



HephIA

Scalable Intelligence

NoSQL

Qu'est ce que HephIA?

Les risques liés à l'utilisation des IA génératives

Une adoption rapide des IA génératives

Selon Gartner,

- **25% des collaborateurs** utilisent déjà les IA génératives tel que ChatGPT.
- Il seront **plus de 50% d'ici 2025**.



Les problèmes de confidentialité

À chaque utilisation des IA génératives, des informations sont envoyées aux LLM (large language Model) processées et conservées par des tiers, sans connaître les conditions de conservation de ces informations.

Cela pose de nombreux problèmes de confidentialité:

- ✗ Protection de la propriété intellectuelle
- ✗ Protection des données de l'entreprise
- ✗ Compliance GRPD

Franceinfo

Intelligence artificielle : "J'ai commencé à utiliser ChatGPT à l'insu de mon employeur"

Des travailleurs se servent de l'intelligence artificielle générative pour optimiser leur travail, sans forcément en parler à leur supérieur hiérarchique. Une pratique sous les radars qui n'est pas sans risques.

Des réponses possibles

Mais pas toujours simples à mettre en oeuvre

☐ Interdiction d'utiliser les LLM

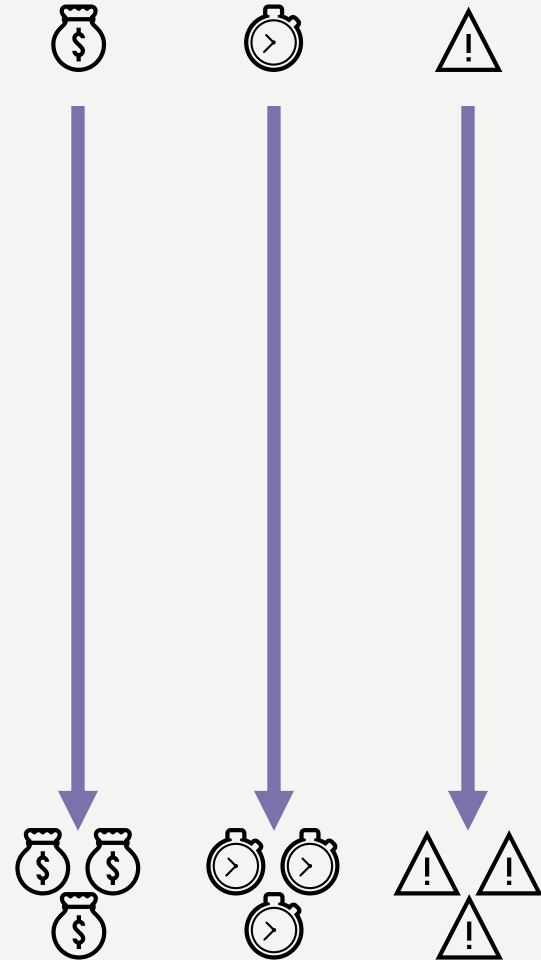
Vos collaborateurs les utiliseront quoiqu'il arrive, car ils sont de plus en plus nombreux, puissants et pertinents.

☐ GPT entreprise, etc.

Long et coûteux à mettre en place
Vos données sont retenues par le LLM
Vos employés utiliseront d'autres LLM

☐ Votre modèle personnalisé

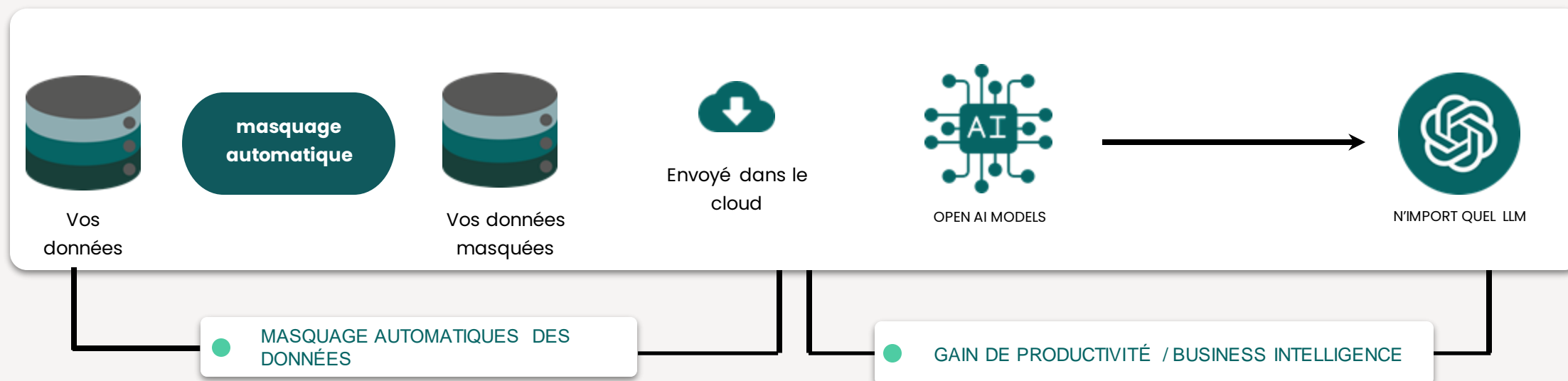
Long et coûteux à mettre en place
C'est une activité à part entière
Le succès n'est pas garanti



SafeSphere™ :

L'assistant d'IA génératif (ChatGPT)
qui assure la sécurité de vos informations
pour une productivité maximale

Description de SafeSphère: le masquage de données



Protégez les données de votre entreprise avec des filtres personnalisés
Augmentez la productivité de vos employés avec des contextes personnalisés

Hephia: Une deeptech reconnue, en pointe sur l'IA

Equipe de direction



Anthony Coutant, CEO

PHD in AI
Background in autonomous vehicles Lab



Yann Girard - COO

Pioneer in Machine learning since 1998
Lead Data teams designing AI entreprise products



Arnaud Besnard - CCO

Pioneer in Machine learning since 1998
Lead Data teams designing AI entreprise products



Mustapha Lebbah - Lead Scientist

AI Professor (Paris Saclay University). 20 years of research in machine learning & AI in entreprise labs (Banking, industry)



Votre serviteur



Brice FOTZO – Ops Guy
brice.fotzo@hephia.com

ERG\NEO

bpi**france**

Initiative
GRANDES ÉCOLES
& UNIVERSITÉS

ÉCOLE
POLYTECHNIQUE

LES
DEEP
TECH

La
FRENCH TECH
SEED

SAFRAN

POCLAIN
Hydraulics

DAVID UVSQ

incub'
Université
Sorbonne
Paris Nord

amias

JEI
Jeune
Entreprise
Innovante

Région
Île de France

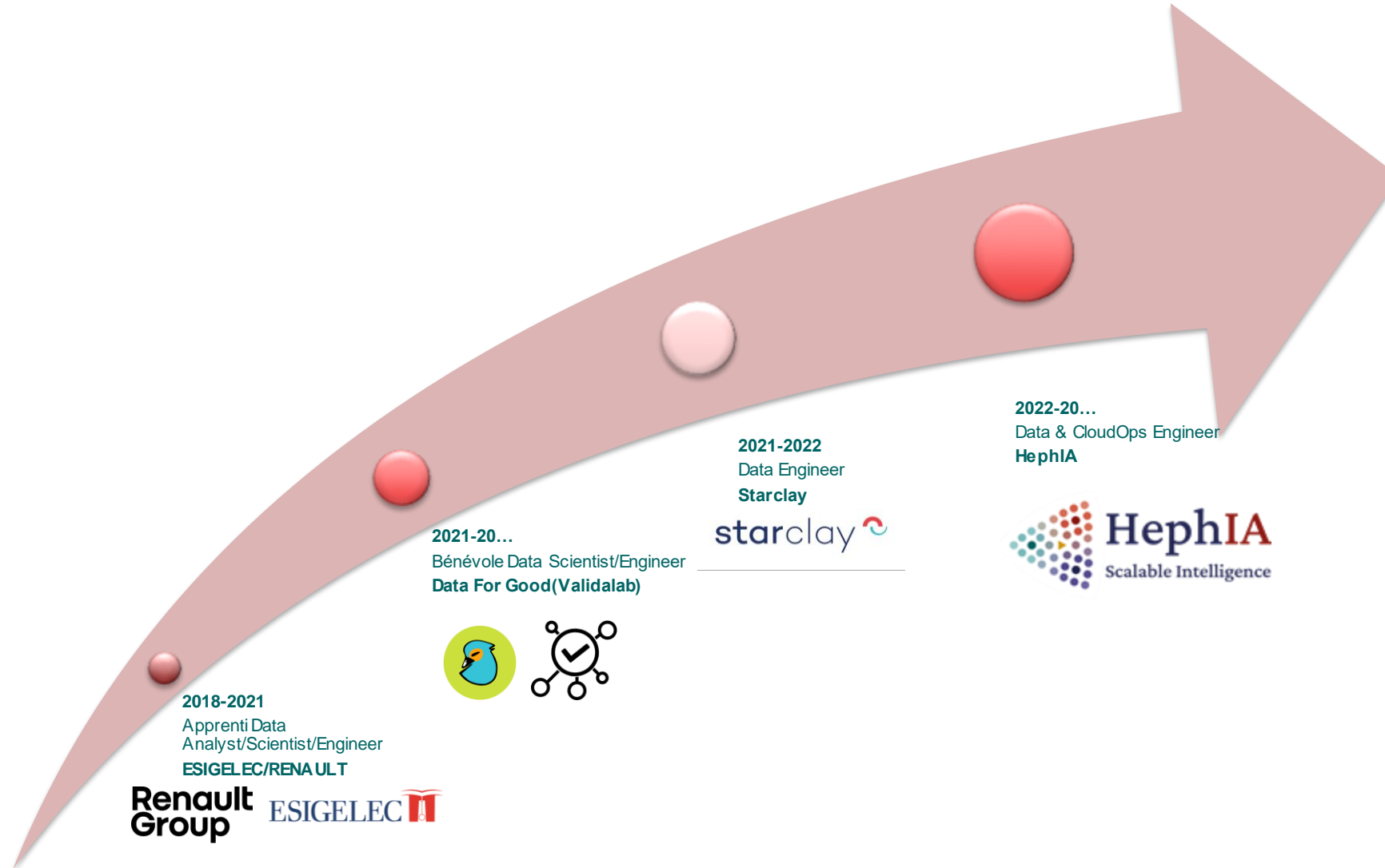
Université
Sorbonne
Paris Nord

EPN

CNRS

G

LA
FRENCH TECH
PARIS-SACLAY



Et vous?



06/11/2023

Introduction NoSQL
BDD Documents
(Mongo)

20/11/2023

Déploiement de
BDD NoSQL

13/11/2023

BDD
graphes(Neo4j)

07/12/2023

Contrôle

1 – Introduction aux BDD NoSQL

SQL (Structured Query Language)

Conçu en 1974, normalisé en 1986

Utilisé dans les SGBD(R)



- ✗ Incapacité à gérer de **très grands volumes** de données à des débits extrêmes
- ✗ Certains types de données ne sont pas adaptés



1- Introduction aux BDD NoSQL

Qu'est ce que le NoSQL?



11 Juin 2009, à San Francisco

Meetup/conference informelle

Organisé par Johan Oskarsson



No SQL? ⚡ Not only SQL?

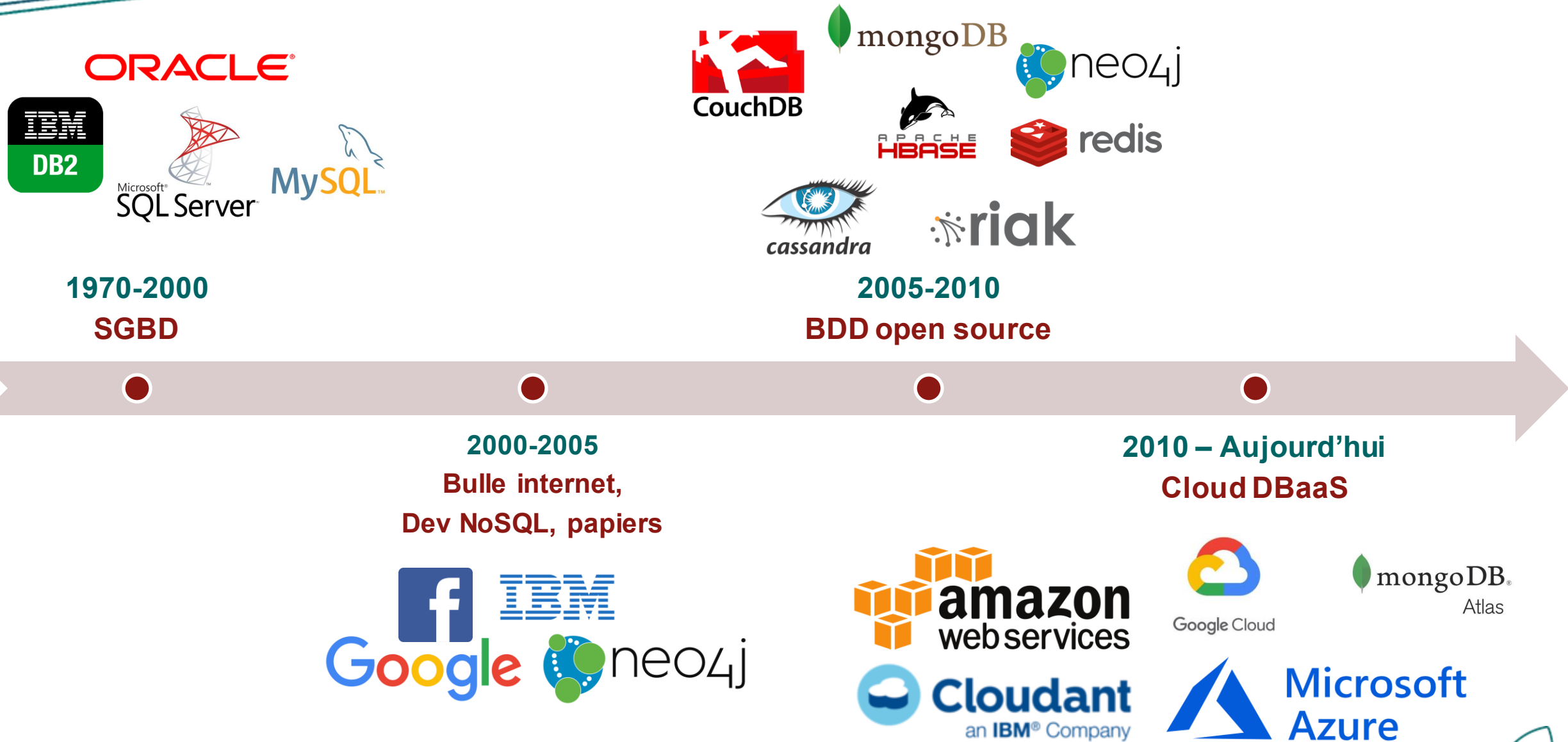
- N'utilisent pas un modèle relationnel (ni le langage SQL)
- Open source
- Conçus pour tourner sur des clusters puissants
- Basés sur les besoins du web au 21^è siècle
- Pas de schema, permet l'ajout de champs/dimensions sans contrôles

Non relationnelle?



1- Introduction aux BDD NoSQL

Qu'est ce que le NoSQL?

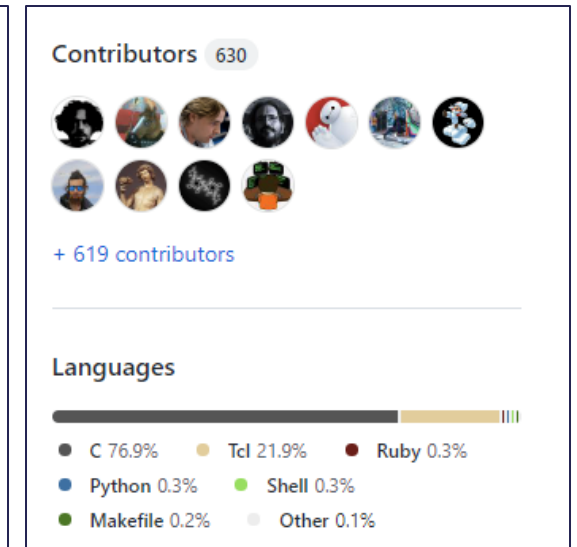
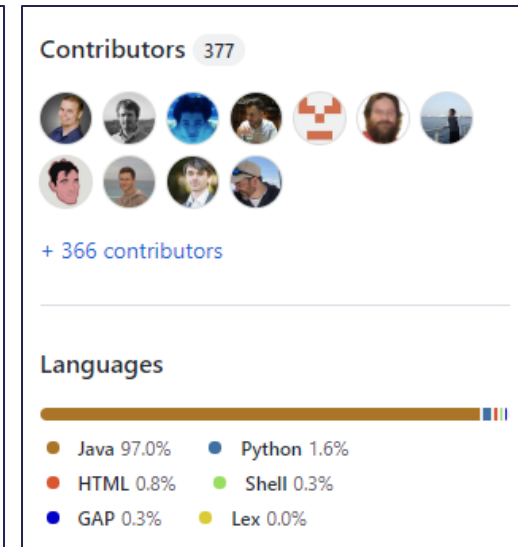
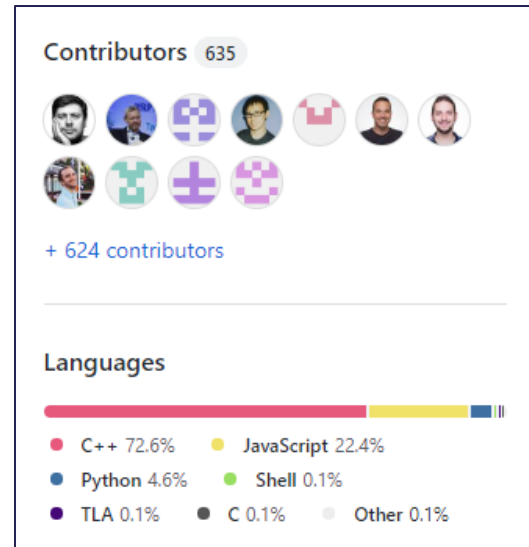
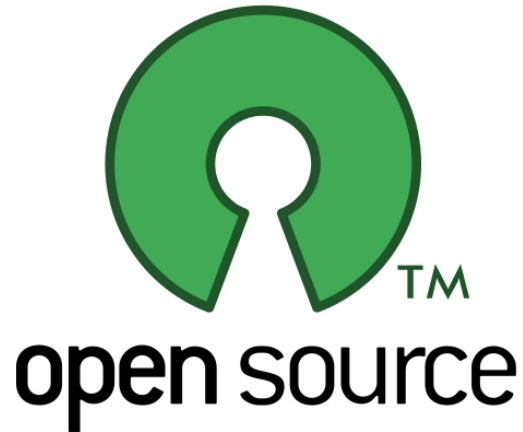


2 – Caractéristiques des BDD NoSQL

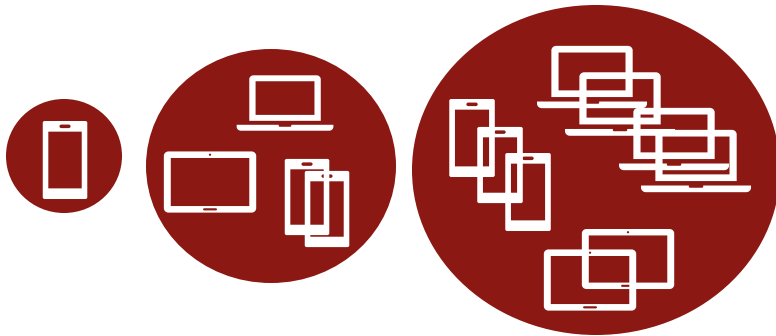
2- Caractéristiques des BDD NoSQL

Communautés impliquées et actives

- Proviennent de l'Open Source ou ont une version Open Source
- Ont été utilisé et exploité de manière open source
- Le support des communautés open source est fondamental pour la croissance du secteur



Scalabilité



Verticale



8vCPUs, 16Go



4vCPUs, 8Go



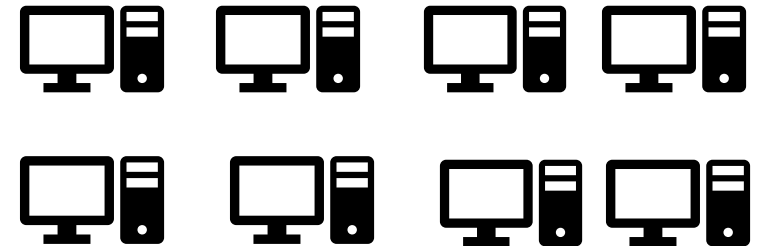
2vCPUs, 4Go



Horizontale



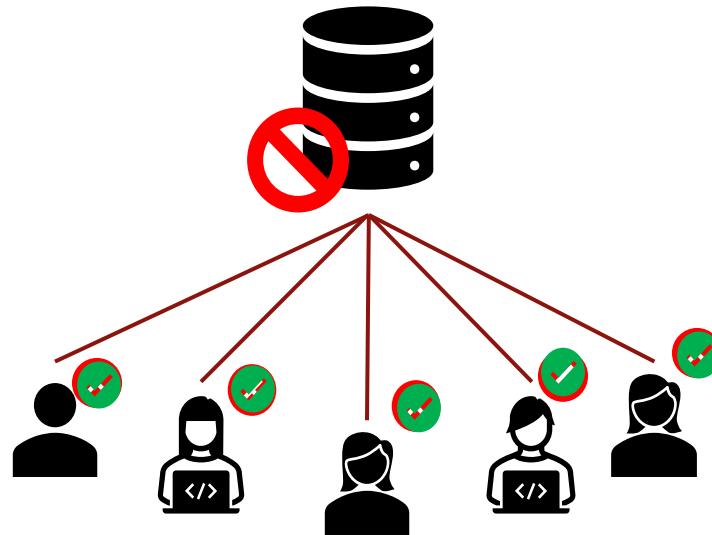
2vCPUs, 4Go * 8



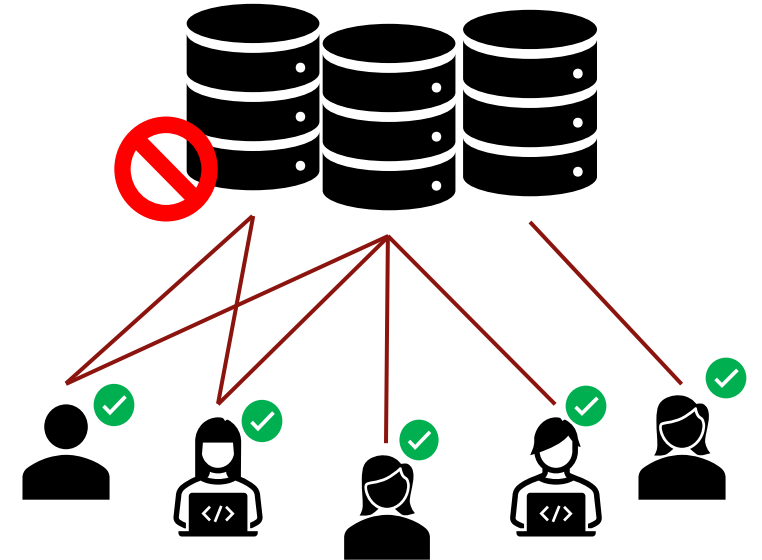
Disponibilité



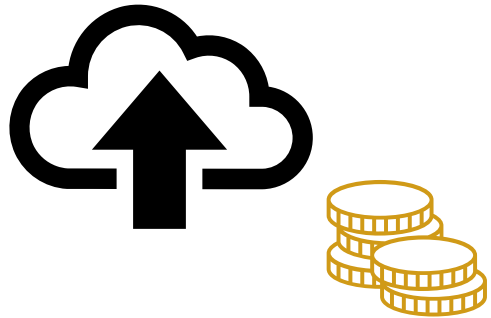
Serveur unique



Cluster de serveurs



Cloud & coûts



- Utilisent des architectures Cloud
- Conçu avec le paradigme Cloud
- Moins chers (serveurs pas chers, open source)
- Implementation facile et peu couteuse en ressources humaines



MongoDB Enterprise Advanced

Enterprise Edition deployed with Oracle RAC

Licence

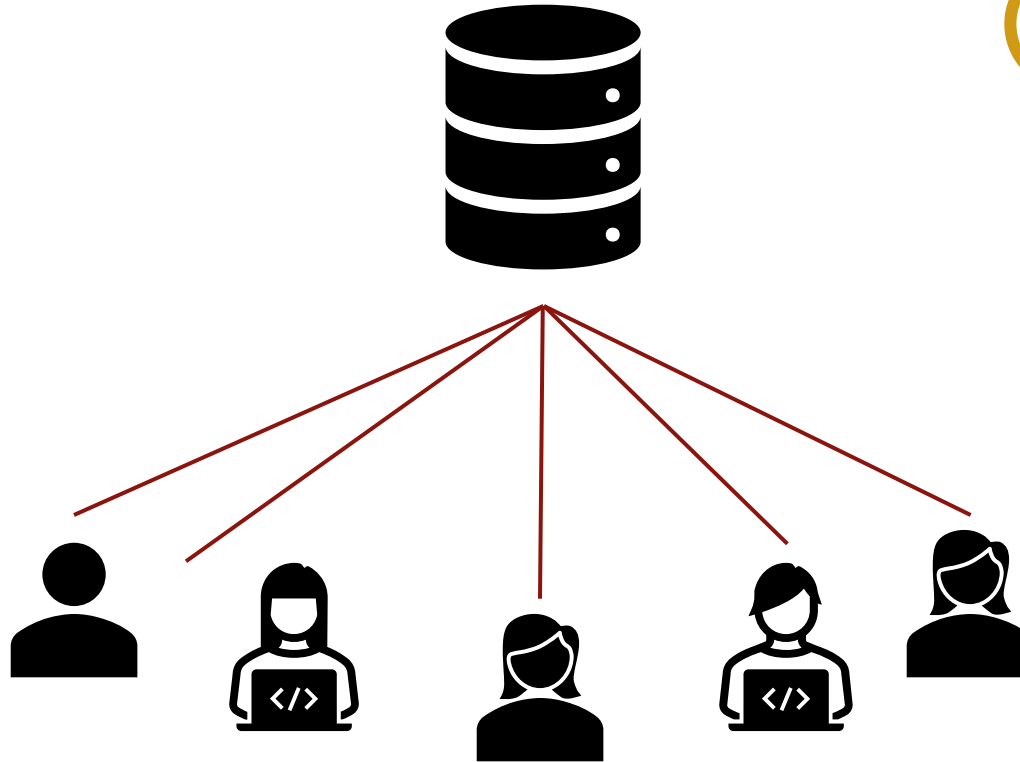
0\$

\$47,500 per unit (sockets * cores per socket * core factor)

Performance



Réponse rapide



Haute concurrence



Flexibilité



- Schéma flexibles
- Types de données variés
- Indexage spécifique

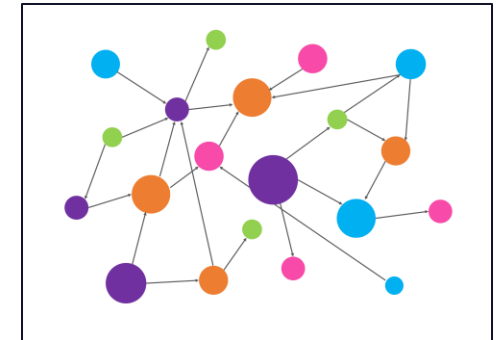
Clé-valeur

key	value
firstName	Bugs
lastName	Bunny
location	Earth

Document

```
{
  "firstName": "Brice",
  "lastName": "De Nice",
  "enrolledDate": ISODate("1990-01-01T14:45:00.000Z"),
  "email": "brice.denice@beach-esigelec.com",
  "hourlySalary": 50.25,
  "teacherId": 20101214,
  "courses": ["surf", "cool-attitude"],
  "address": {
    "city": "Nice",
    "country": "FR",
    "complement": "Plage des Bains militaires"
  }
}
```

Graph



3 – Les types de BDD NoSQL

3- Les types de BDD NoSQL

Les 4 types de base de données NoSQL

Clé-valeur

key	value
firstName	Bugs
lastName	Bunny
location	Earth

Colonnes

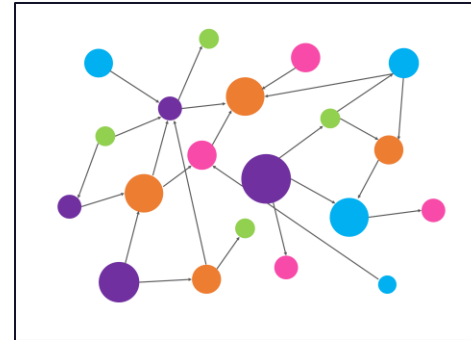
Row-oriented			
ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

Column-oriented	
Name	ID
John	001
Karen	002
Bill	003

Grade	ID
Senior	001
Freshman	002
Junior	003

GPA	ID
4.00	001
3.67	002
3.33	003

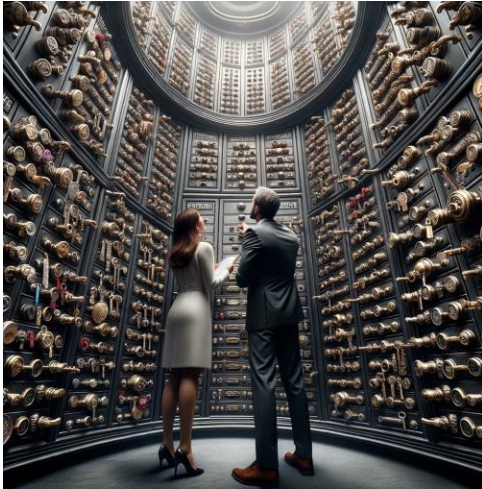
Graph



Document

```
{
  "firstName": "Brice",
  "lastName": "De Nice",
  "enrolledDate": ISODate("1990-01-01T14:45:00.000Z"),
  "email": "brice.denice@beach-esigelec.com",
  "hourlySalary": 50.25,
  "teacherId": 20101214,
  "courses": ["surf", "cool-attitude"],
  "address": {
    "city": "Nice",
    "country": "FR",
    "complement": "Plage des Bains militaires"
  }
}
```





Avantages	Inconvénients
Haute performance: Très rapides pour les opérations de lecture et d'écriture	Fonctionnalités limitées: Pas de requêtes complexes ni de jointures.
Évolutivité: Facilité de mise à l'échelle horizontale	Pas de Relations: Inadapté pour modéliser des relations complexes.
Simplicité: Facile à utiliser et à mettre en œuvre.	Consistance Eventuelle: Ne garantit pas toujours la consistance immédiate des données.

Cas d'usages

Adaptés:

- Cache:** Stockage temporaire de données fréquemment utilisées. (panier, profil user)
- Session Utilisateur:** Stockage rapide et facile des données de session.

Non adaptés:

- Reporting:** Pas idéal pour des requêtes analytiques complexes.
- CRM:** Ne convient pas si de nombreuses relations entre les données sont nécessaires.





Avantages	Inconv�nients
Performance analytique: Rapide pour les requ�tes sur quelques colonnes.	Requ�tes transactionnelles: Peu optimis�es pour les op�rations multi-colonnes.
Compression: Efficace gr�ce � l'homog�n�it� des donn�es.	Co�t d'insertion: Lenteur pour les op�rations sur plusieurs colonnes.
�volutivit�: Facile � mettre � l'�chelle horizontalement.	
Flexibilit� sch�matique: Ajout de colonnes � la vol�e.	

Cas d'usages

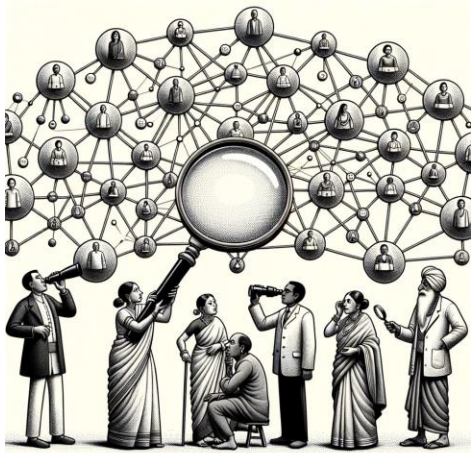
Adapt s:

- Analyse en temps r el: Id ales pour l'analyse rapide de donn es.
- Syst mes de Recommandation: Efficaces pour l'agr gation de donn es.

Non adapt s:

- OLTP: Pas optimis es pour les op rations ACID.
- Relations complexes: Limit es pour les jointures et relations.





Avantages

Relations complexes: Idéal pour modéliser des relations complexes.

Requêtes de Graphe: Langages de requête spécialisés pour des opérations complexes.

Intuitivité: Le modèle de graphe est souvent plus naturel pour représenter des données interconnectées.

Inconvénients

Performance: Peut être plus lent pour des requêtes qui ne nécessitent pas de relations.

Distributivité: N'est pas nativement shardable. Risque de perte des relations

Coût de stockage: Peut nécessiter plus d'espace de stockage pour stocker les relations.

Cas d'usages

Adaptés:

- **Réseaux Sociaux:** Idéal pour modéliser les relations entre les personnes.
- **SEO:** Utile pour comprendre les relations entre les pages web et leur contenu.

Non adaptés:

- **Systèmes OLAP:** Pas optimisé pour des requêtes analytiques sur de gros volumes de données.
- **Logistique:** Pas idéal si les relations entre les entités ne sont pas cruciales.



3- Les types de BDD NoSQL

Orienté Document



Avantages	Inconvénients
Flexibilité schématique: Pas de schéma fixe	Relations: pas idéales pour des relations complexes entre documents.
Requêtes complexes: Supporte de requêtes plus complexes grâce à la structure imbriquée des documents.	Incohérence: Risque d'incohérence des données si la structure des documents change au fil du temps.
Performance: Optimisé pour des opérations de lecture et écriture rapides sur des documents complets.	Stockage: Le stockage peut être plus volumineux en raison de la répétition des noms de champs.

Cas d'usages

Adaptés:

- **CMS:** Idéal pour des systèmes de gestion de contenu où chaque document peut avoir une structure différente.
- **Catalogues de produits:** Utile pour stocker des informations sur des produits aux attributs très variés.

Non adaptés:

- **Systèmes financiers:** Pas idéal pour des transactions ACID strictes.
- **Réseaux Sociaux:** Limité pour gérer les relations complexes entre utilisateurs.



4- Comment choisir?



Critères	Key-Value	Column-family	Document	Graph
Volume de données	Moyen à Grand	Grand	Moyen à Grand	Moyen
Complexité des relations	Faible	Faible	Moyenne	Elevée
Vitesse d'écriture et de lecture	Très élevée	Très élevée	Moyenne	Moyenne à Faible
Requêtes complexes	Non	Non	Oui	Oui
Flexibilité du schéma	Elevée	Moyenne	Très élevée	Très élevée
Transactions ACID	Non	Non	Non	Oui
Cohérence	Configurable	Configurable	Configurable	Forte
Disponibilité	Haute	Haute	Très haute	Configurable
Partition Tolérant	Oui	Oui selon config	Oui	Non sauf configuration



4 – Introduction aux BDD Documents (Mongo)

4- Introduction aux BDD documents (Mongo)

Qu'est ce que MongoDB?

BDD NoSQL orientée **documents**(JSON)

Modélisation à l'écriture

```
{
  "title": "Once Upon a Time in the West",
  "year": 1968,
  "rated": "PG-13",
  "runtime": 175,
  "countries": ["Italy", "USA", "Spain"],
  "genres": ["Western"],
  "director": "Sergio Leone",
  "writers": ["Sergio Donati", "Sergio Leone", "Dario Argento", "Bernardo Bertolucci"],
  "actors": ["Claudia Cardinale", "Henry Fonda", "Jason Robards", "Charles Bronson"],
}
```

Mantra:

Les données interrogées ensemble, devraient être stockées ensemble

Dénormalisation:

Redondance autorisée

Structure modifiable à l'écriture



Database: **schoolManagement**

Collection: **students**

Documents

```
{
  "firsrstName": "Encorvou",
  "lastName": {
    "firsrstName": "Dora",
    "lastName": {
      "firsrstName": "Diego",
      "lastName": {
        "firsrstName": "Son",
        "lastName": "Goku",
        "email": "son.goku@esigelec.com",
        "studentId": 20225454816
      }
    }
  }
}
```

Collection: **teachers**

Documents

```
{
  "firsrstName": "Tortue",
  "lastName": "Geniale",
  "email": {
    "teacherId": {
      "firsrstName": "Brice",
      "lastName": "De Nice",
      "email": {
        "teacherId": {
          "firsrstName": "Tremai",
          "lastName": "Dayo",
          "email": "tremai.dayo@beach-esigelec.com",
          "teacherId": 20105454812,
          "courses": ["sword", "philosophy", "jedi"]
        }
      }
    }
  }
}
```



4- Introduction aux BDD documents (Mongo)







Les types de données supportés par Mongo DB

MongoDB prend en charge tous les types qui relèvent du type BSON

byte	1 byte (8-bits)
int32	4 bytes (32-bit signed integer, two's complement)
int64	8 bytes (64-bit signed integer, two's complement)
uint64	8 bytes (64-bit unsigned integer)
double	8 bytes (64-bit IEEE 754-2008 binary floating-point)
decimal128	16 bytes (128-bit IEEE 754-2008 decimal floating-point)
date	8 bytes(64-bit integer)
objectId	12 bytes(4-byte timestamp value, 5-byte random value, and 3-byte incrementing counter)
array	Storage is based on data (A byte array uses 1 byte, a short array uses 2 bytes, and an integer array uses 4 bytes)

```
{
  "firsstName": "Brice",
  "lastName": "De Nice",
  "enrolledDate": ISODate("1990-01-01T14:45:00.000Z"),
  "email": "brice.denice@beach-esigelec.com",
  "hourlySalary": 50.25,
  "teacherId": 20101214,
  "courses": ["surf", "cool-attitude"],
  "address": {
    "city": "Nice",
    "country": "FR",
    "complement": "Plage des Bains militaires"}
}
```



-  Très utilisé et bien documenté
-  Schéma flexible et évolutif
-  Approche code-first
-  Requête complexe sur les valeurs
-  Hautement disponible
-  Horizontalement scalable





Vue unique

Vue en temps-réel de différentes sources de données



IoT(Internet of Things)

Analyser et exploiter les données relevées



Mobile Apps

Développement rapide et facile des applications



Personnalisation

Contenu personnalisé spécifique aux utilisateurs



Jeux-vidéos

Jeux vidéos mondiaux, scalables, rapides, robustes...



Analytique

Concevoir des applications orientées analytiques





Les démarrages de projets de **Dev** sont souvent complexes

Les problèmes de démarrage

Le développement logiciel moderne exige rapidité, efficacité et apprentissage continu face à l'évolution constante des technologies.

Cela pose de nombreux problèmes:

- ✗ Choix d'outils et stack long
- ✗ Configurations et mises en place difficiles
- ✗ Omissions de composants importants

Des réponses possibles



Pas de recommandation

Vous devez savoir à l'avance quel template vous voulez ou les analyser les templates et choisir s'il y en a un qui vous convient

Flexibilité limitée

Les templates ne sont pas toujours tenus à jour et il n'est pas toujours évident de le faire soit même.

Bases de connaissances inexistantes

Ne fournis pas de mécanismes pour apprendre des erreurs passées ou des succès collectifs.



Codez propre, créez plus, apprenez
toujours.

Description de Gocod: la creation assistée



● Création assistée de projets

Créer votre projet en 2-3 minutes en répondant à quelques questions.

● Bases de connaissances

Transformez vos erreurs et bug de code en une source de connaissance riche

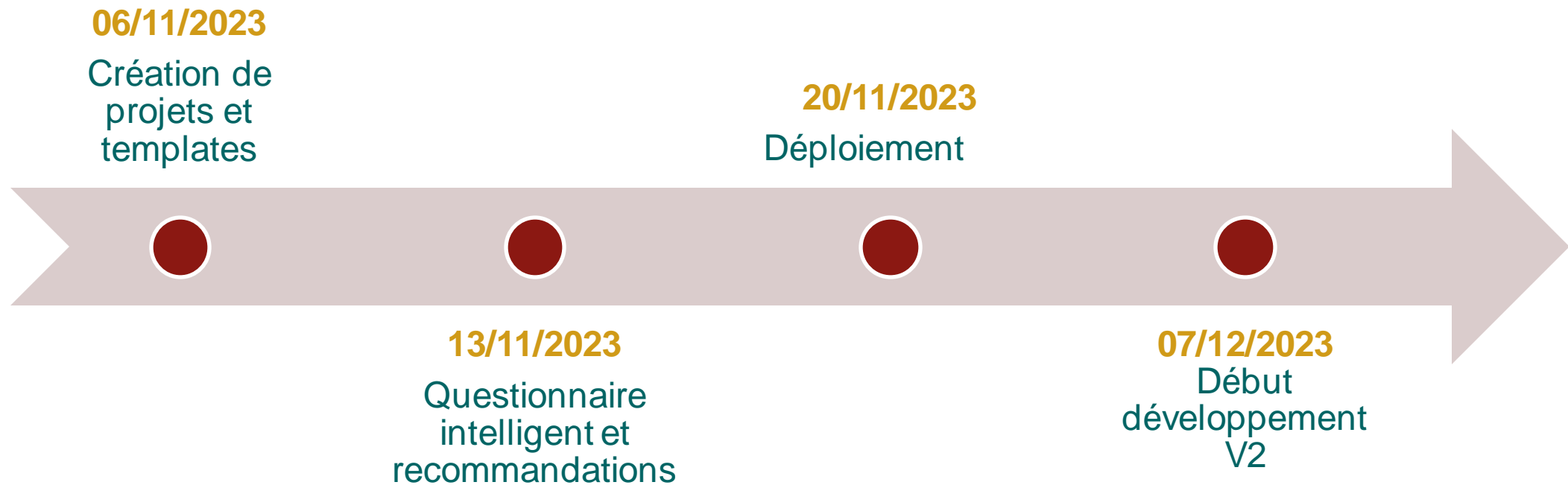
● Recommandations personnalisées

Recevez des recommandations d'outils et best-practices à intégrer

● Optimisation du Workflow

Réduisez vos temps de configuration initial, avec un démarrage de projet en quelques minutes.

Roadmap produit



Go TP!

Recap TP!

Setup, Mongo Basics, Pymong

● Setup facile via Docker

Pas besoin de télécharger plein de fichiers

● CRUD

Des opérations de création, lecture, modification et suppression simples

● Inspection du Docker Compose

Permet de lancer plusieurs services à la fois

● Suite

GoCod



Démarrage projet Gocod



● Constitution des groupes

● Acceptation de l'assignement

https://classroom.github.com/a/3D0_Stan

● Consignes détaillées

<https://github.com/nosql-esigelec/GoCod#readme>

● Livrables

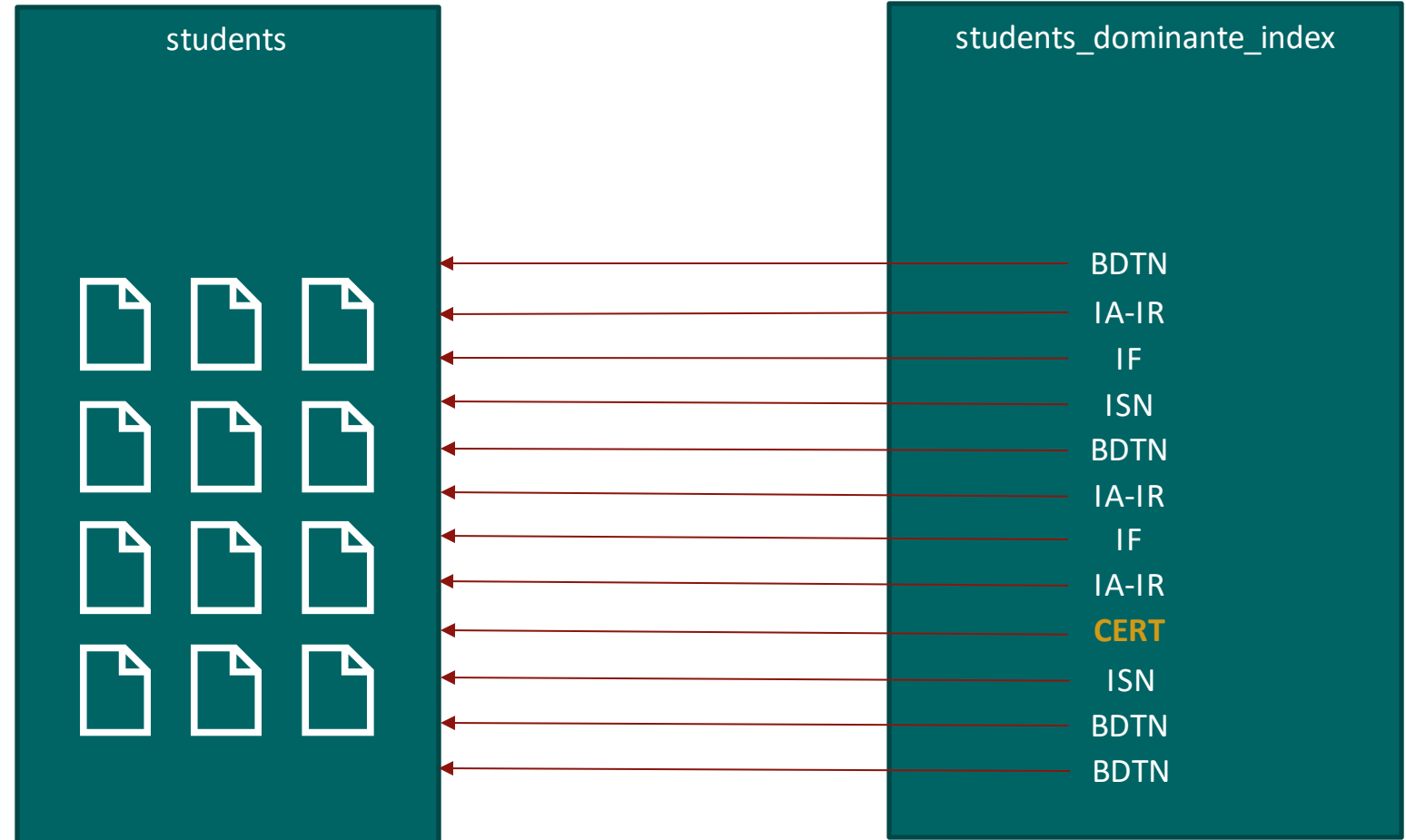
https://github.com/nosql-esigelec/inge3a_24/wiki/Ressources

5 – Introduction aux BDD Documents (Mongo)

- Limite le nombre de documents à scanner
- Optimise et accélère les requêtes

Différents types d'index

- Indexes à champ unique
- Indexes à champs multiples
- Indexes textuel
- Indexes géographiques



Create

- **insertOne()** : Ajoute un document unique
- **insertMany()** : Ajoute plusieurs documents
- **Créer avec insertOne**: Créer le film inception paru en 2010
`db.mflix.insertOne({ title: "Inception", year: 2010 })`
- **Créer avec insertMany()** : Créer les films The Matrix et Interstellar

```
> db.mflix.insertMany([  
  { title: "The Matrix", year: 1999 },  
  { title: "Interstellar", year: 2014 }  
])
```



Read

- **find()** : Récupère des documents
- **findOne()**: Récupère un document
- **Projection**: Sélection de champs spécifiques
- Trier: **sort()**, Limiter: **limit()**, Passer: **skip()**

- Recherche avec **projection**: Afficher les titres de films de genre Sci-Fi
`db.mflix.find({ genre: "Sci-Fi" }, {"title": 1})`
- Recherche avec **tri** et **limite** : Trier les 5 derniers(selon l'année) films Sci-Fi
`db.mflix.find({ genre: "Sci-Fi" }).sort({ year: -1 }).limit(5)`



Update

- **updateOne()** : Met à jour un seul document
- **updateMany()** : Met à jour plusieurs documents
- Opérateurs:
 - \$set** : Modification de champs spécifiques
 - \$push** : Ajout d'éléments à une liste
 - \$pull** : Retrait d'éléments d'une liste
- Mettre à jour avec **updateMany()** : Dénier un Award à tous les films Sci-Fi

```
db.mflix.updateMany({ genre: "Sci-Fi" }, { $set: { award: true } })
db.mflix.updateMany(
    { title: { $in: ["Inception", "The Wolf of Wall Street"] } },
    { $push: { cast: "Leonardo DiCaprio" } }
)
```



Delete

- **deleteOne()** : Supprime un document unique
- **deleteMany()** : Supprime plusieurs documents
- Supprimer avec **deleteMany()** :
`db.mflix.deleteMany({ year: { $lt: 2000 } })`

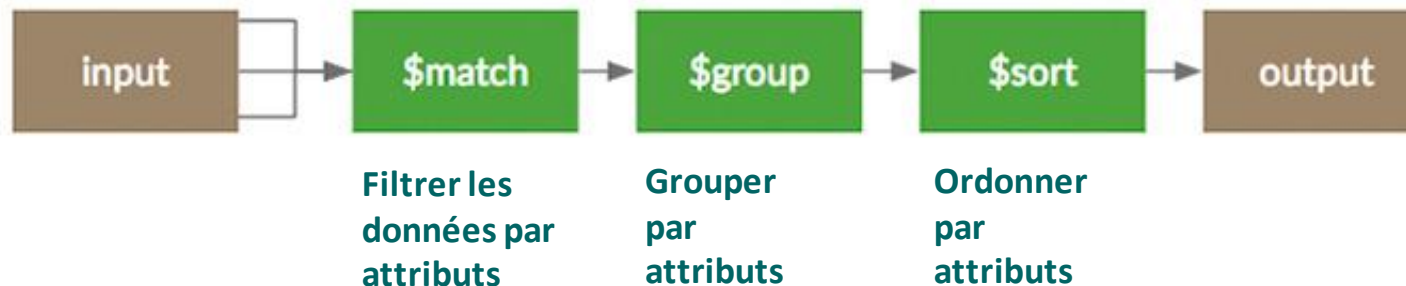


- Chaînes de traitements
- Composés d'étapes de traitement (input => traitement => output)
- Chaque unité de traitement fait intervenir une expression
- Expressions: fonctions de traitement(\$match, \$project, \$group, \$sort, etc...)



mongoDB

Aggregation Framework



w:1 est writeConcern par défaut, il s'assure les données ont été écrites par au moins un noeud(en l'occurrence le noeud primaire qui reçoit en premier les données)

w:majority s'assure que les écritures ont été faites par la majorité des nœuds
Moins rapide mais sûre et surtout plus cohérent(consitent)

w:0 ne s'assure pas qu'il y ait eu une écriture par un quelconque noeud
Très rapide mais moins cohérent, moins durable, à utiliser dans des cas bien spécifiques, envoi de signal(IoT)



1 – BASE vs ACID

A

Atomic

Tout ou rien,
Chaque transaction est **OK** ou **NOK**.

C

Consistent

Intégrité des données conservées après
chaque transaction.

I

Isolated

Les transactions sont complètement isolées
et ne peuvent avoir d'effet sur d'autre.

D

Durable

Les données reliées à une transaction sont
persistantes et sont retrouvables quelque
soit le moment.

VS

BA

Basically Available

Toute requête reçoit une réponse. Y compris
'Success' ou 'Failed'.

S

Soft state

Le système peut changer d'état sans
intervention/événement due à la consistance
éventuelle.

E

Eventually consistent

Si le serveur ne reçoit plus de requête, on est
consistant. Par contre pendant qu'il en reçoit,
on l'est en partie.



2- Théorème CAP

3- Théorème CAP?

Consistance, Disponibilité ou Tolérance aux partitions?

C

Consistency

Tous les clients/nœuds du système voient les mêmes données au même moment.

A

Availability

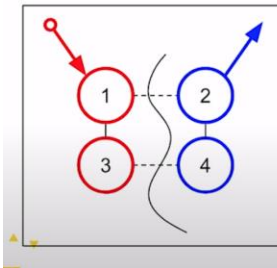
Chaque requête doit avoir une réponse (succès ou échec) y compris lorsqu'il y a des nœuds en échec.

P

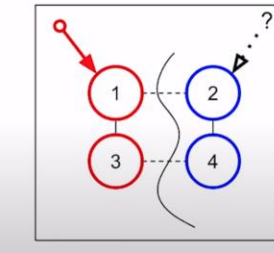
Partition tolerant

Aucune défaillance/échec de tout ou partie des nœuds du cluster ne doit empêcher le système de répondre correctement.

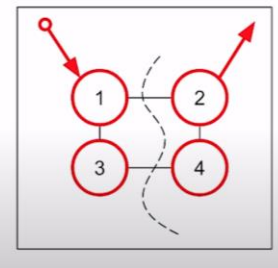
Partition Tolerant +
Available = **Not Consistent**



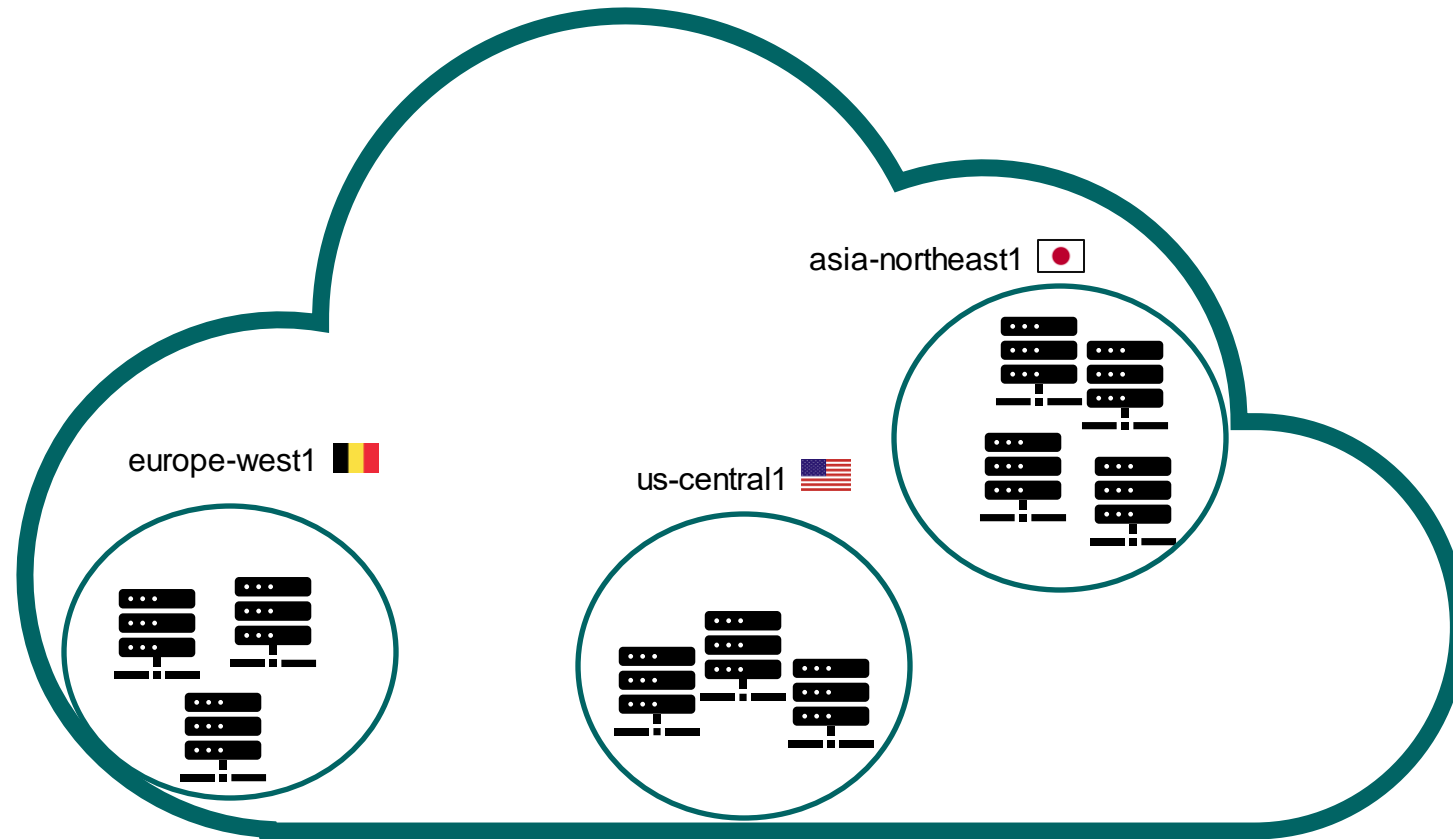
Partition Tolerant +
Consistent = **Not Available**



Consistent + Available =
Not Partition Tolerant

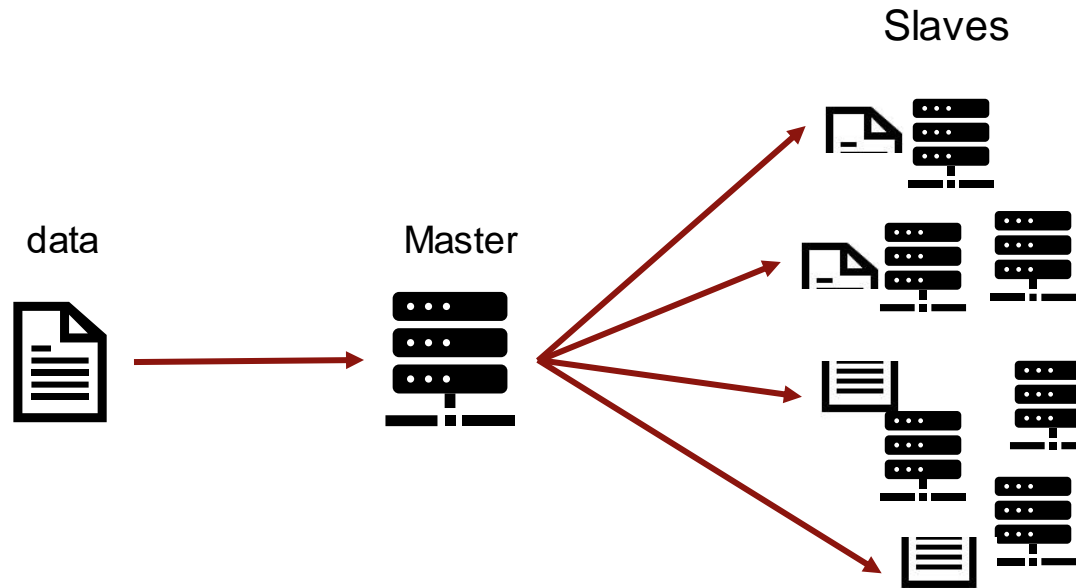


3- Stockage distribué



Partitionnement/Sharding

Replication  Backup



- + Robustesse et disponibilité
- + Performances améliorées
- + Temps de requêtage réduit
- + Facilité de scaling
- + Disponibilité continue



4- Déploiement des bases de données

5- Déploiement de bases de données

Enjeux et questions à se poser

Enjeux majeurs:

- Où héberger la BDD?
- Comment gérer la BDD? En interne? En externe?



Garanti:

- ✓ Uptime
- ✓ Disponibilité
- ✓ Scalabilité

Do-It-Yourself	DataBase as a Service
Hardware <ul style="list-style-type: none">- Achat du matériel- Installation du matériel- Entretien du matériel	Hardware <ul style="list-style-type: none">- Géré par DBaaS
Software <ul style="list-style-type: none">- Installation et gestion des systèmes d'exploitation;- Installation et configuration des bases de données- Administration, sécurité, réseaux, etc...	Software <ul style="list-style-type: none">- Géré par DBaaS
Application <ul style="list-style-type: none">- Conception de l'architecture, le modèle de données, etc...- Conception de l'application	Application <ul style="list-style-type: none">- Conception de l'architecture, le modèle de données, etc...- Conception de l'application, modèle de données



5- Jointures(lookup)

5- Jointures

Comment faire des jointures en MongoDB

- Aggregation pipeline
- Expression de traitement: **\$lookup**

Exemple: Collecter les commentaires pour un post

```
[
  {
    "title": "my first post",
    "author": "Jim",
    "likes": 5
  },
  {
    "title": "my second post",
    "author": "Jim",
    "likes": 2
  },
  {
    "title": "hello world",
    "author": "Joe",
    "likes": 3
  }
]
```

Posts

```
[
  {
    "postTitle": "my first post",
    "comment": "great read",
    "likes": 3
  },
  {
    "postTitle": "my second post",
    "comment": "good info",
    "likes": 0
  },
  {
    "postTitle": "my second post",
    "comment": "i liked this post",
    "likes": 12
  },
  {
    "postTitle": "hello world",
    "comment": "not my favorite",
    "likes": 8
  },
  {
    "postTitle": "my last post",
    "comment": null,
    "likes": 0
  }
]
```

Commentaires

```
db.posts.aggregate([
  {
    $lookup:
    {
      from: "comments",
      localField: "title",
      foreignField: "postTitle",
      as: "comments"
    }
  }
])
```

Requête

- from**: la collection cible de notre jointure
- localField**: le champ que nous voulons joindre dans la collection locale (la collection sur laquelle on exécute la requête .ie **posts**)
- foreignField**: le champ que nous voulons joindre dans la collection cible (dans notre cas **comments**)
- as**: le nom du champ de résultat



5- Jointures

Comment faire des jointures en MongoDB?

- Aggregation pipeline
- Expression de traitement: **\$lookup**

Exemple: Collecter les commentaires pour un post

```
[
  {
    "title": "my first post",
    "author": "Jim",
    "likes": 5
  },
  {
    "title": "my second post",
    "author": "Jim",
    "likes": 2
  },
  {
    "title": "hello world",
    "author": "Joe",
    "likes": 3
  }
]
```

Posts

```
[
  {
    "postTitle": "my first post",
    "comment": "great read",
    "likes": 3
  },
  {
    "postTitle": "my second post",
    "comment": "good info",
    "likes": 0
  },
  {
    "postTitle": "my second post",
    "comment": "i liked this post",
    "likes": 12
  },
  {
    "postTitle": "hello world",
    "comment": "not my favorite",
    "likes": 8
  },
  {
    "postTitle": "my last post",
    "comment": null,
    "likes": 0
  }
]
```

Commentaires

```
db.posts.aggregate([
  {
    $lookup:
    {
      from: "comments",
      localField: "title",
      foreignField: "postTitle",
      as: "comments"
    }
  }
])
```

Requête

```
[
  {
    "title": "my first post",
    "author": "Jim",
    "likes": 5,
    "comments": [
      {
        "postTitle": "my first post",
        "comment": "great read",
        "likes": 3
      }
    ]
  },
  ]
```

Résultat



Jointure avec conditions

Exemple: Collecter les commentaires pour chaque post lorsque les commentaires sont likés que les posts

```
db.posts.aggregate([
  {
    $lookup:
    {
      from: "comments",
      let: { post_likes: "$likes", post_title: "$title" },
      pipeline: [
        {
          $match:
          {
            $expr:
            {
              $and:
              [
                { $gt: ["$likes", "$$post_likes"] },
                { $eq: ["$$post_title", "$postTitle"] }
              ]
            }
          }
        },
        {
          $project:
          {
            _id: 1,
            as: "comments"
          }
        }
      ]
    }
  },
  {
    $project:
    {
      _id: 1,
      comments: 1
    }
  }
])
```

Requête

- **let (optional)**: une expression déclarant les variables à utiliser dans l'étape de pipeline. Cela permet au pipeline, d'accéder aux champs de la collection locale (**posts** dans notre cas)
- **pipeline**: une aggregation pipeline à exécuter sur la collection à joindre (**comments**)



Jointure avec conditions

Exemple: Collecter les commentaires pour chaque post lorsque les commentaires sont liés que les posts

```
db.posts.aggregate([
  {
    $lookup:
    {
      from: "comments",
      let: { post_likes: "$likes", post_title: "$title" },
      pipeline: [
        {
          $match:
          {
            $expr:
            {
              $and:
              [
                { $gt: ["$likes", "$$post_likes"] },
                { $eq: ["$$post_title", "$postTitle"] }
              ]
            }
          }
        },
        {
          $project:
          {
            _id: 1,
            post_likes: 1,
            post_title: 1,
            comment: 1
          }
        }
      ],
      as: "comments"
    }
  }
])
```

Requête

Résultat

```
[
  {
    "title": "my second post",
    "author": "Jim",
    "likes": 2,
    "comments": [
      {
        "postTitle": "my second post",
        "comment": "i liked this post",
        "likes": 12
      }
    ]
  },
  {
    "title": "hello world",
    "author": "Joe",
    "likes": 3,
    "comments": [
      {
        "postTitle": "hello world",
        "comment": "not my favorite",
        "likes": 8
      }
    ]
  }
]
```



Go TP!