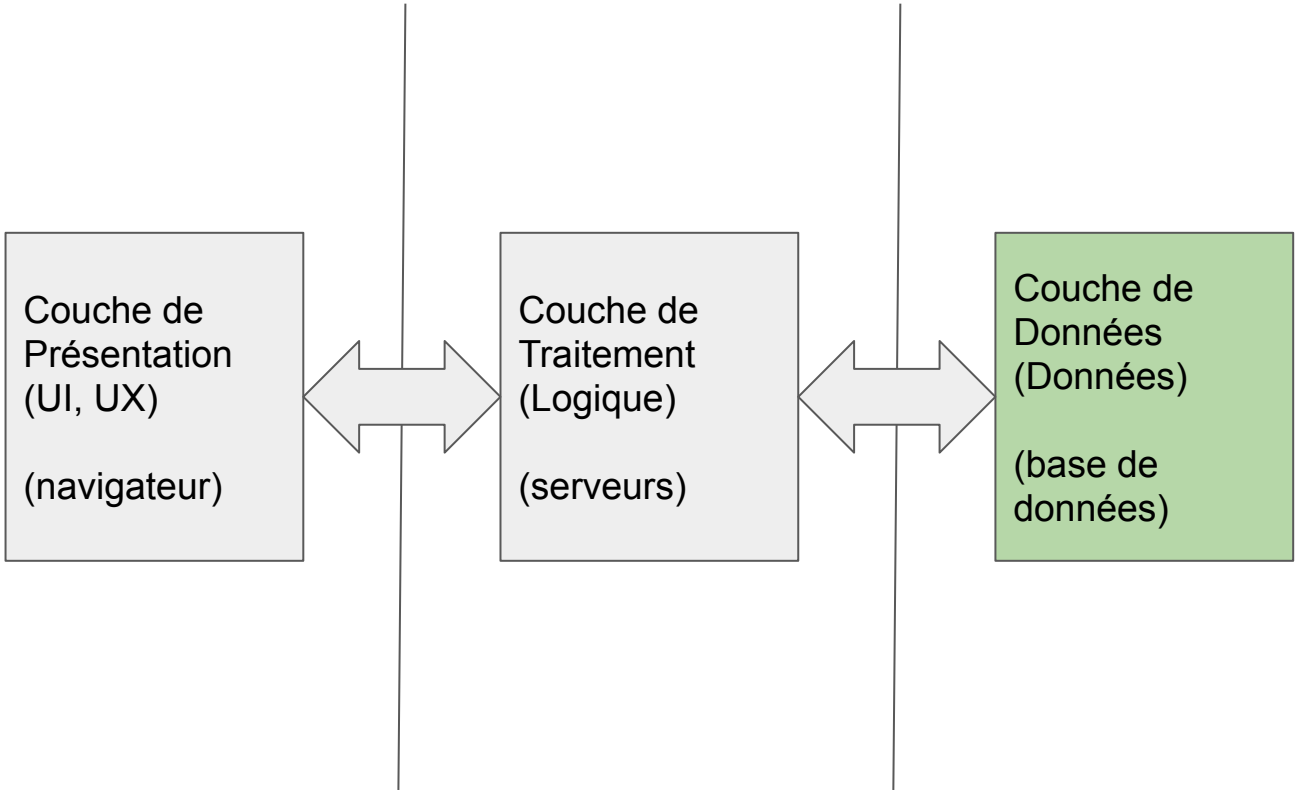
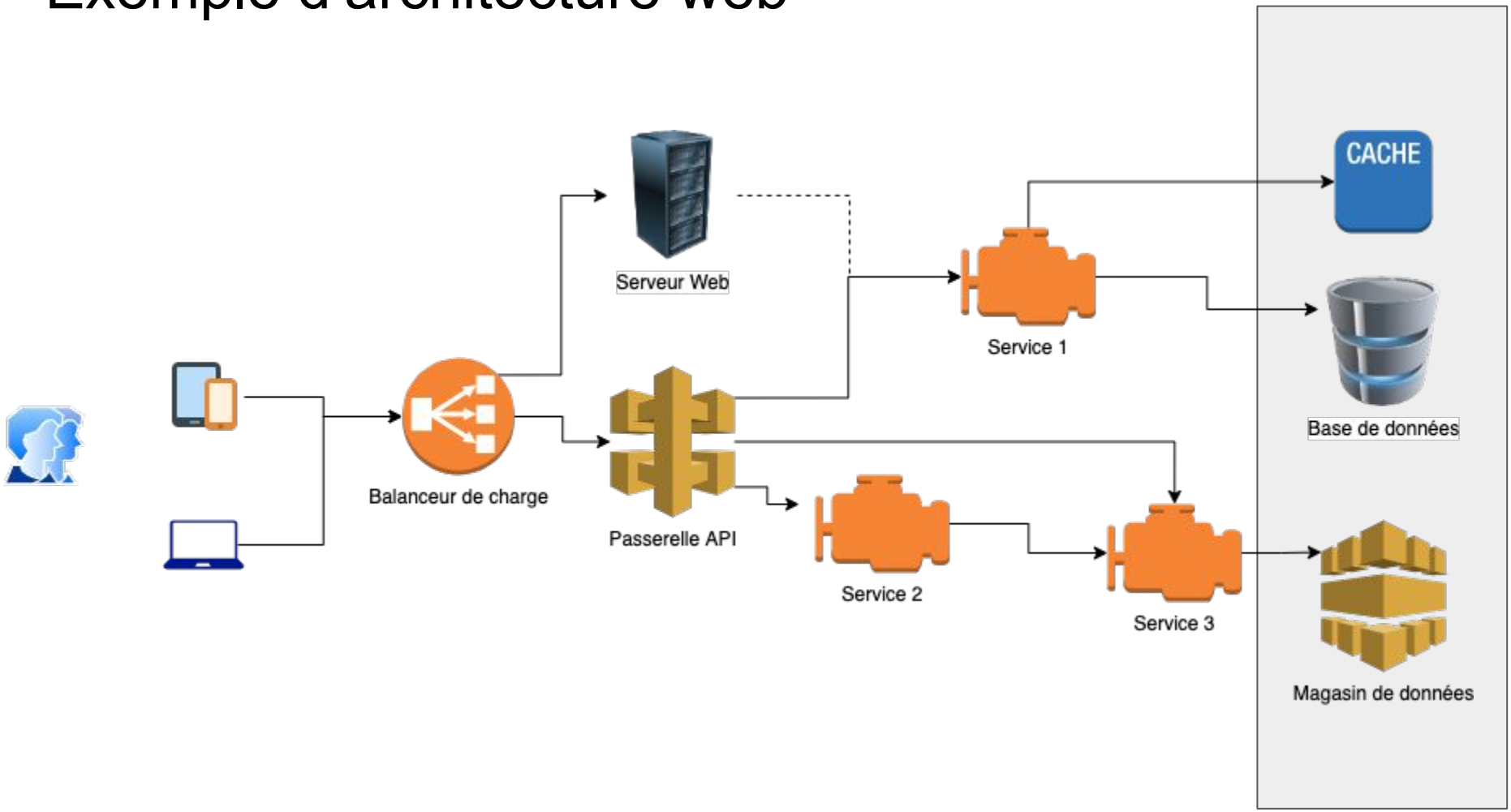
Couche de données

Plan et objectifs

- Explorer les options de persistance de données
- Bases de données relationnelles
- Bases de données non-relationnelles
- Outils de mapping objet-relationnel
- Gestion de fichiers et ressources binaires


Couche de Présentation (UI, UX)
(navigateur)
Couche de Traitement (Logique)
(serveurs)
Couche de Données (Données)
(base de données)

Exemple d’architecture web



Couche de données

L’objectif de la **couche de données** est de gérer la **persistance des données** du système et d’en offrir l’**accès**.

La couche de données ne devrait *idéalement* pas contenir de logique d’affaire.

Lors de la construction de cette couche, le focus est mis sur la **modélisation des différentes entités du système** (incluant les contraintes et leurs relations) ainsi que l’**optimisation des patrons d’accès et de recherche**.

Modélisation de données

Avant même de choisir une option de stockage de données, il est important de bien **analyser le domaine** et la logique d'affaire en lien avec le système que l'on désire bâtir.

Il est primordial de bien **identifier les entités** (concepts, objets), leurs attributs et leurs **relations** (et cardinalité des relations) pour construire un modèle.

Cette étape d'analyse est cruciale: une modification dans le modèle peut avoir de grandes répercussions dans tout le système, car, bien que l'implémentation puisse différer, conceptuellement les mêmes *entités - relations* se retrouvent souvent dans toutes les couches du système.

Patrons d’accès aux données

Il est aussi important de bien comprendre les différents patrons d’accès aux données requis pour les différents scénarios d’utilisation.

Ceci nous permet de choisir la ou les technologies de stockage les plus appropriées et d’optimiser la configuration et l’implémentation du modèle pour faciliter l’accès (e.g. sélection d’index, insertion partielle).

Persistance dans le temps

La couche de données détermine aussi le type de persistance des différentes entités.

Cette persistance peut être temporaire (cache) ou permanente (base de données).

Il est possible qu'une entité soit stockée de façon temporaire dans un système pour rendre son accès plus rapide (cache), mais qu'elle soit aussi stockée de façon permanente pour assurer sa pérennité.

Selon les besoins en cohérence, les données en cache peuvent expirer au bout d'un certain temps ou être invalidées par l'application lorsqu'un changement est détecté.

Un ensemble de solution

La couche de données peut contenir plusieurs solutions différentes de stockage de données.

Par exemple, un système pourrait avoir:

- une ou plusieurs bases de données relationnelles
- une ou plusieurs caches
- un ou plusieurs engins de recherche
- un ou plusieurs systèmes de gestion de fichiers

Base de données relationnelle (SGBDR)

Basé sur un **modèle de tables** possédant un **schéma** qui décrit les champs, les types et leurs contraintes.

Typiquement, chaque entité possède un **identificateur unique** (clé) qui est utilisé pour référencer un objet particulier dans la table.

Les relations sont définies en utilisant ces clés uniques.

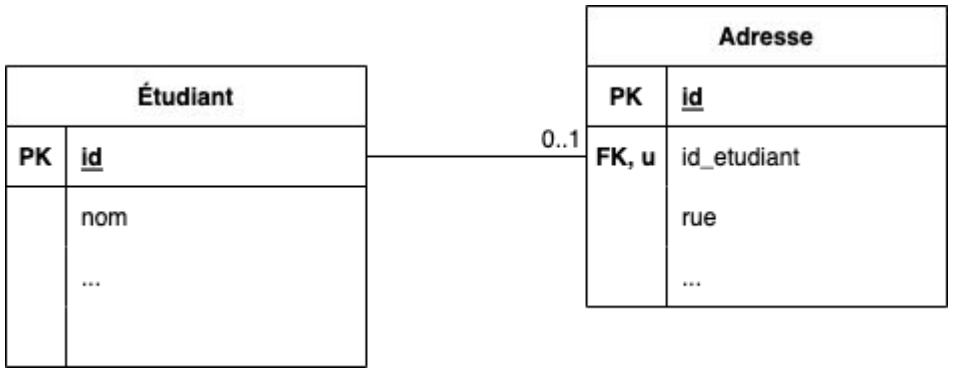
Selon la cardinalité de la relation, elle peut être exprimée soit par une référence directe (un attribut qui réfère à une entité spécifique d'une autre table), soit par une table qui lie les clés de plusieurs enregistrements entre eux.

Relation 1 à 1

Pour créer une relation de 1 à 1 entre deux entités, on utilise la clé primaire dans la 2e table (qui devient une clé étrangère) et on y ajoute un contrainte d'unicité.

Exemple

- Une table Étudiant: [id, nom, prénom, …]
- Une table Adresse: [id, id_etudiant, adresse, ville, …]
- La table Adresse utilise une clé étrangère (id_etudiant) qui fait référence à la clé primaire de la table Étudiant.
- La table Adresse ajoute une contrainte d'unicité sur l'attribut id_etudiant.

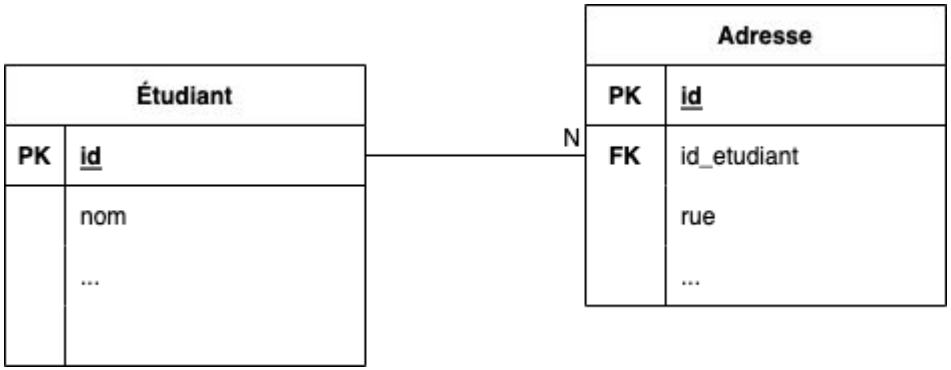


Relation 1 à n

Pour créer une relation de 1 à n entre deux entités, on utilise la clé primaire dans la 2e table (clé étrangère).

Exemple

- Une table Étudiant: [id, nom, prénom, …]
- Une table Adresse: [id, id_etudiant, adresse, ville, …]
- La table Adresse utilise une clé étrangère (id_etudiant) qui fait référence à la clé primaire de la table Étudiant.
- Si un étudiant peut avoir plusieurs adresses, aucune contrainte d'unicité est ajoutée pour l'attribut id_etudiant.

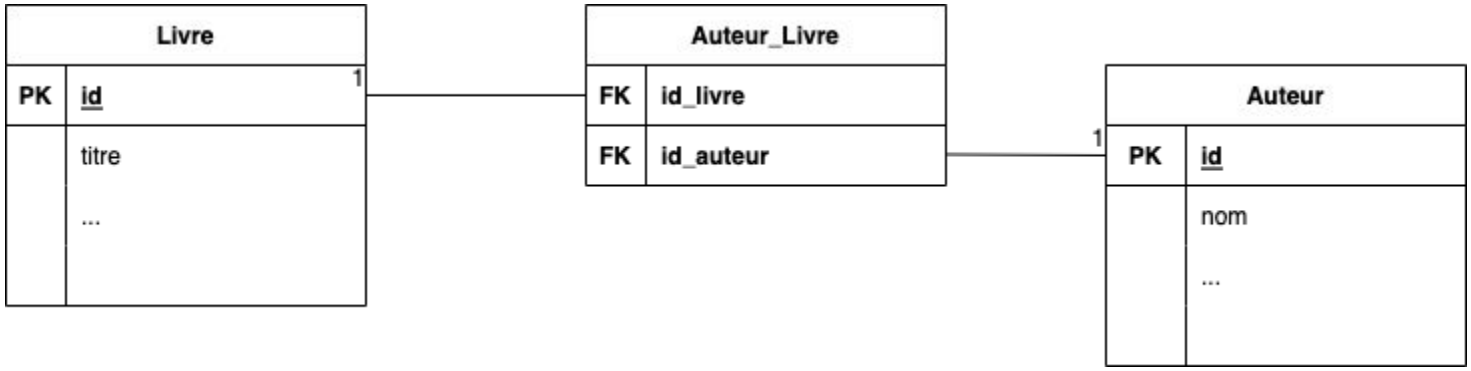


Relation n à m

Pour créer une relation de n à m entre deux entités, on utilise une table de jonction qui fait référence aux clés primaires des deux tables.

Exemple

- Une table Livre: [id, titre, date_de_publication, …]
- Une table Auteur: [id, nom, prénom, …]
- Une table Auteur_Livre: [id_livre, id_auteur]
- La table Auteur_Livre utilise une clé étrangère de la table Livre (id_livre) et y associe une clé étrangère (id_auteur) de la table Auteur.
- Il est possible d'ajouter des contraintes d'unicité sur l'une et/ou l'autre pour changer la cardinalité de n - m à 1 - n, n - 1 ou 1 - 1.



Propriétés ACID

Quatre propriétés définissent les transactions des bases de données relationnelles : atomicité, cohérence, isolement et durabilité.

- **Atomicité**: Tout ou rien
- **Cohérence**: Les contraintes seront toujours respectées
- **Isolement:** Le résultat d'une transaction n'est visible qu'une fois qu'elle est complétée et plusieurs transactions simultanées n'interfèrent pas entre elles.
- **Durabilité**: Une fois complété, l'état final de l'opération est enregistré sur disque.

La discordance objet-relationnel

L’utilisation d’une base de données (BD) relationnelle via un langage orienté objet amène certaines difficultés de conversion entre les deux mondes :

- Granularité: Il est possible que dans le monde objet, plusieurs classes servent à décrire un enregistrement de la BD.
- Sous-type: Le monde objet défini des sous-classes (hiérarchie) ce qui n’est pas le cas dans la BD.
- Identité: Les contraintes d’unicité n'existe plus une fois dans le monde objet.
- Association: Les associations entre entités ne sont pas définies de la même façon (table de jointure, champ dans un objet).
- Navigation: Dans une BD, pour suivre les relations, on navigue d’une table à l’autre. Dans le monde objet, on suit les références directement.

Outils de mapping objet-relationnel (ORM)

Un **outil de mapping objet-relationnel** est une **couche logiciel** (typiquement une bibliothèque) qui fait le **lien entre les objets définis dans le langage de programmation et le modèle de tables défini dans la base de données relationnelle**.

L'objectif est de **simplifier la conversion** pour que le programmeur demeure seulement dans le monde objet, ce qui entraîne un gain de productivité et permet de standardiser et optimiser certains patrons d'accès (dans la bibliothèque).

Souvent considérés comme plus lents (que de bâtir les requêtes SQL à la main), les outils permettent de **programmer à un niveau d'abstraction plus élevé** et de ne pas se soucier des détails d'accès avec la BD (jusqu'à ce qu'un problème survienne).

Avantages des SGBDRs

- Langage de requêtes (SQL) très puissant et versatile
- Assure la cohérence des données (schéma stricte)
- Transactions et changements atomiques
- Outils d’administration très évolués et puissants
- Technologie mature et prouvée

Désavantages des SGBDRs

- Modélisation pas toujours adaptée (e.g. structure récursive ou graphe) et contrainte par un schéma fixe
- L'évolution d'un schéma demande souvent une migration de donnée
- La modification d'une table qui contient beaucoup de données devient coûteuse (verrouillage de table)
- Mise à l'échelle horizontale difficile, car les propriétés ACID doivent être maintenues (principal/subordonné, réplication, partitionnement).
- Certaines requêtes peuvent être très coûteuses (e.g. verrouillage, sans index).

Des compromis

Maintenant, il existe beaucoup d'autres options pour le stockage de données. Selon les besoins du système, on peut troquer certaines caractéristiques pour une autre. Il n'existe pas de solution parfaite. Quelques exemples de compromis:

- schéma stricte ou schéma flexible
- requêtes flexibles ou mise à l'échelle simple
- transactionnalité ou disponibilité

Bases de données de type clé-valeur

Base de données simple pour stocker des valeurs adressées par une clé (i.e. un dictionnaire).

Les données peuvent être en mémoire (type cache) ou persistées selon des règles (persistance périodique, après chaque opération, etc).

Le format de la clé est libre.

Le format des valeurs dépend de la technologie, mais la base de données peut accepter différents types et offrir différentes fonctionnalités selon le type (e.g. ajout d’un item à une liste de façon atomique).

Démo Redis

Bases de données orientés documents

Les données sont stockées sous forme de **documents structurés** (style JSON) qui sont **groupés par collections** au lieu de tables.

Les documents d'une même collection n'ont pas nécessairement exactement les mêmes champs (hétérogènes).

Les documents sont accédés par le nom de la collection, puis la clé du document ou par une recherche (e.g. timestamp > 1664299885).

Les relations entre les entités sont définies par une référence vers un id d'un autre document comme dans un SGBDR, mais l'existence de la clé n'est pas nécessairement validée par la base de données.

Quelques bonnes vidéos sur les patrons de conception dans un monde non-relationnel: NoSQL Data Modelling with Redis.

Modélisation: relation 1 à 1

Dans une base de données non relationnelle, différentes stratégies sont possibles pour modéliser les relations.

Pour une relation 1 à 1,

- deux collections indépendantes où l'id de l'entité principale est un attribut de l'entité secondaire (comme pour un SGBDR)
- une collection et une sous collection où le nombre d'entité dans la sous collection est toujours 1 (la collection **edudiants** contient les données sur les étudiants et la collection **etudiants/*id*/adresse** contient l'adresse)
- une collection où l'on insert l'entité secondaire à l'intérieur de l'entité principale (une seule collection **etudiants**, mais l'enregistrement contient l'adresse)

Modélisation: relation 1 à n

Pour une relation 1 à n:

- deux collections indépendantes où l'id de l'entité principale est un attribut de l'entité secondaire (similaire au cas avec un SGBDR)
- une collection et une sous collection où le nombre d'entité dans la sous collection est variable (**etudiants/*id*/adresses**)
- une seule collection où l'on insert les entités secondaires à l'intérieur de l'entité principale (si le nombre d'entités secondaires est limité) sous forme d'une liste.

```
{
    "nom":"Fred",
    "adresses":[
        ...
    ]
}
```

Modélisation: relation n à m

Pour une relation n à m:

- deux collections indépendantes où chaque document contient une liste d'id correspondant aux autres entités associées (possibilité de le faire que d'un seul côté)

/livres

```
{
   "titre":"Le titre",
   "auteurs":[
      "idAuteur1",
      "idAuteur2",
      "..."
   ]
}
```

/auteurs

```
{
   "nom":"A. Uteur",
   "livres":[
      "idLivre1",
      "idLivre2",
      "..."
   ]
}
```

Modélisation - Insertion partielle

Pour optimiser les performances d'une application, il est aussi possible d'insérer (dupliquer) une partie des informations d'une autre entité dans l'entité principale.

En général, on choisit les informations nécessaires à l'affichage de l'entité principale et on va chercher les données complètes de l'entité secondaire seulement au besoin.

Avantages:

- Limite le nombre d'opérations lectures
- Plus rapide pour l'affichage

Désavantages:

- Duplication de donnée qu'il faut synchroniser
- Si la vue change, il est possible que le modèle ne soit plus aussi optimal

Exemple: Insertion Partielle

/auteurs

```
{
   "nom":"A. Uteur",
   "livres":[
      {
         "id":"idLivre1",
         "titre":"Le titre"
      },
      {
         "id":"idLivre2",
         "titre":"L'autre titre"
      },
      "..."
   ]
}
```

Avantages d’une base de données orientée documents

- Modèle plus souple
- Parfois plus intuitif
- Plus facile de mettre à l’échelle (partitionnement des collections et clés)
- Certaines options offrent des transactions
- L’évolution du modèle est plus simple, mais requiert parfois une migration de données
- Possibilité de dupliquer les données pour plus de performance (e.g. une publication peut contenir une partie des informations de son auteur)

Désavantages d’une base de données orientée documents

- Comme il n’y a pas de schéma, les données peuvent être hétérogènes et plus difficilement manipulables par les clients
- Il est aussi plus facile d’affecter négativement la cohérence des données (attributs avec un mauvais type, mauvais nom d’attribut en écriture, etc)
- Langage de requêtes souvent moins puissant que pour un SGBDR
- Plus difficile de maintenir l’intégrité des relations entre documents (via id)
- Pas toujours l’option de joindre des tables facilement (fait à la main ou duplication de données)
- Si duplication de données, il faut un mécanisme de synchronisation (cohérence à terme)

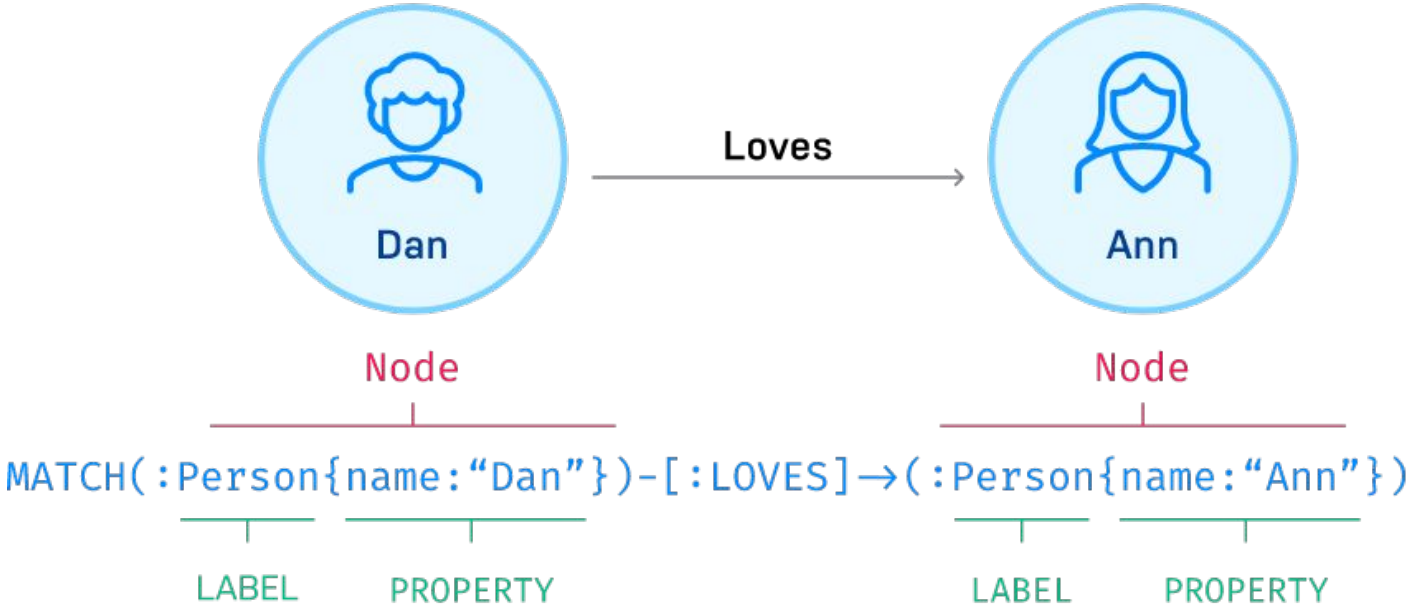
Démo Firestore

Base de données orientée graphe

Les données sont stockées sous forme de graphe:

- noeuds - enregistrements
- arcs - relations

Chacun pouvant contenir des attributs.



Avantages d’une base de données orientée graphe

- Flexibilité du schéma (objets hétérogènes)
- Excellent pour modéliser des entités qui ont différents types de relations entre elles
- Permet d’effectuer des requêtes qui traversent le graphe simplement

Désavantages d’une base de données orientée graphe

- Typiquement moins rapide, mais optimisée pour certains types de requêtes (e.g. les amis de mes amis)
- Mise à l’échelle et partitionnement plus difficile
- Différents langages de requêtes selon la technologie
- Limitation sur le nombre de noeuds (selon la technologie et $)

Démo neo4j

Base de données de recherche

Type de base de données non-relationnelle de type document ayant pour but de permettre des recherches ‘texte-libre’.

Chaque document contient différents attributs qui ont diverses fonctions :

- recherche texte-libre
- filtre
- ordonnancement (*ranking*)
- affichage

Ces bases de données permettent à l’utilisateur d’entrer des mots-clés et la base de données trouvera les documents les plus pertinents.

Données binaires et fichiers

Il est parfois nécessaire de stocker des images ou des fichiers associés à certaines entités.

Il existe différentes solutions:

- Utiliser un champ binaire (BLOB, binary large object) si disponible, mais attention aux limites de taille
- Stocker sur disque sur un serveur (attention à la mise à l'échelle, permission, backup, disponibilité)
- Utiliser sur une solution de stockage de fichier infonuagique (attention aux coûts)

Si on ne stocke pas l'image ou le fichier dans l'entité elle-même, il faut faire attention à maintenir le lien (identificateur du fichier dans l'entité) et s'assurer de bien nettoyer les fichiers ou images une fois qu'elles ne sont plus utilisés (via une modification ou suppression de l'entité).

Liste de vérification

Voici une liste de points à considérer lors de la sélection d'un système de stockage de données.

- Type de schéma - strict ou flexible
- Formats de données possibles (binaire, structure de données)
- Limitation de taille (limitation par champ, par document)
- Support pour les clés étrangères
- Type d’index (performance)
- Type de persistance et garanties (en mémoire, sur disque, etc)
- Support pour les transactions (ACID)
- Support pour la réplication (performance, mise à l’échelle)
- Support pour le partitionnement (performance, mise à l’échelle)
- Options de sécurité
- Langage de requêtes (options et limitations)
- Type de mise à l’échelle possible (verticale et horizontale)