Université de Ngaoundéré
Institut Universitaire de Technologie

Base de Données non structurées (NoSQL)

Chapitre 2 Modèles de données agrégées

Pr. Dayang Paul
Maître de conférence

Année académique
2024/2025

INTRODUCTION

Un modèle de données est le modèle par lequel nous percevons et manipulons nos données. Pour les personnes qui utilisent une base de données, le modèle de données décrit comment nous interagissons avec les données.

Il se distingue du modèle de stockage, qui décrit comment la base de données stocke et manipule les données en interne.

Dans un monde idéal, nous devrions ignorer le modèle de stockage, mais dans la pratique, nous avons besoin d'en avoir au moins une idée, principalement pour obtenir des performances décentes.

INTRODUCTION

Le terme "modèle de données" désigne souvent le modèle des données spécifiques d'une application. Un développeur peut montrer un diagramme entité-relation de sa base de données et s'y référer comme à son modèle de données contenant des clients, des commandes, des produits, etc.

Cependant, dans ce cours, nous utiliserons principalement le terme "modèle de données" pour désigner le modèle par lequel la base de données organise les données, ce que l'on pourrait appeler plus formellement un métamodèle.

INTRODUCTION

Le modèle de données dominant des deux dernières décennies est le modèle de données relationnel, qui se visualise le mieux comme un ensemble de tableaux, un peu comme une page de tableur. Chaque tableau comporte des lignes, chaque ligne représentant une entité d'intérêt.

L'une des évolutions les plus évidentes de **NoSQL** est l'abandon du modèle relationnel. Chaque solution NoSQL utilise un modèle différent, que nous avons classé en quatre catégories largement utilisées dans l'écosystème NoSQL : clé-valeur, document, famille de colonnes et graphe. Parmi celles-ci, les trois premières partagent, une caractéristique commune de leurs modèles de données que nous appellerons orientation agrégée.

1- AGREGATS

Le modèle relationnel prend les informations que nous voulons stocker et les divise en tuples. Un tuple est une structure de données limitée : Il capture un ensemble de valeurs, de sorte que vous ne pouvez pas imbriquer un tuple dans un autre pour obtenir des enregistrements imbriqués, ni mettre une liste de valeurs ou de tuples dans une autre.

L'**orientation agrégée** adopte une approche différente. Elle reconnaît que, souvent, vous souhaitez opérer sur des données dans des unités dont la structure est plus complexe qu'un ensemble de tuples. Il peut être pratique de penser en termes d'enregistrement complexe qui permet d'imbriquer des listes et d'autres structures d'enregistrement.

1- AGREGATS

Nous allons voir comment les bases de données de type clé-valeur, document et colonne font toutes appel à cet enregistrement plus complexe.

Cependant, il n'existe pas de terme commun pour désigner cet enregistrement complexe; dans ce cadre ce cours, nous utilisons le terme **agrégat**. L'agrégat est un terme qui provient de la conception pilotée par le domaine [Evans].

*« L'**agrégat** est une collection d'objets connexes que nous souhaitons traiter comme une unité. »*

- La manipulation des données
- La gestion de la cohérence.

1- AGREGATS

En général, nous aimons mettre à jour les agrégats avec des opérations atomiques et communiquer avec notre stockage de données en termes d'agrégats.

Le fait de traiter les agrégats facilite grandement le fonctionnement de ces bases de données dans un cluster, puisque l'agrégat constitue une unité naturelle pour **la réplication et le sharding**.

- **La réplication** permet d'assurer la disponibilité constante du système.
- **Le sharding** est une méthode qui permet de partitionner un ensemble de données venant d'une même base de données

1.1. Exemple de relations et d'agrégats

Supposons que nous devions construire un site Web de e-commerce; nous allons vendre des articles directement aux clients sur le Web. Il faut:

- Stocker des informations sur les utilisateurs,
- Notre catalogue de produits,
- Les commandes,
- Les adresses d'expédition,
- Les adresses de facturation et les données de paiement

Nous pouvons utiliser ce scénario pour modéliser les données en utilisant un model de données relationnel ainsi que des modèles de données NoSQL et parler de leurs avantages et inconvénients.

1.1. Exemple de relations et d'agrégats

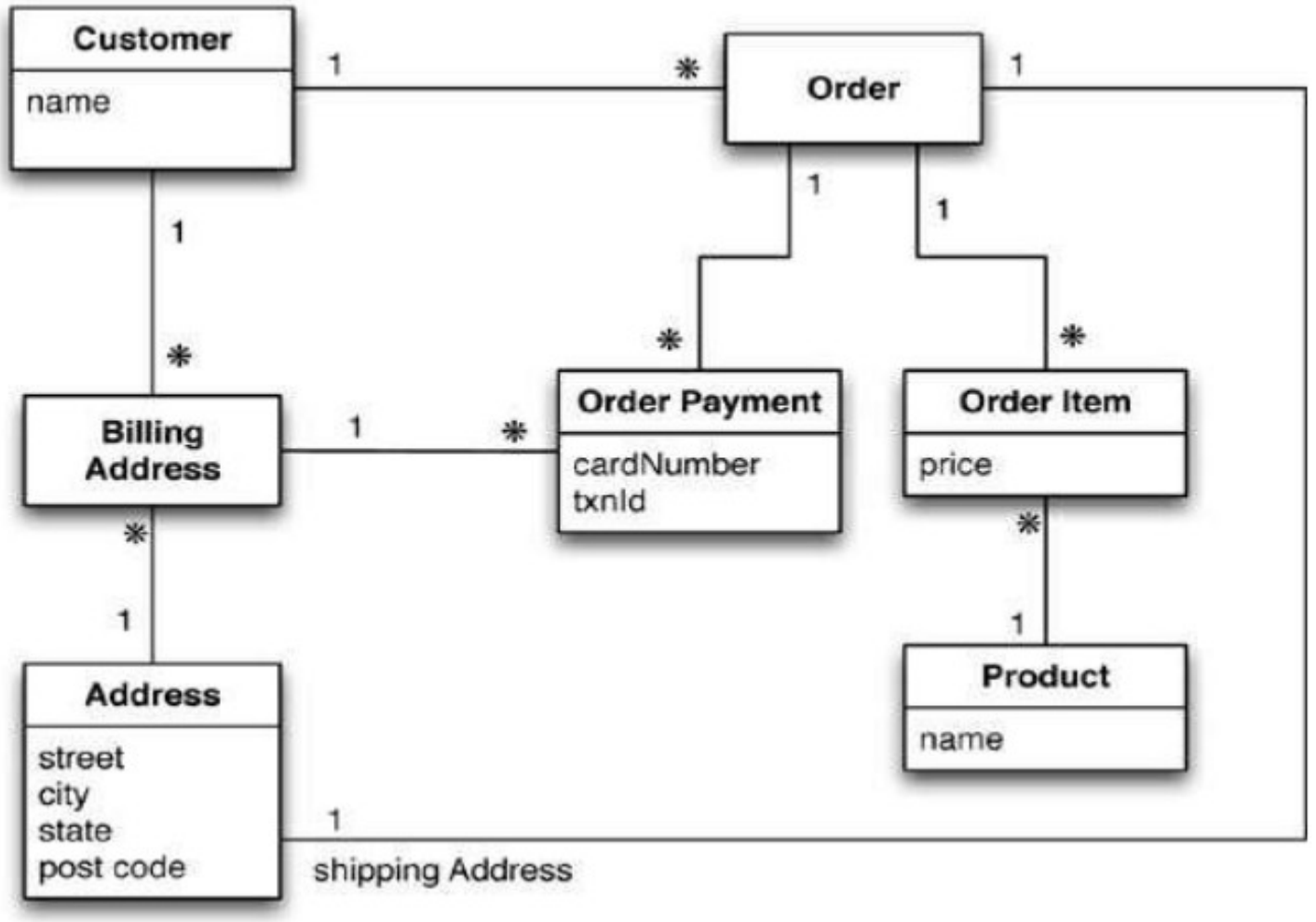

Figure : 1. Modèle de données orienté autour d'une base de données relationnelle (UML)

1.1. Exemple de relations et d'agrégats

Customer

Id	Name
1	Martin

Orders

Id	CustomerId	ShippingAddressId
99	1	77

Product

Id	Name
27	NoSQL Distilled

BillingAddress

Id	CustomerId	AddressId
55	1	77

OrderItem

Id	OrderId	ProductId	Price
100	99	27	32.45

Address

Id	City
77	Chicago

OrderPayment

Id	OrderId	CardNumber	BillingAddressId	txnId
33	99	1000-1000	55	abelif879rft

Figure 2. Données typiques utilisant un modèle de données SGBDR

1.1. Exemple de relations et d'agrégats

Voyons maintenant à quoi peut ressembler ce modèle lorsque nous pensons en termes plus agrégés;

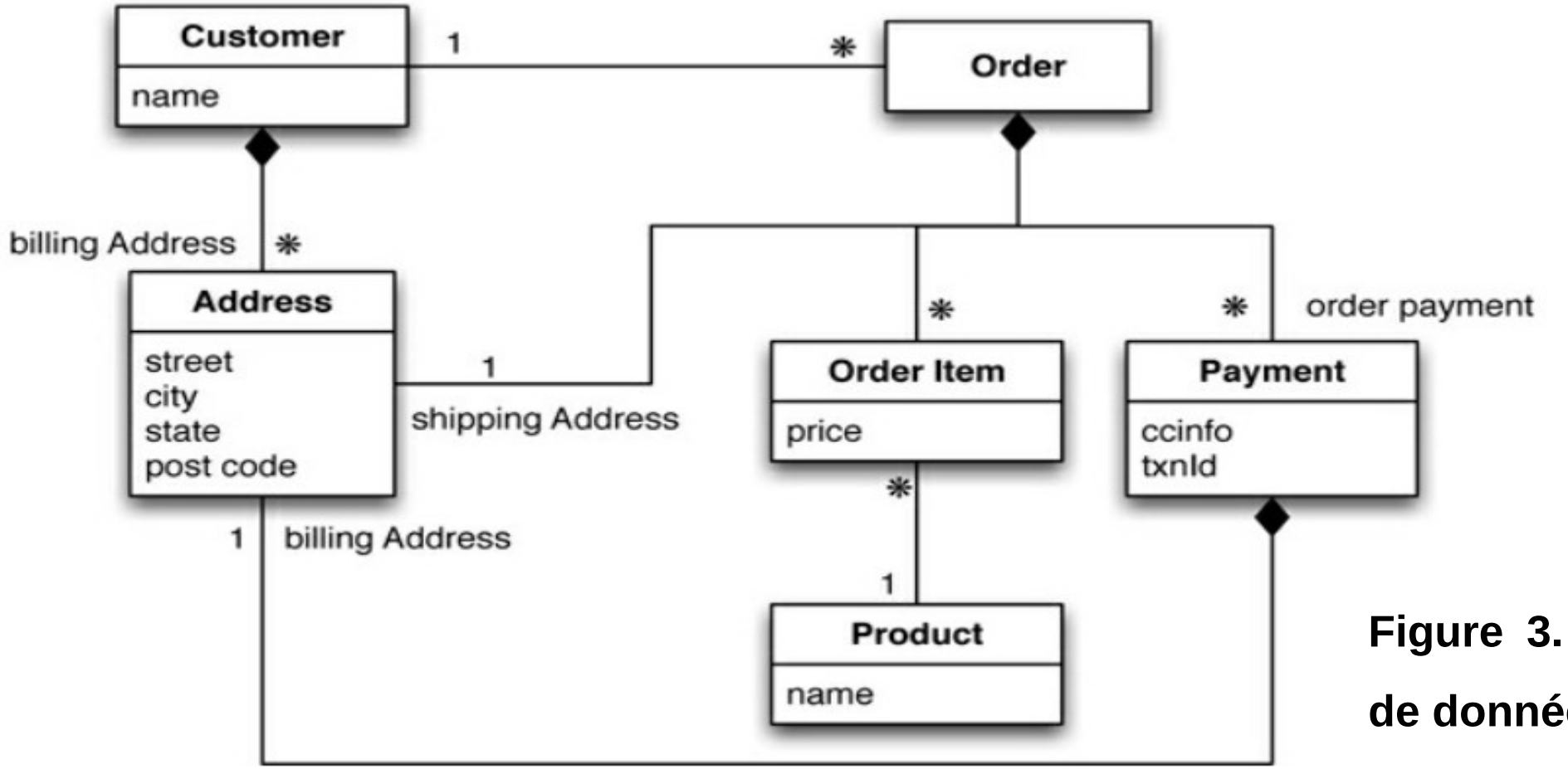

Figure 3. Un modèle de données agrégé

1.1. Exemple de relations et d'agrégats

Encore une fois, nous avons quelques données d'exemple, que nous allons montrer en format JSON car c'est un format commun représentation des données en NoSQL.

```
// in customers
{
"id":1,
"name":"Martin",
"billingAddress":[{"city":"Chicago"}]
}

// in orders
{
"id":99,
"customerId":1,
"orderItems":[
  {
  "productId":27,
  "price": 32.45,
  "productName": "NoSQL Distilled"
    }
  ],
"shippingAddress":[{"city":"Chicago"}]
"orderPayment":[
  {
    "ccinfo":"1000-1000-1000-1000",
    "txnId":"abelif879rft",
    "billingAddress": {"city": "Chicago"}
  }
  ],
}
```

1.1. Exemple de relations et d'agrégats

Dans ce modèle, nous avons deux agrégats principaux : le client et la commande. Nous avons utilisé le marqueur de composition en UML pour montrer comment les données s'intègrent dans la structure d'agrégation.

- Le client contient une liste d'adresses de facturation ;
- La commande contient une liste d'articles commandés,
- Une adresse de livraison et des paiements.
- Le paiement lui-même contient une adresse de facturation pour ce paiement.

Un seul enregistrement d'adresse logique apparaît trois fois dans les données de l'exemple, mais au lieu d'utiliser des ID, il est traité comme une valeur et copié à chaque fois. Cela correspond au domaine dans lequel nous ne voudrions pas que l'adresse d'expédition, ni l'adresse de facturation du paiement, changent.

1.1. Exemple de relations et d'agrégats

Dans une base de données relationnelle, nous ferions en sorte que les lignes d'adresse ne soient pas mises à jour dans ce cas, en créant une nouvelle ligne à la place. Avec l'agrégat, nous pouvons copier toute la structure de l'adresse dans l'agrégat si nous en avons besoin.

- Le lien entre le client et la commande ne se trouve pas dans l'un ou l'autre des agrégats, mais dans une relation entre agrégats.
- Le lien d'un élément de commande passerait dans une structure d'agrégat distincte pour les produits,

Nous avons montré le nom du produit comme faisant partie de l'élément de la commande ici.

1.1. Exemple de relations et d'agrégats

Ce type de dénormalisation est similaire aux compromis avec les bases de données relationnelles, mais il est plus courant avec les agrégats car nous voulons minimiser le nombre d'agrégats auxquels nous accédons pendant une interaction de données.

Ce qu'il faut retenir ici, ce n'est pas tant la façon particulière dont nous avons tracé la limite de l'agrégat que le fait que vous devez penser à accéder à ces données, mais plutôt l'intégrer dans votre réflexion lors du développement du modèle de données de l'application. Nous pourrions en effet tracer les limites de notre agrégat différemment, en plaçant toutes les commandes d'un client dans l'agrégat client.

1.1. Exemple de relations et d'agrégats

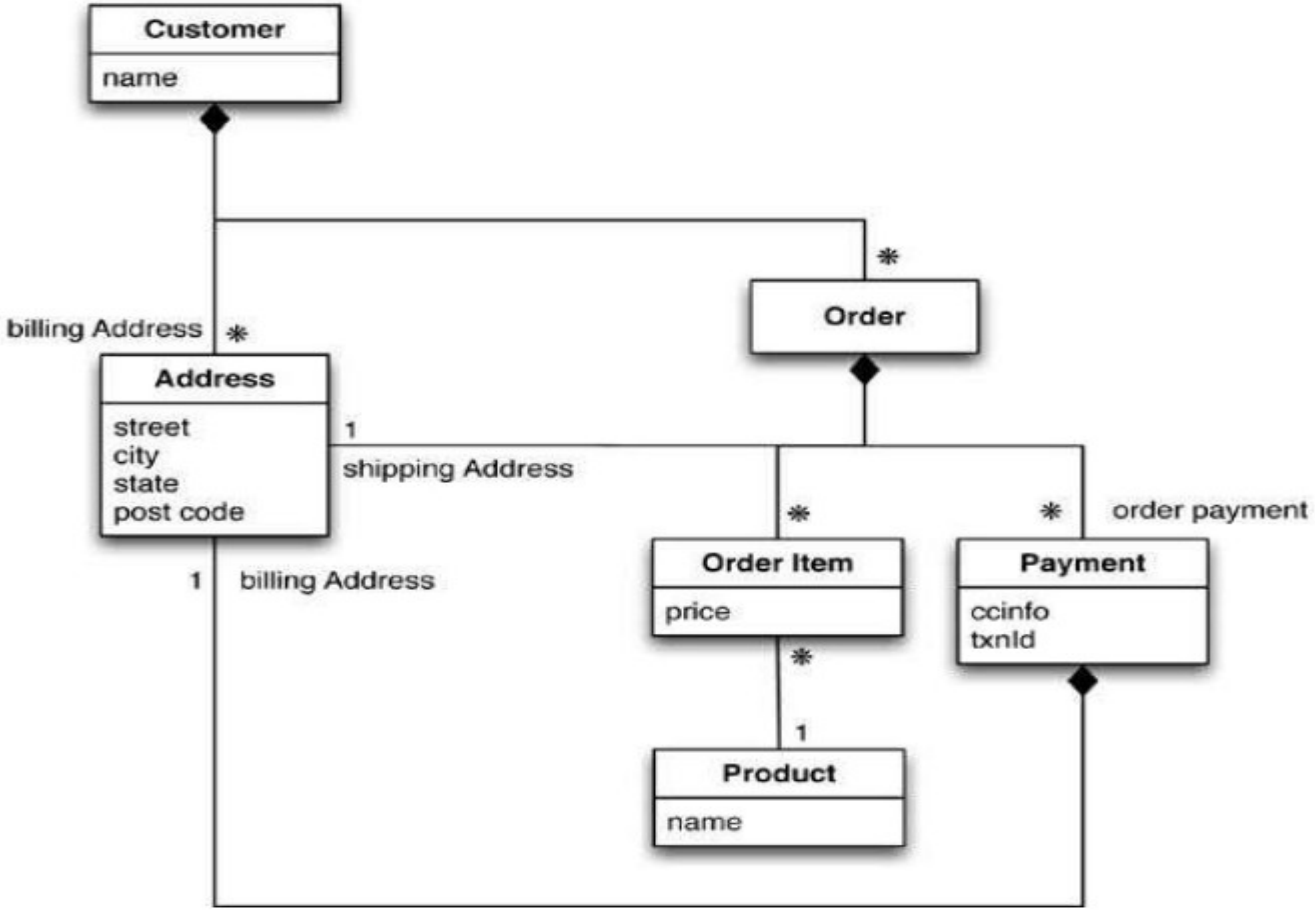

Figure 4. Embarquer tous les objets pour le client et les commandes du client en utilisant le modèle de données

1.1. Exemple de relations et d'agrégats

```
// in customers
{
"customer" :
"id" : 1,
"nom" : "Martin",
" billingAddress " : [ {"city" :"Chicago"}] ,
"orders" : [
  {
    "id":99,
    "customerid":1,
    "orderitems" : [
     {
         "productid":27,
         "price" : 32.45,
         "productName" : " NoSQL
      }
   ],
"shippingAddress" : [ {"city" : "Chicago"}]
"orderPayment" : [
 {
      "ccinfo":"1000-1000-1000-1000",
     "txnid" : "abelif879rft",
     "billingAddress" : {"city" : "Chicago"}
  } ],
}]
}
}
```

1.1. Exemple de relations et d'agrégats

Comme la plupart des choses en modélisation, il n'y a pas de réponse universelle sur la façon de tracer les limites de vos agrégats. Cela dépend entièrement de la façon dont vous avez tendance à manipuler vos données. Si vous avez tendance à accéder à un client et à toutes ses commandes en même temps, vous préférerez un seul agrégat.

Cependant, si vous avez tendance à vous concentrer sur l'accès à une seule commande à la fois, vous préférerez avoir des agrégats séparés pour chaque commande.

Naturellement, ceci est très spécifique au contexte ; certaines applications préféreront l'un ou l'autre, même au sein d'un seul système, ce qui est exactement la raison pour laquelle de nombreuses personnes préfèrent l'ignorance des agrégats.

1.2. Conséquences de l'orientation globale

Si le mappage relationnel capture raisonnablement bien les différents éléments de données et leurs relations, il le fait sans aucune notion d'entité agrégée. Dans notre langage de domaine, nous pourrions dire qu'une commande est constituée d'**éléments de commande, d'une adresse de livraison et d'un paiement**.

Cela peut être exprimé dans le modèle relationnel en termes de relations de clé étrangère, mais rien ne permet de distinguer les relations qui représentent des agrégations de celles qui n'en représentent pas.

Par conséquent, la base de données ne peut pas utiliser la connaissance de la structure des agrégats pour l'aider à stocker et à distribuer les données.

1.2. Conséquences de l'orientation globale

Diverses techniques de modélisation des données ont fourni des moyens de marquer les structures agrégées ou composites. Le problème, est que les modélisateurs fournissent rarement une sémantique sur ce qui différencie une relation d'agrégat d'une autre ; lorsqu'il y a une sémantique, elle varie.

Lorsque nous travaillons avec des bases de données orientées agrégat, nous avons une sémantique plus claire à prendre en compte en nous concentrant sur l'unité d'interaction avec le stockage des données.

Il ne s'agit cependant pas d'une propriété logique des données : Il s'agit de savoir comment les données sont utilisées par les applications - une préoccupation qui se situe souvent en dehors des limites de la modélisation des données.

1.2. Conséquences de l'orientation globale

Les bases de données relationnelles n'ont aucun concept d'agrégat dans leur modèle de données, c'est pourquoi nous les appelons **agrégat-ignorant.** Dans le monde NoSQL, les bases de données de graphes sont également ignorantes de l'agrégation. Le fait d'être ignorant en matière d'agrégats n'est pas une mauvaise chose. Il est souvent difficile de bien tracer les limites des agrégats, en particulier si les mêmes données sont utilisées dans de nombreux contextes différents.

Une commande constitue un bon agrégat lorsqu'un client passe et révise des commandes, et lorsque le détaillant traite des commandes.

1.2. Conséquences de l'orientation globale

Cependant, si un détaillant souhaite analyser ses ventes de produits sur les derniers mois, l'agrégat de commande devient un problème. Pour accéder à **l'historique des ventes** de produits, vous devrez fouiller dans chaque agrégat de la base de données.

Ainsi, une structure d'agrégat peut faciliter certaines interactions de données mais constituer un obstacle pour d'autres.

Un modèle **agrégat-ignorant** vous permet d'examiner facilement les données de différentes manières, c'est donc **le meilleur choix** lorsque vous ne disposez pas d'une structure primaire pour manipuler vos données.

1.2. Conséquences de l'orientation globale

L'argument décisif en faveur de l'**orientation agrégée** est qu'elle facilite grandement l'exécution sur un cluster, qui, comme vous vous en souvenez, est l'argument décisif pour l'essor de **NoSQL**.

Si nous fonctionnons sur un cluster, nous devons minimiser le nombre de nœuds que nous devons interroger lorsque nous collectons des données.

En incluant explicitement des agrégats, nous donnons à la base de données des informations importantes sur les éléments de données qui seront manipulés ensemble et qui devraient donc se trouver sur le même nœud.

1.2. Conséquences de l'orientation globale

Les bases de données relationnelles vous permettent de manipuler n'importe quelle combinaison de lignes de n'importe quelle table en une seule transaction. De telles transactions sont appelées **transactions ACID :** Atomic, Consistent, Isolated, and Durable. ACID est un acronyme plutôt artificiel ; le point essentiel est l'atomicité.

1.3. Modèles de données clés-valeurs et documents

Nous avons dit précédemment que les bases de données **clés valeurs et de documents** étaient fortement orientées vers **les agrégats**. Ces deux types de bases de données sont constitués d'un grand nombre d'agrégats, chaque agrégat possédant **une clé ou un identifiant** qui permet d'accéder aux données.

Les deux modèles diffèrent dans la mesure où, dans une base de données clés-valeurs, l'agrégat **est opaque** pour la base de données, c'est-à-dire qu'il s'agit d'une grosse masse de bits sans signification. En revanche, une base de données de documents est capable de voir une structure dans l'agrégat.

1.3 Modèles de données clés-valeurs et documents

L'avantage de l'opacité est que nous pouvons stocker ce que nous voulons dans l'agrégat. La base de données peut imposer une limite de taille générale, mais pour le reste, nous sommes totalement libres.

Une base de données documentaire impose des limites à ce que nous pouvons y placer, en définissant les structures et les types autorisés. En contrepartie, cependant, nous bénéficions d'une plus grande flexibilité d'accès.

Avec un model clé-valeur, nous ne pouvons accéder à un agrégat que par une recherche basée sur sa clé. Avec une base de données documentaire, nous pouvons soumettre des requêtes à la base de données en fonction des champs de l'agrégat, nous pouvons récupérer une partie de l'agrégat plutôt que la totalité, et la base de données peut créer des index basés sur le contenu de l'agrégat.

1.3. Modèles de données clés-valeurs et documents

En pratique, la frontière entre valeur-clé et document est un peu floue. Les gens placent souvent un champ d'identification dans une base de données de documents pour effectuer une recherche de type valeur-clé.

Les bases de données classées comme bases de données clés-valeurs peuvent vous permettre de structurer les données au-delà d'un simple agrégat opaque. Par exemple:

- **Riak** permet d'ajouter des métadonnées aux agrégats pour l'indexation et les liens inter-agrégats,
- **Redis** permet de décomposer l'agrégat en listes ou en ensembles.

Vous pouvez prendre en charge l'interrogation en intégrant des outils de recherche tels que **Solr**.

1.3. Modèles de données clés-valeurs et documents

Malgré ce flou, la distinction générale reste valable. Avec les bases de données **clés-valeurs**, nous nous attendons à rechercher des agrégats en utilisant **une clé**. Avec les bases de données de documents, nous nous attendons à soumettre une forme de requête basée sur la structure interne du document ; il peut s'agir d'une clé, mais il est plus probable que ce soit autre chose.

1.4. Model de la famille des colonnes

L'une des premières bases de données NoSQL influentes a été **BigTable** de Google. Son nom évoque une structure tabulaire qu'elle réalise avec des colonnes éparses et sans schéma. Comme vous le verrez plus tard, il n'est pas utile de considérer cette structure comme une table ; il s'agit plutôt d'une carte à deux niveaux.

Ce qui les différenciait, c'était la manière dont ils stockaient physiquement les données. La plupart des bases de données ont une ligne comme unité de stockage, ce qui favorise notamment les performances d'écriture.

Cependant, il existe de nombreux scénarios où les écritures sont rares, mais où vous devez souvent lire quelques colonnes de plusieurs lignes à la fois. Dans cette situation, il est préférable de stocker des groupes de colonnes pour toutes les lignes comme unité de stockage de base - c'est pourquoi ces bases de données sont appelées **column stores**

1.4. Model de la famille des colonnes

La meilleure façon d'envisager le modèle de famille de colonnes est peut-être de le considérer comme une structure d'agrégat à deux niveaux.

Dans le cas d'un stockage clé-valeur, la première clé est souvent décrite comme un identifiant de ligne, reprenant l'agrégat d'intérêt. La différence avec les structures de type famille de colonnes est que cet agrégat de ligne est lui même formé d'une carte de valeurs plus détaillées.

Ces valeurs de second niveau sont appelées colonnes. Outre l'accès à la ligne dans son ensemble, les opérations permettent également de sélectionner une colonne particulière.

1.4. Model de la famille des colonnes

Pour obtenir le nom d'un client particulier. Vous pouvez effectuer une opération du type get ('1234', 'nom').

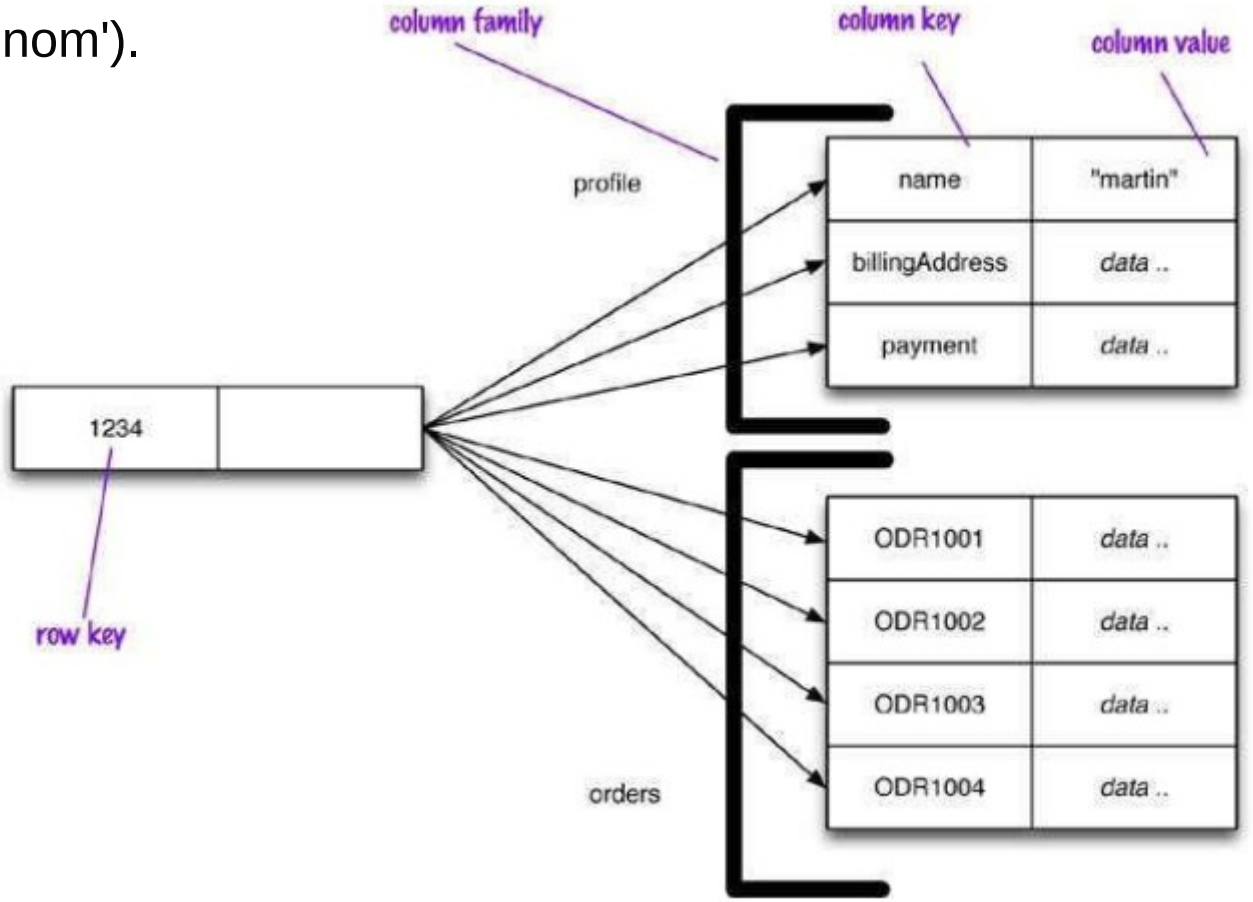

Figure 5. Représentation des informations sur les clients dans une structure de type colonne-famille

1.4. Model de la famille des colonnes

Les bases de données à famille de colonnes organisent leurs colonnes en familles de colonnes. Chaque colonne doit faire partie d'une seule famille de colonnes, et la colonne fait office d'unité d'accès, en partant du principe que les données d'une famille de colonnes particulière seront généralement consultées ensemble.

Cela vous donne également deux façons de penser à la façon dont les données sont structurées.

- **Orienté lignes** : Chaque ligne est un agrégat (par exemple, un client avec l'ID de 1234) avec des familles de colonnes représentant des groupes de données.
- **Orienté colonnes** : Chaque famille de colonnes définit un type d'enregistrement avec une ligne pour chacun des enregistrements.

1.4. Model de la famille des colonnes

Ce dernier aspect reflète la nature colonnaire des bases de données à famille de colonnes. Puisque la base de données connaît ces regroupements communs de données, elle peut utiliser cette information pour son comportement de stockage et d'accès.

Les familles de colonnes donnent une qualité bidimensionnelle aux bases de données à base de colonnes.

Cette terminologie est celle établie par **Google Bigtable et HBase**, mais **Cassandra** voit les choses légèrement différemment. Dans Cassandra, une ligne n'apparaît que dans une seule famille de colonnes, mais cette famille de colonnes peut contenir des supercolonnes, c'est-à-dire des colonnes qui contiennent des colonnes imbriquées. Les supercolonnes de Cassandra sont l'équivalent des familles de colonnes classiques de Bigtable.

1.4. Model de la famille des colonnes

Bien qu'il soit utile de distinguer les familles de colonnes par leur nature large ou étroite, il n’y a aucune raison technique pour qu'une famille de colonnes ne puisse pas contenir à la fois des colonnes de type champ et des colonnes de type liste, bien que cela perturbe l'ordre de tri.

1.5. Résumé des bases de données orientées vers les agrégats

À ce stade, nous avons abordé suffisamment de sujets pour vous donner un aperçu raisonnable des trois différents styles de modèles de données orientés agrégats et de leurs différences.

1.5. Résumé des bases de données orientées vers les agrégats

Ce qu'ils partagent tous est la notion d'agrégat indexé par une clé que vous pouvez utiliser pour la recherche. L'agrégat est un élément central de l'exécution sur un cluster, car la base de données veillera à ce que toutes les données d'un agrégat soient stockées ensemble sur un seul nœud. L'agrégat agit également comme une unité atomique pour les mises à jour, fournissant une quantité utile, bien que limitée, de contrôle transactionnel.

Au sein de cette notion d'agrégat, nous avons quelques différences. Le modèle de données clé-valeur traite l'agrégat comme un ensemble opaque, ce qui signifie que vous ne pouvez effectuer une recherche de clé que pour l'ensemble de l'agrégat.

Qu’est-il des modelés des graphes?