



HephIA
Scalable Intelligence

NoSQL

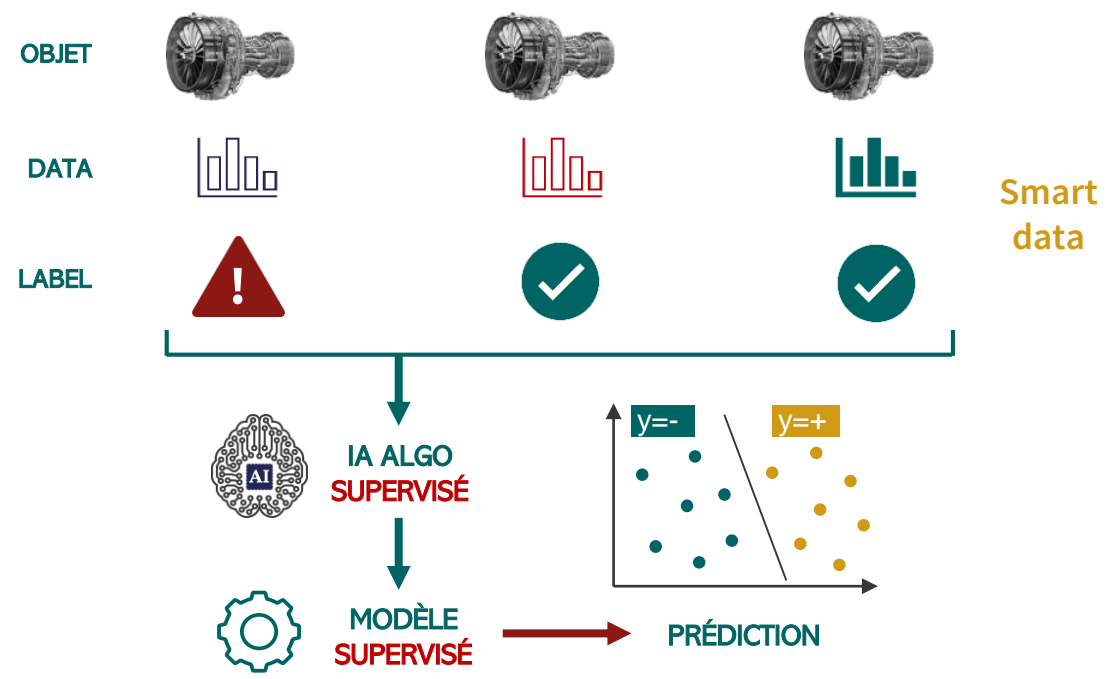
A propos de HephIA?



Motivation

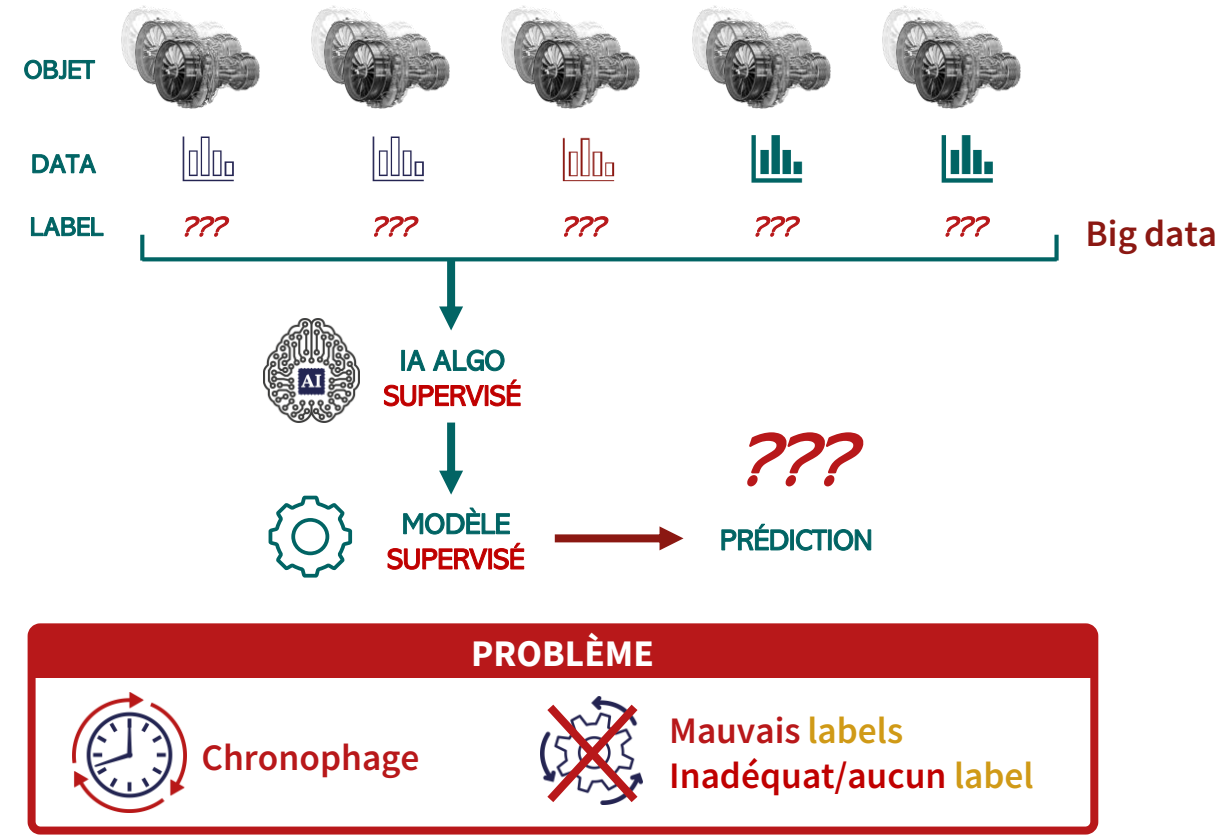
Cas d'utilisation : Jumeau numérique pour décarboner les moteurs d'avions

Le projet



Dans un cas idéal, les données sont bien rangées, en quantité juste suffisantes, représentatives du problème à traiter, étiquetées pour réaliser de belles prédictions grâce à l'IA supervisée.

La réalité



En réalité, les données sont massives, chaotiques, difficiles à explorer manuellement et 97% des données n'ont pas d'étiquette, rendant toute utilisation d'une IA supervisée prédictive fastidieuse.

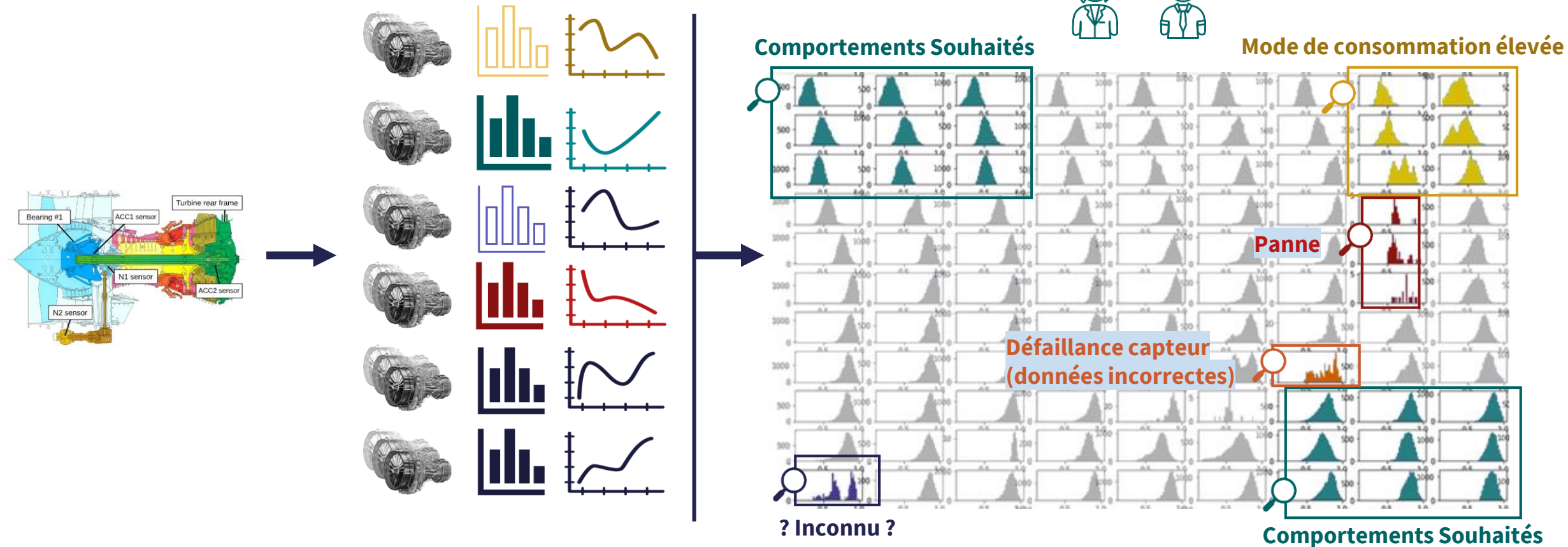


HephIA : gouvernance Big data grâce à compréhension INTÉGRALE de la donnée

Amont (Audit Capacité) | Pendant (Gain Temps) | Aval (Audit Qualité)

Exemple de Surveillance de l'état des turboréacteurs d'avions avec l'aide de l'IA non supervisée

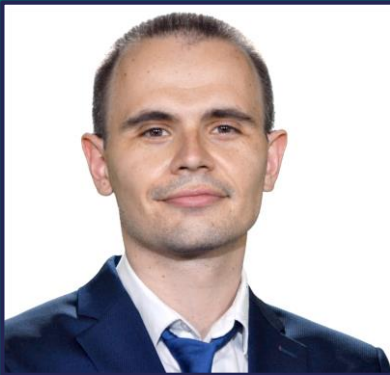
Jumeau numérique —> Des millions d'exemples —> Résumé en quelques archétypes expressifs



LES DONNÉES SE METTENT À PARLER



HephIA : 3 doctorats en IA & 1 expert en ingénierie des données de l'industrie



**Anthony COUTANT,
Ph.D.**

- ▶ Ph.D. IA & Master MIAGE
- ▶ Expert en IA et génie logiciel
- ▶ > 7 ans XP dans les projets académiques et industriels
- ▶ Vision de l'outil / du produit



Gaël BECK, Ph.D.

- ▶ Ph.D. @Scale d'IA non supervisée
- ▶ > 4 ans XP champ Segmentation de marché par IA
- ▶ Développeur principal C4E



Mustapha LEBBAH, PhD.

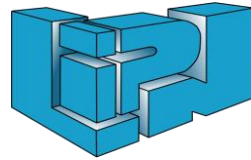
- ▶ HDR Non Supervisé IA @Scale
- ▶ Maître de Conférences au LIPN et à l'USPN
- ▶ > 20 ans R&D projets académiques & industriels : forte expertise
- ▶ Solides partenariats historiques avec le secteur privé



Yann GIRARD

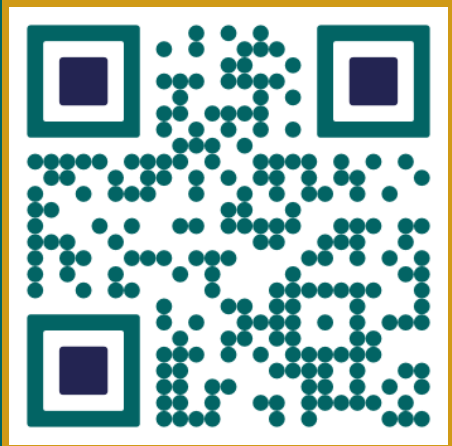
- ▶ > 20 ans XP en Machine learning
- ▶ Expert en adaptation de la data science aux domaines métiers et aux défis business
- ▶ Responsable d'équipes de données et d'industrialisation de projets

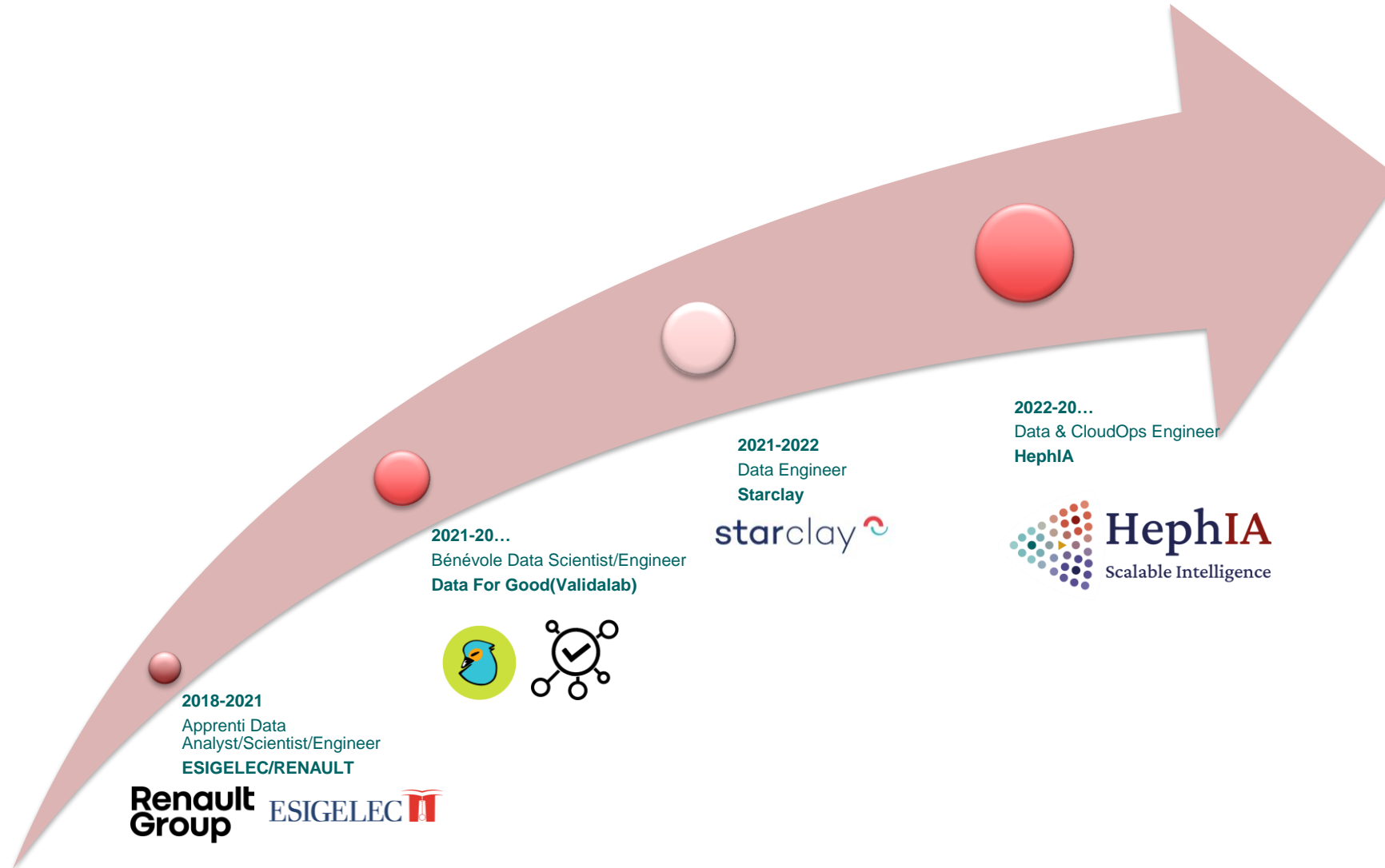
Nos partenaires



HephIA
Scalable Intelligence

Brice FOTZO
Data & CloudOps Engineer
brice.fotzo@hephia.com
+33 6 60 88 46 39
hephia.com





Renault Group ESIGELEC



1 – Introduction aux BDD NoSQL

SQL (Structured Query Language)

Conçu en 1974, normalisé en 1986

Utilisé dans les SGBD(R)

The Oracle logo, consisting of the word "ORACLE" in a bold, red, sans-serif font.

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



1- Introduction aux BDD NoSQL

Qu'est ce que le NoSQL?



11 Juin 2009, à San Francisco

Meetup/conference informelle

Organisé par Johan Oskarsson

Discussions au tour de BDD non relationnelles



NoSQL?

No SQL  Not only SQL

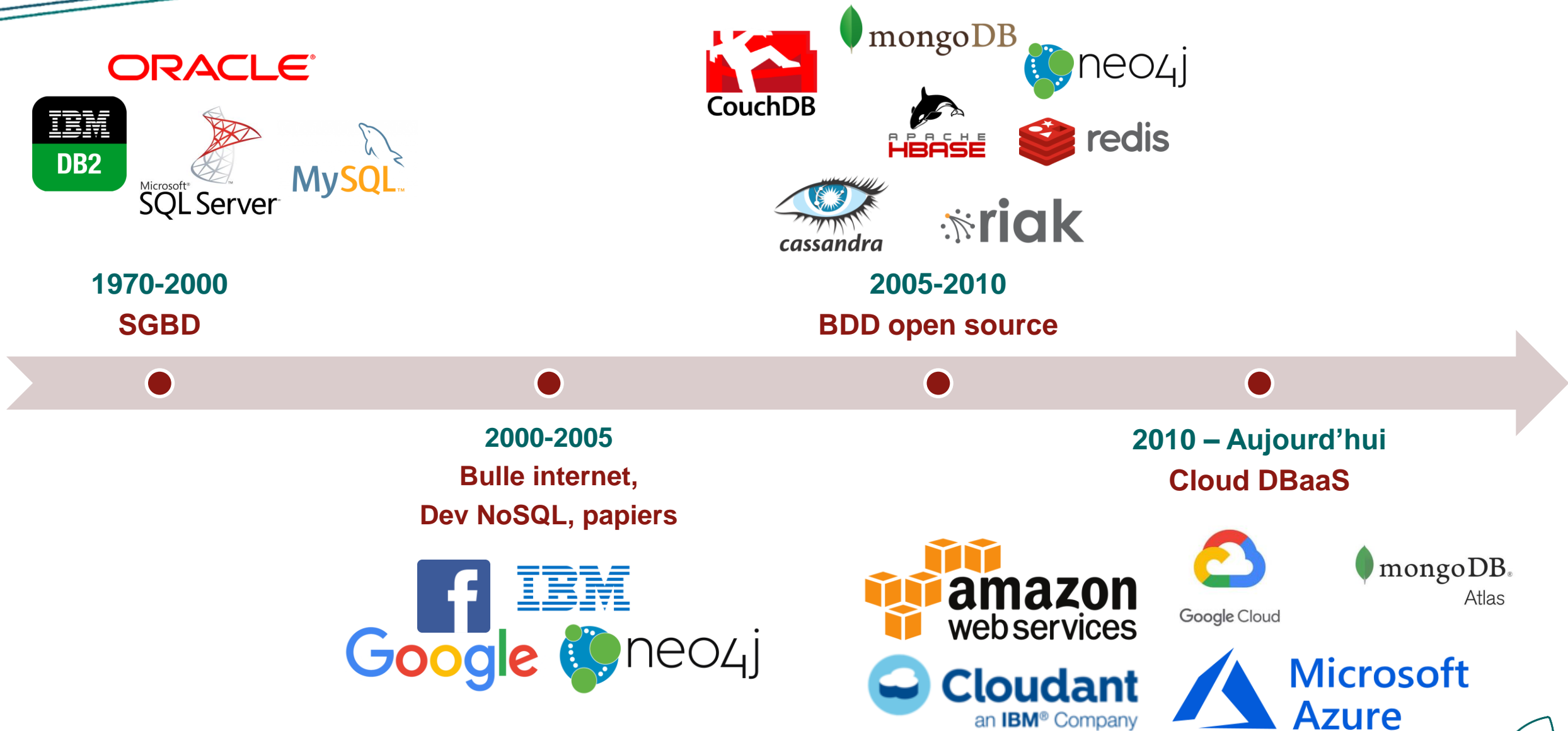
- N'utilisent pas un modèle relationnel (ni le langage SQL)
- Open source
- Conçus pour tourner sur des clusters puissants
- Basés sur les besoins du web au 21^è siècle

Non relationnelle?



1- Introduction aux BDD NoSQL

Qu'est ce que le NoSQL?

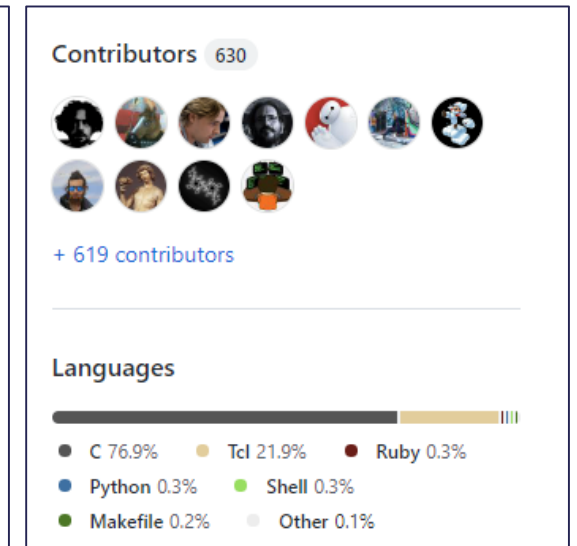
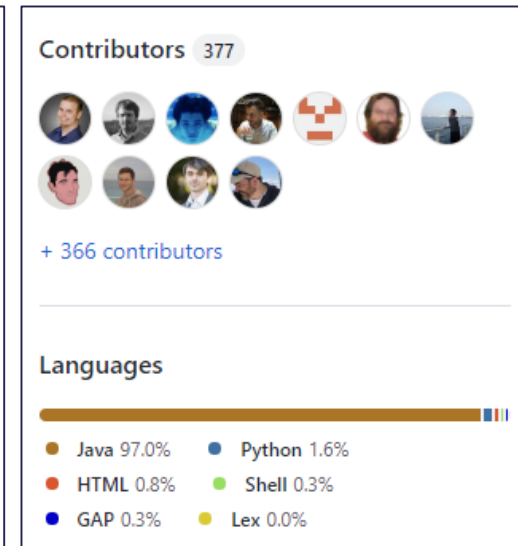
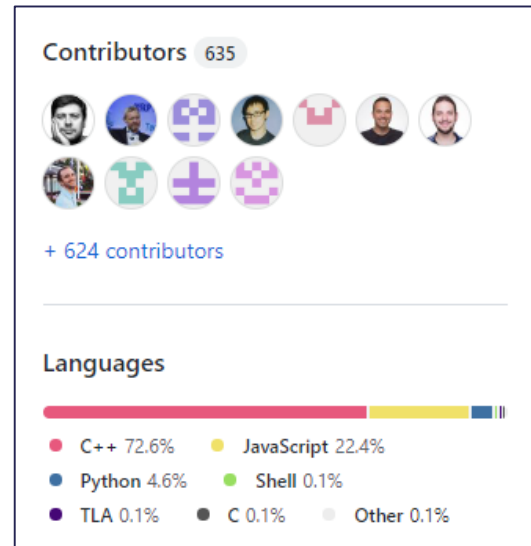
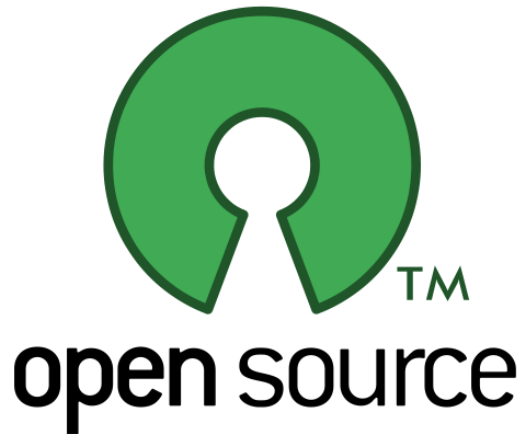


2 – Caractéristiques des BDD NoSQL

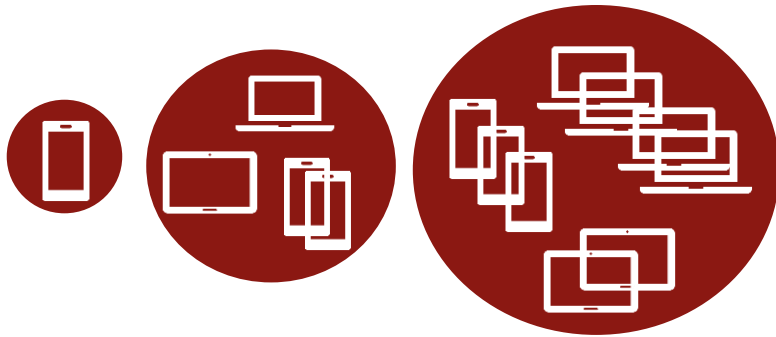
2- Caractéristiques des BDD NoSQL

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



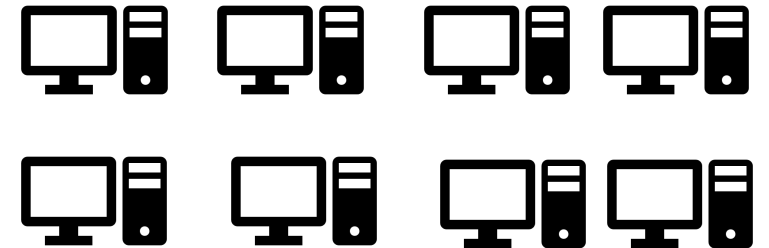
2vCPUs, 4Go



Horizontale



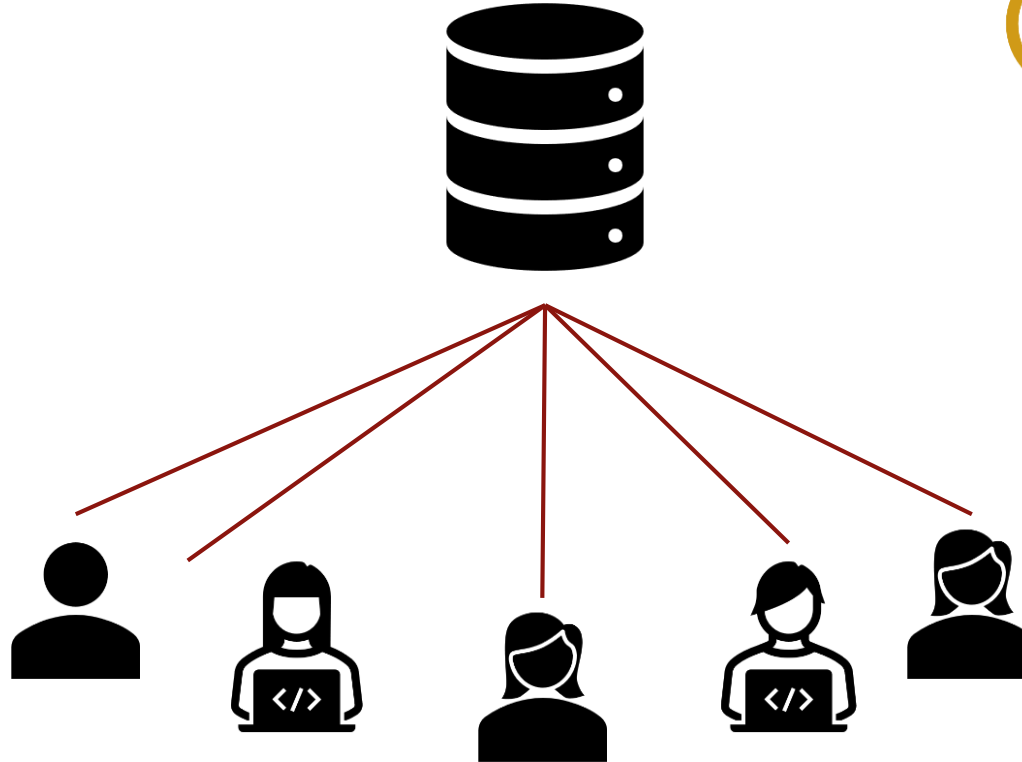
2vCPUs, 4Go * 8



Performance



Réponse rapide



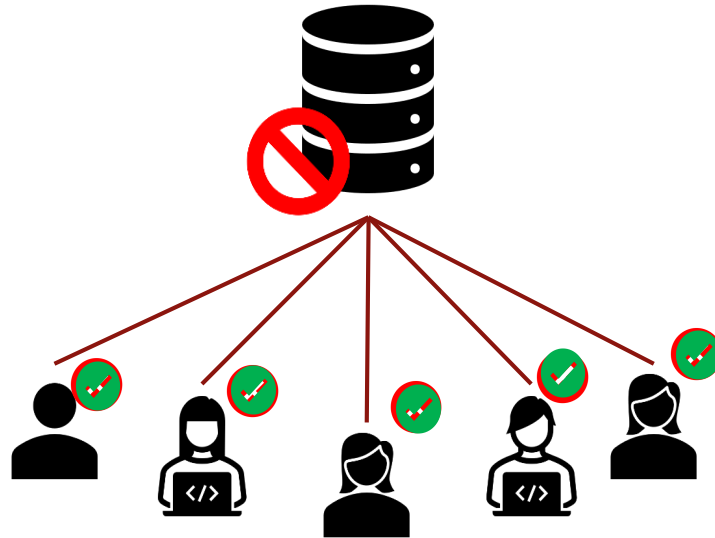
Haute concurrence



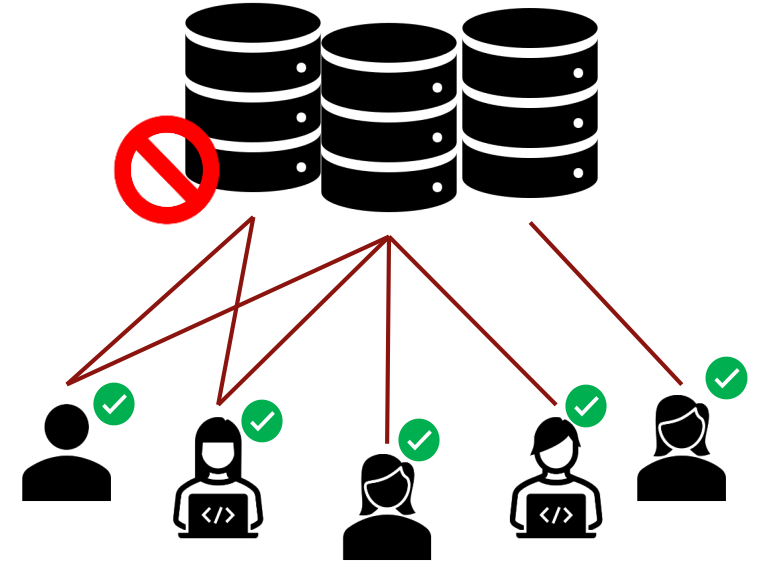
Disponibilité



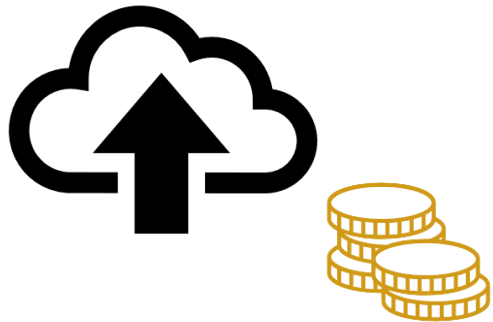
Serveur unique



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)



Flexibilité



- Schéma flexibles
- Types de données variés
- Indexage spécifique

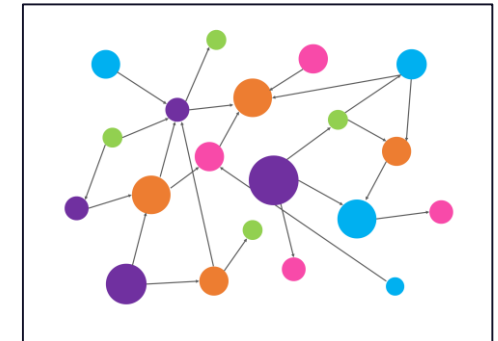
Clé-valeur

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

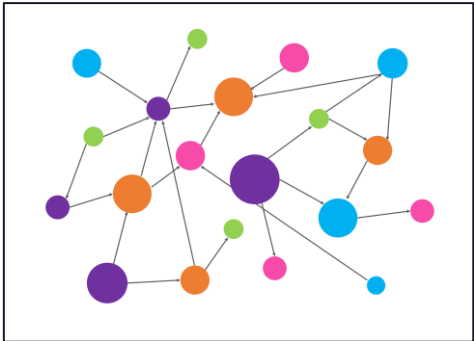
Row-oriented			
ID	Name	Grade	GPA
001	John	Senior	4.00
002	Karen	Freshman	3.67
003	Bill	Junior	3.33

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
Moins complexe en architecture	Pas fait pour des requêtes complexes
Représentée comme des hashmap	atomiques pour les opérations à clé unique seulement
Idéal pour des opérations de CRUD basiques	Moins de flexibilité pour l'indexing et le requêtage des données
Passe bien à l'échelle	
Facilement shardable	

Use cases:

- Performance rapide pour des opérations de CRUD sur des données non-interconnectées
 - Ex: stocker et retrouver des info de sessions pour des app web
- Stocker des données de profil d'utilisateur et préférences
- Panier de boutiques en ligne

Use cases non adaptés:

- Données interconnectées avec une relation many-to-many
 - Réseaux sociaux
 - Systèmes de recommandation
- Haut niveau de consistance requis pour des transactions multi-opération avec des clés multiples
 - Besoin d'une BDD ACID ready
- Quand on exécute des requêtes basée sur des valeurs contre les clés
 - Plutôt utiliser le type document ici



Orienté colonnes



Avantages	Inconvénients
Enregistre les données en colonnes ou groupes de colonnes	Moins efficaces à l'insertion des données
Rends les requêtes basées sur les dimensions plus rapides	
Idéal pour des opérations de CRUD basiques	
Passe bien à l'échelle	
Facilement shardable	

Use cases:

- Enorme volume de données "sparse" séparé, échantillonné, etc...
- Supportent le déploiement à travers plusieurs cluster de nœuds
- Peut-être utilisé pour du logging d'événements et des blogs
- Adaptés pour des cas d'usage de comptage sur colonne
- Les colonnes ont un paramètre TTL (time to live), utile pour des cas d'usage avec expiration des données (périodes d'essai, etc...)

Use cases non adaptés:

- ACID transactions
 - Reads and writes sont atomiques au niveau de la ligne (seulement)
- Au début du développement, les modèles de requêtes peuvent changer et demander énormément de changements
 - cela peut coûter cher et ralentir la production²



Graph



Avantages	Inconvénients
ACID friendly contrairement aux autres BDD NoSQL	Ne scale pas horizontalement
Impressionnant quand les données ont une structure graphe	Cheminer à travers un graph avec des nœuds partagés à travers divers serveurs peut être très couteux
	N'est pas shardable(ne shard pas bien)

Use cases:

- Données très connectées et reliées
- Réseaux sociaux
- Routing, spatial, map apps
- Systèmes de recommandation

Use cases non adaptés:

- Les cas couverts par les autres cas d'usage NoSQL
- Besoin de scaler horizontalement
- Mise à jour toutes les données ou un échantillon de nœuds avec paramètre donnée
 - Cela peut s'avérer très difficile et pas évident/trivial



3- Les types de BDD NoSQL

Orienté Document



Avantages	Inconvénients
Valeurs visibles et requêtables	Possible incohérence des données
Chaque donnée est considérée comme document(JSON/XML)	
Chaque document offre un schéma flexible	
Peut être indexé de manière personnalisée	
Scalable horizontalement	
Supporte le sharding	

Use cases:

- Event logging pour des apps et process
- Blog posts(chaque utilisateur, post, commentaire, like) représenté comme un document
- Metadonnées pour le web, applications mobiles, conçus avec Internet à l'esprit(JSON, RESTful APIs, données non structurées)

Use cases non adaptés:

- Transaction ACID
 - Ne supporte pas les transactions agissant sur de multiples document à la fois
- Données orientées agrégats
 - Données qui tombent naturellement dans un modèle tabulaire



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



4- Introduction aux BDD documents (Mongo)

Notions de base de MongoDB?

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": {
    "firsrstName": "Brice",
    "lastName": "De Nice",
    "email": {
      "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



4- Introduction aux BDD documents (Mongo)

Quand utiliser MongoDB?



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

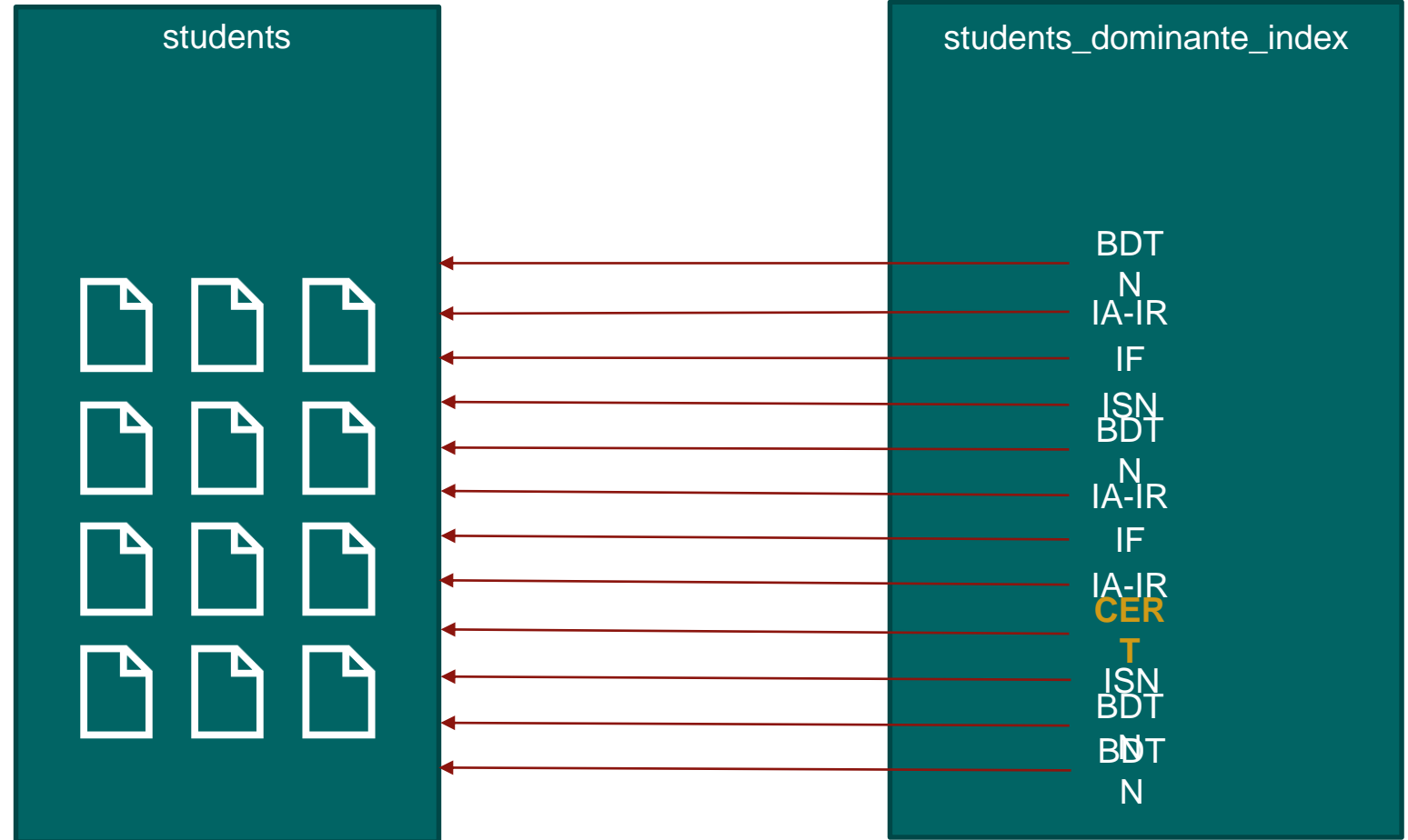
Concevoir des applications orientées analytiques



- 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

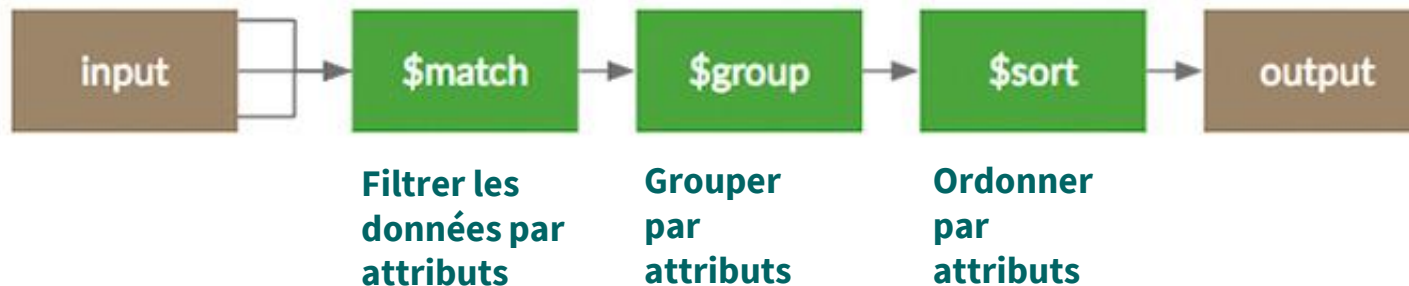


- Chaînes de traitements
- Composés d'étapes de traitement (input => traitement => output)
- Chaque unité de traitement fait intervenir une expression
- Expressions: fonctions de traitement(\$match, \$project, \$group, \$sort, etc...)



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)





Go TP!

1- Déploiement des bases de données

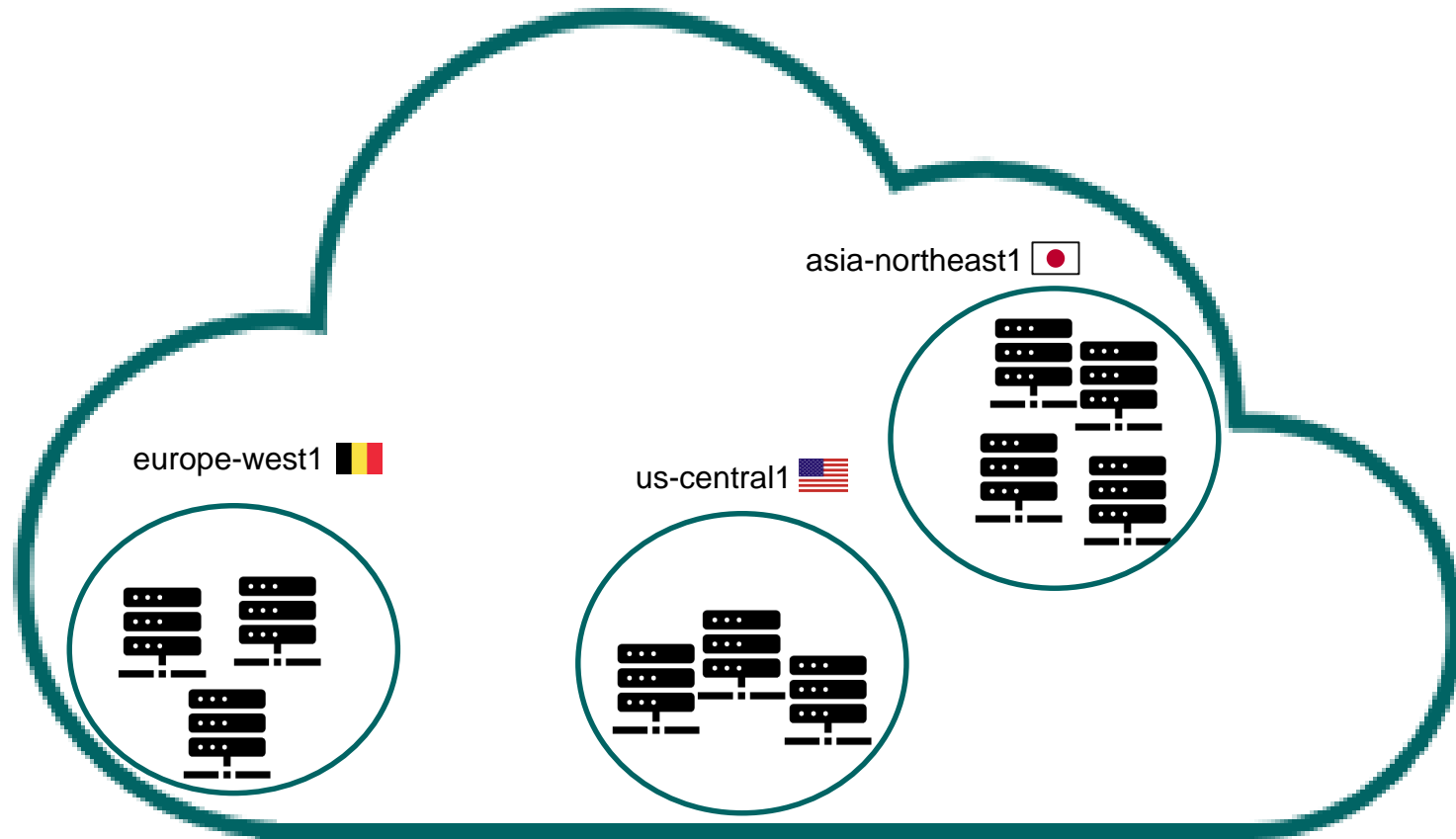


- Garanti:
- ✓ Uptime
 - ✓ Disponibilité
 - ✓ Scalabilité

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 les 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

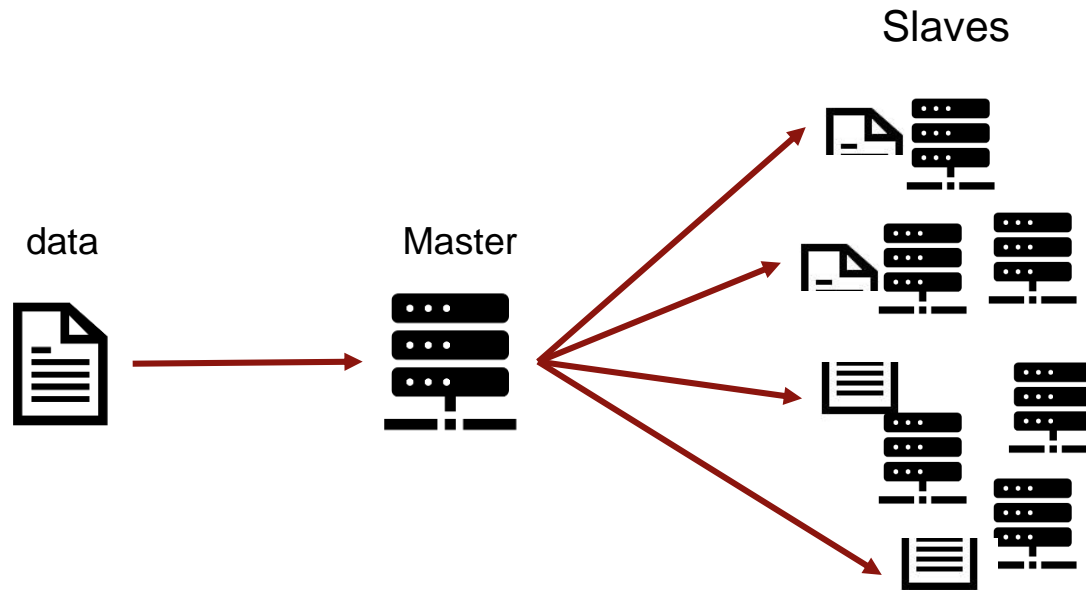


2- Stockage distribué



Partitionnement/Sharding

Replication  Backup



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



3 – 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
'Sucess' ou 'Failed'.

S

Soft state

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

E

Eventually consitent

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



4- 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

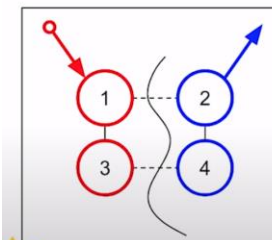
Aucune défaillance/échec de tout ou partie des nœuds du cluster ne doit empêcher le système de répondre correctement.



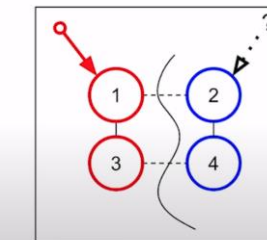
Eric Brewer

Il est impossible sur un système informatique de calcul distribué de garantir simultanément la cohérence, la disponibilité et la tolérance au partitionnement.

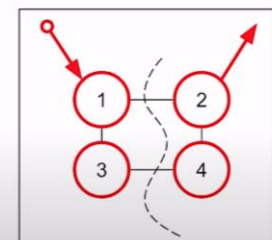
Partition Tolerant +
Available = **Not Consistent**



Partition Tolerant +
Consistent = **Not Available**



Consistent + Available =
Not Partition Tolerant





Go TP!

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**: the 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 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,
            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**: un aggregation pipeline à exécuter sur la collection à joindre (**comments**)



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"] }
              ]
            }
          }
        },
        {
          $project:
          {
            _id: 0,
            postTitle: "$_id"
          }
        }
      ],
      as: "comments"
    }
  }
])
```

Requête

```
[
  {
    "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
      }
    ]
  }
]
```

Résultat





Go TP!