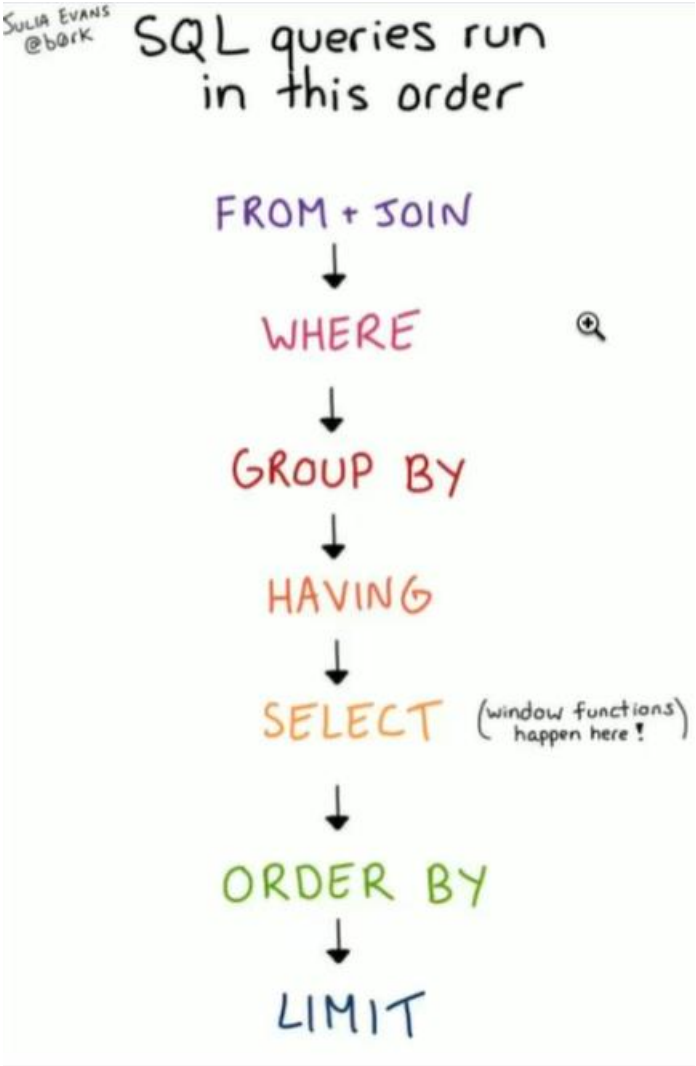
Points à savoir sur Nosql

Pas de schema en nosql

Conçu pour la scalabilité horizontale : peut s'adapter au nombre grandissant de requêtes

Notions SQL



ALTER = changer la structure de la table => risque de perte de données

Importance de la modélisation de données en amont

Offset = utilisé dans la gestion de la pagination

Nor: aucune des conditions ne doit être vraie

Projection: possibilité de ne pas sélectionner tous les champs (lorsqu'on ne fait pas de SELECT *)

Équivalence SQL

Table: Collection

Row: Document

Column: Field

Index: Index

Index: Repère sur une donnée qui améliore les performances de lecture

Join: $Lookup / embedding

Primary Key: _id field

Schema: Fixe(SQL) vs Flexible (Mongo)

Relations: Jointures explicite s vs Documents imbriqués

Scalabilité: Verticale vs Horizontale

Transactions: ACIS complet vs ACID par document (avant v4.0)

Requêtes: SQL vs Syntaxe orientées objet

Find() : équivalent du SELECT *

UpdateOne/Many(): équivalent de l'UPDATE

DeleteOne/Many(): équivalent du DELETE

Projection :2e param find où on précise les champs

Transaction: Succession d'opérations SQL

BSON

JSON étendu spécialement conçu pour mongoDB, il ajute des fonctions et une syntaxe différente du JSON

https://www.mongodb.com/docs/manual/reference/bson-types/?utm_source=compass&utm_medium=product

Set up un cluster mongodb

Pour créer une base de données il faut au mois une collection

Pour créer une collection il fut au moins un Document à l'intérieur

Ajout d'un utilisateur

```
db.utilisateurs.insertOne({prenom: "maxime", nom: "tuconnais", age: 20})
```

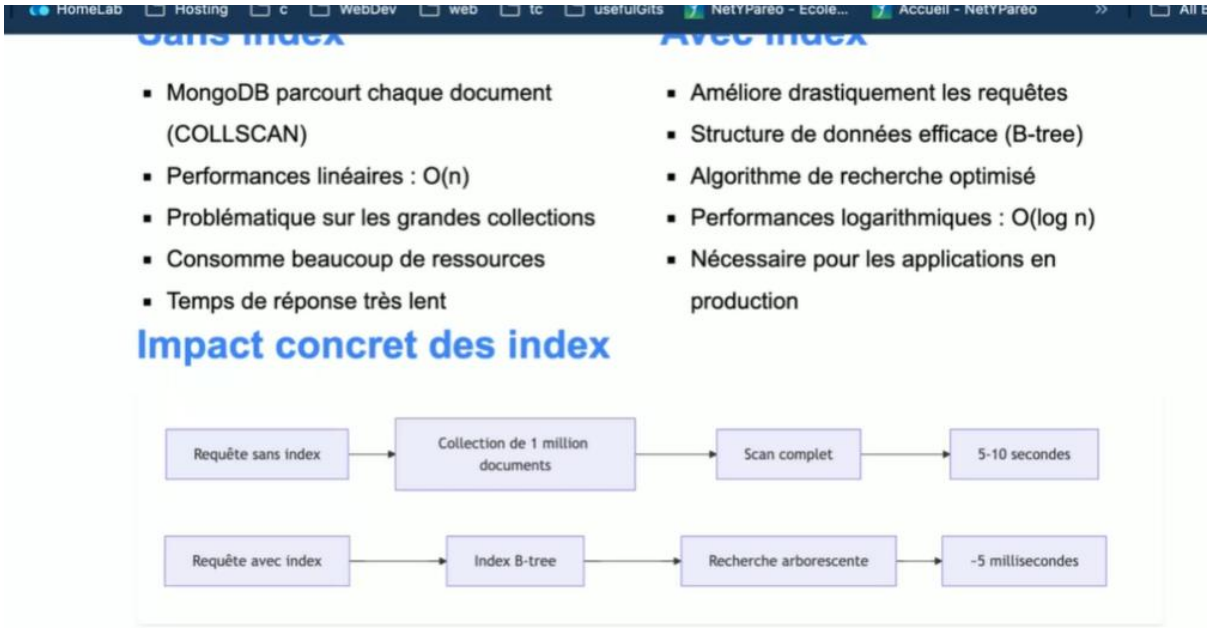
Vérifier si 2 entrées sont dans un tableau (tags)

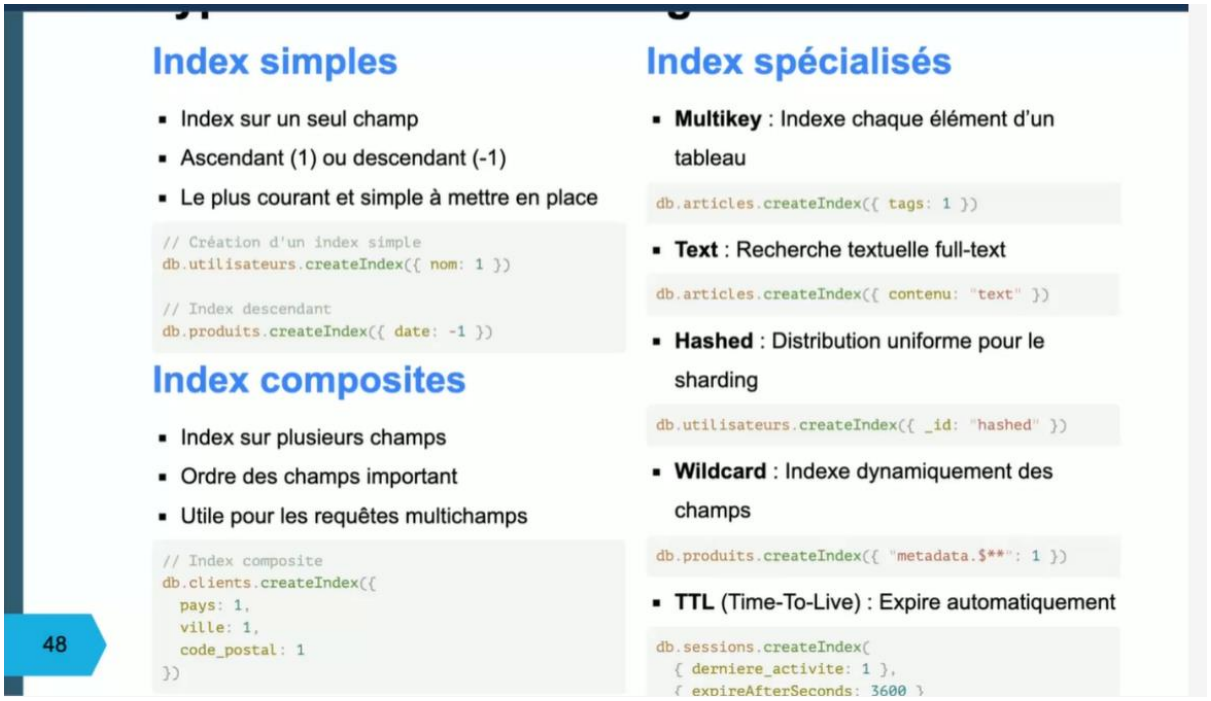
```
db.produits.find({ tags: { $all: ["cuisine", "ustensiles"] } })
```

Update un user en ajoutant une entrée dans table

```
db.utilisateurs.updateOne({nom: "Dupont"}, {$push: {tags: "nouveautag"}})
```

Indexation


HomeLab
Hosting
c
WebDev
web
tc
usefulGits
NetYPareo - Ecole...
Accueil - NetYPareo
Sans index
Avec index
MongoDB parcourt chaque document (COLLSCAN)
Performances linéaires : O(n)
Problématique sur les grandes collections
Consomme beaucoup de ressources
Temps de réponse très lent
Améliore drastiquement les requêtes
Structure de données efficace (B-tree)
Algorithme de recherche optimisé
Performances logarithmiques : O(log n)
Nécessaire pour les applications en production
Impact concret des index
Requête sans index
Collection de 1 million documents
Scan complet
5-10 secondes
Requête avec index
Index B-tree
Recherche arborescente
-5 millisecondes


Index simples
Index sur un seul champ
Ascendant (1) ou descendant (-1)
Le plus courant et simple à mettre en place
// Création d'un index simple
db.utilisateurs.createIndex({ nom: 1 })
// Index descendant
db.produits.createIndex({ date: -1 })
Index composites
Index sur plusieurs champs
Ordre des champs important
Utile pour les requêtes multichamps
// Index composite
db.clients.createIndex({
pays: 1,
ville: 1,
code_postal: 1
})
Index spécialisés
Multikey : Indexe chaque élément d'un tableau
db.articles.createIndex({ tags: 1 })
Text : Recherche textuelle full-text
db.articles.createIndex({ contenu: "text" })
Hashed : Distribution uniforme pour le sharding
db.utilisateurs.createIndex({ _id: "hashed" })
Wildcard : Indexe dynamiquement des champs
db.produits.createIndex({ "metadata.$**": 1 })
TTL (Time-To-Live) : Expire automatiquement
db.sessions.createIndex(
{ derniere_activite: 1 },
48

Création et gestion des index

Création d'index

```
// Index simple avec options
db.produits.createIndex(
  { nom: 1 },
  {
    background: true,
    unique: true,
    sparse: false,
    name: "idx_nom"
  }
)
```

Options importantes

- **background**: Création en arrière-plan
- **unique**: Valeurs uniques uniquement
- **sparse**: Ignore les documents sans le champ
- **partialFilterExpression**: Indexe un sous-ensemble
- **name**: Nom personnalisé pour l'index

Administration des index

```
// Lister tous les index d'une collection
db.produits.getIndexes()

// Statistiques d'utilisation des index
db.produits.aggregate([
  { $indexStats: {} }
])

// Supprimer un index
db.produits.dropIndex("idx_nom")

// Reconstruire tous les index
db.produits.reIndex()
```

Bonnes pratiques

- Indexer les champs utilisés fréquemment en filtres
- Indexer les champs de tri et d'agrégation
- L'ordre des champs doit correspondre aux requêtes

Analyse des performances avec explain()

Modes d'explain()

```
// Mode queryPlanner (par défaut)
db.utilisateurs.find({ age: { $gt: 25 } })
  .explain()

// Mode executionStats
db.utilisateurs.find({ age: { $gt: 25 } })
  .explain("executionStats")

// Mode allPlansExecution
db.utilisateurs.find({ age: { $gt: 25 } })
  .explain("allPlansExecution")
```

Étapes d'exécution (stages)

- **COLLSCAN** : Scan complet (problématique)
- **IXSCAN** : Utilisation d'un index
- **FETCH** : Récupération des documents

Métrique à surveiller

- **nReturned** : Nombre de documents retournés
- **totalKeysExamined** : Entrées d'index examinées
- **totalDocsExamined** : Documents examinés
- **executionTimeMillis** : Temps d'exécution
- **stage** : Type d'opération utilisée

Interprétation

- Idéal : totalKeysExamined ≈ nReturned
- Problème : totalDocsExamined >> nReturned
- Signe de couverture d'index :

Aggrégation

```
db.collection.aggregate([
  { $stage1: { /* paramètres */ } },
  { $stage2: { /* paramètres */ } },
  /* ... plus d'étapes ... */
])
```

Étapes principales (Stages)

- **$match** : Filtre les documents (comme find())
- **$group** : Regroupe par clé(s) et applique des opérations
- **$project** : Sélectionne et transforme les champs
- **$sort** : Trie les résultats
- $limit * *et * *skip : Pagination
- **$unwind** : "Déplie" les tableaux
- **$lookup** : Joint avec une autre collection

```
db.collection.aggregate(
  [ /* pipeline */ ],
  {
    allowDiskUse: true,
    maxTimeMS: 60000,
    collation: { locale: "fr" },
    hint: { indexName: 1 }
  }
)
```

Considérations importantes

- Limite de 100MB par étape (sauf avec allowDiskUse)
- Traitement document par document
- Utilisation des index uniquement aux premières étapes
- Optimisation de l'ordre des étapes
- Complexité croissante avec la taille des données
- Possibilité de créer une vue basée sur une

c