MongoDB

Une introduction au NoSQL

C'est parti! →

Formation MongoDB

Jour 1 : Fondamentaux et Requêtes

Appuyez sur Espace pour passer à la slide suivante →

Conclusion de la journée

Synthèse et travail personnel pour demain

Récapitulatif des concepts clés

Concepts fondamentaux

- Base de données NoSQL orientée document
- Flexibilité du schéma et modèle de données JSON/BSON
- Structure: bases de données, collections, documents
- Atlas comme solution cloud vs installation locale

Requêtes avancées

- Opérateurs de comparaison: $eq, $gt, $lt, $in…
- Opérateurs logiques: $and, $or, $not, $nor
- Requêtes sur tableaux: $all, $elemMatch, $size
- Requêtes sur documents imbriqués
- Projection, pagination et tri

CRUD operations

- Create: insertOne(), insertMany()
- Read: find(), findOne()

Interfaces

- MongoDB Shell (mongosh)
- MongoDB Compass

Pour la prochaine session

Travail personnel

- Terminer les exercices du TP
- Explorer la documentation officielle de MongoDB
- Réfléchir aux cas d'utilisation personnels/professionnels

Questions à se poser

- Comment optimiser mes requêtes complexes ?
- Comment modéliser mes données relationnelles en NoSQL ?

Préparez-vous pour demain

Nous aborderons :

- L'indexation et l'optimisation des performances
- Les requêtes géospatiales
- Le framework d'agrégation

Ressources supplémentaires

- Documentation MongoDB CRUD

Introduction au NoSQL et à MongoDB

Les fondamentaux pour comprendre la révolution des bases de données non-relationnelles

Qu’est-ce que le NoSQL ?

NoSQL signifie "Not Only SQL" - une famille de bases de données qui s’éloigne du modèle relationnel traditionnel.

Caractéristiques principales

- Schéma flexible ou absent
- Conçu pour la scalabilité horizontale
- Optimisé pour des modèles de données spécifiques
- Compromis dans la cohérence (CAP théorème)

Types de bases NoSQL

- **Document** : MongoDB, CouchDB
- **Clé-valeur** : Redis, DynamoDB
- **Colonne** : Cassandra, HBase
- **Graphe** : Neo4j, OrientDB

Comparaison avec les bases relationnelles

Concept SQL	Concept MongoDB	Description
Database	Database	Conteneur physique pour les collections
Table	Collection	Groupe de documents MongoDB
Row	Document	Enregistrement unique dans une collection
Column	Field	Paire clé-valeur dans un document
Index	Index	Améliore les performances des requêtes
JOIN	$lookup & Embedding	Association entre documents
Primary Key	_id Field	Identifiant unique pour chaque document

Pourquoi MongoDB ?

Forces

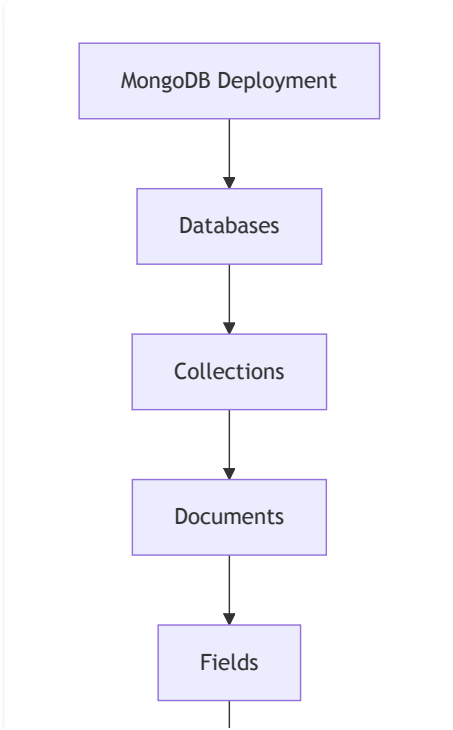
- Schéma flexible adaptatif
- Modèle de données intuitif (JSON)
- Performances élevées en lecture/écriture
- Scalabilité horizontale (sharding natif)
- Requêtes riches et expressives
- Indexation avancée
- Distribution géographique
- Support de transactions multi-documents

Cas d'usage

- Applications web et mobiles
- Gestion de contenus (CMS)
- E-commerce (catalogues produits)
- IoT et données en temps réel
- Big Data et analytique
- Gestion de métadonnées
- Stockage de données de configuration
- Caching et sessions

Architecture de MongoDB

Structure logique



Le Format BSON

BSON (Binary JSON) est le format de stockage et d'échange de données utilisé par MongoDB.

Caractéristiques

- Extension binaire de JSON
- Encodage plus efficace en espace
- Support de types additionnels
- Optimisé pour la traversée rapide
- Conçu pour la sérialisation/désérialisation rapide

Types de données BSON

- Types de base: String, Number, Boolean, Null
- Types étendus:
  - ObjectId (identifiant unique sur 12 octets)
  - Date (timestamp UNIX en millisecondes)
  - BinData (données binaires)
  - RegExp (expressions régulières)
  - Timestamp (horodatage interne)
  - NumberDecimal, NumberLong, NumberInt

Exemple de document BSON:

```
{
```

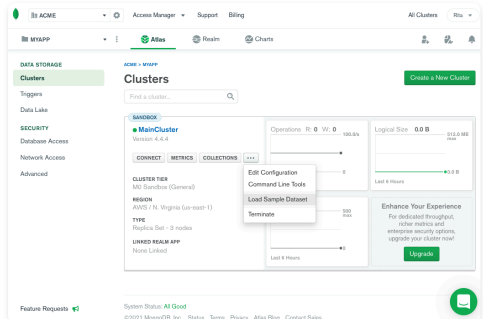
Installation et premiers pas avec MongoDB

Configuration de l'environnement, interfaces et concepts fondamentaux

Options d’installation

MongoDB Atlas (Cloud)

1. Création d’un compte sur mongodb.com/atlas
2. Déploiement d’un cluster gratuit (M0)
3. Configuration du réseau (liste blanche IP)
4. Création d’un utilisateur pour la connexion
5. Obtention de la chaîne de connexion



Installation locale

Windows:

- Téléchargement de l’installateur MSI
- Assistant d’installation
- Option "MongoDB as a Service"

macOS:

```
brew tap mongodb/brew
brew install mongodb-community
```

Linux (Ubuntu):

```
wget -qO - https://www.mongodb.org/static/pgp/serve
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb
sudo apt-get update
```

Interfaces MongoDB

MongoDB Shell (mongosh)

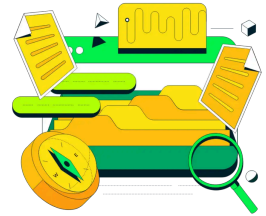
Interface ligne de commande interactive avec syntaxe JavaScript:

```
// Connexion
mongosh "mongodb+srv://user:pass@cluster.mongodb.net

// Liste des bases de données
show dbs

// Utilisation d'une base
use ma_base

// Liste des collections
show collections

// Aide
db.help()
```

MongoDB Compass

Interface graphique officielle pour:

- Explorer les données visuellement
- Construire et tester des requêtes
- Analyser les performances
- Gérer les index
- Importer/exporter des données



Opérations CRUD dans MongoDB

Create, Read, Update, Delete - Les fondamentaux de la manipulation de données

Create : Insertion de documents

insertOne()

Insertion d'un document unique:

```
db.utilisateurs.insertOne({
  nom: "Dupont",
  prenom: "Jean",
  email: "jean.dupont@example.com",
  age: 35,
  actif: true
})
```

Résultat:

```
{
  acknowledged: true,
          insertedId: ObjectId("...")
}
```

insertMany()

Insertion de plusieurs documents:

```
db.utilisateurs.insertMany([
  {
    nom: "Martin",
    prenom: "Sophie",
    email: "sophie.martin@example.com",
    age: 28
  },
  {
    nom: "Dubois",
    prenom: "Pierre",
    email: "pierre.dubois@example.com",
    age: 42
  }
])
```

Résultat:

Read : Lecture de documents

find()

Récupération de multiples documents:

```
// Tous les documents
db.utilisateurs.find()

// Avec un filtre
db.utilisateurs.find({ age: { $gt: 30 } })

// Avec projection (sélection des champs)
db.utilisateurs.find(
        { actif: true },
        { nom: 1, email: 1, _id: 0 }
)
```

Méthodes de curseur:

```
db.utilisateurs.find()
        .sort({ age: -1 })
```

findOne()

Récupère un seul document:

```
// Premier document correspondant
db.utilisateurs.findOne({ nom: "Dupont" })

// Avec critères multiples
db.utilisateurs.findOne({
  nom: "Dupont",
  actif: true
})
```

Méthodes de comptage

```
// Compter tous les documents
db.utilisateurs.countDocuments()

// Compter avec filtre
db.utilisateurs.countDocuments({ age: { $lt: 30 } })
```

Update : Mise à jour de documents

updateOne()

Met à jour le premier document correspondant:

```
db.utilisateurs.updateOne(
        { email: "jean.dupont@example.com" },
        { $set: { age: 36, derniere_connexion: new D
)
```

updateMany()

Met à jour tous les documents correspondants:

```
db.utilisateurs.updateMany(
        { age: { $lt: 18 } },
        { $set: { categorie: "junior" } }
)
```

replaceOne()

Remplace un document entier:

```
db.utilisateurs.replaceOne(
        { _id: ObjectId("...") },
        {
          nom: "Dupont",
          prenom: "Jean-Pierre",
          email: "jp.dupont@example.com",
          age: 36
        }
)
```

Options de mise à jour

- `upsert` : insertion si n'existe pas (défaut: false)
- `multi` : mise à jour de tous les documents

Opérateurs de mise à jour

Opérateurs de champs

- `$set` : définit une valeur

```
{ $set: { categorie: "premium" } }
```

- `$unset` : supprime un champ

```
{ $unset: { temporaire: "" } }
```

- `$rename` : renomme un champ

```
{ $rename: { "ancien_nom": "nouveau_nom" } }
```

- `$inc` : incrémente une valeur numérique

```
{ $inc: { compteur: 1, score: 5 } }
```

- `$mul` : multiplie une valeur numérique

```
{ $mul: { prix: 1.1 } } // +10%
```

Opérateurs de tableaux

- `$push` : ajoute un élément à un tableau

```
{ $push: { tags: "nouveau-tag" } }
```

- `$addToSet` : ajoute sans dupliquer

```
{ $addToSet: { categories: "sport" } }
```

- `$pull` : retire des éléments du tableau

```
{ $pull: { tags: "obsolete" } }
```

- `$pop` : retire le premier (-1) ou dernier (1) élément

```
{ $pop: { historique: 1 } }
```

- Avec modificateurs

Delete : Suppression de documents

deleteOne()

Supprime le premier document correspondant:

```
// Suppression par ID
db.utilisateurs.deleteOne(
        { _id: ObjectId("...") }
)

// Suppression par critère
db.utilisateurs.deleteOne(
        { email: "jean.dupont@example.com" }
)
```

deleteMany()

Supprime tous les documents correspondants:

```
// Suppression conditionnelle
db.utilisateurs.deleteMany(
```

drop()

Supprime une collection entière:

```
// Suppression de la collection
db.utilisateurs.drop()
```

Points importants

- Les suppressions sont définitives
- La haute disponibilité peut compliquer la suppression
- Les suppressions conservent les index et métadonnées

Alternatives à la

Requêtes avancées

Filtres complexes, opérateurs et fonctions avancées pour extraire précisément l’information

Opérateurs de comparaison

Opérateurs de base

- `$eq` : égal (défaut)

```
{ age: { $eq: 30 } }  // équivalent à { age: 30 }
```

- `$ne` : non égal

```
{ status: { $ne: "inactif" } }
```

- `$gt` : supérieur à

```
{ age: { $gt: 18 } }
```

- `$gte` : supérieur ou égal à

```
{ age: { $gte: 18 } }
```

- `$lt` : inférieur à

```
{ prix: { $lt: 100 } }
```

Opérateurs d'appartenance

- `$in` : dans une liste de valeurs

```
{ categorie: { $in: ["A", "B", "Premium"] } }
```

- `$nin` : non présent dans une liste

```
{ categorie: { $nin: ["Archivé", "Supprimé"] } }
```

Combinaison d'opérateurs

```
// Prix entre 10 et 50
db.produits.find({
  prix: {
    $gte: 10,
    $lte: 50
  }
})
```

Opérateurs logiques

$and (ET logique)

Implicite lorsque plusieurs conditions sont dans le même objet:

```
// Implicite (recommandé si possible)
db.utilisateurs.find({
  age: { $gt: 18 },
  actif: true
})

// Explicite (nécessaire pour conditions multiples s
db.utilisateurs.find({
  $and: [
    { age: { $gt: 18 } },
    { age: { $lt: 65 } }
  ]
})
```

$not (NON logique)

$or (OU logique)

Au moins une condition doit être vraie:

```
db.utilisateurs.find({
  $or: [
    { categorie: "premium" },
    { credits: { $gt: 100 } }
  ]
})
```

$nor (NI…NI logique)

Aucune des conditions ne doit être vraie:

```
db.utilisateurs.find({
  $nor: [
    { bloque: true },
    { supprime: true }
  ]
```

Requêtes sur les tableaux

Recherche simple

Correspond si le tableau contient la valeur:

```
// Utilisateurs avec le tag "premium"
db.utilisateurs.find({ tags: "premium" })
```

$all (tous les éléments)

Correspond si le tableau contient tous les éléments spécifiés:

```
// Utilisateurs avec TOUS ces tags
db.utilisateurs.find({
  tags: { $all: ["premium", "newsletter"] }
})
```

$size (taille exacte)

$elemMatch (correspondance complexe)

Correspond si au moins un élément du tableau satisfait tous les critères:

```
// Produits avec au moins une note ≥ 4 ET vérifiée
db.produits.find({
  avis: {
    $elemMatch: {
      note: { $gte: 4 },
      verifie: true
    }
  }
})
```

Opérateurs de position

Requêtes sur les documents imbriqués

Notation par points

Accès aux champs imbriqués:

```
// Utilisateurs qui vivent à Paris
db.utilisateurs.find({
  "adresse.ville": "Paris"
})

// Accès à un niveau plus profond
db.utilisateurs.find({
  "preferences.notifications.email": true
})
```

Recherche sur documents entiers

Correspondance exacte (y compris l'ordre des champs):

Combinaison de critères

```
// Plusieurs conditions sur sous-documents
db.utilisateurs.find({
  "adresse.ville": "Paris",
  "adresse.codePostal": { $in: ["75001", "75002"] },
  "profil.experience": { $gte: 5 }
})
```

Requêtes sur tableaux d'objets

```
// Commandes avec produit spécifique
db.commandes.find({
  "produits.id": "P12345"
})

// Commandes avec produit spécifique ET quantité > 2
db.commandes.find({
  produits: {
```

Projection et limitation des résultats

Projection (sélection des champs)

```
// Inclusion (1)
db.utilisateurs.find(
        { age: { $gt: 18 } },
        { nom: 1, email: 1 }
)

// Exclusion (0)
db.utilisateurs.find(
        { age: { $gt: 18 } },
        { mot_de_passe: 0, __v: 0 }
)

// Avec champs imbriqués
db.utilisateurs.find(
        { actif: true },
        { "nom": 1, "adresse.ville": 1 }
)
```

Limitation et pagination

```
// Limiter le nombre de résultats
db.produits.find().limit(10)

// Sauter des résultats (offset)
db.produits.find().skip(20)

// Pagination (page 3, 10 éléments par page)
db.produits.find().skip(20).limit(10)
```

Tri des résultats

```
// Tri ascendant (1)
db.produits.find().sort({ prix: 1 })

// Tri descendant (-1)
db.produits.find().sort({ date_creation: -1 })

// Tri multi-critères
db.produits.find().sort({
```

Travaux Pratiques

Mise en pratique des concepts vus aujourd'hui

TP 1 : Configuration de l’environnement

Objectif

Mettre en place un environnement MongoDB fonctionnel et se familiariser avec l’interface.

Étapes

1. **Création d’un compte MongoDB Atlas**
   - Inscription sur mongodb.com/atlas
   - Déploiement d’un cluster gratuit (M0)
2. **Configuration des accès**
   - Création d’un utilisateur de base de données
   - Configuration des règles réseau (whitelist IP)
3. **Connexion avec MongoDB Compass**

Tâches à réaliser

4. **Création de vos premières collections**
   - Créer une base de données "formation"
   - Créer deux collections : "utilisateurs" et "produits"
5. **Importation de données initiales**
   - Télécharger les fichiers d’exemple fournis
   - Utiliser la fonctionnalité d’import de Compass
6. **Première exploration**
   - Identifier la structure des documents
   - Utiliser l’interface de filtrage de Compass
   - Tester quelques requêtes simples

TP 2 : Manipulation de données avec les opérations CRUD

Exercice 1 : Création d'un jeu de données

1. Création d'une collection "ecommerce_produits"
2. Insertion d'au moins 10 produits avec:

- nom, description, prix, stock
- catégorie et sous-catégorie
- caractéristiques techniques (objet imbriqué)
- commentaires/avis (tableau d'objets)
- tags (tableau simple)

Exercice 3 : Mises à jour

1. Augmenter le prix de tous les produits d'une catégorie de 5%
2. Ajouter un champ "promotion" à certains produits
3. Ajouter un nouveau tag à tous les produits d'une catégorie
4. Mettre à jour le stock après une "vente"

Exercice 4 : Requêtes complexes

1. Trouver les produits disponibles avec tag1 ET

Modélisation des données dans MongoDB

Approches de modélisation

Documents embarqués (dénormalisation)

- Regroupement des données liées dans un seul document
- Lectures plus rapides et atomiques
- Idéal pour les relations "un-à-plusieurs" limitées

```
{
  _id: ObjectId("..."),
         titre: "Smartphone XYZ",
         prix: 699,
         fabricant: {
```

Références (normalisation)

- Séparation des données dans différentes collections
- Utilisation de références entre documents
- Idéal pour les relations "plusieurs-à-plusieurs"

```
// Collection produits
{
  _id: ObjectId("p1"),
         titre: "Smartphone XYZ",
        prix: 699,
        fabricant_id: ObjectId("m1")
}

// Collection fabricants
{
  _id: ObjectId("m1"),
         nom: "TechCorp",
        pays: "USA",
```

Patterns de modélisation

Pattern "Un-à-Plusieurs" (Embarqué)

Exemple : Articles et commentaires

```
{
  _id: ObjectId("..."),
         titre: "Introduction à MongoDB",
         auteur: "Jean Dupont",
         date: ISODate("2023-01-15"),
         contenu: "...",
         commentaires: [
    {
      auteur: "Marie",
      texte: "Article très instructif",
      date: ISODate("2023-01-16")
    },
    {
      auteur: "Paul",
      texte: "Merci pour ces explications",
```

Pattern "Plusieurs-à-Plusieurs" (Référence)

Exemple : Étudiants et cours

```
// Collection cours
{
  _id: ObjectId("c1"),
          titre: "MongoDB Avancé",
        description: "...",
        credits: 3
}

// Collection étudiants
{
  _id: ObjectId("e1"),
          nom: "Dupont",
        prenom: "Jean",
        cours_inscrits: [
  ObjectId("c1"),
```

Considérations pratiques pour la modélisation

Cardinalité et croissance

- **Un-à-un (1:1)** : Généralement embarqué, sauf si données rarement utilisées
- **Un-à-plusieurs (1:N)** :
  - Si N est petit et stable → Embarqué
  - Si N est grand ou variable → Références
- **Plusieurs-à-plusieurs (N:M)** : Généralement références

Schéma polymorphe

Différents types de documents dans la même

Pattern d'attributs calculés

Stockage des agrégats pré-calculés:

```
// Produit avec compteurs pré-calculés
{
  _id: ObjectId("..."),
          nom: "Laptop Pro",
        prix: 1299,
        // Valeurs pré-calculées mises à jour à chaq
        nb_vues: 1528,
        nb_commentaires: 43,
        note_moyenne: 4.7
}
```

Compromis à considérer

Bonnes pratiques en production

Conception du schéma

- Prioriser l'usage et les requêtes pour la conception
- Éviter la profondeur excessive dans les documents
- Limiter la taille des tableaux (risque de dépassement)
- Choisir des noms de champs courts mais descriptifs
- Documenter le schéma et ses évolutions

Gestion des identifiants

Optimisation des performances

- Embarquer les données fréquemment accédées ensemble
- Éviter les requêtes gigantesques (utiliser skip/limit/pagination)
- Indexer les champs fréquemment interrogés ou triés
- Surveiller la taille des documents avec des tableaux
- Planifier la distribution des données (sharding)

Sécurité dans MongoDB

Authentification

- **Utilisateurs et rôles**: Créer des utilisateurs dédiés avec privilèges minimaux

```
db.createUser({
  user: "app_user",
  pwd: "secure_password",
  roles: [
    { role: "readWrite", db: "app_database" }
  ]
})
```

- **Méthodes d'authentification**:
  - SCRAM (par défaut)
  - X.509 certificats
  - LDAP (Enterprise)

Autorisation

- **Rôles prédéfinis**:
  - `read` , `readWrite`
  - `dbAdmin` , `userAdmin`
  - `clusterAdmin` , `backup` , `restore`
- **Rôles personnalisés**:

```
db.createRole({
  role: "reportingRole",
  privileges: [
    {
      resource: { db: "sales", collection: "order
      actions: [ "find" ]
    }
  ],
```

Haute disponibilité et réplication

Principes de la réplication

- Ensemble de réplicas (Replica Set)
- 1 nœud primaire (Primary) + N nœuds secondaires
- Élection automatique d'un nouveau primaire en cas de défaillance
- Garantie de cohérence (CP dans le théorème CAP)

Configuration d'un Replica Set

```
rs.initiate({
  _id: "rs0",
  members: [
    { _id: 0, host: "mongodb0.example.net:27017" },
    { _id: 1, host: "mongodb1.example.net:27017" },
    { _id: 2, host: "mongodb2.example.net:27017" }
  ]
})
```

Types de nœuds secondaires

- **Standard**: Réplique complète, peut devenir primaire
- **Priority 0**: Ne peut pas devenir primaire

Performances et optimisation

Analyse des performances

- **Outils de diagnostic**:
  - `explain()`
  - `db.collection.stats()`
  - Profiler MongoDB

```
db.setProfilingLevel(1, { slowms: 100 })
```

  - MongoDB Atlas Monitoring

Stratégies d'optimisation

- Requêtes adaptées aux index
- Limitation des résultats (ne récupérer que le nécessaire)

Bonnes pratiques d'indexation

- Indexer les champs fréquemment interrogés
- Créer des index composites pour les requêtes complexes
- Éviter les index inutilisés ou redondants
- Surveiller la taille et l'impact des index
- Construire les index en arrière-plan pour la production

```
db.users.createIndex(
        { "last_login": 1 },
        { background: true }
)
```

Sauvegarde et récupération

Stratégies de sauvegarde

- **Méthodes**:
  - Sauvegarde logique (mongodump/mongorestore)
  - Sauvegarde par copie de fichiers
  - Sauvegarde par snapshot de volume
  - Backup continu avec MongoDB Cloud/Atlas

Sauvegarde avec mongodump

```
# Sauvegarde complète d'une base
```

Restauration des données

```
# Restauration complète
mongorestore --uri="mongodb://user:pwd@host:port" /b

# Restauration avec options spécifiques
mongorestore --drop --nsInclude="app.*" /backup/path
```

Points à considérer

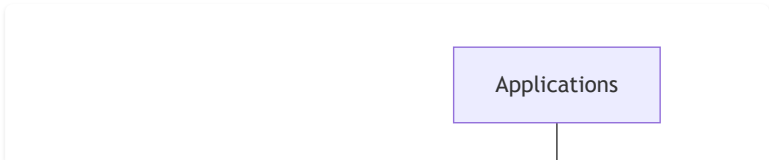
- Fréquence des sauvegardes adaptée à la criticité
- Sauvegardes complètes vs incrémentielles
- Test régulier des procédures de restauration
- Réplication comme complément (pas comme remplacement)
- Stratégie de rétention (combien de temps

MongoDB en environnement distribué

Sharding (partitionnement)

- Distribution horizontale des données sur plusieurs serveurs
- Permet de gérer des volumes de données très importants
- Augmente les capacités de lecture/écriture

Architecture du sharding



Clés de sharding

- **Choix crucial** pour la distribution et les performances
- Types de distribution:
  - Hashed: distribution uniforme
  - Ranged: requêtes par plages optimisées

```
// Sharding d'une collection
sh.shardCollection(
        "database.collection",
        { user_id: "hashed" } // ou { timestamp: 1 }
)
```

Considérations

- Opérations de resharding complexes
- Transactions distribuées (depuis MongoDB

Microservices et MongoDB

Approches d'architecture

- **Database-per-Service**:
  - Une base par microservice
  - Isolation et autonomie
  - Réduction des conflits de schéma
  - Évolutivité indépendante
- **Collection-per-Service**:
  - Collections dédiées par service
  - Base de données partagée
  - Transactions multi-collections possibles
  - Gestion simplifiée

Défis et solutions

- **Requêtes distribuées**:
  - API Gateway
  - CQRS (Command Query Responsibility Segregation)
  - Vues matérialisées
- **Cohérence des données**:
  - Cohérence éventuelle
  - Compensation/Saga pattern
  - Outbox pattern
- **Évolution de schéma**:

Conclusion de la journée

Synthèse et travail personnel pour demain

Récapitulatif des concepts clés

Concepts fondamentaux

- Base de données NoSQL orientée document
- Flexibilité du schéma et modèle de données JSON/BSON
- Structure: bases de données, collections, documents
- Atlas comme solution cloud vs installation locale

Requêtes avancées

- Opérateurs de comparaison: $eq, $gt, $lt, $in…
- Opérateurs logiques: $and, $or, $not, $nor
- Requêtes sur tableaux: $all, $elemMatch, $size
- Requêtes sur documents imbriqués
- Projection, pagination et tri

CRUD operations

- Create: insertOne(), insertMany()
- Read: find(), findOne()

Modélisation et bonnes pratiques

- Documents embarqués vs références

Pour la prochaine session

Travail personnel

- Terminer les exercices du TP
- Explorer la documentation officielle de MongoDB
- Réfléchir aux cas d'utilisation personnels/professionnels

Questions à se poser

- Comment optimiser mes requêtes complexes ?
- Comment modéliser mes données relationnelles en NoSQL ?

Préparez-vous pour demain

Nous aborderons :

- L'indexation et l'optimisation des performances
- Les requêtes géospatiales
- Le framework d'agrégation

Ressources supplémentaires

- Documentation MongoDB CRUD

layout:

Learn More

Documentation · GitHub · Showcases

Powered by Slidev