Université de Ngaoundéré
Institut Universitaire de Technologie

Base de Données non structurées (NoSQL)

Chapitre 3 Plus de détails sur les modèles de données

Pr. Dayang Paul
Maître de conférence

Année académique
2024/2025

INTRODUCTION

Jusqu'à présent, nous avons couvert la caractéristique principale de la plupart des bases de données NoSQL : leur utilisation d'agrégats et la façon dont les bases de données orientées agrégats modélisent les agrégats de différentes manières.

Si **les agrégats** sont un élément central de l'histoire des bases de données NoSQL, la modélisation des données ne s'arrête pas là et nous allons explorer ces concepts plus en profondeur dans ce chapitre.

1. Les relations

Les agrégats sont utiles dans la mesure où ils regroupent des données auxquelles on accède généralement ensemble. Mais il existe encore de nombreux cas où des données apparentées sont consultées différemment. Prenons l'exemple de la relation entre un client et toutes ses commandes. Certaines applications voudront:

- Accéder à l'historique des commandes chaque fois qu'elles accèdent au client ;
- cela va de pair avec la combinaison du client et de son historique de commandes en un seul agrégat.

D'autres applications, en revanche, veulent ***traiter les commandes individuellement et modélisent donc les commandes comme des agrégats indépendants***.

1. Les relations

Dans ce cas, vous voudrez des agrégats distincts pour les commandes et les clients, mais avec une sorte de lien entre eux, de sorte que tout travail sur une commande puisse consulter les données du client. La façon la plus simple de fournir un tel lien est d'**intégrer l'ID** du client dans les données agrégées de la commande.

Ainsi, si vous avez besoin des données de l'enregistrement du client, vous pouvez:

- **Lire la commande,**
- **Trouver l'ID du client,**
- **Appeler à nouveau la base de données pour lire les données du client.**

Cela fonctionnera, et sera parfait dans de nombreux scénarios, mais la base de données ignorera la relation dans les données.

1. Les relations

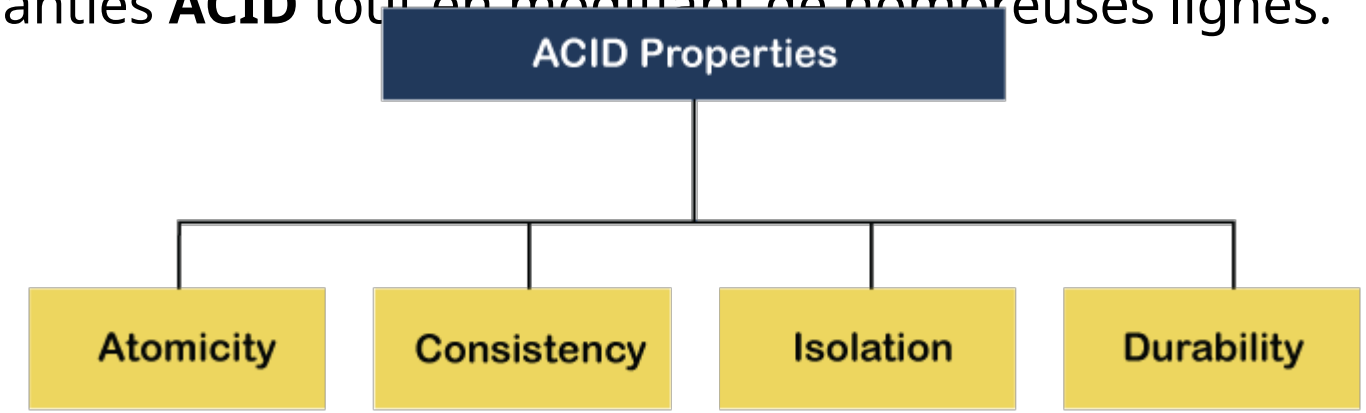
Par conséquent, de nombreuses bases de données, et même **les modèles clé-valeurs**, fournissent des moyens de rendre ces relations visibles pour la base de données. **Les modèles orienté documents** mettent le contenu de l'agrégat à la disposition de la base de données **sous forme d'index et de requêtes.**

Un aspect important des relations entre agrégats est la façon dont ils gèrent les mises à jour. Les bases de données orientées agrégats traitent l'agrégat comme l'unité de recherche de données.

Par conséquent, l'atomicité n'est supportée que dans le contenu d'un seul agrégat. Si vous mettez à jour plusieurs agrégats à la fois, vous devez faire face à un échec en cours de route.

1. Les relations

Les bases de données relationnelles vous aident dans ce domaine en vous permettant de modifier plusieurs enregistrements en une seule transaction, en fournissant des garanties **ACID** tout en modifiant de nombreuses lignes.



Tout cela signifie que les bases de données orientées agrégats deviennent plus difficiles à manier lorsque vous devez opérer sur plusieurs agrégats. Il existe plusieurs façons de gérer ce problème, que nous explorerons plus tard dans ce chapitre.

2. Bases de données graphiques

La plupart des bases de données NoSQL ont été inspirées par la nécessité de fonctionner sur des clusters, ce qui a conduit à des modèles de données orientés vers l'agrégation de grands enregistrements avec des connexions simples. Les bases de données graphiques sont motivées par une frustration différente par rapport aux bases de données relationnelles.

- **Un modèle opposé**
- **De petits enregistrements avec des interconnexions complexes**

2. Bases de données graphiques

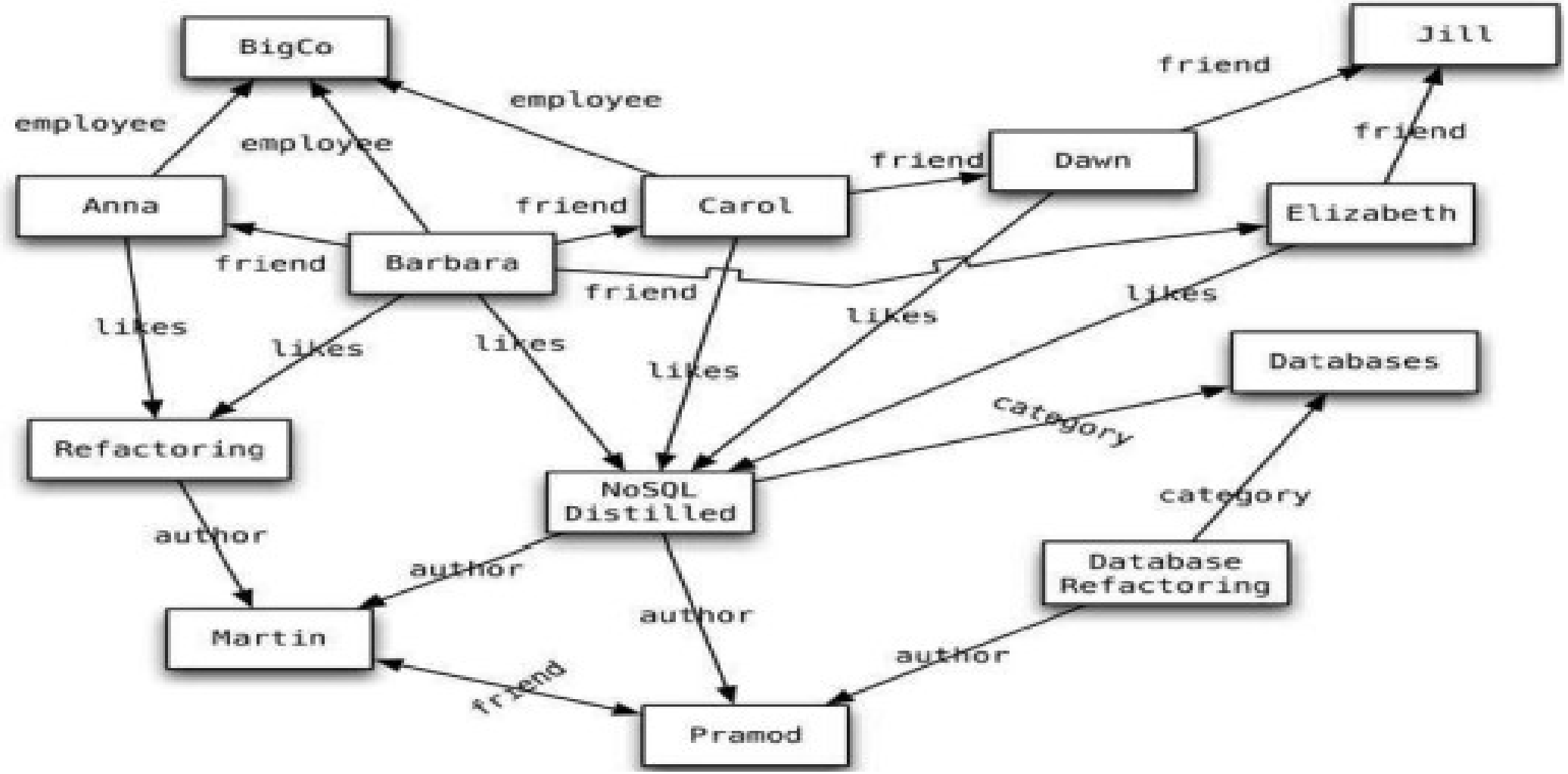


Figure 1. Un exemple de structure de graphe

2. Bases de données graphiques

Dans la figure 1, nous avons un réseau d'informations dont les nœuds sont très petits (rien de plus qu'un nom) mais il existe une riche structure d'interconnexions entre eux. Avec cette structure, nous pouvons poser des questions telles que "trouver les livres de la catégorie Bases de données qui sont écrits par l’amie d’un de mes amis.

Les bases de données graphiques sont spécialisées dans la capture de ce type d'informations, mais à une échelle beaucoup plus grande que celle d'un diagramme lisible. C'est l'idéal pour capturer toute donnée constituée de relations complexes telles que des réseaux sociaux, des préférences de produits ou des règles d'éligibilité.

2. Bases de données graphiques

Le modèle de données fondamental d'une base de données de graphes est très simple : des nœuds reliés par des arêtes (également appelées arcs). Au-delà de cette caractéristique essentielle, il existe une grande variété de modèles de données - en particulier, les mécanismes dont vous disposez pour stocker les données dans vos nœuds et vos arcs.

- **FlockDB,** est simplement constitué de nœuds et d'arêtes sans aucun mécanisme pour des attributs supplémentaires ;
- **Neo4J,** vous permet d'attacher des objets Java en tant que propriétés aux nœuds et aux arêtes, sans schéma.
- **Infinite Graph,** stocke vos objets Java, qui sont des sous-classes de ses types intégrés, en tant que noeuds et arêtes.

2. Bases de données graphiques

Une fois que vous avez construit un graphe de nœuds et d'arêtes, une base de données de graphes vous permet d'interroger ce réseau avec des opérations d'interrogation conçues pour ce type de graphe.

C'est là qu'interviennent les différences importantes entre les bases de données graphiques et relationnelles. Bien que les bases de données relationnelles puissent mettre en œuvre des relations à l'aide de clés étrangères, les jointures nécessaires pour naviguer peuvent devenir assez coûteuses, ce qui signifie que les performances sont souvent médiocres pour les modèles de données hautement connectés.

2. Bases de données graphiques

Les bases de données graphiques rendent la traversée des relations très bon marché. Cela s'explique en grande partie par le fait que les bases de données graphiques déplacent la majeure partie du travail de navigation dans les relations du moment de la requête au moment de l'insertion. Cela s'avère naturellement payant dans les situations où les performances d'interrogation sont plus importantes que la vitesse d'insertion.

La plupart du temps, vous trouvez des données en naviguant dans le réseau d'arêtes, avec des requêtes telles que "dites-moi tout ce qu'**aiment** Anna et Barbara". Cependant, vous avez besoin d'un point de départ, c'est pourquoi certains nœuds peuvent être indexés par un attribut tel que l'**ID**. Vous pouvez donc commencer par une recherche par ID (c'est-à-dire rechercher les personnes nommées "Anna" et "Barbara") et ensuite commencer à utiliser les bords.

2. Bases de données graphiques

Cependant, les bases de données graphiques s'attendent à ce que la plupart de vos requêtes portent sur la navigation dans les relations. L'accent mis sur les relations rend les bases de données graphiques très différentes des bases de données orientées agrégats.

Cette différence de modèle de données a également des conséquences sur d'autres aspects ; vous constaterez que ces bases de données sont plus susceptibles de fonctionner sur un seul serveur plutôt que d'être distribuées sur des grappes.

Les transactions ACID doivent couvrir plusieurs nœuds et bords pour maintenir la cohérence. La seule chose qu'elles ont en commun avec les bases de données orientées agrégats est leur rejet du **modèle relationnel** et le regain d'attention qu'elles ont reçu à peu près en même temps que le reste du domaine NoSQL.

3. Bases de données sans schéma

Un thème commun à toutes les formes de bases de données NoSQL est qu'elles sont sans schéma. Lorsque vous souhaitez stocker des données dans une base de données relationnelle:

- **Vous devez d'abord définir un schéma,**
- **Une structure définie pour la base de données qui indique quelles tables existent,**
- **Quelles colonnes existent et quels types de données chaque colonne peut contenir.**

Avant de stocker des données, vous devez définir le schéma correspondant

3. Bases de données sans schéma

Avec les bases de données NoSQL, le stockage des données est beaucoup plus décontracté.

- Un **modèle clé-valeur,** vous permet de stocker toutes les données que vous souhaitez sous une clé.
- Le **modèle orienté documents,** fait effectivement la même chose, puisqu'elle n'impose aucune restriction sur la structure des documents que vous stockez.
- Les **bases de données à colonnes,** vous permettent de stocker n'importe quelle donnée sous n'importe quelle colonne.
- Les **bases de données orienté graphe,** vous permettent d'ajouter librement de nouvelles arêtes et d'ajouter librement des propriétés aux nœuds et aux arêtes comme vous le souhaitez.

3. Bases de données sans schéma

Outre la gestion des modifications, un magasin sans schéma facilite également le traitement des **données non uniformes**, c'est-à-dire des données dont chaque enregistrement comporte un ensemble de champs différent. Un schéma place toutes les lignes d'une table dans un cadre, ce qui devient gênant si vous avez différents types de données dans différentes lignes.

L'absence de schéma permet d'éviter cela, en permettant à chaque enregistrement de contient juste ce dont il a besoin - ni plus, ni moins.

L'absence de schéma est séduisante et permet certainement d'éviter de nombreux problèmes liés aux bases de données à schéma fixe, mais elle entraîne également certains problèmes.

3. Bases de données sans schéma

Si tout ce que vous faites est de stocker des données et de les afficher dans un rapport sous la forme d'une simple liste de lignes **Nom du champ** : valeur, alors un schéma ne fera que vous gêner. Mais généralement, nous utilisons nos données plus que cela, et nous le faisons avec **des programmes** qui ont besoin de savoir que l'adresse de facturation s'appelle **billingAddress** etc.

Le fait d'avoir le schéma implicite dans le code de l'application entraîne certains problèmes. Cela signifie que pour comprendre les données présentes, vous devez creuser dans le code de l'application.

Essentiellement, une base de données sans schéma déplace le schéma dans le code de l'application qui y accède. Cela devient problématique si plusieurs applications, développées par des personnes différentes, accèdent à la même base de données. Ces problèmes peuvent être réduits grâce à deux approches

3. Bases de données sans schéma

La première consiste à encapsuler toute l'interaction avec la base de données dans une seule application et à l'intégrer à d'autres applications à l'aide de services Web. Cette approche correspond bien à la préférence actuelle de nombreuses personnes pour l'utilisation de services Web à des fins d'intégration.

Une autre approche consiste à délimiter clairement les différentes zones d'un agrégat auxquelles les différentes applications peuvent accéder. Il peut s'agir de différentes sections dans une base de données documentaire ou de différentes familles de colonnes dans une base de données à familles de colonnes.

3. Bases de données sans schéma

Bien que les fans de NoSQL reprochent souvent aux schémas relationnels de devoir être définis dès le départ et d'être inflexibles, ce n'est pas vraiment vrai. Les schémas relationnels peuvent être modifiés à tout moment avec des commandes SQL standard.

Si nécessaire, vous pouvez créer de nouvelles colonnes de manière ad hoc pour stocker des données non uniformes. Nous n'avons que rarement vu cela se faire, mais cela a raisonnablement bien fonctionné là où nous l'avons fait.

Cependant, la non-uniformité de vos données est une bonne raison de privilégier une base de données sans schéma.

4. Vues matérialisées

Lorsque nous avons parlé des modèles de données orientés agrégats, nous avons souligné leurs avantages. Si vous voulez accéder à des commandes, il est utile d'avoir toutes les données d'une commande contenues dans un seul agrégat qui peut être stocké et consulté comme une unité.

Mais l'orientation vers les agrégats **présente un inconvénient** correspondant :

« Que se passe-t-il si un chef de produit veut savoir combien un article particulier a été vendu au cours des deux dernières semaines ? »

L'orientation agrégat joue alors contre vous, vous obligeant à lire potentiellement chaque commande de la base de données pour répondre à la question. Vous pouvez réduire cette charge en construisant un index sur le produit, mais vous travaillez toujours contre la structure agrégée.

4. Vues matérialisées

Les bases de données relationnelles présentent un avantage à cet égard, car leur absence de structure agrégée leur permet de prendre en charge l'accès aux données de différentes manières.

En outre, elles fournissent un mécanisme pratique qui vous permet d'examiner les données différemment de la façon dont elles sont stockées : les vues. Une vue est comme une table relationnelle (c'est une relation) mais elle est définie par un calcul sur les tables de base.

Lorsque vous accédez à une vue, la base de données calcule les données de la vue une forme pratique d'encapsulation.

4. Vues matérialisées

Les vues fournissent un mécanisme permettant de cacher au client si les données sont des données dérivées ou des données de base - on ne peut pas éviter le fait que certaines vues sont coûteuses à calculer.

Pour y faire face, on a inventé les vues matérialisées, qui sont des vues calculées à l'avance et mises en cache sur le disque. Les vues matérialisées sont efficaces pour les données qui sont lues intensivement.

Bien que les bases de données NoSQL n'aient pas de vues, elles peuvent avoir des requêtes précalculées et mises en cache, et elles réutilisent le terme " vue matérialisée " pour les décrire.

C'est également un aspect beaucoup plus central pour les bases de données orientées agrégat que pour les systèmes relationnels, puisque la plupart des applications devront traiter certaines requêtes qui ne s'adaptent pas bien à la structure agrégée.

4. Vues matérialisées

Il existe deux stratégies générales pour construire une vue matérialisée.

La première est l'approche **eager**, qui consiste à mettre à jour la vue matérialisée en même temps que ses données de base. Dans ce cas, l'ajout d'une commande mettrait également à jour les agrégats de l'historique des achats pour chaque produit.

Cette approche convient lorsque la lecture de la vue matérialisée est plus fréquente que l'écriture et que l'on souhaite que la vue matérialisée soit aussi fraîche que possible. L'approche de la base de données d'application est précieuse ici car elle permet de s'assurer plus facilement que toute mise à jour des données de base met également à jour les vues matérialisées.

4. Vues matérialisées

Vous pouvez construire des vues matérialisées en dehors de la base de données en lisant les données, en calculant la vue et en la sauvegardant dans la base de données. Le plus souvent, les bases de données prennent en charge la construction de vues matérialisées eux-mêmes.

Dans ce cas, vous fournissez le calcul qui doit être effectué, et la base de données exécute1 le calcul lorsque cela est nécessaire en fonction de certains paramètres que vous configurez.

Comme nous l'avons mentionné précédemment, lorsque nous modélisons des agrégats de données, nous devons tenir compte de la manière dont les données seront lues ainsi que des effets secondaires sur les données liées à ces agrégats.

5. Modélisation de l'accès aux données

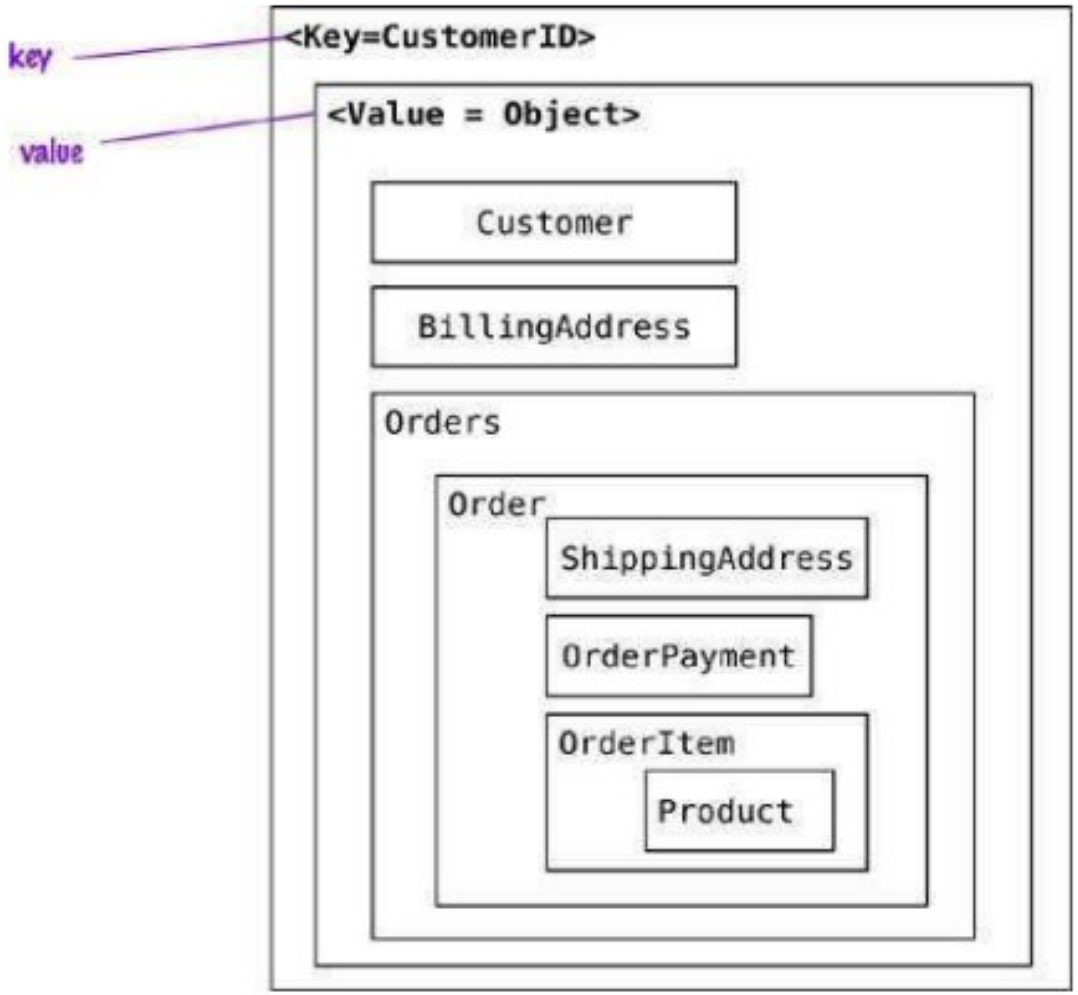


Figure 2. Embarquez tous les objets pour le client et ses commandes.

5. Modélisation de l'accès aux données

Dans ce scénario, l'application peut lire les informations sur le client et toutes les données connexes en utilisant la clé. Si les exigences sont de lire les commandes ou les produits vendus dans chaque commande, l'objet doit être lu et ensuite analysé du côté client pour construire les résultats.

Lorsque des références sont nécessaires, nous pouvons passer à des **modèles documents**, puis effectuer des requêtes à l'intérieur des documents, ou même modifier les données du **modèle clé-valeur** pour diviser l'objet valeur en objets **Client** et **Commandes**, puis maintenir les références de ces objets entre eux.

5. Modélisation de l'accès aux données

Avec les références (voir ***Figure 3***), nous pouvons maintenant trouver les commandes indépendamment du client, et avec la référence **orderId** dans le client, nous pouvons trouver toutes les commandes pour le client.

L'utilisation d'agrégats de cette manière permet d'optimiser la lecture, mais nous devons pousser la référence **orderIdreference** dans Client à chaque fois qu'une nouvelle commande est créée.

5. Modélisation de l'accès aux données

```
# Customer object
{
"customerId": 1,
"customer": {
        "name": "Martin",
        "billingAddress": [{"city": "Chicago"}],
        "payment": [{"type": "debit","ccinfo": "1000-1000-1000-1000"}],                "orders":
[{"orderId":99}]
  }
}
```

5. Modélisation de l'accès aux données

```
# Order object
{
    "customerId": 1,
     "orderId": 99,
                "order":{ "orderDate":"Nov-20-2011",
            "orderItems":[{"productId":27, "price": 32.45}],
            "orderPayment":[{"ccinfo":"1000-1000-1000-1000",
                        "txnId":"abelif879rft"}],
            "shippingAddress":{"city":"Chicago"}
            }
}
```

5. Modélisation de l'accès aux données

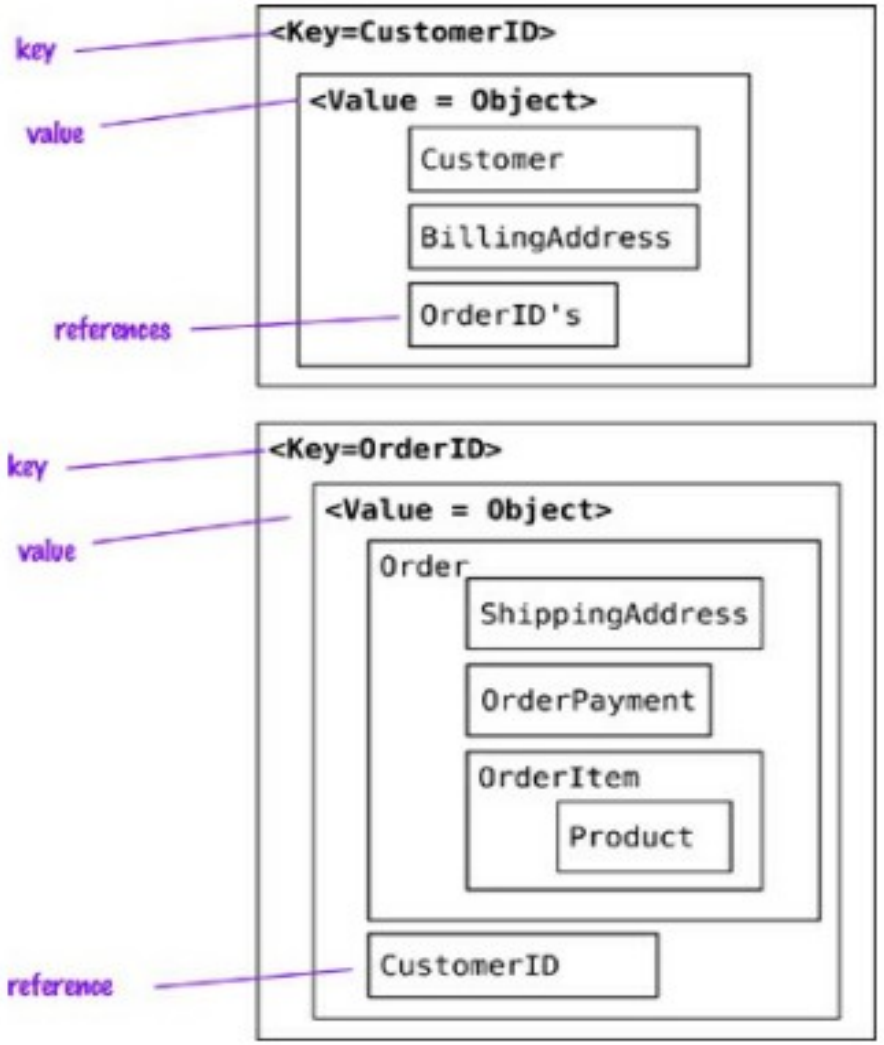


Figure 3. Le client est stocké séparément de la commande.

5. Modélisation de l'accès aux données

Les agrégats peuvent également être utilisés pour obtenir des analyses; par exemple, ***une mise à jour de l'agrégat peut fournir des informations sur les commandes qui contiennent un produit donné.***

Cette dénormalisation des données permet un accès rapide aux données qui nous intéressent et constitue la base de l'analyse en temps réel.

L'analyse

Les entreprises ne doivent plus compter sur les exécutions par lots de fin de journée pour alimenter les tables de l'entrepôt de données et générer des analyses; elles peuvent désormais remplir ce type de données, pour de multiples types de besoins, lorsque le client passe sa commande.

5. Modélisation de l'accès aux données

```
{
   "itemid":27,
   "orders":{99,545,897,678}
}
```

```
{
   "itemid":29,
   "orders":{199,545,704,819}
}
```

Dans les documents, puisque nous pouvons interroger à l'intérieur des documents, il est possible de **supprimer les références aux commandes** de l'**objet client**. Cette modification permet de ne pas mettre à jour l'**objet client** lorsque de nouvelles commandes sont passées par le client. commandes sont passées par le client.

5. Modélisation de l'accès aux données

```
# Objet client
{
   "customerid": 1,
    "name" : "Martin",
    "billingAddress" : [ {"city" : "Chicago"} ],
    "payment" : [ {"type" : "débit",
            "ccinfo" : "1000-1000-1000-1000"} ]
 }
```

```
# Objet de la commande
{
     "orderid" : 99,
     "customerid" : 1,
     "orderDate" : "Nov-20- 2011",
       "orderitems" : [ { "Id. produit" : 27, "prix" : 32.
45}],
     "orderPayment" : [
           { "ccinfo" : "1000-1000-1000-1000",
              "txnid" : "abelif879rft"}],
      "shippingAddress":{"city" : "Chicago"}
}
```

5. Modélisation de l'accès aux données

Lors de la modélisation pour les **modèles de données orienté colonnes**, nous avons l'avantage que les colonnes sont **ordonnées**, ce qui nous permet de nommer les colonnes qui sont fréquemment utilisées afin qu'elles soient recherchées en premier.

Lorsque vous utilisez les familles de colonnes pour modéliser les données, n'oubliez pas de le faire en fonction des exigences de votre requête et non dans le but d'écrire ;

la règle générale est de faciliter l'interrogation et de dénormaliser les données pendant l'écriture.

Comme vous pouvez l'imaginer, il existe plusieurs façons de modéliser les données. L'une d'entre elles consiste à stocker le client et la commande dans des familles de colonnes différentes

5. Modélisation de l'accès aux données

Ici, il est important de noter que la référence à toutes les commandes passées par le client se trouve dans la famille de colonnes client. D'autres dénormalisations similaires sont généralement effectuées afin d'améliorer les performances des requêtes.

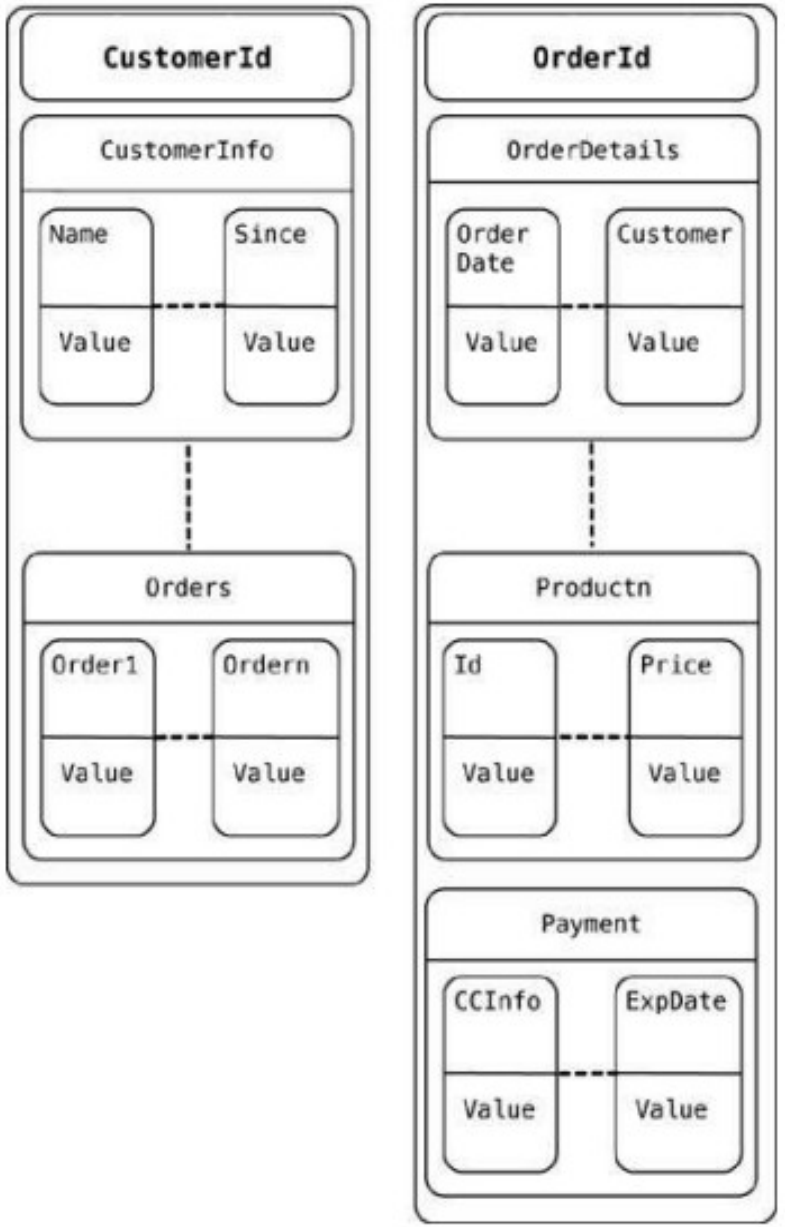


Figure 4. Vue conceptuelle d'un modèle de données en colonnes

5. Modélisation de l'accès aux données

Lorsque nous utilisons des bases de données de graphes pour modéliser les mêmes données, nous modélisons tous les objets comme des nœuds et les relations entre eux comme des relations ; ces relations ont des types et une signification directionnelle.

Chaque nœud a des relations indépendantes avec d'autres nœuds. Ces relations portent des noms tels que **ACHATÉ, PAYÉ, ou APPARTENANCE** (voir Figure 5) ;
ces noms de relations nous permettent de parcourir le graphe.

5. Modélisation de l'accès aux données

Ce type de traversée des relations est très facile avec les bases de données graphiques. Il est particulièrement pratique lorsque vous avez besoin d'utiliser les données pour recommander des produits aux utilisateurs ou pour trouver des modèles dans les actions entreprises par les utilisateurs.

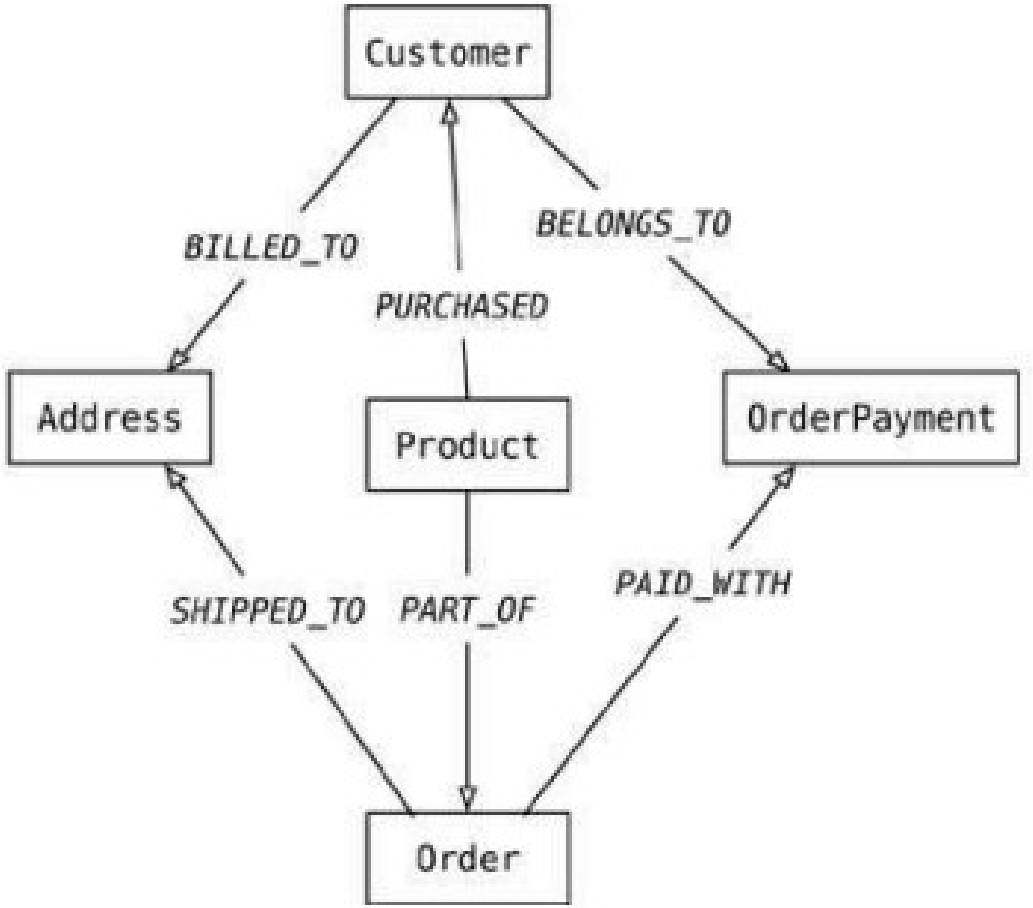


Figure 5. Modèle graphique des données du commerce électronique

Résume

- Les bases de données **orientées agrégats** rendent les relations inter-agrégats plus difficiles à gérer que les relations intra-agrégats.
- Les bases de données **graphiques** organisent les données sous forme de graphes de nœuds et d'arêtes; elles fonctionnent mieux pour les données qui présentent des structures de relations complexes.
- Les bases de données sans schéma vous permettent d'ajouter librement des champs aux enregistrements, mais il existe généralement un schéma implicite attendu par les utilisateurs des données.
- Les bases de données orientées agrégats calculent souvent des vues matérialisées pour fournir des données organisées différemment de leurs agrégats primaires. Ceci est souvent fait avec des calculs de type map-reduce

Etant donné que les bases de données NoSQL on été mise en place pour s’executer dans les clusters,

Quel modèle de distributions utilise-telle?