

I- Initiation à MapReduce en mode centralisé avec CouchDB

Vidéo 1 : Présentation du système

CouchDB est un système de gestion de base de données documentaires. Ce n'est pas le système le plus répandu, mais il a l'avantage d'être simple à installer et facile à utiliser.

CouchDB expose ses fonctionnalités à travers une API REST, les verbes essentiels à connaître :

- **GET** : permet de demander une ressource, ou plus précisément d'obtenir sa représentation (par exemple lire un document).
- **PUT** : sert à créer une ressource ou la remplacer si elle existe déjà (utile pour insérer ou mettre à jour un document entier).
- **POST** : utilisé pour envoyer des données à une ressource, en général pour demander au serveur d'effectuer une opération ou de créer une ressource sans en préciser l'identifiant.
- **DELETE** : permet de supprimer une ressource.

Documentation CouchDB : [Overview — Apache CouchDB® 3.5 Documentation](#)

Accéder à CouchDB :

Dans le terminal :

```
docker volume create couchdb_data
```

Puis

```
docker run --name couchdb -e COUCHDB_USER=cath -e  
COUCHDB_PASSWORD=catherine -p 5984:5984 -v couchdb_data:/opt/couchdb/data -d  
couchdb
```

```
C:\Users\Cath>docker run --name couchdb -e COUCHDB_USER=cath -e COUCHDB_PASSWORD=catherine -p 5984:5984 -v couchdb_data:  
51b27282aaec: Pull complete  
2897b671832f: Pull complete  
cd726490d7d0: Pull complete  
7f82cbc9af1f: Pull complete  
ae4ce04d0e1c: Pull complete  
87d5a41c1790: Pull complete  
609e5bd9a7b1: Pull complete  
8856ade7c27a: Pull complete  
4b62383183a1: Pull complete  
716c80c273f8: Pull complete  
f6c9c88e767d: Pull complete  
Digest: sha256:a2c8839c86304962ae60df41c9aa51907925b7ca022b69acfcd45663a89daa77  
Status: Downloaded newer image for couchdb:latest  
63501170fb62ce2702465ce521755fcda45f190dd4019b574724b3c39c62eea
```

Le conteneur s'appelle : couchdb

Nous avons 2 variables d'environnements : nom d'utilisateur et mot de passe.

L'interface graphique : localhost:5984/_utils/

On utilise un client, ici curl pour communiquer avec l'API REST

Première requête pour vérifier que CouchDB fonctionne et que nous avons accès au serveur avec nos identifiants : `curl -X GET http://login:mot_de_passe@localhost:5984`

```
curl -X GET http://cath:catherine@localhost:5984
```

```
C:\Users\Cath>curl -X GET http://cath:catherine@localhost:5984
{"couchdb": "Welcome", "version": "3.5.1", "git_sha": "44f6a43d8", "uuid": "4338e8955690265c63ca1b3307e81d4d", "features": ["access-ready", "partitioned", "pluggable-storage-engines", "reshard", "scheduler"], "vendor": {"name": "The Apache Software Foundation"}}
```

Créer une ressource film : `curl -X PUT http://cath:catherine@localhost:5984/films`

On reçoit un document JSON comme confirmation

```
C:\Users\Cath>curl -X PUT http://cath:catherine@localhost:5984/films
{"ok":true}
```

Récupérer une ressource : `curl -X GET http://cath:catherine@localhost:5984/films`

```
C:\Users\Cath>curl -X GET http://cath:catherine@localhost:5984/films
{"instance_start_time": "1765299882", "db_name": "films", "purge_seq": "0-g1AAAABPeJzLYWBgYMpgTmHgzcPy09JdcjLz8gvLskBCeexAEmGBiD1HwiyEhlwqEtikSkqHKMgCAIT2GV4", "update_seq": "0-g1AAACLeJzLYWBgYMpgTmHgzcPy09JdcjLz8gvLskBCeexAEmGBiD1HwiyMpgTGXKBAuxmlsmmmqlp6HpwmJLIkfSPoj3RwiItNdkEXXEWACMxKyE", "sizes": {"file": 16692, "external": 0, "active": 0}, "props": {}, "doc_del_count": 0, "doc_count": 0, "disk_format_version": 8, "compact_running": false, "cluster": {"q": 2, "n": 1, "w": 1, "r": 1}}
```

Insérer un nouveau document dans la base films :

```
curl -X PUT http://cath:catherine@localhost:5984/films/doc -H "Content-Type: application/json" -d '{"cle": "valeur"}'
```

Remarque :

Dans CouchDB, comme dans MongoDB, aucun schéma n'est imposé contrairement à Cassandra, qui fait partie des rares systèmes NoSQL à exiger un schéma, ce qui peut finalement constituer un avantage.

L'intérêt d'un système NoSQL réside dans la possibilité de stocker des documents qui n'ont pas forcément la même structure. Cependant, cette liberté a des conséquences : pour exploiter ces données, l'application doit effectuer tout le travail de traitement. Lorsqu'un document est récupéré, l'application doit elle-même interpréter sa structure, gérer les différences entre documents et compenser l'absence de schéma défini. En termes de performances, cette situation n'est pas idéale, même si le système autorise l'insertion de documents hétérogènes.

En réalité, même lorsqu'aucun schéma n'est officiellement imposé, un schéma finit toujours par exister d'une manière implicite dès que le système est utilisé. Sans cela, chaque application devrait définir sa propre organisation des données, ce qui n'est pas acceptable. Une base de données doit être conçue pour exister indépendamment des applications, de manière pérenne et cohérente. Elle ne doit pas dépendre de chaque application développée au-dessus d'elle.

```
C:\Users\Cath>curl -X PUT http://cath:catherine@localhost:5984/films/doc -H "Content-Type: application/json" -d "{\"cle\":\"valeur\"}" {"ok":true,"id":"doc","rev":"1-0f3beea1048634b34d8091e22ab3c6fb"}
```

On rajoute -H "Content-Type: application/json" car sinon nous avons l'erreur invalid UTF-8 JSON car couchDB ne reconnaît pas que c'est du JSON

On vérifie que ca a bien été inséré avec un GET :

```
C:\Users\Cath>curl -X GET http://cath:catherine@localhost:5984/films/doc {"_id":"doc","_rev":"1-0f3beea1048634b34d8091e22ab3c6fb","cle":"valeur"}
```

- **_id** est l'identifiant unique du document dans CouchDB.
Il permet de retrouver, modifier ou supprimer le document.
Son rôle est comparable à celui d'une clé primaire dans une base relationnelle.
Si aucun identifiant n'est fourni lors de l'insertion, CouchDB en génère un automatiquement.
- **_rev** représente la révision du document, c'est-à-dire sa version actuelle.
CouchDB utilise cette information pour gérer les mises à jour, éviter les conflits et garantir la cohérence des données.
Chaque modification du document crée une nouvelle révision (par exemple : 1-..., 2-..., etc.).

Insérons un autre document :

(On veille bien à changer le nom du document sinon nous recevrons un message d'erreur)

```
C:\Users\Cath>curl -X PUT http://cath:catherine@localhost:5984/films/doc1 -H "Content-Type: application/json" -d "{\"nom\":\"youcef\"}" {"ok":true,"id":"doc1","rev":"1-7935b7a8d47af3f4100e1b57f8d24f3e"}
```

Insérer un document sauvegardé dans un fichier JSON :

```
curl -X POST http://cath:catherine@localhost:5984/exemple -d @movie:10098.json -H "Content-Type: application/json"
```

Dans CouchDB, lorsqu'aucun identifiant n'est fourni lors de l'insertion d'un document, la base de données attribue automatiquement un identifiant unique. Cette fonctionnalité existe également dans MongoDB. D'un point de vue pratique, l'attribution automatique d'identifiants est généralement préférable à la création manuelle d'une clé par l'utilisateur.

La raison est liée au rôle d'un système de gestion de base de données, en particulier dans un contexte distribué. Un SGBD NoSQL est conçu pour gérer des données réparties sur plusieurs serveurs, garantir la cohérence, organiser le stockage sur disque et assurer la continuité du service même en cas de panne d'un nœud. La génération des identifiants fait partie des mécanismes nécessaires à cette gestion : elle permet d'éviter les collisions, facilite la répartition des données dans une infrastructure distribuée et assure la bonne coordination entre les serveurs.

Insérer d'une collection de documents :

```
curl -X POST http://cath:catherine@localhost:5984/films/_bulk_docs -d  
@films_couchdb.json -H "Content-Type: application/json"
```

Insertion bien effectuée

```
C:\Users\Cath> curl -X POST http://cath:catherine@localhost:5984/films/_bulk_docs -d @films_couchdb.json  
-H "Content-Type: application/json"  
[{"ok":true,"id":"movie:11","rev":"1-c404285f85cc593f351855821ebe5fc7"}, {"ok":true,"id":"movie:24","rev":"1-7f851a642ab7108adad8354952dac560"}, {"ok":true,"id":"movie:28","rev":"1-5ef74f3007d597da5c1a41d73e00f308"}, {"ok":true,"id":"movie:33","rev":"1-210992fbdd105dd91ceb02a1f6b1811d"}, {"ok":true,"id":"movie:38",
```

On peut également voir cela sur l'interface graphique

Voici la structure d'un film :

```
1  [ ]  
2  "_id": "movie:10098",  
3  "_rev": "1-a9d1a0a8ec461bac3ce06dee5b3e6a4",  
4  "title": "Le Kid",  
5  "year": 1921,  
6  "genre": "Comédie",  
7  "summary": "Un pauvre vitrier recueille un enfant abandonné par sa mère victime d'un s",  
8  "country": "US",  
9  "director": {  
10   "_id": "artist:13848",  
11   "last_name": "Chaplin",  
12   "first_name": "Charlie",  
13   "birth_date": 1889  
14 },  
15  "actors": [  
16    {  
17      "last_name": "Chaplin",  
18      "first_name": "Charlie",  
19      "birth_date": 1889  
20    },  
21    {  
22      "last_name": "Coogan",  
23      "first_name": "Jackie",  
24      "birth_date": 1914  
25    },  
26    {  
27      "last_name": "Purviance",  
28      "first_name": "Edna",  
29      "birth_date": 1895  
30    }  
31  ]  
32 ]
```

Remarque :

Dans CouchDB (comme dans d'autres bases NoSQL documentaires), un document peut contenir des données **imbriquées** : par exemple, un film peut intégrer directement la liste de ses acteurs, son genre, son année, etc., dans une seule structure. Cette organisation permet de créer des documents complexes, mais elle **viole la première forme normale**, qui est obligatoire dans les bases de données relationnelles.

Dans un modèle relationnel, une telle structure n'est pas autorisée : les données doivent être séparées en plusieurs tables, reliées par des clés étrangères, afin d'éviter la redondance et de garantir la cohérence. Dans CouchDB, au contraire, la duplication d'informations est possible et même courante. Cela implique cependant un risque : une même donnée (par exemple le nom d'un acteur) peut être répétée dans plusieurs documents et écrite différemment, ce qui peut entraîner des incohérences.

Le **point essentiel à retenir** est donc le suivant :

la grande différence entre un modèle documentaire et un modèle relationnel réside dans la possibilité d'imbriquer des données et d'accepter la redondance, au détriment des règles strictes de normalisation.

Récupérer un document sachant son identifiant :

```
curl -X GET http://cath:catherine@localhost:5984/films/movie:10098
```

```
C:\Users\Cath>curl -X GET http://cath:catherine@localhost:5984/films/movie:10098
{"_id": "movie:10098", "_rev": "1-a9ad1a0a8ec461bac3ce06dee5b3e6a4", "title": "Le Kid", "year": 1921, "genre": "Comédie", "summary": "Un pauvre vitrier recueille un enfant abandonné par sa mère victime d'un séducteur. L'enfant casse des carreaux pour aider son père adoptif, qui l'arrache à des dames patronnesses, puis le rend à sa mère, devenue riche.", "country": "US", "director": {"_id": "artist:13848", "last_name": "Chaplin", "first_name": "Charlie", "birth_date": 1889}, "actors": [{"last_name": "Chaplin", "first_name": "Charlie", "birth_date": 1889}, {"last_name": "Coogan", "first_name": "Jackie", "birth_date": 1914}, {"last_name": "Purviance", "first_name": "Edna", "birth_date": 1895}]}]
```

Supprimer un document sachant son identifiant :

```
curl -X DELETE "http://cath:catherine@localhost:5984/films/movie:10098?rev=1-a9ad1a0a8ec461bac3ce06dee5b3e6a4"
```

Dans CouchDB, chaque document possède une révision (`_rev`).

Cette révision est obligatoire pour toute modification ou suppression.

Supprimer un document sans fournir sa révision est impossible, car CouchDB doit vérifier que l'on supprime bien la dernière version du document afin d'éviter les conflits dans un système distribué.

En pratique, il faut toujours :

1. Récupérer la révision avec un GET ;
2. Utiliser cette révision dans la commande DELETE.

```
C:\Users\Cath>curl -X DELETE "http://cath:catherine@localhost:5984/films/movie:10098?rev=1-a9ad1a0a8ec461bac3ce06dee5b3e6a4"
{"ok":true,"id":"movie:10098","rev":"2-7ad622f3e243feb7e2e2ad9e24c9ba99"}
```

Vidéo 2 :

MapReduce est un des concepts les plus utilisés en Big Data pour effectuer des calculs parallèles. Il repose sur la définition de deux fonctions :

- Map
- Reduce

Pourquoi utiliser CouchDB pour la démonstration ?

Deux raisons :

- CouchDB est libre (open source) et propose un moteur d'exécution MapReduce.
- Il expose une API REST, ce qui le rend facile à appréhender et à prendre en main.

Et surtout (raison principale) :

- Avec CouchDB, on n'est pas obligé de définir la fonction Reduce : on peut définir uniquement Map.
- Cela permet de sauvegarder la fonction et de visualiser les résultats intermédiaires, ce qui est présenté comme rare (voire unique) par rapport à d'autres systèmes.

Rappel : principe général de MapReduce

On travaille sur une collection de documents (comme dans les bases documentaires MongoDB et CouchDB).

Les documents sont indépendants, ce qui permet d'effectuer des traitements en parallèle.

Fonction Map

- La fonction Map s'applique à chaque document, un par un, de manière indépendante.
- Elle prend un document en paramètre et réalise un traitement dessus.
- Elle peut produire :
 - aucun résultat,
 - un résultat,
 - ou plusieurs résultats.

C'est donc une fonction de transformation.

Notion de groupes (clés) : La fonction Map définit des groupes à l'aide de clés (identifiants de groupe).

=> Ces clés servent ensuite à regrouper les données pour la suite du traitement.

Phase intermédiaire : Sort & Shuffle

Après Map, il existe une phase intermédiaire :

- Sort & Shuffle : tri + redistribution des résultats

Cette étape sert à trier et envoyer les résultats vers les différents serveurs d'une grappe (cluster).

Démonstration de la vidéo :

Même si le concept est prévu pour une grappe, ici la démonstration est faite sur :

- un seul nœud / un seul serveur

L'objectif est surtout de montrer :

- comment définir la fonction Map,
- comment définir la fonction Reduce,
- et comprendre l'intérêt du mécanisme.

Lancement de CouchDB : Une fois CouchDB démarré, on peut l'utiliser via une interface graphique : localhost:5984/_utils/

La fonction map traite les données document par document, de manière indépendante. Elle est appliquée automatiquement sur chaque document de la collection. Dans MapReduce, la fonction map ne fait pas un return comme en programmation classique : elle émet des résultats (des documents intermédiaires), car ces résultats seront ensuite regroupés avec tous ceux qui ont le même identifiant (clé) et pourront être traités par un autre nœud / serveur pour faire une agrégation (par exemple une somme).

1. Rappel sur le rôle de la fonction map

- Elle prend un document en paramètre.
- Elle traite les documents un par un, indépendamment (point essentiel).
- Elle sert à transformer le document : on peut faire du nettoyage, de la restructuration, etc.
- Elle émet une clé et une valeur (résultat intermédiaire) destinées à être regroupées et traitées plus tard.

2. Exemple de fonction map par défaut dans l'interface

Dans l'interface CouchDB, une fonction map par défaut peut émettre :

- la clé : l'identifiant du document
- la valeur : 1

3. Objectif : calculer le nombre de films par année

La question posée est de calculer le nombre de films par année, en imaginant une exécution parallèle :

- chaque nœud (serveur) traite un sous-ensemble de documents (un lot de films)
- pour chaque film, la fonction map extrait des informations simples

Dans l'exemple, on extrait notamment :

- l'année de sortie du film
- le titre du film

Le **résultat intermédiaire** est préparé pour le regroupement :

- la clé intermédiaire correspond à l'année
- les résultats sont déjà triés/regroupés par année (on observe que tous les films de la même année apparaissent ensemble, par exemple 1950, puis les années 40, etc.)
- on peut choisir le nombre de résultats affichés dans l'interface

Ensuite, ces résultats intermédiaires sont préparés pour la suite : une fonction de hachage pourra être utilisée pour envoyer chaque groupe de résultats vers un serveur/nœud de la grappe afin qu'il fasse l'agrégation.

4. Résultats intermédiaires et particularité CouchDB

Les résultats de la fonction map peuvent être sauvegardés dans CouchDB, ce qui permet de visualiser les résultats intermédiaires. L'intervenant indique qu'à sa connaissance c'est une particularité rare (notamment parmi des bases NoSQL).

5. Passage à la fonction reduce (agrégation)

Après map, on applique une fonction reduce pour répondre à la question initiale : le nombre de films par année.

- on prend **la clé intermédiaire** (ici l'année)
- tous les films sortis la même année sont agrégés
- on calcule une agrégation simple, typiquement une somme, pour obtenir le nombre total par année

	id	key	value
□	_design/demoMapReduce	_design/demoMapReduce	{ "rev": "1-8d361a23b4cb8e213f0868ea3d2742c..." }
□	doc	doc	{ "rev": "1-0f3beea1048634b34d8091e22ab3c6f..." }
□	doc1	doc1	{ "rev": "1-7935b7a8d47af3f4100e1b57f8d24f3e" }
□	movie:1018	movie:1018	{ "rev": "1-d092ae64609792fc8d09e01726f8ff9e" }
□	movie:10238	movie:10238	{ "rev": "1-04e18ea88d56259144d7b646b144c3..." }
□	movie:103	movie:103	{ "rev": "1-06e8ddd6bb2f56dacaccd39cb970ee8f" }
□	movie:10362	movie:10362	{ "rev": "1-9045bb46faeb05af6837d75f365e4503" }
□	movie:103731	movie:103731	{ "rev": "1-087c137274fb02c1844a882a10d3e45..." }
□	movie:106	movie:106	{ "rev": "1-02252f92774e5f478f0c62e89a755827" }
□	movie:10669	movie:10669	{ "rev": "1-e55281901af86aac6e1cfb80e4f2ec4a" }

Calculer nombre de films par année :

La fonction Map :

```
Function (doc) {emit(doc.year, doc.title); }
```

	id	key	value
□	doc	null	null
□	doc1	null	null
□	movie:631	1927	L'Aurore
□	movie:832	1931	M le maudit
□	movie:3082	1936	Les Temps modernes
□	movie:43884	1936	La Charge de la brigade légère
□	movie:777	1937	La Grande Illusion
□	movie:223	1940	Rebecca
□	movie:596	1940	The Grapes of Wrath
□	movie:914	1940	Le Dictateur
□	movie:11462	1941	Soupçons
□	movie:289	1942	Casablanca
□	movie:29084	1943	Le Corbeau

⇒ on voit que la phase de groupement, Sort est déjà appliqué.

Appliquer la fonction Reduce pour calculer le nombre de film par année

```
function (keys, values) {
    return values.length;
}
```

1931	1
1936	2
1937	1
1940	2
1941	1
1942	1
1943	1
1944	1
1946	1
1947	1
1948	1

Calculer nombre de films pour chaque acteur :

```
function(doc)
```

```
{
```

```
    emit({ "prénom": doc.actors[i].first_name, "nom": doc.actors[i].last_name }, doc.title);
```

```
}
```

id	key	value
movie:1443	{ "prénom": "A.J.", "nom": "Cook" }	Virgin suicides
movie:155	{ "prénom": "Aaron", "nom": "Eckhart" }	The Dark Knight : Le Chevalier noir
movie:46738	{ "prénom": "Abdelghafour", "nom": "Elaaziz" }	Incendies
movie:773	{ "prénom": "Abigail", "nom": "Breslin" }	Little Miss Sunshine
movie:140607	{ "prénom": "Adam", "nom": "Driver" }	Star Wars : Le Réveil de la Force
movie:181808	{ "prénom": "Adam", "nom": "Driver" }	Star Wars : Les Derniers Jedi
movie:181812	{ "prénom": "Adam", "nom": "Driver" }	Star Wars, épisode IX
movie:245703	{ "prénom": "Adam", "nom": "Driver" }	Midnight Special
movie:857	{ "prénom": "Adam", "nom": "Goldberg" }	Il faut sauver le soldat Ryan
movie:21575	{ "prénom": "Adel", "nom": "Bencherif" }	Un prophète

La fonction Reduce :

```
function (keys, values) {
```

```
    return values.length;
```

```
}
```

key	value
{ "prénom": "A.J.", "nom": "Cook" }	1
{ "prénom": "Aaron", "nom": "Eckhart" }	1
{ "prénom": "Abdelghafour", "nom": "Elaaziz" }	1
{ "prénom": "Abigail", "nom": "Breslin" }	1
{ "prénom": "Adam", "nom": "Driver" }	2
{ "prénom": "Adam", "nom": "Goldberg" }	1
{ "prénom": "Adel", "nom": "Bencherif" }	1

II- MapReduce avec MongoDB

1. Compter le nombre total de film

Définissons la function Map dans une variable :

```
var map1 = function () { emit("total", 1); };
```

Définissons la function Reduce dans une variable :

```
var reduce1 = function (k, vals) { return Array.sum(vals); };
```

On lance MapReduce :

```
db.films.mapReduce(map1, reduce1, { out: "q1_total_films" });
```

Pour voir le résultat :

```
db.q1_total_films.find();
```

```
DeprecationWarning: Collection.mapReduce() is deprecated. Use an aggregation instead.
See https://mongodb.com/docs/manual/core/map-reduce for details.
[ { _id: 'total', value: 278 } ]
```

On peut voir que l'utilisation de MapReduce est dépréciée, MongoDB recommande d'utiliser l'agrégation, mais l'objectif de notre TP est de se familiariser avec l'écriture de fonctions MapReduce dans MongoDB donc on ne prendra pas en compte la recommandation.

2. Compter le nombre de films par genre

```
var map2 = function () { emit(this.genre, 1); };
```

```
var reduce2 = function (k, vals) { return Array.sum(vals); };
```

```

db.films.mapReduce(map2, reduce2, { out: "q2_films_par_genre" });

db.q2_films_par_genre.find().sort({ value: -1 });

lesfilms> var map2 = function () { emit(this.genre, 1); };
... var reduce2 = function (k, vals) { return Array.sum(vals); };
...
... db.films.mapReduce(map2, reduce2, { out: "q2_films_par_genre" });
... db.q2_films_par_genre.find().sort({ value: -1 });
...
[
  { _id: 'Drame', value: 96 },
  { _id: 'Action', value: 36 },
  { _id: 'Crime', value: 29 },
  { _id: 'Comédie', value: 25 },
  { _id: 'Aventure', value: 22 },
  { _id: 'Drama', value: 14 },
  { _id: 'Thriller', value: 10 },
  { _id: 'Science-Fiction', value: 9 },
  { _id: 'Horreur', value: 8 },
  { _id: 'Mystère', value: 6 },
  { _id: 'Fantastique', value: 4 },
  { _id: 'Romance', value: 3 },
  { _id: 'Western', value: 3 },
  { _id: 'Adventure', value: 3 },
  { _id: 'Musique', value: 2 },
  { _id: 'Fantasy', value: 2 },
  { _id: 'Mystery', value: 1 },
  { _id: 'Comedy', value: 1 },
  { _id: 'War', value: 1 },
  { _id: 'Guerre', value: 1 }
]

```

3. Compter le nombre de films par réalisateur

```

var map3b = function () {
  if (!this.director) return;
  var name = [this.director.first_name, this.director.last_name].filter(Boolean).join(
    "").trim();
  emit(name, 1);
};

var reduce3b = function (k, vals) { return Array.sum(vals); };

```

```

db.films.mapReduce(map3b, reduce3b, { out: "q3_films_par_realisateur_nom_cle" });
db.q3_films_par_realisateur_nom_cle.find().sort({ value: -1 });

```

```

...
[
  { _id: 'Steven Spielberg', value: 13 },
  { _id: 'Alfred Hitchcock', value: 10 },
  { _id: 'Woody Allen', value: 8 },
  { _id: 'Stanley Kubrick', value: 7 },
  { _id: 'Christopher Nolan', value: 7 },
  { _id: 'Quentin Tarantino', value: 7 },
  { _id: 'Roman Polański', value: 7 },
  { _id: 'Francis Ford Coppola', value: 6 },
  { _id: 'Ridley Scott', value: 6 },
  { _id: 'Paul Verhoeven', value: 5 },
  { _id: 'Ingmar Bergman', value: 5 },
  { _id: 'Michael Mann', value: 5 },
  { _id: 'George Lucas', value: 5 },
  { _id: 'David Fincher', value: 4 },
  { _id: 'François Truffaut', value: 4 },
  { _id: 'Martin Scorsese', value: 4 },
  { _id: 'Charlie Chaplin', value: 4 },
  { _id: 'David Cronenberg', value: 4 },
  { _id: 'Denis Villeneuve', value: 4 },
  { _id: 'Sydney Pollack', value: 3 }
]

```

4. Compter le nombre d'acteurs uniques qui apparaissent dans tout les films

```

lésfilms> var map4 = function () {
...   var set = {};
...   (this.actors || []).forEach(function (a) {
...     var k = [a.last_name, a.first_name, a.birth_date].join("|");
...     set[k] = 1;
...   });
...   emit("uniqueActors", set);
... };
...
... var reduce4 = function (k, vals) {
...   var out = {};
...   vals.forEach(function (v) {
...     for (var kk in v) out[kk] = 1;
...   });
...   return out;
... };
...
... var finalize4 = function (k, reduced) {
...   return Object.keys(reduced).length;
... };
...
... db.films.mapReduce(map4, reduce4, { out: "q4_nb_acteurs_uniques", finalize: finalize4 });
... db.q4_nb_acteurs_uniques.find();
...
[ { _id: 'uniqueActors', value: 1210 } ]

```

5. Nombre de films par année

```

lesfilms> var map5 = function () { emit(this.year, 1); };
... var reduce5 = function (k, vals) { return Array.sum(vals); };
...
... db.films.mapReduce(map5, reduce5, { out: "q5_films_par_annee" });
... db.q5_films_par_annee.find().sort({ _id: 1 });
...
[
  { _id: 1921, value: 1 }, { _id: 1927, value: 1 },
  { _id: 1931, value: 1 }, { _id: 1936, value: 2 },
  { _id: 1937, value: 1 }, { _id: 1940, value: 3 },
  { _id: 1941, value: 1 }, { _id: 1942, value: 1 },
  { _id: 1943, value: 1 }, { _id: 1944, value: 1 },
  { _id: 1946, value: 2 }, { _id: 1947, value: 1 },
  { _id: 1948, value: 1 }, { _id: 1949, value: 1 },
  { _id: 1950, value: 2 }, { _id: 1951, value: 1 },
  { _id: 1952, value: 2 }, { _id: 1954, value: 2 },
  { _id: 1955, value: 1 }, { _id: 1956, value: 2 }
]
Type "it" for more

```

6. Note moyenne par films avec le tableau grade

```

lesfilms> var map6 = function () {
...   var s = 0, c = 0;
...   (this.grades || []).forEach(function (g) { s += g.note; c += 1; });
...   emit(this._id, { title: this.title, sum: s, count: c });
... };

...
... var reduce6 = function (k, vals) {
...   var title = null, sum = 0, count = 0;
...   vals.forEach(function (v) {
...     title = title || v.title;
...     sum += v.sum; count += v.count;
...   });
...   return { title: title, sum: sum, count: count };
... };

...
... var finalize6 = function (k, v) {
...   return { title: v.title, avg: (v.count ? v.sum / v.count : null), countGrades: v.count };
... };

...
... db.films.mapReduce(map6, reduce6, { out: "q6_moyenne_par_film", finalize: finalize6 });
... db.q6_moyenne_par_film.find();
...
[
  {
    _id: 'movie:1578',
    value: { title: 'Raging Bull', avg: 48.5, countGrades: 4 }
  },
  {
    _id: 'movie:33',
    value: { title: 'Impitoyable', avg: 36.5, countGrades: 4 }
  },

```

7. Calculer la note moyenne par genre :

```

var map7 = function (){
  var s = 0, c = 0;
  (this.grades || []).forEach(function (g) { s += g.note; c += 1; });
  emit(this.genre, { sum: s, count: c });
};

var reduce7 = function (k, vals){
  var sum = 0, count = 0;

```

```

vals.forEach(function (v) { sum += v.sum; count += v.count; });

return { sum: sum, count: count };

};

var finalize7 = function (k, v) { return v.count ? (v.sum / v.count) : null; };

db.films.mapReduce(map7, reduce7, { out: "q7_moyenne_par_genre", finalize:
finalize7 });

db.q7_moyenne_par_genre.find().sort({ value: -1 });

...
[  

{ _id: 'Histoire', value: 83.75 },  

{ _id: 'War', value: 70 },  

{ _id: 'Guerre', value: 69.5 },  

{ _id: 'Adventure', value: 62.91666666666664 },  

{ _id: 'Science-Fiction', value: 54.94444444444444 },  

{ _id: 'Aventure', value: 54.10227272727273 },  

{ _id: 'Crime', value: 53.060344827586206 },  

{ _id: 'Fantasy', value: 52.5 },  

{ _id: 'Mystère', value: 51.29166666666664 },  

{ _id: 'Comedy', value: 50.5 },  

{ _id: 'Drame', value: 50.375 },  

{ _id: 'Musique', value: 49.75 },  

{ _id: 'Romance', value: 48.5 },  

{ _id: 'Action', value: 48.22222222222222 },  

{ _id: 'Western', value: 47.83333333333336 },  

{ _id: 'Fantastique', value: 47 },  

{ _id: 'Thriller', value: 46.75 },  

{ _id: 'Drama', value: 45.910714285714285 },  

{ _id: 'Comédie', value: 45.4 },  

{ _id: 'Horreur', value: 45.34375 }
]
Type "it" for more

```

8. Note moyenne par réalisateur

```

lesfilms> var map8b = function () {
...   if (!this.director) return;
...
...   var name = [this.director.first_name, this.director.last_name]
...     .filter(Boolean)
...     .join(" ")
...     .trim();
...
...   var s = 0, c = 0;
...   (this.grades || []).forEach(function (g) { s += g.note; c += 1; });
...
...   emit(name, { sum: s, count: c });
...
... };
...
... var reduce8b = function (k, vals) {
...   var sum = 0, count = 0;
...   vals.forEach(function (v) { sum += v.sum; count += v.count; });
...   return { sum: sum, count: count };
...
... };
...
... var finalize8b = function (k, v) {
...   return v.count ? (v.sum / v.count) : null;
...
... };
...
... db.films.mapReduce(map8b, reduce8b, {
...   out: "q8_moyenne_par_realisateur_nom_cle",
...   finalize: finalize8b
...

```

```

db.films.mapReduce(map8b, reduce8b, {
  out: "q8_moyenne_par_realisateur_nom_cle",
  finalize: finalize8b
});

db.q8_moyenne_par_realisateur_nom_cle.find().sort({ value: -1 });

...
[
  { _id: 'William Wyler', value: 79.25 },
  { _id: 'Wolfgang Petersen', value: 75.25 },
  { _id: 'Mati Diop', value: 75 },
  { _id: 'Andrew Davis', value: 73.75 },
  { _id: 'Jean-Pierre Melville', value: 72.91666666666667 },
  { _id: 'Fritz Lang', value: 68.25 },
  { _id: 'Charles Laughton', value: 68 },
  { _id: 'Sergio Leone', value: 68 },
  { _id: 'Michel Gondry', value: 67.625 },
  { _id: 'John Huston', value: 66.5 },
  { _id: 'John Cassavetes', value: 66.33333333333333 },
  { _id: 'Luc Besson', value: 65 },
  { _id: 'Philippe Lioret', value: 64.75 },
  { _id: 'Thierry de Peretti', value: 63.25 },
  { _id: 'David Lean', value: 63.25 },
  { _id: 'Peter Jackson', value: 62.91666666666664 },
  { _id: 'Sydney Pollack', value: 62.83333333333336 },
  { _id: 'Stéphane Brizé', value: 62.5 },
  { _id: 'Nicolas Pariser', value: 62.5 },
  { _id: 'François Ozon', value: 62.5 }
]

```

9. Film avec la note maximum la plus élevée

```

lesfilms> var map9 = function () {
...   var m = null;
...   (this.grades || []).forEach(function (g) { m = (m === null) ? g.note : Math.max(m, g.note); });
...   emit("bestFilm", { title: this.title, year: this.year, maxNote: m });
... };
...
... var reduce9 = function (k, vals) {
...   var best = null;
...   vals.forEach(function (v) {
...     if (!best || (v.maxNote !== null && v.maxNote > best.maxNote)) best = v;
...   });
...   return best;
... };
...
... db.films.mapReduce(map9, reduce9, { out: "q9_meilleur_film_max_note" });
... db.q9_meilleur_film_max_note.find();
...
[
  {
    _id: 'bestFilm',
    value: { title: 'Star Wars, épisode IX', year: 2019, maxNote: 100 }
  }
]

```

10. Nombre de notes supérieures à 70 dans tous les films.

```

lesfilms> var map10 = function () {
...   var c = 0;
...   (this.grades || []).forEach(function (g) { if (g.note > 70) c += 1; });
...   emit("gt70", c);
... };
... var reduce10 = function (k, vals) { return Array.sum(vals); };
...
... db.films.mapReduce(map10, reduce10, { out: "q10_nb_notes_sup_70" });
... db.q10_nb_notes_sup_70.find();
...
[ { _id: 'gt70', value: 317 } ]

```

11. Tous les acteurs par genre, sans doublons

```

lesfilms> var map11 = function () {
...   var set = {};
...   (this.actors || []).forEach(function (a) {
...     var k = [a.last_name, a.first_name, a.birth_date].join("|");
...     set[k] = 1;
...   });
...   emit(this.genre, set);
... };
...
... var reduce11 = function (k, vals) {
...   var out = {};
...   vals.forEach(function (v) { for (var kk in v) out[kk] = 1; });
...   return out;
... };
...
... var finalize11 = function (k, reduced) {
...   return Object.keys(reduced); // liste d'acteurs uniques (clé "Nom|Prénom|Birth")
... };
...
... db.films.mapReduce(map11, reduce11, { out: "q11_acteurs_par_genre", finalize: finalize11 });
... db.q11_acteurs_par_genre.find();
...
...
[
{
  _id: 'Science-Fiction',
  value: [
    'Pearce|Guy|1967',
    'Theron|Charlize|1975',
    'Harris|Sean|1966',
    'Fassbender|Michael|1977',
    'Elba|Idris|1972',
    'Rapace|Noomi|1979',
    'Marshall-Green|Logan|1976',
    'Hutchinson|Lucy|',
    'Ford|Harrison|1942',
    'Wright|Robin|1966',
    'Leto|Jared|1971',
    'Gosling|Ryan|1980',
    'Hoeks|Sylvia|1983',
    'de Armas|Ana|1988',
    'Davis|Mackenzie|1987',

```

12. Acteurs apparaissant dans le plus grand nombre de films

```

Type "it" for more
lesfilms> var map12 = function () {
...   var seen = {};
...   (this.actors || []).forEach(function (a) {
...     var k = [a.last_name, a.first_name, a.birth_date].join("|");
...     seen[k] = 1;
...   });
...   for (var k in seen) emit(k, 1); // 1 fois par film
... };
...
... var reduce12 = function (k, vals) { return Array.sum(vals); };
...
... db.films.mapReduce(map12, reduce12, { out: "q12_nb_films_par_acteur" });
...
... // max
... var top = db.q12_nb_films_par_acteur.find().sort({ value: -1 }).limit(1).toArray()[0].value;
... // acteurs au max
... db.q12_nb_films_par_acteur.find({ value: top });
...
[ { _id: 'Ford|Harrison|1942' , value: 13 } ]

```

13. Les films par lettre de grade majoritaire (A , B , C , etc.)

```

lesfilms> var map13 = function () {
...   var freq = {};
...   (this.grades || []).forEach(function (g) {
...     freq[g.grade] = (freq[g.grade] || 0) + 1;
...   });
...
...   var maj = null, best = -1;
...   for (var gr in freq) {
...     if (freq[gr] > best) { best = freq[gr]; maj = gr; }
...   }
...
...   emit(maj, [this.title]);
... };
...
... var reduce13 = function (k, vals) {
...   var out = [];
...   vals.forEach(function (arr) { out = out.concat(arr); });
...   return out;
... };
...
... db.films.mapReduce(map13, reduce13, { out: "q13_films_par_grade_majoritaire" });
... db.q13_films_par_grade_majoritaire.find();

```

```

{
  _id: 'F',
  value: [
    'Les Misérables',
    'Knives Out',
    'Le Grand Bain',
    'Première année',
    'Prometheus',
    "Jusqu'à la garde",
    'Victoria',
    '3 Billboards : Les Panneaux de la vengeance',
    'Blade Runner 2049',
    'Hôtel des Amériques',
    'The Dark Knight Rises',
    'Les Vikings',
    'Un prophète',
    'Soupçons',
    'Le Fugitif',
    'La Collectionneuse',
    'Jour de fête',
    'La nuit du chasseur',
    'Barry Lyndon',
    'On connaît la chanson',
    "Claire's Knee",
    '58 minutes pour vivre',
    'Les Hommes du président',
    'Duel',
    'American Graffiti',
    'Chinatown',
    'Shining',
    "Out of Africa : Souvenirs d'Afrique",
    'Fenêtre sur cour',
  ],
}

```

14. Note moyenne par année de sortie des films

```

lesfilms> var map14 = function () {
...   var s = 0, c = 0;
...   (this.grades || []).forEach(function (g) { s += g.note; c += 1; });
...   emit(this.year, { sum: s, count: c });
... };
...
... var reduce14 = function (k, vals) {
...   var sum = 0, count = 0;
...   vals.forEach(function (v) { sum += v.sum; count += v.count; });
...   return { sum: sum, count: count };
... };
...
... var finalize14 = function (k, v) { return v.count ? (v.sum / v.count) : null; };
...
... db.films.mapReduce(map14, reduce14, { out: "q14_moyenne_par_annee", finalize: finalize14 });
... db.q14_moyenne_par_annee.find().sort({ _id: 1 });

```

```
[  
  { _id: 1921, value: 51.25 },  
  { _id: 1927, value: 33.5 },  
  { _id: 1931, value: 68.25 },  
  { _id: 1936, value: 58 },  
  { _id: 1937, value: 46.5 },  
  { _id: 1940, value: 47.08333333333336 },  
  { _id: 1941, value: 59 },  
  { _id: 1942, value: 24.75 },  
  { _id: 1943, value: 47.5 },  
  { _id: 1944, value: 55.75 },  
  { _id: 1946, value: 51.375 },  
  { _id: 1947, value: 47.5 },  
  { _id: 1948, value: 30.75 },  
  { _id: 1949, value: 23.5 },  
  { _id: 1950, value: 55.875 },  
  { _id: 1951, value: 57 },  
  { _id: 1952, value: 38.875 },  
  { _id: 1954, value: 40.25 },  
  { _id: 1955, value: 68 },  
  { _id: 1956, value: 57.375 }  
]
```