Bases de Données NoSQL - TP4

MAP REDUCE
COUCHDB et MONGODB

Prénom: Abdelillah

Nom: SELHI

N°Étudiant: 12206456

Groupe: INFOA3

Introduction

Le modèle **MapReduce** est un mécanisme de programmation distribuée conçu pour le traitement de très grands ensembles de données (**Big Data**) sur des clusters de machines. Son architecture permet de paralléliser les calculs, offrant ainsi une solution robuste et évolutive pour l'analyse de données massives et non structurées.

Utilité :

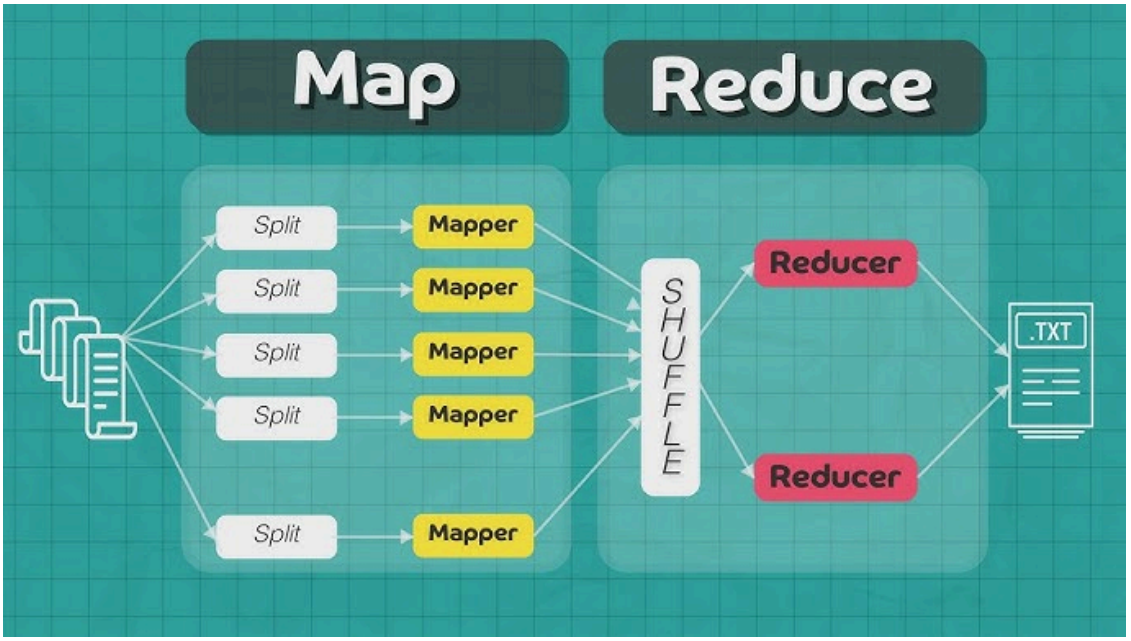
- Il permet une gestion et une analyse efficace des données qui dépassent les capacités des systèmes de bases de données relationnelles traditionnels.
- Il garantit la tolérance aux pannes en distribuant les tâches sur plusieurs nœuds.
- Il est le fondement de nombreux écosystèmes Big Data (comme Hadoop).

Map :

- **Description :** C'est la première étape du processus. Elle prend les données d'entrée, les divise en blocs (split) et les traite pour générer un ensemble de paires clé-valeur intermédiaires.
- **Utilité :** Elle sert principalement à filtrer, organiser et transformer les données brutes en un format adapté à l'agrégation.

Reduce :

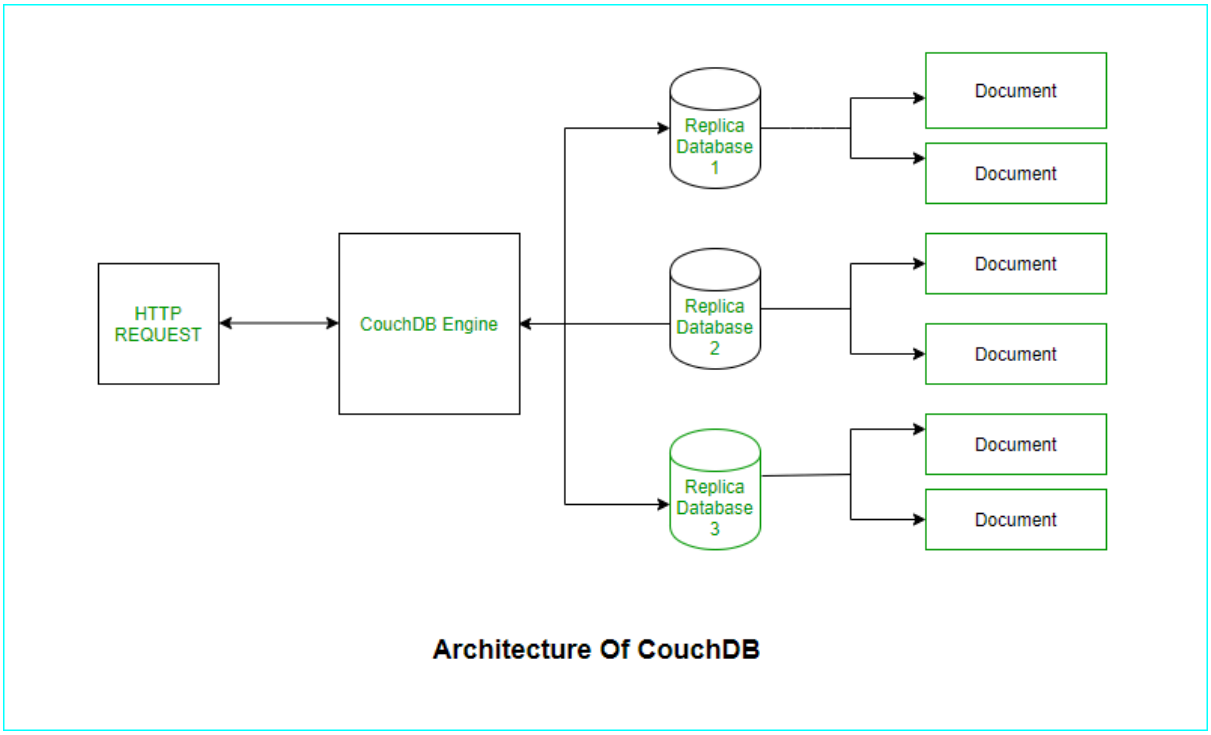
- **Description :** C'est la seconde étape. Elle reçoit toutes les valeurs associées à une même clé intermédiaire et effectue une opération d'agrégation ou de consolidation (somme, comptage, moyenne, etc.).
- **Utilité :** Elle synthétise les données pour fournir le résultat final du traitement.



source:https://www.youtube.com/watch?v=cHGaQz0E7AU

CouchDB :

- **Description :** CouchDB est une base de données NoSQL orientée document qui met l'accent sur la simplicité d'accès via HTTP/REST, une idée centrale est de proposer une architecture "web-native" : les opérations CRUD se font via des requêtes HTTP. Elle stocke les données sous forme de documents JSON et se distingue par son modèle de cohérence optimiste, sa réplication facile et son architecture décentralisée.
- **Utilité :**
    - Elle offre une grande flexibilité de schéma grâce à son format JSON.
    - Elle est particulièrement adaptée aux applications mobiles et déconnectées grâce à sa réplication.
    - Elle utilise MapReduce comme mécanisme principal pour créer des vues et exécuter des requêtes d'agrégation sur les données stockées.



source: https://media.geeksforgeeks.org/wp-content/uploads/20200626162258/CouchDB1.png

Installation de CouchDB

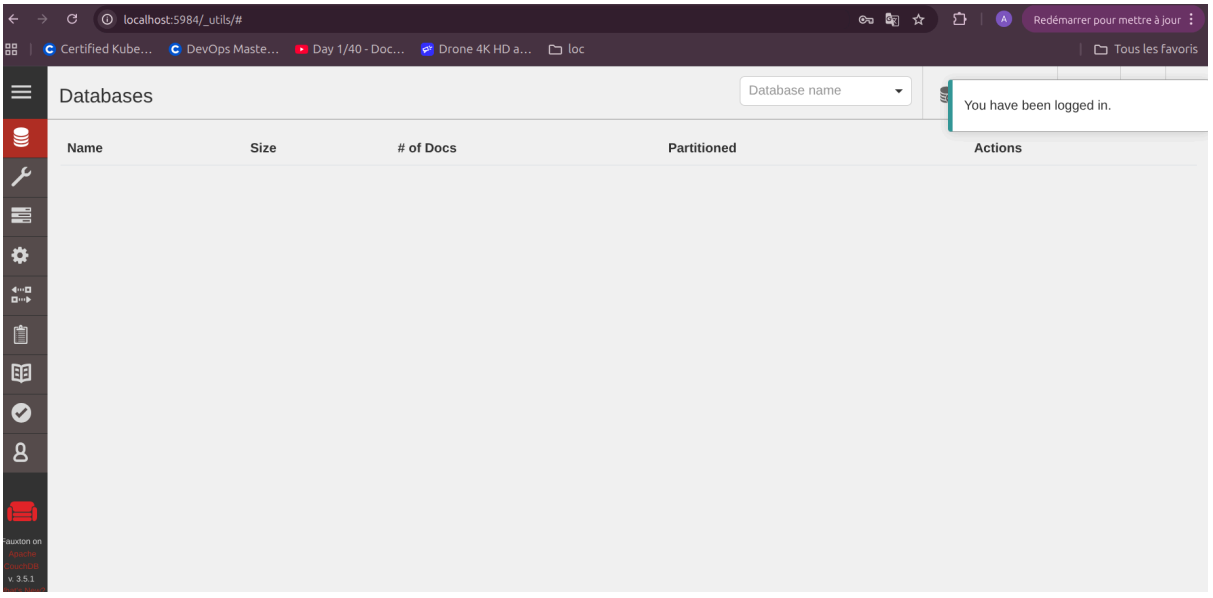
Exécuter la commande:

```
docker run -d \
--name couchdb \
-p 5984:5984 \
-e COUCHDB_USER=admin \
-e COUCHDB_PASSWORD=admin \
couchdb:latest
```

```
abdelillah@abdelillah-HP-250-G9:~/Documents/INFOA3/NoSQL/NoSQL/tp3$ docker run -d \
--name couchdb \
-p 5984:5984 \
-e COUCHDB_USER=admin \
-e COUCHDB_PASSWORD=admin \
couchdb:latest
Unable to find image 'couchdb:latest' locally
latest: Pulling from library/couchdb
51b27282aaec: Download complete
ae4ce04d0e1c: Extracting 3 s
716c80c273f8: Download complete
7f82cbc9af1f: Download complete
cd726490d7d0: Downloading [==================================================>]  105.5MB/105.5MB
609e5bdfa7b1: Download complete
4b62383183a1: Download complete
87d5a41c1790: Download complete
8856ade7c27a: Download complete
2897b671832f: Download complete
f6c9c88e767d: Download complete
```

```
abdelillah@abdelillah-HP-250-G9:~/Documents/INFOA3/NoSQL/NoSQL/tp3$ docker ps
CONTAINER ID   IMAGE           COMMAND                  CREATED              STATUS              PORTS                                       NAMES
69f5409045da   couchdb:latest  "tini -- /docker-ent…"   About a minute ago   Up About a minute   0.0.0.0:5984->5984/tcp, [::]:5984->5984/tcp   couchdb
```

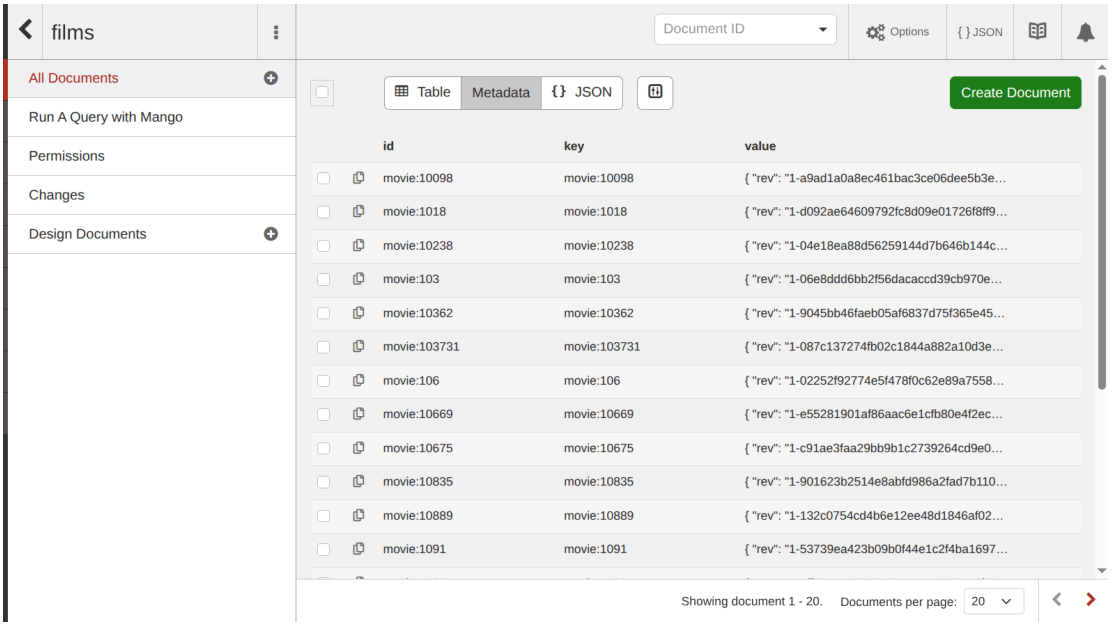
CouchDB fournit une interface web d'administration accessible: **http://localhost:5984/_utils**



Création de la base de données FILMS et insertions des données depuis le fichier JSON

```
abdelillah@abdelillah-HP-250-G9:~/Documents/INFOA3/NoSQL/NoSQL/tp3$ curl -u admin:admin -X POST "http://localhost:5984/films/_bulk_docs" -H "Content-Type
: application/json" -d @films.json
[{"ok":true,"id":"movie:11","rev":"1-c404285f85cc593f351855821ebe5fc7"},{"ok":true,"id":"movie:24","rev":"1-7f851a642ab7108adad8354952d4c560"},{"ok":true
,"id":"movie:28","rev":"1-5ef74f3007d597da5c1a41d73e00f308"},{"ok":true,"id":"movie:33","rev":"1-210992fbbd105dd91ceb02a1f6b1811d"},{"ok":true,"id":"movi
e:38","rev":"1-902184f7cc63bc4f802f3b33ffd2eb27"},{"ok":true,"id":"movie:59","rev":"1-2ca4990b59fbaee2006e0b0ef74481d0"},{"ok":true,"id":"movie:62","rev"
:"1-89d7541cf67625fbeda283dadf7294bb"},{"ok":true,"id":"movie:74","rev":"1-ea1b40608799bd603ebc7f82cf4511ac"},{"ok":true,"id":"movie:75","rev":"1-522355c
47de05179064d6218542b6ca7"},{"ok":true,"id":"movie:77","rev":"1-85291b834cfda40e18739d1b37d6deef"},{"ok":true,"id":"movie:78","rev":"1-c2b126bd26e20d4256
ea573e5bc5a11a"},{"ok":true,"id":"movie:85","rev":"1-ae348bef3600f3a445ed329201ccd191"},{"ok":true,"id":"movie:87","rev":"1-8981cf7f2d13f2d0004ae8c4a8440
```

Les données sont maintenant visibles sur l'interface visuelle:



Modèle proposé pour le problème:

Aux origines du MapReduce : soit une matrice M de dimension $N \times N$ représentant des liens d'un très grand nombre de pages web (soit N). Chaque lien est étiqueté par un poids (son importance).

1. Proposer un modèle, sous forme de ducuments structurés, pour représenter une telle matrice (s'inspirer du cas Page Rank du moteur de recherche Google, vu en cour). Soit C la collection ainsi obtenue.
2. La ligne i peut être vue comme un vecteur à N dimensions décrivant la page P_i. Spécifiez le traitement MapReduce qui calcule la norme de ces vecteurs à partir des documents de la collection C. La norme d'un vecteur $V(v_1, v_2, \ldots, v_N)$ est le scalaire $||V|| = \sqrt{v_1^2 + v_2^2 + \ldots + v_N^2}$
3. Nous voulons calculer le produit de la matrice M avec un vecteur de dimension N, $W(w_1, w_2, \ldots, w_N)$. Le résultat est un vecteur ϕ = $\sum_{j=1}^{N} M_{ij} w_j$, On suppose que le vecteur W tient en mémoire RAM et est accessible comme variable statique par toutes les fonctions de Map ou de Reduce. Spécifiez le traitement MapReduce qui implante ce calcul.

1) Je propose un **document json** pour chaque case de la matrice:

$M_{12,431}$=0.37 est équivalent à **{ "i": 12, "j": 431, "value": 0.37 }**

2) MapReduce pour la norme de la ligne i

Map

```
function (doc) {
  emit(doc.i, doc.value * doc.value);
}
```

Reduce

```
function (keys, values, rereduce) {
  return Math.sqrt(sum(values));
}
```

3) MapReduce pour ϕi=∑MijWj

Map (W accessible via la mémoire)

W est une variable globale, W = {431: 0.8, 98: 0.1, ...}

```
function (doc) {
  emit(doc.i, doc.value * W[doc.j]);
}
```

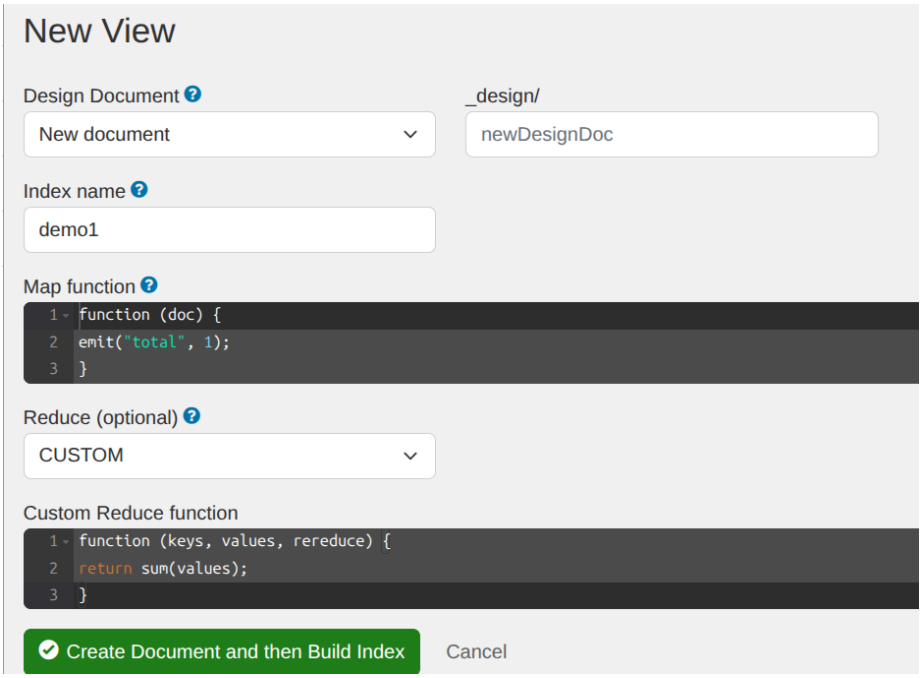
Reduce

```
function (keys, values, rereduce) {
  return sum(values);
}
```

Exercices MapReduce avec CouchDB:

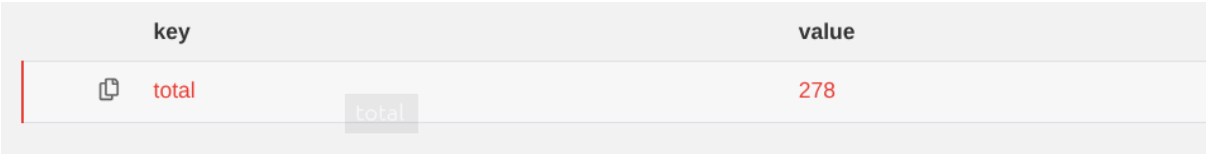
1 - Nombre total de films
Map:
```
function (doc) {
        emit("total", 1);
}
```
Reduce:
```
function (keys, values, rereduce) {
        return sum(values);
}
```

Ces fonctions sont insérés via l'interface visuelles en créant une vue:



Il faut également cocher reduce sinon seulement le résultat de map est renvoyé:

Le résultat:



2- Nombre de films par genre
Map:

```
function (doc) {
    emit(doc.genre, 1);
}
```

Reduce:

```
function (keys, values, rereduce) {
   return sum(values);
}
```

key	value
Action	36
Adventure	3
Aventure	22
Comédie	25
Comedy	1
Crime	29
Drama	14
Drame	96
Fantastique	4
Fantasy	2
Guerre	1
Histoire	1
Horreur	8

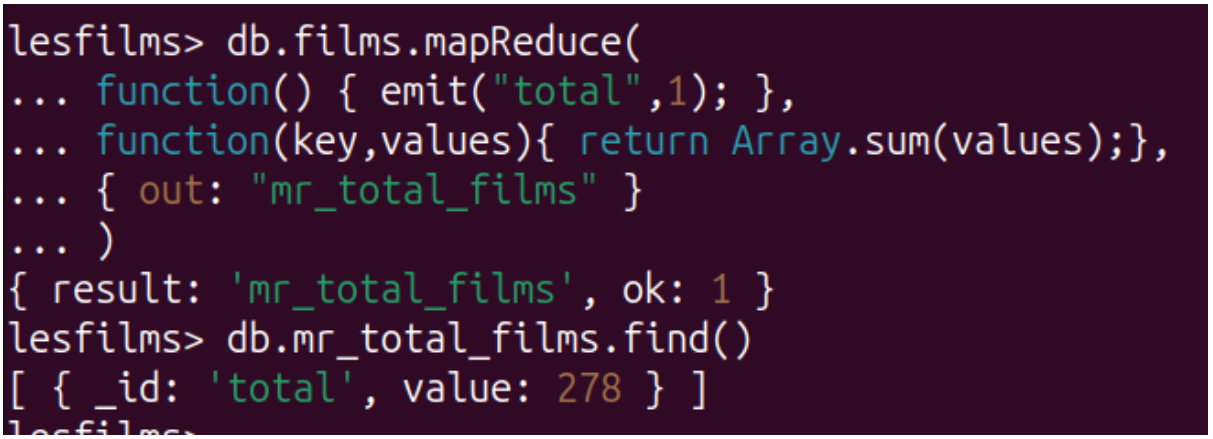
Aventure

Map Reduce avec mongoDB:

Dans CouchDB, on “crée” MapReduce en enregistrant des fonctions map/reduce dans une view stockée dans la base, puis on interroge cette view via HTTP comme un index réutilisable.

Dans MongoDB, on exécute MapReduce directement sur une collection avec une commande (MapReduce) et le résultat est écrit dans une collection de sortie, sans créer une view persistante de la même manière.

1- Compter le nombre total de films



2- Nombre de films par genre

```
[direct: mongos] mabasefilms> db.films.mapReduce(
...   function () { emit(this.genre, 1); },
...   function (key, values) { return Array.sum(values); },
...   { out: "mr_films_par_genre" }
... )
... db.mr_films_par_genre.find().sort({ value: -1 })
...
[
  { _id: 'Horreur', value: 167141 },
  { _id: 'Action', value: 166892 },
  { _id: 'Aventure', value: 166671 },
  { _id: 'Drame', value: 166548 },
  { _id: 'Comédie', value: 166421 },
  { _id: 'Science-Fiction', value: 166327 }
]
```

3- Nombre de films par réalisateur

```
lesfilms> db.films.mapReduce(
... function() { emit(this["director"], 1); },
... function(key,values) { return Array.sum(values); },
... { out: "mr_films_par_realisateur" }
... )
{ result: 'mr_films_par_realisateur', ok: 1 }
lesfilms> db.mr_films_par_realisateur.find()
[
  {
    _id: {
      _id: 'artist:291263',
      last_name: 'Peele',
      first_name: 'Jordan',
      birth_date: 1979
    },
    value: 1
  },
  {
    _id: {
      _id: 'artist:7396',
      last_name: 'Farrelly',
```

4- Nombre d'acteurs uniques

```
db.films.mapReduce(
  function () {
    if (!this.actors) return;
    this.actors.forEach(a => emit(a, 1));
  },
  function (key, values) { return 1; },
  { out: "mr_acteurs_uniques" }
)
db.mr_acteurs_uniques.countDocuments({})
```

```
lesfilms> db.films.mapReduce(
...   function () {
...     if (!this.actors) return;
...     this.actors.forEach(a => emit(a, 1));
...   },
...   function (key, values) { return 1; },
...   { out: "mr_acteurs_uniques" }
... )
... db.mr_acteurs_uniques.countDocuments({})
...
1210
```

5- Nombre de films par année

```
lesfilms> db.films.mapReduce( function () {emit(this["year"], 1);}, function
 (key, values) {return Array.sum(values);}, { out: "mr_films_par_annee" } )
{ result: 'mr_films_par_annee', ok: 1 }
lesfilms> db.mr_films_par_annee.find().sort({id:1})
[
  { _id: 2000, value: 5 }, { _id: 1950, value: 2 },
  { _id: 1944, value: 1 }, { _id: 1946, value: 2 },
  { _id: 1952, value: 2 }, { _id: 1955, value: 1 },
  { _id: 1997, value: 6 }, { _id: 1968, value: 2 },
  { _id: 2018, value: 8 }, { _id: 1948, value: 1 },
  { _id: 1927, value: 1 }, { _id: 1965, value: 1 },
  { _id: 2012, value: 4 }, { _id: 1974, value: 4 },
  { _id: 1990, value: 7 }, { _id: 1981, value: 3 },
  { _id: 2013, value: 3 }, { _id: 2017, value: 9 },
  { _id: 1957, value: 3 }, { _id: 1931, value: 1 }
]
```

6- Note moyenne par film

```
db.films.mapReduce(
  function () {
    for (var i = 0; i < this.grades.length; i++) {
      emit(this.title, { sum: this.grades[i].note, count: 1 });
    }
  },
  function (key, values) {
    var res = { sum: 0, count: 0 };
    values.forEach(function (v) {
      res.sum += v.sum;
      res.count += v.count;
    });
    return res;
  },
  {
    out: "mr_moy_film",
    finalize: function (key, res) {
      res.avg = res.count ? (res.sum / res.count) : null;
      return res;
    }
  }
)
db.mr_moy_film.find().limit(10)
```

```
[
  { _id: 'La Mouche', value: { sum: 152, count: 4, avg: 38 } },
  {
    _id: 'No Country For Old Men',
    value: { sum: 223, count: 4, avg: 55.75 }
  },
  {
    _id: 'Star Wars : Les Derniers Jedi',
    value: { sum: 155, count: 4, avg: 38.75 }
  },
  {
    _id: 'Eternal Sunshine of the Spotless Mind',
    value: { sum: 213, count: 4, avg: 53.25 }
  },
  {
    _id: 'La liste de Schindler',
    value: { sum: 322, count: 4, avg: 80.5 }
  },
  { _id: 'Comme un avion', value: { sum: 110, count: 4, avg: 27.5 } },
```

10. Nombre de notes strictement supérieures à 70

```
db.films.mapReduce(
  function () {
    this.grades.forEach(function (g) {
      if (g.note > 70) emit(1, 1);
    });
  },
  function (keys, values) {
    return Array.sum(values);
  },
  { out: "mr_high_grades" }
)
```

```
lesfilms> db.films.mapReduce(
...     function () {
...       this.grades.forEach(function (g) {
...         if (g.note > 70) emit(1, 1);
...       });
...     },
...     function (keys, values) {
...       return Array.sum(values);
...     },
...     { out: "mr_high_grades" }
... )
...
{ result: 'mr_high_grades', ok: 1 }
lesfilms> db.mr_high_grades.find()
[ { _id: 1, value: 317 } ]
lesfilms>
```

11- Acteurs par genre (sans doublons)

```
db.films.mapReduce(
function () {
    if (!this.actors || !this.genre) return;
    for (var i = 0; i < this.actors.length; i++) {
            var a = this.actors[i];
            var full = a.first_name + " " + a.last_name;
            emit(this.genre, full);
    }
  },

  function (genre, actors) {
    var seen = {};
    var out = [];
    actors.forEach(function (name) {
      if (!seen[name]) { seen[name] = true; out.push(name); }
    });
    return out;
  },
  { out: "mr_acteurs_par_genre" }
)
db.mr_acteurs_par_genre.find().limit(10)
```

```
[
  {
    _id: 'Mystère',
    value: [
      'Allison Williams',      'Caleb Landry Jones',
      'Daniel Kaluuya',        'Bradley Whitford',
      'Catherine Keener',      'Cedric Hardwicke',
      'Dame May Whitty',       'Nigel Bruce',
      'Joan Fontaine',         'Cary Grant',
      'Barbara Bel Geddes',    'Kim Novak',
      'James Stewart',         'Jackie Sawiris',
      'Madison Eginton',       'Nicole Kidman',
      'Sydney Pollack',        'Tom Cruise',
      'Leo G. Carroll',        'Martin Landau',
      'Eva Marie Saint',       'James Mason',
      'Harriet Sansom Harris', 'Stephen Tobolowsky',
      'Mark Boone Junior',     'Joe Pantoliano',
      'Carrie-Anne Moss',      'Guy Pearce'
    ]
  },
  {
    _id: 'Comédie',
    value: [
      'Camille Cottin',       'Benjamin Biolay',      'Vincent Lacoste',
      'Marie-Christine Adam', 'Chiara Mastroianni',   'Carole Bouquet',
```

12- Acteurs apparaissant dans le plus grand nombre de films

```
db.films.mapReduce(
  function () {
    var movieId = this._id;
    var seen = {};
    for (var i = 0; i < this.actors.length; i++) {
      var actor = this.actors[i].first_name + " " + this.actors[i].last_name;
      if (!seen[actor]) {
        emit(actor, 1);
        seen[actor] = true;
      }
    }
  },
  function (keys, values) {
    return Array.sum(values);
  },
  { out: "mr_actor_movie" }
)
```

```
lesfilms> db.mr_actor_movie.find()
[
  { _id: 'Susan Sarandon', value: 1 },
  { _id: 'Brent Briscoe', value: 1 },
  { _id: 'Josiane Balasko', value: 2 },
  { _id: 'Chris Tucker', value: 1 },
  { _id: 'Isabella Rossellini', value: 1 },
  { _id: 'Tatsuya Nakadai', value: 1 },
  { _id: 'François Périer', value: 1 },
  { _id: 'Charles Laughton', value: 1 },
  { _id: 'Judith Anderson', value: 1 },
  { _id: 'Marianne Basler', value: 1 },
  { _id: 'Max Minghella', value: 1 },
  { _id: 'Jan Malmsjö', value: 1 },
  { _id: 'Bruno Podalydès', value: 1 },
  { _id: 'Andréa Ferréol', value: 1 },
  { _id: 'Martin Sheen', value: 1 },
  { _id: 'Christopher Lee', value: 2 },
  { _id: 'James Gleason', value: 1 },
  { _id: 'Carole Bouquet', value: 1 },
  { _id: 'Rod Taylor', value: 1 },
  { _id: 'Richard Roundtree', value: 1 }
]
```

6- Note moyenne par film:

```
lesfilms> db.films.mapReduce(
... function() { emit("total",1); },
... function(key,values){ return Array.sum(values);},
... { out: "mr_total_films" }
... )
{ result: 'mr_total_films', ok: 1 }
lesfilms> db.mr_total_films.find()
[ { _id: 'total', value: 278 } ]
lesfilms> db.films.mapReduce(
...   function () {
...     if (this.grades && this.grades.length > 0) {
...       var total = 0;
...       this.grades.forEach(function(g) {
...         total += g.note;
...       });
...       emit(this._id, { sum: total, count: this.grades.length });
...     }
...   },
...   function (keys, values) {
...     var result = { sum: 0, count: 0 };
...     values.forEach(function(v) {
...       result.sum += v.sum;
...       result.count += v.count;
...     });
...     return result;
...   },
...   {
...     out: "mr_average_grades"
...   }
... )
```

```
[
  { _id: 'movie:8741', value: { sum: 102, count: 4 } },
  { _id: 'movie:103731', value: { sum: 259, count: 4 } },
  { _id: 'movie:10669', value: { sum: 204, count: 4 } },
  { _id: 'movie:335984', value: { sum: 105, count: 4 } },
  { _id: 'movie:6977', value: { sum: 223, count: 4 } },
  { _id: 'movie:9208', value: { sum: 272, count: 4 } },
  { _id: 'movie:343702', value: { sum: 110, count: 4 } },
  { _id: 'movie:2860', value: { sum: 241, count: 4 } },
  { _id: 'movie:42661', value: { sum: 242, count: 4 } },
  { _id: 'movie:1572', value: { sum: 234, count: 4 } },
  { _id: 'movie:345', value: { sum: 238, count: 4 } },
  { _id: 'movie:3112', value: { sum: 272, count: 4 } },
  { _id: 'movie:8217', value: { sum: 178, count: 4 } },
  { _id: 'movie:562', value: { sum: 189, count: 4 } },
  { _id: 'movie:74', value: { sum: 150, count: 4 } },
  { _id: 'movie:2252', value: { sum: 217, count: 4 } },
  { _id: 'movie:329', value: { sum: 357, count: 4 } },
  { _id: 'movie:15383', value: { sum: 278, count: 4 } },
  { _id: 'movie:389044', value: { sum: 147, count: 4 } },
  { _id: 'movie:496243', value: { sum: 115, count: 4 } }
]
```

7- Note moyenne par genre:

```
lesfilms> db.films.mapReduce(
...   function () {
...     if (this.grades && this.grades.length > 0 && this.genre) {
...       var total = 0;
...       this.grades.forEach(function(g) {
...         total += g.note;
...       });
...       emit(this.genre, { sum: total, count: this.grades.length });
...     }
...   },
...   function (keys, values) {
...     var result = { sum: 0, count: 0 };
...     values.forEach(function(v) {
...       result.sum += v.sum;
...       result.count += v.count;
...     });
...     return result;
...   },
...   {
...     out: "mr_average_by_genre"
...   }
... )
{ result: 'mr_average_by_genre', ok: 1 }
```

On aura le sum et count et la moyenne c'est **SUM/COUNT**

```
[
  { _id: 'Romance', value: { sum: 582, count: 12 } },
  { _id: 'Adventure', value: { sum: 755, count: 12 } },
  { _id: 'Drame', value: { sum: 19344, count: 384 } },
  { _id: 'Science-Fiction', value: { sum: 1978, count: 36 } }
  { _id: 'Histoire', value: { sum: 335, count: 4 } },
  { _id: 'Horreur', value: { sum: 1451, count: 32 } },
  { _id: 'Thriller', value: { sum: 1870, count: 40 } },
  { _id: 'Action', value: { sum: 6944, count: 144 } },
  { _id: 'Crime', value: { sum: 6155, count: 116 } },
  { _id: 'Mystère', value: { sum: 1231, count: 24 } },
  { _id: 'Comédie', value: { sum: 4540, count: 100 } },
  { _id: 'Mystery', value: { sum: 62, count: 4 } },
  { _id: 'Drama', value: { sum: 2571, count: 56 } },
  { _id: 'Comedy', value: { sum: 202, count: 4 } },
  { _id: 'Aventure', value: { sum: 4761, count: 88 } },
  { _id: 'War', value: { sum: 280, count: 4 } },
  { _id: 'Science Fiction', value: { sum: 171, count: 4 } },
  { _id: 'Musique', value: { sum: 398, count: 8 } },
  { _id: 'Western', value: { sum: 574, count: 12 } },
  { _id: 'Guerre', value: { sum: 278, count: 4 } }
]
```

8- Note moyenne par réalisateur:

```
lesfilms> db.films.mapReduce(
...   function () {
...     if (this.grades && this.grades.length > 0 && this.director) {
...       var total = 0;
...       this.grades.forEach(function(g) {
...         total += g.note;
...       });
...       var directorName = this.director.first_name + " " + this.director.last_name;
...       emit(directorName, { sum: total, count: this.grades.length });
...     }
...   },
...   function (keys, values) {
...     var result = { sum: 0, count: 0 };
...     values.forEach(function(v) {
...       result.sum += v.sum;
...       result.count += v.count;
...     });
...     return result;
...   },
...   {
...     out: "mr_average_by_director"
...   }
... )
...
{ result: 'mr_average_by_director', ok: 1 }
lesfilms> db.mr_average_by_director.find()
[
  { _id: 'Andrei Tarkovsky', value: { sum: 591, count: 12 } },
  { _id: 'David Lynch', value: { sum: 209, count: 8 } },
  { _id: 'Charles Laughton', value: { sum: 272, count: 4 } },
  { _id: 'Bruno Podalydès', value: { sum: 110, count: 4 } },
  { _id: 'William Wyler', value: { sum: 317, count: 4 } },
  { _id: 'Sydney Pollack', value: { sum: 754, count: 12 } },
  { _id: 'Jordan Peele', value: { sum: 162, count: 4 } },
  { _id: 'Jean-Luc Godard', value: { sum: 495, count: 12 } },
  { _id: 'Peter Weir', value: { sum: 141, count: 4 } },
  { _id: 'Thierry de Peretti', value: { sum: 253, count: 4 } },
  { _id: 'Céline Sciamma', value: { sum: 211, count: 4 } },
  { _id: 'Sergio Leone', value: { sum: 272, count: 4 } },
  { _id: 'Lana Wachowski', value: { sum: 187, count: 4 } },
  { _id: 'Xavier Legrand', value: { sum: 199, count: 4 } },
  { _id: 'Thomas Lilti', value: { sum: 288, count: 8 } },
  { _id: 'Fernando Meirelles', value: { sum: 149, count: 4 } },
  { _id: 'Stanley Kubrick', value: { sum: 1674, count: 28 } },
  { _id: 'François Truffaut', value: { sum: 898, count: 16 } },
  { _id: 'Gilles Lellouche', value: { sum: 163, count: 4 } },
  { _id: 'Christophe Honoré', value: { sum: 226, count: 4 } }
]
```

9- Film avec la note maximale:

Il faut regarder la note maximale par film puis afficher en ordre décroissant

```
lesfilms> db.films.mapReduce(
...   function () {
...     if (this.grades && this.grades.length > 0) {
...       var maxNote = this.grades[0].note;
...       this.grades.forEach(function(g) {
...         if (g.note > maxNote) maxNote = g.note;
...       });
...       emit(this._id, maxNote);
...     }
...   },
...   function (keys, values) {
...     return Math.max.apply(null, values);
...   },
...   { out: "mr_max_grade_per_film" }
... )
```

```
lesfilms> db.mr_max_grade_per_film.find().sort({value:-1})
[
  { _id: 'movie:181812', value: 100 },
  { _id: 'movie:145', value: 100 },
  { _id: 'movie:9361', value: 100 },
  { _id: 'movie:424', value: 100 },
  { _id: 'movie:269', value: 100 },
  { _id: 'movie:891', value: 100 },
  { _id: 'movie:754', value: 100 },
  { _id: 'movie:604', value: 100 },
  { _id: 'movie:10675', value: 100 },
  { _id: 'movie:247', value: 100 },
  { _id: 'movie:1891', value: 100 },
  { _id: 'movie:777', value: 100 },
  { _id: 'movie:6075', value: 100 },
  { _id: 'movie:558', value: 100 },
  { _id: 'movie:832', value: 100 },
  { _id: 'movie:103', value: 100 },
  { _id: 'movie:665', value: 100 },
  { _id: 'movie:1091', value: 100 },
  { _id: 'movie:9208', value: 99 },
  { _id: 'movie:5503', value: 99 }
]
```

14- Calculer la note moyenne par année de sortie des films:

```
lesfilms> db.films.mapReduce(
...   function () {
...     if (this.grades && this.grades.length > 0 && this.year) {
...       var total = 0;
...       this.grades.forEach(function(g) {
...         total += g.note;
...       });
...       emit(this.year, { sum: total, count: this.grades.length });
...     }
...   },
...   function (keys, values) {
...     var result = { sum: 0, count: 0 };
...     values.forEach(function(v) {
...       result.sum += v.sum;
...       result.count += v.count;
...     });
...     return result;
...   },
...   {
...     out: "mr_average_by_year"
...   }
... )
...
{ result: 'mr_average_by_year', ok: 1 }
lesfilms> db.mr_average_by_year.find()
[
  { _id: 2000, value: { sum: 816, count: 20 } },
  { _id: 1950, value: { sum: 447, count: 8 } },
  { _id: 1944, value: { sum: 223, count: 4 } },
  { _id: 1946, value: { sum: 411, count: 8 } },
  { _id: 1952, value: { sum: 311, count: 8 } },
  { _id: 1955, value: { sum: 272, count: 4 } },
  { _id: 1997, value: { sum: 1121, count: 24 } },
  { _id: 1968, value: { sum: 501, count: 8 } },
  { _id: 2018, value: { sum: 1501, count: 32 } },
  { _id: 1948, value: { sum: 123, count: 4 } },
  { _id: 1927, value: { sum: 134, count: 4 } },
  { _id: 1965, value: { sum: 107, count: 4 } },
  { _id: 2012, value: { sum: 804, count: 16 } },
  { _id: 1974, value: { sum: 725, count: 16 } },
  { _id: 1990, value: { sum: 1370, count: 28 } },
  { _id: 1981, value: { sum: 790, count: 12 } },
  { _id: 2013, value: { sum: 575, count: 12 } },
  { _id: 2017, value: { sum: 1795, count: 36 } },
  { _id: 1957, value: { sum: 668, count: 12 } },
  { _id: 1931, value: { sum: 273, count: 4 } }
]
```

15- Identifier les réalisateurs dont la note moyenne de tous leurs films est supérieure à 80:

```
lesfilms> db.films.mapReduce(
...   function () {
...     if (this.grades && this.grades.length > 0 && this.director) {
...       var total = 0;
...       this.grades.forEach(function(g) {
...         total += g.note;
...       });
...       var directorName = this.director.first_name + " " + this.director.last_name;
...       emit(directorName, { sum: total, count: this.grades.length });
...     }
...   },
...   function (keys, values) {
...     var result = { sum: 0, count: 0 };
...     values.forEach(function(v) {
...       result.sum += v.sum;
...       result.count += v.count;
...     });
...     return result;
...   },
...   {
...     out: "mr_average_by_director"
...   }
... )
```

Ceci renvoie pour chaque réalisateur la somme de ses grade et le nombre de notes puis il faudra diviser pour avoir la moyenne et classer en ordre décroissant

```
lesfilms> db.mr_average_by_director.find().forEach(doc => {
...   var average = doc.value.sum / doc.value.count;
...   if (average > 80) {
...     print(doc._id + " : " + average);
...   }
... });
```

Aucun réalisateur n'a strictement > 80 dans toutes les notes