_pk* pour le hub COURSIER et *vehicule_pk* pour le hub VEHICULE.

4.4.3 Modélisation des satellites

Les hubs et les links de notre modèle étant déjà définis, la dernière étape de la modélisation consiste à modéliser les satellites afin de pouvoir fournir les données contextuelles de la transaction étudiée. Comme on peut s'en rendre compte, jusque-là nous n'avons fait appel à aucun attribut contextuel lors de la modélisation des hubs et des links. Les hubs et les links sont construits uniquement à partir des clés fonctionnelles qui représentent les objets métiers et leurs relations. A présent, nous allons utiliser les attributs contextuels des objets métiers pour modéliser le satellite.

Le choix des satellites à modéliser est basé sur un principe simple. Il doit y avoir au moins un satellite pour chaque hub et chaque link constituant le modèle. Bien entendu, dans le modèle Data Vault, on peut toujours concevoir plusieurs satellites pour un même hub ou pour un même link. Suivant le principe énoncé ci-dessus, nous devons avoir dix satellites au minimum. Les dix satellites que nous avons conçus sont présentés sur la figure 9 ci-dessous.

Figure 9 : Modélisation des satellites



En observant la structure de chacun des satellites présentés sur cette figure nous pouvons savoir sans grande difficulté à quel hub/link il est relié. Pour cela, il suffit de comparer les clés primaires. En effet, la clé primaire d'un satellite est toujours la même que celui du hub ou du link auquel il est relié. Par exemple la clé primaire de l'entité SAT_PRODUIT (*produit_pk)* est le même que hub PRODUIT. Donc le hub PRODUIT est bien le hub de SAT_PRODUIT. Rappelons une fois de plus qu'un hub peut avoir plusieurs satellites mais un satellite ne peut avoir plusieurs hubs. Sinon, la relation doit passer par un link. De même qu'on a un SAT_PRODUIT pour le hub PRODUIT, nous avons SAT_CLIENT pour le hub CLIENT. La clé primaire qui lie ces deux entités est *client_pk*. Les satellites SAT_ENTREPOT, SAT_PREPARATEUR, SAT_COURSIER, SAT_VEHICULE, SAT_COMMANDE sont les autres satellites qui sont directement reliés à un hub. Les satellites qui sont reliés à un link sont SAT_DETAILS, SAT_PREPARATION et SAT_LIVRAISON. Comme pour les hubs, la clé primaire d'un link correspond toujours à la clé primaire du satellite auquel il est relié. On peut remarquer par exemple que la

clé primaire de SAT_DETAILS est le même que celui LINK_DETAILS (*details_pk*). De même, la clé primaire de SAT_PREPARATION est le même que celui LINK_PREPARATION (*preparatif_pk*). Il en est de même pour SAT_LIVRAISON et LINK_LIVRAISON qui sont deux objets reliés par la clé primaire *livraison_pk*.

Comme nous l'avons déjà rappelé à plusieurs reprises, le rôle d'un satellite est d'exposer les attributs descriptifs d'un objet métier. Pour cette raison, les informations contenues dans un satellite sont contextuelles, c'est-à-dire qu'elles sont susceptibles de changer cours du temps. Mais en plus des informations contextuelles, les satellites peuvent contenir aussi des métadonnées. Il s'agit notamment de la date de chargement de chaque ligne de données (*timestamp_insertion*) mais aussi le système source ayant fourni la donnée. Contrairement aux hubs et links où il est moins pertinent de renseigner le système source (pour des raisons que nous avons déjà plus haut), pour les satellites il est extrêmement important sinon capital de renseigner le système source. Etant donné que plusieurs système sources peuvent alimenter un même satellite, renseigner le système source permet d'avoir une traçabilité des données. A tout moment, on peut reconstituer l'historique complète des données renvoyées par une source donnée. Pour cette raison, une colonne supplémentaire doit être rajoutée dans la structure de chaque satellite afin de capter le système source fournisseur de la donnée.

Il est aussi possible d'ajouter une surrogate key (sk) dans un satellite pour servir de clé substitution à la clé primaire dont la complexité peut causer des problèmes de performance dans certaines situations. Dans tous les satellites que nous avons modélisés dans ce travail, nous avons ajouté une colonne *source* pour pouvoir renseigner le système source mais aussi une colonne *timestamp_insertion* pour capturer la date de chargement. Mais nous avons préféré ne pas ajouter de clé de substitution (surrogate key) car une telle clé n'apporte aucune amélioration significative dans la performance du modèle.

Date de validité fonctionnelle

La notion de validité fonctionnelle d'une donnée stockée dans un satellite est une notion importante car elle est en rapport direct avec les besoins métiers. De nombreuses modélisations des satellites ne prennent pas en compte cette notion et se limitent à renseigner une date de chargement des données (voir par exemple la

colonne *timestamp_insertion*). Cette approche peut être suffisante lorsque le Data Warehouse a été conçu dans une perspective organisationnelle plus générale sans une orientation métier précise. Cependant lorsqu'un Data Warehouse est conçu pour une finalité métier bien défini, la notion de validité fonctionnelle devient centrale lors de la modélisation des satellites.

La vocation des satellites étant de stocker l'historique complète de toutes les versions des données issues des systèmes sources, déterminer la validité de telle ou telle version de données face à un besoin métier spécifié devient obligatoire. Par exemple, lorsqu'un besoin métier est centré sur les reportings de type mensuel, trimestriel ou annuel, seules les versions les plus récentes entrent en ligne de compte. Il est rare de faire appel à toute l'historique des données stockées dans le Data Warehouse. Néanmoins, il arrive qu'un reporting métier fasse appel à des données sur une période spécifique du temps. Tout cela soulève la question de la validité des données dans le temps pour un besoin métier donné. C'est pourquoi, il est important de modéliser les satellites en prenant en compte la notion de validité temporelle d'une donnée.

Dans la structure d'un satellite, il est souvent important d'ajouter une colonne date qui détermine la période de validité d'une ligne de donnée. Cette colonne est qualifiée de date fonctionnelle. Tout comme les clés fonctionnelles, la date fonctionnelle est une information qui est récupérée auprès des acteurs métiers lors de la phase de modélisation. Les dates de validité fonctionnelles les plus couramment utilisées sont la date de reporting, la date de calcul, la date de publication. En prenant par exemple le cas de la date de reporting, il est fréquent de voir que les métiers poussent les données vers le Data Warehouse à l'issue de chaque cycle de reporting. Les reportings sont souvent réalisés suivant un rythme mensuel, trimestriel ou même mensuel. Etant donné que les données que toutes les versions des données sont empilées, le Data Warehouse garde l'historique des données issues de tous les reportings. Dès lors, il devient nécessaire de rajouter une colonne qui permet d'identifier de tel ou tel reporting quel que soit la date à laquelle les données ont été chargées.

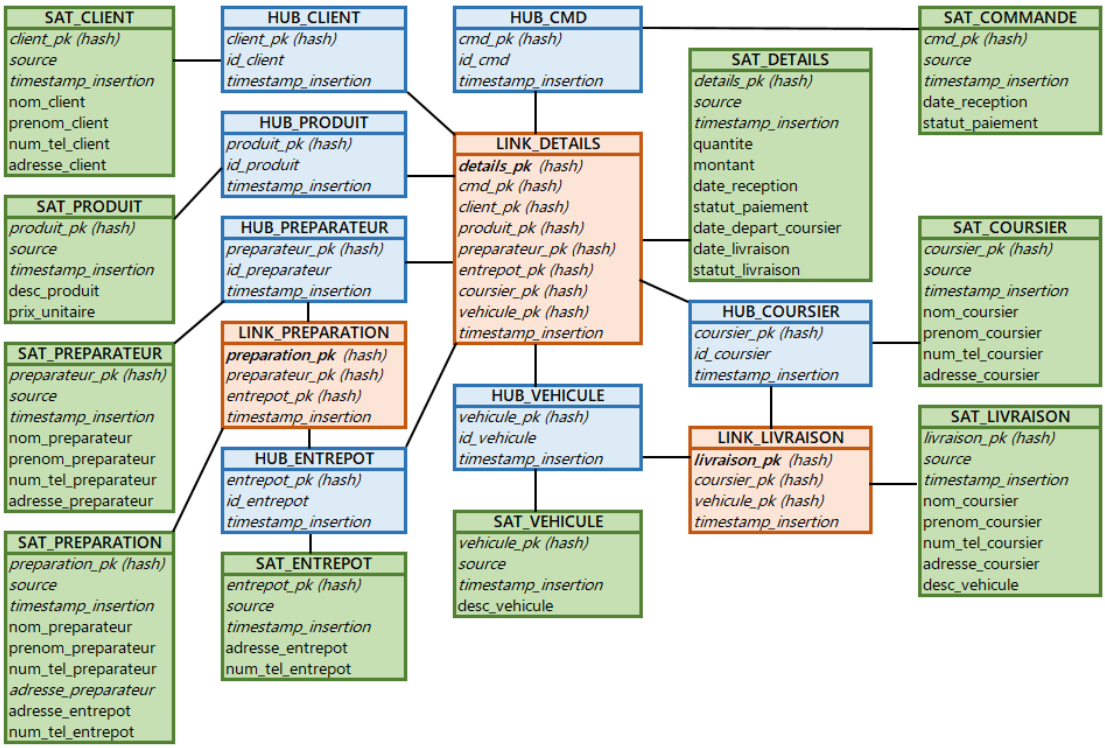
En somme lors de la modélisation d'un satellite, il est fortement conseillé de rajouter une colonne permettant de capter la date de validité de chaque ligne. Cette colonne peut être nommée ***dt_validite***. Elle est souvent calculée à partir d'une colonne préexistante dans les tables sources. Mais elle peut aussi être calculée suivant une règle

métier définie à l'avance. Par exemple, pour toutes les données chargées au cours d'un mois donné, la colonne ***dt_validite*** peut être fixée au premier jour ou au dernier jour ouvré du mois. Elle peut être fixée aussi au premier jour ou au dernier du trimestre courant, etc.. De toutes les manières, le choix de la date fonctionnelle ou de son mode de calcul doit toujours être fait de commun accord avec les experts métiers.

4.4.4 Vue d'ensemble du modèle

Les hubs, les links et les sats étant modélisés, voici ci-dessous le modèle Data Vault pouvant être tiré de notre exemple d'illustration.

Figure 10 : Vue d'ensemble du modèle Data Vault



4.4.5 Implémentation du modèle

Les codes sources de l'implémentation de ce modèle en langage pyspark est disponible sur ce repo github

4.5 Exposition des données: Business Vault et Data Mart

Tout modèle conçu suivant un MCD (Modèle Conceptuel des Données) et implémenté via un MPD (Modèle Physique des Données) a vocation être mis à la disposition des

utilisateurs en vue de son exploitation. Le Data Vault ne fait pas exception à cette règle. Cependant, compte tenu de son architecture fragmentée (hub, link, sat), le requêtage des données s'avère parfois très ardu. Par exemple un satellite impliquant un link complexe peut voir ses clés fonctionnelles dispatchées dans plusieurs hubs. Dans une telle situation, vouloir reconstituer l'ensemble des informations nécessite d'élaborer une requête à jointures multiples. Parfois, pour la reconstitution des informations impliquant plusieurs satellites, il faut parcourir l'ensemble du modèle. Ce qui ne représente pas une situation idéale pour les utilisateurs métiers. C'est pour faire face à ces difficultés d'accès aux données sur la couche basse du modèle (encore appelée *Raw Vault*) que des couches supplémentaires doivent être envisagées. Les solutions couramment utilisées sont soit la mise en place d'un *Business Vault* ou la mise en place d'un *Data Mart*.

4.5.1 Business Vault

Le Data Vault est construit sur deux couches distinctes : le Raw Vault et le Business Vault. Le Raw Vault représente la couche basse du Data Vault. C'est l'espace de stockage des données brutes telles qu'elles sont reçues des systèmes sources. Le Raw Vault est formé par les hubs, links et sats modélisés tels que nous venons de le voir. Le Business Vault, quant à lui, est une couche supplémentaire mise en place en vue de simplifier l'accès aux données par les utilisateurs et la création de rapports.

L'architecture du Business Vault est constituée par des objets spécifiques que sont notamment les vues (simples ou matérialisées) et quelques tables spécialisées : satellites adaptés, tables Point-In-Time, tables Bridge, etc..(voir les détails de ces objets plus bas).

D'une manière générale, le Business Vault est situé sur le même schéma (database) que le Raw Vault. Il n'est pas nécessaire de mettre en place un schéma dédié au Business Vault. Celui-ci doit faire partie intégrante de l'architecture du Data Warehouse. Les objets du Business Vault rafraichis à chaque fois que de nouvelles données sont chargées dans le Raw Vault.

4.5.1.1 Tables et vues matérialisées

Les objets du Business Vault sont habituellement construits à partir d'un certain nombre de requêtes pré-formulées qui implémentent des règles métiers pour corriger, harmoniser et organiser les données selon les besoins exprimés. Les requêtes pré-formulées s'avèrent très utilises notamment dans les cas où l'extraction des données nécessite une jointure entre plusieurs objets du Raw Vault.

Une approche couramment utilisée pour construire les objets du business vault est d'utiliser des vues simples ou vues matérialisées pour extraire les données. Toutefois, lorsque la requête d'extraction devient plus complexe et exige l'application de plusieurs règles métiers, il est préférable d'utiliser un traitement batch spécifique afin d'extraire les données souhaitées et les stocker dans des tables physiques situées sur le schéma aux côtés des objets du Raw Vault. On parle alors de satellite calculé ou *satellite adapté*.

4.5.1.2 Table Point-In-Time (PIT)

Depuis le Data Vault 2.0, deux types de tables spécialisées ont été rajoutée à l'architecture du Business Vault pour améliorer la performance des requêtes d'extraction lancées sur les objets du Raw Vault. Il s'agit notamment des table Point-In-Time (PIT) et des tables Bridge (BT).

Une table PIT est une utilisée pour accélérer une requête d'extraction impliquant un hub et plusieurs satellites. En effet un hub peut avoir plusieurs Sats et cela pour plusieurs raisons. Plusieurs systèmes sources peuvent alimenter un même hub alors que chaque source aliment son propre satellite à cause du fait les sources contiennent des attributs différents. Les données provenant d'un même système source peuvent servir à alimenter plusieurs satellites suite à un regroupement des attributs. Par exemple, pour un client on peut distinguer les attributs de contacts, attributs biographiques, etc. Chaque groupe d'attributs peut alimenter un satellite dédié. De plus, chaque satellite peut évoluer à son propre rythme. Par exemple, certains satellites sont alimentés quotidiennement alors que d'autres le sont à un rythme hebdomadaire ou mensuel. Dans ce genre de situation, lancer une requête d'extraction impliquant plusieurs satellites reliés à ce hub peut s'avérer très couteuse. Mais la mise en place d'un table PIT peut être une solution permettant d'alléger ces coûts.

La table PIT est une table qui fait la correspondance des dates de validité fonctionnelle[2] entre tous les satellites pour une ligne de clé du hub. Pour donner un cas concret d'une table PIT, examinons le tableau ci-dessous.

pk_hub	dt_calcul_PIT	dt_validite_sat1	dt_validite_sat2	dt_validite_sat3
20874e9418	2021-10-30	2021-10-29	2021-10-01	2021-10-15
20874e9418	2021-11-25	2021-10-29	2021-11-01	2021-10-30
20874e9418	2022-01-10	2021-12-31	2021-12-15	2021-12-15
20874e9418	2022-03-05	2022-02-15	2022-01-13	2022-02-01

Ce tableau est une extraction des données à partir d'une table PIT pour une valeur de clé. Cette table PIT fait la correspondance de la date de validité fonctionnelle des données pour trois satellites sat1, sat2 et sat2. Pour rappel, la date de validité fonctionnelle (*dt_validite*) est une colonne renseignée dans chaque satellite pour indiquer la période sur laquelle les données chargées sont valides d'un point de vue métier. A chaque rechargement du satellite suite à l'arrivée de nouvelles données, la colonne *dt_validite* est mise à jour. Ainsi, pour chaque ligne de données dans le satellite une *dt_validite* est renseignée pour chaque version de donnée. Très souvent la valeur de *dt_validite* est calculée à partir d'une colonne déjà existante dans les données sources ou suivante une règle de gestion fournie par le métier. Mais puisque le Raw Vault empile toutes les versions dans les satellites, il arrive parfois que deux lignes de données pour une même valeur de clé aient la même date de validité. Dans cette situation, c'est la ligne qui a la plus grande valeur de *timestamp_insertion* qui doit être retenue lorsqu'on souhaite extraire les données les plus récentes.

Pour revenir à la structure de la table PIT ci-dessous, on remarque tout d'abord qu'une table PIT contient une colonne qui fournit les valeurs de clé venant de hub (*pk_hub*). Cette colonne récence toutes les valeurs de la clé primaire (pk) du hub et à chaque lancement du traitement d'alimentation de la table PIT, une nouvelle ligne est insérée dans la table PIT pour chaque valeur de clé. D'abord, à chaque chargement de la table PIT, on renseigne une colonne *dt_calcul_PIT* dont la valeur sera la même pour toutes les valeurs de clé. Dans le tableau ci-dessus, on remarque que le premier chargement

[2] Voir la notion de *date de validité fonctionnelle* à la fin de la section consacrée à la modélisation des satellites.

de la table PIT a été effectué à la date du *2021-10-30*. En plus de la colonne *dt_calcul_PIT* qui est une date technique, on renseigne pour chaque satellite la dernière valeur de la date de validité fonctionnelle pour chaque valeur de clé. Par exemple, au premier chargement de la table PIT (*dt_calcul_PIT=2021-10-30*), les dates de validité pour la clé *20874e9418* sont *2021-10-29* pour le sat1, *2021-10-01* pour le sat2 et *2021-10-15* pour le sat3. D'un point de vue métier, la mise en correspondance des dates de validité pour une même valeur de clé permet de réconcilier les attributs venant de plusieurs satellites à un certain point du temps. D'où la notion de ***Point-In-Time***. Une table PIT permet, à tout moment, de faire correspondre les bonnes versions de tous les attributs à partir des satellites.

Grâce à la table PIT, l'écriture des requêtes d'extraction à partir des satellites multiples se trouve considérablement simplifiée et leur exécution plus rapide. Voici ci-dessous un exemple de requête d'extraction basée sur la table PIT présentée ci-dessous.

```
SELECT H.PK, H.ID, S1.*, S2.*,S3.*
FROM HUB H
JOIN SAT1 S1 ON (S1.PK=H.PK)
JOIN SAT2 S2 ON (S2.PK=H.PK)
JOIN SAT3 S3 ON (S3.PK=H.PK)
JOIN PIT P ON (P.PK_HUB=H.PK)
AND P.DT_CALCUL_PIT= '2022-01-10'
AND S1.DT_VALIDITE= P.DT_VALIDITE_SAT1
AND S2.DT_VALIDITE= P.DT_VALIDITE_SAT2
AND S3.DT_VALIDITE= P.DT_VALIDITE_SAT3 ;
```

La requête d'extraction ci-dessous permet remonter les données réconciliées des satellites à la date du 2022-10-10. A ce point du temps, les dates de validité fonctionnelle des trois satellites réconciliés sont 2021-12-31 pour sat1, 2021-12-15 pour sat2 et 2021-12-15 pour sat3.

En résumé, la table PIT est une table spécialisée du Business Vault dont le rôle est d'établir une correspondance entre les dernières dates de validité des satellites pour une même valeur de clé de hub. L'avantage de la mise en place d'une table PIT est de

simplifier les requêtes d'extraction impliquant plusieurs satellites contenant les données dont les dates de validité métier sont différentes.

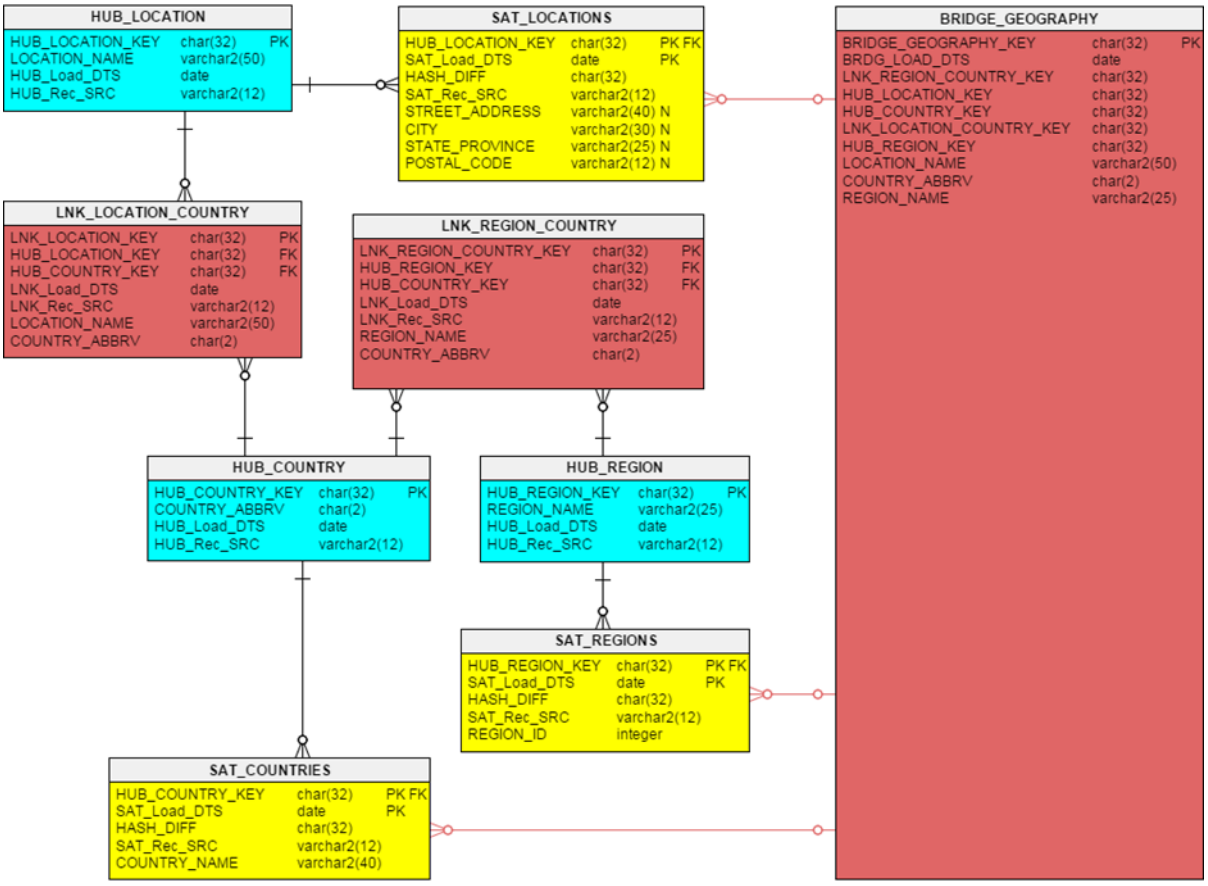
4.5.1.3 Table Bridge (BT)

A l'instar d'une table PIT qui permet de réconcilier plusieurs satellites reliés à un même hub, la table bridge est un second type de tables spécialisées du Business dont le rôle est de relier plusieurs hubs et links. Tout comme une table PIT, l'objectif d'une table bridge est de simplifier les requêtes d'extractions des objets sur le Raw Vault lorsque ces requêtes impliquent plusieurs hubs/links.

Il existe une grande similarité entre la structure des tables de links du Raw Vault et la structure des tables bridge. Rappelons que sur le Raw Vault, les tables de links servent à relier plusieurs hubs lorsque cette relation doit déboucher sur la création d'un satellite dédié. A la différence des tables links, les tables bridge servent à lier plusieurs hubs même si ces hubs n'ont pas de relations directes dans l'architecture du Raw Vault. Les tables bridge servent également à relier plusieurs tables links même si celles-ci n'ont pas de relations directes. De même une table bridge permet de relier plusieurs hubs et plusieurs links pris ensemble.

L'utilisation des tables bridges permet de raccourcir le chemin d'accès aux données lorsque ces données sont dispersées à différents endroits du modèle. La figure ci-dessous fournit un exemple d'illustration de la table bridge.

Figure 11 : Table Bridge



Source : Graziano (2015)

La table bridge permet d'accéder aux données dans les satellites sans avoir à passer par les hubs et les links associés. Les tables bridges rassemblent au même endroit les clés primaires des hubs et des links mais aussi les clés fonctionnelles naturelles des hubs. Grâce à ces informations, on peut directement faire les jointures avec les satellites concernés pour remonter les données sans avoir à passer par les hubs et les links. Cela permet ainsi de limiter le nombre de jointure entre tables, améliorant par là même la performance de l'exécution des requêtes d'extraction.

Par exemple, en partant du cas présenté sur la figure 11, on peut accéder aux données des satellites en utilisant la requête ci-dessous.

```
SELECT B.LOCATION_NAME,S1.STREET_ADDRESS
,S1.CITY, S1.STATE_PROVINCE,S1.POSTAL_CODE
,S2.COUNTRY_NAME, B.REGION_NAME, S3.REGION_ID
FROM BRIDGE_GEOGRAPHY B
JOIN SAT_LOCATIONS S1
ON B.HUB_LOCATION_KEY = S1.HUB_LOCATION_KEY
JOIN SAT_COUNTRIES S2
ON B.HUB_COUNTRY_KEY = S2.HUB_COUNTRY_KEY
JOIN SAT_REGIONS S3
ON B.HUB_REGION_KEY = S3.HUB_REGION_KEY ;
```

Source : Graziano (2015)

Remarquons que depuis le Data Vault 2.0, beaucoup de tables de links peuvent être désormais modélisés de sorte à jouer le rôle de tables bridges. Pour positionner une table link comme une table bridge, il suffit de la dénormaliser en ramenant les clés fonctionnelles naturelles de tous les hubs reliés au niveau du link. Pour rappel, les links classiques sont des tables normalisées qui ne contiennent que la clé primaire (hashée) et des clés étrangères (hashées) correspondant aux clés primaires des hubs mis en relation. Pour dénormaliser cette table, il suffit de ramener au niveau du link, les clés fonctionnelles (valeurs en clair) qui, habituellement, ne sont renseignées que dans les hubs. Un link dénormalisé permet alors d'accéder aux données en faisant une jointure simple le satellite sans passer par le hub. Ainsi, la dénormalisation d'un link permet d'obtenir une table bridge.

4.5.2 Data Mart

L'utilisation de Data Marts constitue une autre voie pour exposer les données stockées dans un Data Vault. Cependant l'utilisation des Data Marts est recommandée dans les cas où les données nécessitent des traitements plus complexes (harmonisation et agrégation) mais aussi l'application de règles métiers spécifiques.

4.6 Avantages et Inconvénients du modèle Data Vault

4.6.1 Avantages

4.6.1.1 Exhaustivité des données

Les données issues de plusieurs systèmes sources sont stockées et historisées sans aucun écrasement des données déjà chargées. Chaque donnée est chargée dans son format d'origine en renseignant sa date de chargement et son système source. Ce qui permet, à tout moment, de reconstituer l'historique complète des données propre à une source. Un tel mode de stockage facilite l'audit et la traçabilité des informations historiques.

4.6.1.2 Flexibilité et adaptabilité du modèle

Le concept Data Vault offre beaucoup plus de flexibilité dans la mise en place de Data Warehouse. Grâce à sa souplesse, l'architecture du Data Vault s'adapte à tout changement majeur dans la logique métier sans avoir à refondre ou à reconcevoir tout le système.

4.6.1.3 Modèle dynamique et extensible

Le Data Vault permet d'ajouter des nouveaux objets (hubs, links, sats) et des nouvelles sources de données sans remettre en cause l'architecture déjà existante.

4.6.1.4 Rapidité de chargement des données

Le Data Vault permet un chargement parallèle des flux. Par exemple, les flux qui n'alimentent pas les mêmes objets métiers (hubs, links et sats) peuvent être lancés en parallèle. Ce qui permet de gagner en rapidité de chargement.

4.6.2 Inconvénients :

4.6.2.1 Architecture fragmentée

La flexibilité qu'offre le Data Vault est souvent obtenue au prix d'une grande fragmentation de l'architecture. En effet, le découpage des données en plusieurs objets distincts (hubs, links, sats, pits et bridges) conduit à multiplier les tables dans le Data Warehouse. Cette situation cause parfois à une perte de visibilité sur le schéma général du modèle.

4.6.2.2 Difficulté d'accès aux données via le Raw Vault

Il est très difficile pour les utilisateurs d'accéder aux données via le Raw Vault. Compte tenu de la multiplicité des objets, l'extraction des données devient plus complexe. C'est pourquoi un Business Vault est souvent prévu dans l'architecture du Data Vault afin de mettre à la disposition des utilisateurs des objets précalculés pour des besoins spécifiquement exprimés.

4.7 Recommandations

Dans cette section, nous indiquons quelques situations dans lesquelles il est recommandé d'utiliser ou de ne pas utiliser le Data Vault pour mettre en place un Data Warehouse.

4.7.1 Dans quels cas le modèle Data Vault est-il recommandé ?

4.7.1.1 Un Data Warehouse multicanal

Il est recommandé d'utiliser le modèle Data Vault lorsque les données à stocker dans le Data Warehouse proviennent de plusieurs systèmes sources.

4.7.1.2 Un environnement métier changeant et évolutif

Il est également recommandé de mettre en place un modèle Data Vault lorsque votre environnement métier est soumis à des mutations rapides et fréquentes. En effet,

l'architecture Data Vault est conçu de sorte à pouvoir ajouter des nouveaux objets ou des nouvelles sources sans impact majeur sur le système existant.

4.7.1.3 Besoin d'audit et traçabilité des données

Le modèle Data Vault est également mieux indiqué lorsqu'il s'agit de mettre en place un système d'audit et de traçabilité des données. Dans le Data Vault, toutes les lignes sont insérées en renseignant la date de chargement et mais aussi le système ayant fourni la donnée. De plus les données sont historisées sans écrasement, ni mise à jour des versions existantes. Un tel mécanisme facilite la traçabilité des données à des fins réglementaires et de gouvernance de données.

4.7.2 Dans quels cas le Data Vault n'est pas pertinent ?

4.7.2.1 Data Warehouse mono-source

Lorsque le Data Warehouse n'est alimenté que par un seul système source, la pertinence du modèle Data Vault peut être sévèrement questionnée. Dans un Data Warehouse mono-source, il convient d'abord d'examiner la volumétrie des données reçues.

4.7.2.2 Données sources stables dans le temps

Une autre situation dans laquelle le Data Vault est moins indiqué est lorsque les données sources sont statiques ou qu'elles changent très peu dans le temps. C'est le cas par exemple des données de logement, des données relatives aux infrastructures routières ou ferroviaires. Vouloir charger de telles données dans le Data Warehouse suivant le modèle Data Vault aboutirait parfois à une forte duplication des données.

4.7.2.3 Alimenter une base de reporting métier

Il est fortement déconseillé d'utiliser le modèle Data Vault pour alimenter directement une base de reporting métier. Compte tenu du niveau de la fragmentation des données entre plusieurs tables dans le Data Vault, l'accès aux données à partir d'un outil de reporting nécessite de passer par des jointures multiples. Ces jointures réduisent la performance des requêtes et ralentissement la génération des rapports.

5 RESSOURCES DOCUMENTAIRES

Je tiens ici à remercier tous les auteurs de documents techniques, d'articles, de blogs et de ressources numériques que j'ai consulté lors de la revue documentaire mais qui ne figurent pas dans le liste ci-dessous.

Bibliographie

Alrehamy, H. H. et C. Walker (2015). Personal Data Lake With Data Gravity Pull. In IEEE 5th International Conference on Big Data and Cloud Computing (BDCloud 2015), Dalian, China, pp. 160–167.

Dixon, J. (2010). Pentaho, Hadoop and Data Lakes. https ://jamesdixon.wordpress.com/2010/10/14/pentaho-hadoop-and-data-lakes/.

Graziano Kent (2015), The Business Data Vault, accessible en ligne ici

Linstedt D. (2002), Data Vault Series, The Data Administration Newsletter

Linstedt, D. (2011). Super Charge your Data Warehouse : Invaluable Data Modeling Rules to Implement Your Data Vault. CreateSpace Independent Publishing.

Miloslavskaya, N. et A. Tolstoy (2016). Big Data, Fast Data and Data Lake Concepts. Procedia Computer Science 88, 300–305.

Stein, B. et A. Morrison (2014). The enterprise data lake : Better integration and deeper analytics. Technology Forecast, 1. http ://www.pwc.com/us/en/technology-forecast/2014/cloudcomputing/assets/pdf/pwc-technology-forecast-data-lakes.pdf.

Urls utiles :

https://agiledss.com/en/blog/data-vault-whats-to-know-about-this-data-warehouse-vision

https://aginic.com/blog/modelling-with-data-vaults/

https://aptitive.com/blog/3-reasons-to-implement-a-data-vault-model-and-2-reasons-not-to/

https://ithealth.io/data-vault/

https://medium.com/@jryan999/data-vault-an-overview-27bed8a1bf9f

https://www.ladissertation.com/Sciences-et-Technologies/Sciences-Cognitives/Data-Vault-comment-se-distingue-cette-nouvelle-vision-45769.html

https://vertabelo.com/blog/data-vault-series-the-business-data-vault/