



# Le NoSQL et MongoDB

Le Big Data – défis du siècle

# Le Big Data – Défis du siècle : plan

- Introduction
- Le modèle des 3V
- Le mode de stockage
- Cas d'utilisations

# Le Big Data – Introduction

- Pourquoi parle-t-on de Big Data ?

Aujourd'hui, la quantité de données, la vitesse de génération ainsi que la variété dont elles font preuve, a marqué la limite d'utilisation des systèmes de gestion traditionnels.

« Big » car les données générées ces deux dernières années représentent près de 90% de toutes les données créées par l'homme depuis la nuit des temps\*.

\*Données 2015



- Pourquoi les systèmes traditionnels ont-ils atteint leurs limites ?

Les systèmes utilisés sont devenus peu performants pour le traitement de ces données. En effet, celles-ci, en plus d'être nombreuses, sont semi-structurées alors que les systèmes traditionnels ont besoin de données structurées pour être performants.

- Quelles technologies ont résolu une partie de ces défis ?


Le NoSQL est une des techniques qui a été développée par de grands leaders du web (Google, Facebook, Amazon, ...) afin de répondre à leurs besoins internes.



# Le Big Data – Le modèle des « 3V »

Le Big Data a trois caractéristiques : la vitesse, le volume et la variété.

- La vitesse est la vitesse de création et de modification des données.
- Le volume est la quantité de données à traiter, stocker et retrouver.
- La variété des données provient de leur structure qui n'est pas toujours semblable.



# Le Big Data – Le mode de stockage

Les données du Big Data peuvent être stockées dans différentes structures comme le Cloud, les data centers ou dans un système de fichiers distribués.

- Un système de fichiers distribués applique le principe de « Data Locality ». Ce principe signifie que les données sont stockées là où elles seront traitées.  
Ce système privilégie également le stockage des données sur un très grand nombre de petites machines au lieu d'une grosse.





# Le Big Data – Utilisation

Le Big Data est une source d'information sans précédent dans l'histoire de l'humanité.

Il est actuellement utilisé dans :

- le domaine scientifique afin de traiter les données des grosses expérimentations.
- le domaine commercial afin d'aider les décisions d'entreprise en fonction des profils clients analysés, de leurs habitudes de consommation, etc.
- le domaine politique pour influencer l'électorat, orienter un débat, ...



Le NoSQL – L'avenir des bases de données ?

# Le NoSQL : plan

- Introduction / Défis
- Théorème CAP
- Utilisation
- Système de gestion de données
- Schéma et Contenu
- Type de moteur NoSQL

# Le NoSQL – Introduction

- La technologie NoSQL a été créée dans le but de répondre au besoin du Big Data.
- Le NoSQL est un système distribué de gestion de base de données.
- Le NoSQL privilégie une montée en charge horizontale permettant de répondre plus facilement à l'augmentation des quantités de données. Pour parvenir à suivre la vitesse de génération des données, les moteurs NoSQL ont généralement abandonné le modèle relationnel et son schéma structuré ce qui leur permet une plus grande flexibilité.

# NoSQL - Les défis

Les bases de données NoSQL ayant été créés pour répondre au besoin du Big Data, il est évident que ces défis sont ceux du Big Data :

- Capture des données  
Gestion des données semi-structurée fournies par plusieurs types de support.
- Stockage des données
- Recherche rapide
- Partage
- Transfert
- Analyse
- Présentation

# Le NoSQL – Le théorème CAP

- CAP (Consistency, Availability and Partition tolerance) pour Consistance, Disponibilité et tolérance à la Partition.
- Selon ce théorème, les bases de données distribuées ne sont capables de garantir que deux de ces trois points. Ainsi, en privilégiant la disponibilité et la tolérance à la partition, les bases de données NoSQL perdent la capacité de garantir une certaine consistance.
- Pour palier ce problème, les bases de données NoSQL garantissent une cohérence finale des données à travers tous les nœuds.

# NoSQL- Utilisation

Question: Que peut-on faire avec ces données ?

- Le NoSQL a été créé dans le but d'analyser les données. Ainsi, nous pouvons :
  - analyser le comportement d'achat des clients ;
  - analyser les produits les plus vendus ;
  - prendre des décisions en fonction de ces analyses ;
  - produire des factures, etc.
- La différence, entre le NoSQL et les bases de données traditionnelles, est que la difficulté de traitement est laissée au code clients. Ce qui permet une plus grande flexibilité, vitesse de traitement et accessibilité.

# NoSQL - Système de gestion de données


La gestion d'une base de données est laissée à un logiciel spécialisé appelé: *système de gestion de base de données (SGBD)*

- Les SGBD s'occupent de :
  - L'organisation des données.  
Organise les données en entités stockées sur le disque de manière à garantir un accès rapide à celles-ci.
  - La gestion des données.  
Garantir la cohérence finale des données stockées.
  - Assurer l'accès aux données.  
Accès aux données stockées par plusieurs clients et logiciels de traitement simultané.





- La protection contre les incidents  
Les moteurs NoSQL garantissent cette protection par la réplication des données.
- Le partitionnement des données  
Les bases de données NoSQL permettent un partitionnement des données automatiques permettant un gain en vitesse de traitement.
- La réplication des données  
La majorité des moteurs NoSQL permettent une réplication automatique des données sur différents serveurs.



# NoSQL - Schéma et contenu

- La plupart des moteurs NoSQL sont dits « schema-free » ou « schema-less ». Cela signifie que l'existence d'une structure de données prédéfinie n'est pas requise pour l'insertion de données.
- La non-existence d'un schéma permet au NoSQL d'être plus adapté aux méthodes de développement actuel.
- Elle permet également d'optimiser l'espace en enregistrant que ce qui est nécessaire.



# NoSQL - La redondance interne

- Dans les bases de données NoSQL, la redondance d'information est utilisée afin de permettre de meilleures performances dans les opérations de lecture de données en évitant de nombreuses requêtes parfois complexes.
- Le prix de l'espace de stockage par rapport au prix d'opération processeur en est la raison.

# NoSQL - Les structures physiques

Les données étant stockées sur le disque dur, y accéder peut prendre du temps. Comment le diminuer ?

- En NoSQL, les données sont généralement réparties sur plusieurs serveurs qui exécutent une partie de la requête. Cela permet de diminuer le temps d'accès aux données.  
=> Mécanisme de répartition.
- Le NoSQL intègre également un mécanisme emprunté au monde relationnel : l'indexation

L'indexation est un « catalogue » permettant de retrouver plus rapidement une information.

# Le NoSQL – Les types de moteurs

Il existe différents types de moteurs NoSQL :

- orientée colonne  
Ces moteurs sont généralement assez proches conceptuellement des bases de données traditionnelles mais permettent de gérer de plus grosses quantités de données.
- orienté document  
Ces moteurs sont destinés à des besoins plus « modestes » et s'adaptent plus au paradigme de programmation orienté Objet.
- orientée clé-valeur  
Ces moteurs sont des dépôts de données.
- orienté graph  
Ces moteurs sont destinés à être utilisés dans le cadre des relations complexes entre entités.

MongoDB – Un moteur NoSQL

# MongoDB – Un moteur NoSQL : Plan

- Introduction
- Les données
- Les concepts
- La réplication
- La répartition
- Le DDL
- Le DML



# MongoDB - Introduction

# MongoDB – Introduction

- MongoDB est une base de données NoSQL orienté document. Ce moteur a été développé initialement pour des besoins plus modestes. Cependant, il est capable de réaliser une montée en charge horizontale.
- Ce moteur garantit la disponibilité et la tolérance au partitionnement ainsi qu'une cohérence finale des données distribuées.

# MongoDB – Les données



# MongoDB – Les données

- Plan
  - Première approche
    - Les collections
    - Les documents
  - Utilisation
  - Première conclusion
  - Exercices

# MongoDB - Première approche des données

Exemple de données à traiter

Commande N° : 30188		Date : 2/1/2009		
Numéro client	B512			
Nom	GILLET			
Adresse	14, r. de l'Eté			
Localité	Toulouse			
N° PRODUIT	LIBELLE PRODUIT	PRIX	QUANTITE	SOUS-TOTAL
CS464	CHEV. SAPIN 400x6x4	220	180	39600
PA45	POINTE ACIER 45 (20K)	105	22	2310
PA60	POINTE ACIER 60 (10K)	95	70	6650
PH222	PL. HETRE 200x20x2	230	92	21160
TOTAL COMMANDE				69720

# MongoDB - Première approche des données

- À partir de cet exemple, il nous est possible de construire une base de données orientée document.
- En regardant ce bon de commande, on remarque que certaines données ont un lien. Il suffit de regrouper les données reliées au sein d'une même collection.
- Ainsi, cet exemple nous demandera d'avoir plusieurs collections:
  - une collection « clients »
  - une collection « produits »

# MongoDB – Première approche des données

- Les collections:
  - une collection est l'équivalent des tables dans les bases de données traditionnelles dites relationnelles, elles sont donc composées de données regroupées en « document ».
  - Ces collections n'obligent pas les documents à posséder une structure commune.
  - Il est cependant conseillé d'adopter une certaine rigueur dans la structure.



# MongoDB – Première approche des données

---

- Les documents
  - Un document est l'équivalent d'un enregistrement dans les bases de données traditionnelles dites relationnelles, ils composent donc l'unité de stockage.

```
{
  "_id": ObjectId('56d3fc4a9290688a3b724486'),
  "IDENTIFIANT": {
    "NOM": "Exemple",
    "PRENOM": "Document",
    "DATE_NAISSANCE": ISODate('2000-12-31T23:00:00.000Z')
  },
  "ADRESSE": {
    "RUE": "exempleRue",
    "LOCALITE": "exempleLocalite",
    "CODE_POSTALE": 1000,
    "NUM": 10
  },
  "DROIT": [
    "modo",
    "user",
    "redacteur"
  ]
}
```

# MongoDB – Première approche des données

	_id	NOM	ADRESSE	LOCALITE	CAT	COMPTE	COMMANDES
1	B332	MONTI	23,a. Dumont	Geneve	B2	0	
2	B112	HANSENNEA	23,a. Dumont	Poitiers	C1	1250	
3	F010	TOUSSAINT	23,a. Dumont	Poitiers	C1	0	
4	S127	VANDERKA	23,a. Dumont	Namur	C1	-4580	[ 1 element ]
5	B062	GOFFIN	23,a. Dumont	Namur	B2	-3200	
6	B512	GILLET	23,a. Dumont	Toulouse	B1	-8700	[ 1 element ]
7	K111	VANBIST	23,a. Dumont	Lille	B1	720	[ 1 element ]
8	C123	MERCIER	23,a. Dumont	Namur	C1	-2300	
9	C003	AVRON	23,a. Dumont	Toulouse	B1	-1750	
10	K729	NEUMAN	23,a. Dumont	Toulouse		0	
11	C400	FERARD	23,a. Dumont	Poitiers	B2	350	[ 3 elements ]
12	F011	PONCELET	23,a. Dumont	Toulouse	B2	0	[ 1 element ]
13	L422	FRANCK	23,a. Dumont	Namur	C1	0	
14	D063	MERCIER	23,a. Dumont	Toulouse		-2250	
15	F400	JACOB	23,a. Dumont	Bruxelles	C2	0	
16	S712	GUILLAUME	23,a. Dumont	Paris	B1	0	
17	K112	EXO	4/105, r. Ch...	Namur		125	
18	F401	1234	23,a. Dumont	Bruxelles	C2	2523	
19	B632	AROUNDI	23,a. Dumont	Geneve	B2	0.95	

# MongoDB – Première approche des données

	id	CHAMP	NOM	ADRESSE	LOCALITE	CAT	COMPTE	COMMANDES
1	B332		MONTI	23,a. Dumont	Geneve	B2	0	DOCUMENT
2	B112		HANSENNEA	23,a. Dumont	Poitiers	C1	1250	Champ inexistant
3	F010		TOUSSAINT	23,a. Dumont	Poitiers	C1	0	
4	S127		VANDERKA	23,a. Dumont	Namur	C1	-4580	
5	B062		GOFFIN	23,a. Dumont	Namur	B2	-3200	
6	B512		GILLET	23,a. Dumont	Toulouse	B1	-8700	[ 1 element ]
7	K111		VANBIST	23,a. Dumont	Lille	B1	720	[ 1 element ]
8	C123		MERCIER	23,a. Dumont	Namur	C1	-2300	
9	C003		AVRON	23,a. Dumont	Toulouse	B1	-1750	
10	K729		NEUMAN	23,a. Dumont	Toulouse		0	
11	C400		FERARD	23,a. Dumont	Poitiers	B2	350	[ 3 elements ]
12	F011		PONCELET	23,a. Dumont	Toulouse	B2	0	[ 1 element ]
13	L422		FRANCK	23,a. Dumont	Namur	C1	0	
14	D063		MERCIER	23,a. Dumont	Toulouse		-2250	COLLECTION
15	F400		JACOB	23,a. Dumont	Bruxelles	C2	0	
16	S712		GUILLAUME	23,a. Dumont	Paris	B1	0	
17	K112		EXO	4/105, r. Ch...	Namur		125	
18	F401		1234	23,a. Dumont	Bruxelles	C2	2523	
19	B632		AROUNDI	23,a. Dumont	Geneve	B2	0.95	

# MongoDB - Première approche des données

- Exercices
  - Dans le cadre de la collection « clients », quels en serait les composants ?
  - Dans le cadre de la collection « produits », quels en serait les composants ?

# MongoDB - Utilisation

- Le rôle d'une base de données est la mise à disposition et le stockage des données nécessaires au fonctionnement d'une société.
- On peut donc les interroger de manière plus ou moins aisée
  - Quel est le numéro, le nom et le montant du compte de tous les clients habitant à Toulouse ?
    - `db.clients.find({LOCALITE: 'Toulouse'}, {_id: true, NOM: true, COMPTE: true})`
  - Quelles sont les commandes réalisées par les clients de Toulouse
    - `db.clients.find({LOCALITE: 'Toulouse'}, {_id: false, COMMANDES: true})`

# MongoDB - Première conclusion

- Une base de données MongoDB est constituée d'un ensemble de collections
- Chaque collection est composée de documents
- Chaque document constitue un ensemble de données liées entre elles
- Chaque document est unique et identifié par un identifiant

# MongoDB - Première conclusion

- Les données calculables ne sont pas stockées en base de données
- On conserve les données relatives au sein d'un même document.
  - Redondance d'information, mais rapidité de traitement.
- Les bases de données MongoDB s'adaptent aux besoins actuels d'une société. Elles s'adaptent donc plus facilement au modèle de développement « Agile » de par sa flexibilité.
- Ces bases de données demandent une analyse moins rigoureuse puisqu'il est possible d'ajouter ou de retirer des attributs à un ou plusieurs documents.



# MongoDB – Les concepts



# MongoDB – Les concepts

- Plan
  - Illustration
  - Répartition des données
  - Réplication des données
  - Une base de données distribuée
  - Valeur NULL
  - Identifiant
  - Curseur

# MongoDB – Illustration

Visualisation sous  
forme de table

	_id	NOM	ADRESSE	LOCALITE	CAT	COMPTE	COMMANDES
1	B332	MONTI	23,a. Dumont	Geneve	B2	0	
2	B112	HANSENNEA	23,a. Dumont	Poitiers	C1	1250	
3	F010	TOUSSAINT	23,a. Dumont	Poitiers	C1	0	
4	S127	VANDERKA	23,a. Dumont	Namur	C1	-4580	[ 1 element ]
5	B062	GOFFIN	23,a. Dumont	Namur	B2	-3200	
6	B512	GILLET	23,a. Dumont	Toulouse	B1	-8700	[ 1 element ]
7	K111	VANBIST	23,a. Dumont	Lille	B1	720	[ 1 element ]
8	C123	MERCIER	23,a. Dumont	Namur	C1	-2300	
9	C003	AVRON	23,a. Dumont	Toulouse	B1	-1750	
10	K729	NEUMAN	23,a. Dumont	Toulouse		0	
11	C400	FERARD	23,a. Dumont	Poitiers	B2	350	[ 3 elements ]
12	F011	PONCELET	23,a. Dumont	Toulouse	B2	0	[ 1 element ]
13	L422	FRANCK	23,a. Dumont	Namur	C1	0	
14	D063	MERCIER	23,a. Dumont	Toulouse		-2250	
15	F400	JACOB	23,a. Dumont	Bruxelles	C2	0	
16	S712	GUILLAUME	23,a. Dumont	Paris	B1	0	
17	K112	EXO	4/105, r. Ch...	Namur		125	
18	F401	1234	23,a. Dumont	Bruxelles	C2	2523	
19	B632	AROUNDI	23,a. Dumont	Geneve	B2	0.95	

# MongoDB – Illustration

Visualisation sous  
forme de document

```
{
  "_id": ObjectId('52873b364038253faa4bbc0e'),
  "student_id": "STU002",
  "sem": "sem1",
  "english": "A",
  "maths": "A+",
  "science": "A",
  "dateTime": ISODate('1992-07-18T22:00:00.000Z')
}
```

```
{
  "_id": ObjectId('52873b5d4038253faa4bbc0f'),
  "student_id": "STU001",
  "sem": "sem1",
  "english": "A+",
  "maths": "A+",
  "science": "A",
  "dateTime": ISODate('1991-07-18T22:00:00.000Z')
}
```

# MongoDB – Illustration

Visualisation sous forme de liste

```
{ "_id": "ObjectId(52873b364038253faa4bbc0e)", "student_id": "STU002", "sem": "sem1", "english": "A", "maths": "A+", "science": "A", "dateTime": "ISODate(1992-07-18T22:00:00.000Z)" }
{ "_id": "ObjectId(52873b5d4038253faa4bbc0f)", "student_id": "STU001", "sem": "sem1", "english": "A+", "maths": "A+", "science": "A", "dateTime": "ISODate(1991-07-18T22:00:00.000Z)" }
{ "_id": "ObjectId(52873b7e4038253faa4bbc10)", "student_id": "STU003", "sem": "sem1", "english": "A+", "maths": "A", "science": "A+", "dateTime": "ISODate(1991-07-18T22:00:00.000Z)" }
{ "_id": "ObjectId(56b320cc67f06e94da74caf0)", "student_id": "STUD004", "sem": "sem1", "english": "B-", "maths": "A+", "science": "C" }
{ "_id": "STU56bca724a02e2340fd50493e", "name": "Test", "prenom": "id" }
{ "_id": "STU5" }
```

# MongoDB – Répartition de données

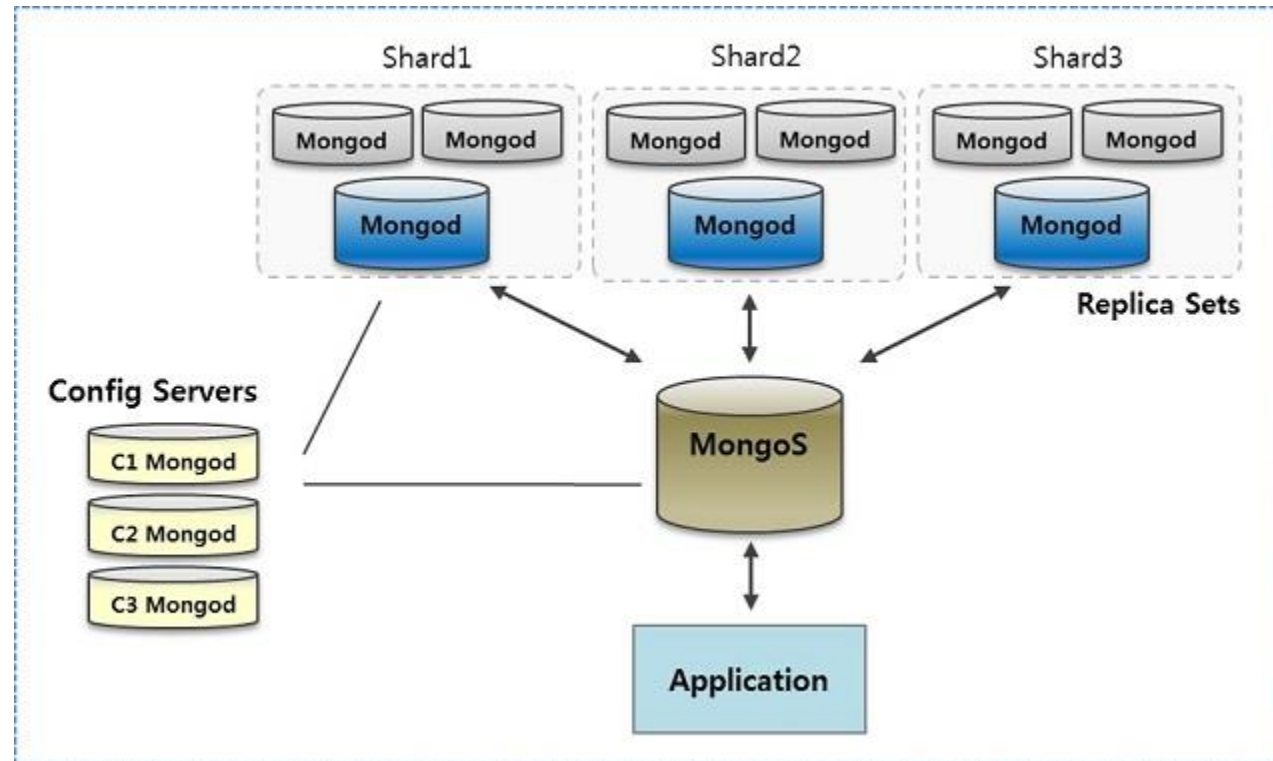
- La répartition est un principe fondamental. Il permet de répartir une collection sur plusieurs serveurs en les répartissant selon un ensemble de clés-valeurs défini par le gestionnaire de base de données.
- Cette répartition possède une architecture maître-esclave. Cela lui permet de répartir la charge de travail sur les esclaves qui n'exécutent que des requêtes affectant leurs contenus.

# MongoDB – Réplication de données

- La réplication est un principe fondamental. Il permet d'effectuer une copie de la base de données contenue dans un serveur. Cela permet une haute disponibilité.
- Les serveurs effectuant une réplication sont regroupés au sein d'un « ensemble de réplication » (replica set) qui possède une architecture maître-esclave avec basculement automatique du maître en cas de défaillance de ce dernier.

# MongoDB – Base de données distribuée

- Organisation d'une base de données distribuée selon l'architecture maître-esclave.





# MongoDB – La valeur « NULL »

Dans les moteurs NoSQL, les données ne doivent pas respecter un schéma préétabli. Dès lors, l'absence d'un attribut, lors d'opérations de lecture, est comprise comme l'association de cet attribut par une valeur « NULL », mais qui n'est pas stocké en mémoire.

# MongoDB –La valeur « NULL »

- Exemple d'utilisation de la valeur « NULL »
  - `db.<collection>.find({<champ>: null})`

Renvoie les documents contenant le champ <champ> et dont la valeur est null, et les documents dont le champ <champ> n'existe pas.
  - `db.<collection>.find({<champ>: {$exists: false}})`

Renvoie que les documents ne contenant pas le champ <champ>.
  - `db.<collection>.find({$and: [{<champ>: {$exist: true}}, {<champ>: null}])`

Renvoie les documents possédant le champ <champ> ayant une valeur « null »

# MongoDB - Identifiant

- Le rôle de l'identifiant dans une base de données est d'identifier de manière unique une entité.
- Ainsi l'identifiant de MongoDB est un nombre de 12 octets constitué d' :
  - un numéro attribué au serveur (3 octets)

Ce numéro est utilisé lors de la répartition des données. Permet de connaître le serveur sur lequel est stocké le document.
  - une date (permettant la gestion de versions des données) (4 octets)

Une date en seconde depuis le 1er janvier 1970 (Création de Linux)
  - un numéro unique généré pour la collection (3 octets)

Compteur commençant à une valeur aléatoire.
  - le « process id » du démon Mongod (2 octets)

# MongoDB - Identifiant

- L'identifiant définit une contrainte d'unicité du document
- Il est généré automatiquement par le moteur ou défini par le code client
- Un identifiant peut être constitué de plusieurs attributs
- L'identifiant d'un autre document peut être inséré dans un document. Ceci permettra de réaliser une requête au serveur sur base de la référence récupérée
  - Cette « jointure » est réalisée par le code client

# MongoDB – Le curseur

- Dans MongoDB, les requêtes renvoient un objet appelé « cursor ». Cet objet contient quelques méthodes permettant de le traiter. Ainsi, en récupérant cet objet dans une variable, nous pouvons l'utiliser pour effectuer des requêtes sur base d'autres requêtes.
- Une variable est un nom symbolique permettant de stocker un objet et ainsi de le réutiliser ultérieurement.
  - Pour déclarer une variable, « var <nom\_symbolique> »
- Exemple
  - var produits = <requête>

# MongoDB – Le DDL



# MongoDB – Le DDL

- Plan
  - Introduction
  - Création de collections
  - Suppression de collections
  - Insertion de documents
  - Suppression de documents
  - Création d'index
  - Suppression d'index
  - Exercices

# Le DDL - Introduction

- Le DDL (Data Description Language) est un langage permettant de définir les structures d'une collection, d'un document.
- MongoDB a été développé dans le paradigme « Objet »  
Il y a un objet parent global: « db ». Celui-ci possède des méthodes et des objets accessibles par l'opérateur « . ». Ainsi, « db.collection » fait appel à l'objet collection de l'objet db. (Important pour la suite)
- Dans MongoDB, le DDL est composé de plusieurs méthodes:
  - la création de collection
  - l'insertion de document
  - la suppression de document
  - la suppression de collection
  - la création et suppression des index



# Le DDL – Création de collections

- Dans MongoDB, la création d'une base de données se fait par son utilisation et l'insertion de données dans celle-ci.
  - use <nom\_base\_de\_données>
- Pour créer une collection dans ce moteur NoSQL, il y a deux possibilités:
  - en utilisant la méthode « createCollection() » de l'objet « db » qui permet de définir différentes options de contrôle sur les documents
  - MongoDB crée automatiquement la collection utilisée lors d'une insertion si elle n'existe pas au sein de la base de données. Les options sont celles par défaut.
- Les collections n'ayant pas de schéma préétabli, seuls leurs noms sont nécessaires à leurs créations.

# Le DDL – Suppression de collections

- La suppression d'une collection se fait facilement au vu de l'inexistence de contrainte de schéma.
- Elle se fait par la méthode « drop() » de l'objet collection dont le nom est la collection à supprimer.

# Le DDL – L'insertion de documents

- L'insertion de document se fait grâce à la méthode « `insert()` » d'une collection.
- L'insertion d'un document se fait toujours entre « `{ }` »
- L'insertion de plusieurs documents peut se faire de deux manières différentes:
  - en utilisant la méthode « `insert([ { }, { }, { }, ... ])` »
  - en réalisant l'importation de données d'un fichier JSON

# Le DDL – L'insertion de documents

- L'insertion de plusieurs documents par la méthode « insert() »

```
db.collection.insert([
    {<document1>},
    {<document2>},
    ...
])
```
- L'insertion de plusieurs documents par Console
  - Dans le fichier binaire du dossier d'installation, il existe un utilitaire : mongoimport.
  - Utilisation:

```
/ « chemin_acces »/mongoimport --dbpath  
« chemin_acces_db » -c « collection » <  
« chemin_fichier json »
```

# Le DDL – L'insertion de documents

### Import documents

local/exercice

Source

☒ Import documents from file  
☐ Import documents from textbox

Name of destination-collection\*

Import mode

☐ Replace documents in collection by import (disabled when unknown collection)  
☒ Append documents in collection with import

Ignore duplicate keys

☒

**GLISSER ou CHARGER votre fichier JSON**  
Drop your file here or use the 'choose file' button

Choose File

No file chosen

The import process may take some time to complete, with a larger number of records or a memory size greater than 100 MB of the collection we recommend mongoexport and mongoimport through the console.

Cancel

Import

# Le DDL – L'insertion de documents

- Les types de données pouvant être insérées sont de type JavaScript.
- Exemple de type:
  - entier, double pour les nombres
  - chaîne de caractère (string)
  - Date (\$date: {'aaaa-mm-jj'})
  - Fichier binaire
  - ...

# Le DDL – La suppression de documents

- Pour réaliser la suppression d'un document dans une collection, il faut exécuter la méthode « remove() »
- Le comportement par défaut de cette méthode est la suppression de tous les documents répondant au critère de recherche.
- Exemple:
  - nous souhaitons supprimer le document contenant un client dont le nom est FERARD.

```
db.clients.remove(  
    {NOM: 'FERARD'},  
    {justOne: true}  
)
```

# Le DDL – La création d'index

- Dans le moteur NoSQL MongoDB, il est possible de créer plusieurs index différents au sein d'un document. Cela permet une amélioration des performances lors de lectures de données.
- Pour créer un index en MongoDB, il faut utiliser la méthode « `createIndex()` » de la collection.
  - Le nombre situé après le champ à indexer est l'ordre d'indexation
    - $> 0$  : ordre croissant
    - $< 0$  : ordre décroissant
- Exemple : `db.clients.createIndex({NOM: 1})`



# Le DDL – La suppression d'index

- La suppression d'un index se fait grâce à la méthode « `dropIndex()` » de la collection
- La structure de cette méthode est la même que celle permettant sa création

# Le DDL - Exercice

Réaliser la base de données permettant le stockage et le traitement d'un client et ses commandes de produits.

- Le nom de la base de données est libre.
- Cette base de données doit contenir deux collections:
  - clients
    - Un client est composé d'un NOM, d'une ADRESSE, d'une LOCALITE d'une CAT, d'un COMPTE et d'un ensemble de COMMANDES.
    - Une COMMANDES est généralement composée d'un NCOM, d'une DATECOM et d'un DETAIL.
    - Un DETAIL est constitué d'un PRODUIT, et d'une QCOM.
  - produits
    - Un produit est défini par un NPROD, un LIBELLE, un PRIX, une DATELASTMODIF et une QSTOCK

# MongoDB – Le DML



# MongoDB – Le DML

- Plan
  - Introduction
  - La récupération de données
    - L'extraction simple
    - L'agrégation
  - Exercice récapitulatif

# Le DML - Introduction

- Le DML (Data Manipulation Language) est un langage permettant de manipuler les données contenues dans une base de données.
- Le DML permet la :
  - récupération des données
  - modification des données

# Le DML – La récupération des données

- Dans MongoDB, il existe trois méthodes de récupération : l'extraction simple, l'agrégation et la « distinct ».
- L'extraction simple permet d'effectuer des requêtes simples de sélection et de projection des données.
  - La projection des données est une sélection des données à renvoyer.
- L'agrégation permet d'effectuer des actions sur les données avant de les renvoyer.
- La méthode « distinct » permet d'éviter les renvois de doublons.

# L'extraction simple

Introduction

Les opérateurs de recherches

Les opérateurs conditionnels

Les opérateurs d'éléments

Les opérateurs d'évaluation

Les opérateurs sur tableau

Les opérateurs de projection

# L'extraction simple – Introduction

- L'extraction simple est une méthode de la collection permettant de renvoyer un ou plusieurs document(s) selon un ou plusieurs critères et d'affichant que les attributs nécessaires.
- Structure générale d'une extraction:  
`db.<collection>.find({<critères>},{<projections>})`
- Exemple d'extraction sans critère et sans projection:
  - requête: Afficher tous les clients.
  - traduction DML: `db.clients.find({}, {})`



# L'extraction simple – Introduction

- Résultat:

```
{ "_id": "B332", "NOM": "MONTI", "ADRESSE": "23, a. Dumont", "LOCALITE": "Geneve", "CAT": "B2", "COMPTE": 0 }
{ "_id": "B112", "NOM": "HANSENNEA", "ADRESSE": "23, a. Dumont", "LOCALITE": "Poitiers", "CAT": "C1", "COMPTE": 1250 }
{ "_id": "F010", "NOM": "TOUSSAINT", "ADRESSE": "23, a. Dumont", "LOCALITE": "Poitiers", "CAT": "C1", "COMPTE": 0 }
{ "_id": "S127", "NOM": "VANDERKA", "ADRESSE": "23, a. Dumont", "LOCALITE": "Namur", "CAT": "C1", "COMPTE": -4580, "COMMANDES": [{"NCOM": 30182, "DETAIL": [{"produit_id": "PA60", "QCOM": 20}]}]}
{ "_id": "B062", "NOM": "GOFFIN", "ADRESSE": "23, a. Dumont", "LOCALITE": "Namur", "CAT": "B2", "COMPTE": -3200 }
{ "_id": "B512", "NOM": "GILLET", "ADRESSE": "23, a. Dumont", "LOCALITE": "Toulouse", "CAT": "B1", "COMPTE": -8700, "COMMANDES": [{"NCOM": 30188, "DETAIL": [{"produit_id": "PA60", "QCOM": 70}, {"produit_id": "CS464", "QCOM": 25}]}]}
{ "_id": "K111", "NOM": "VANBIST", "ADRESSE": "23, a. Dumont", "LOCALITE": "Lille", "CAT": "B1", "COMPTE": 720, "COMMANDES": [{"NCOM": 30178, "DETAIL": [{"produit_id": "CS464", "QCOM": 25}]}]}
{ "_id": "C123", "NOM": "MERCIER", "ADRESSE": "23, a. Dumont", "LOCALITE": "Namur", "CAT": "C1", "COMPTE": -2300 }
{ "_id": "C003", "NOM": "AVRON", "ADRESSE": "23, a. Dumont", "LOCALITE": "Toulouse", "CAT": "B1", "COMPTE": -1750 }
{ "_id": "K729", "NOM": "NEUMAN", "ADRESSE": "23, a. Dumont", "LOCALITE": "Toulouse", "COMPTE": 0 }
{ "_id": "C400", "NOM": "FERARD", "ADRESSE": "23, a. Dumont", "LOCALITE": "Poitiers", "CAT": "B2", "COMPTE": 350, "COMMANDES": [{"NCOM": 30179, "DETAIL": [{"produit_id": "CS262", "QCOM": 60}, {"produit_id": "PS222", "QCOM": 600}]}]}
{ "_id": "F011", "NOM": "PONCELET", "ADRESSE": "23, a. Dumont", "LOCALITE": "Toulouse", "CAT": "B2", "COMPTE": 0, "COMMANDES": [{"NCOM": 30185, "DETAIL": [{"produit_id": "PS222", "QCOM": 600}]}]}
{ "_id": "L422", "NOM": "FRANCK", "ADRESSE": "23, a. Dumont", "LOCALITE": "Namur", "CAT": "C1", "COMPTE": 0 }
{ "_id": "D063", "NOM": "MERCIER", "ADRESSE": "23, a. Dumont", "LOCALITE": "Toulouse", "COMPTE": -2250 }
{ "_id": "F400", "NOM": "JACOB", "ADRESSE": "23, a. Dumont", "LOCALITE": "Bruxelles", "CAT": "C2", "COMPTE": 0 }
{ "_id": "S712", "NOM": "GUILLAUME", "ADRESSE": "23, a. Dumont", "LOCALITE": "Paris", "CAT": "B1", "COMPTE": 0 }
{ "_id": "K112", "NOM": "EXO", "ADRESSE": "4/105, r. Charles Dubois", "LOCALITE": "Namur", "COMPTE": 125 }
{ "_id": "F401", "NOM": "1234", "ADRESSE": "23, a. Dumont", "LOCALITE": "Bruxelles", "CAT": "C2", "COMPTE": 2523 }
{ "_id": "B632", "NOM": "AROUNDI", "ADRESSE": "23, a. Dumont", "LOCALITE": "Geneve", "CAT": "B2", "COMPTE": 0.95 }
```

# L'extraction simple – Introduction

- Exemple d'extraction simple avec critère de sélection
  - Requête: Afficher tous les clients qui n'ont pas de catégorie.
  - Traduction DML: `db.clients.find({CAT: {$exists: false}}, {})`
  - Résultat:

```
{ "_id": "K729", "NOM": "NEUMAN", "ADRESSE": "23, a. Dumont", "LOCALITE": "Toulouse", "COMPTE": 0 }  
{ "_id": "D063", "NOM": "MERCIER", "ADRESSE": "23, a. Dumont", "LOCALITE": "Toulouse", "COMPTE": -2250 }  
{ "_id": "K112", "NOM": "EXO", "ADRESSE": "4/105, r. Charles Dubois", "LOCALITE": "Namur", "COMPTE": 125 }
```

# L'extraction simple – Introduction


- Exemple de requêtes avec projection.
  - Afficher le NOM de tous les clients.
  - Traduction DML: `db.find({}, {NOM: true})`
  - Résultat:

```
{ "_id": "B332", "NOM": "MONTI" }  
{ "_id": "B112", "NOM": "HANSENNEA" }  
{ "_id": "F010", "NOM": "TOUSSAINT" }  
{ "_id": "S127", "NOM": "VANDERKA" }  
{ "_id": "B062", "NOM": "GOFFIN" }  
{ "_id": "B512", "NOM": "GILLET" }  
{ "_id": "K111", "NOM": "VANBIST" }  
{ "_id": "C123", "NOM": "MERCIER" }  
{ "_id": "C003", "NOM": "AVRON" }  
{ "_id": "K729", "NOM": "NEUMAN" }  
{ "_id": "C400", "NOM": "FERARD" }  
{ "_id": "F011", "NOM": "PONCELET" }  
{ "_id": "L422", "NOM": "FRANCK" }  
{ "_id": "D063", "NOM": "MERCIER" }  
{ "_id": "F400", "NOM": "JACOB" }  
{ "_id": "S712", "NOM": "GUILLAUME" }  
{ "_id": "K112", "NOM": "EXO" }  
{ "_id": "F401", "NOM": 1234 }  
{ "_id": "B632", "NOM": "AROUNDI" }
```

# L'extraction simple – Introduction

- Exercices

1. Afficher le NOM et l'ADRESSE des clients habitant à Toulouse.
2. Afficher le NOM, le COMPTE et les COMMANDES des clients habitant à Poitiers.
3. Afficher tous les champs sauf le champ CAT pour tous les clients
4. Afficher le DETAIL des COMMANDES.



# Extraction simple: Les opérateurs de recherche

Les opérateurs \$eq,  
\$gt, \$gte, \$lt, \$lte

L'opérateur \$in

Exercices

# Extraction – Les opérateurs de recherche

L'opérateur  $\$eq$ ,  $\$gt(e)$ ,  $\$lt(e)$ ,

- Ces opérateurs sont utilisés à des fins de comparaison.
- $\$eq$  signifie égale/équivalente.
- $\$gt$  et  $\$gte$  signifient « plus grand » et « plus grand ou égal »
- $\$lt$  et  $\$lte$  signifient « plus petit » et « plus petit ou égal »
- La structure générale de cet opérateur est:  
    {<opérateur>: <valeur>}

# Extraction – Les opérateurs de recherche

- Exemple de requête `$eq`:
  - requête: Afficher les clients dont le nom est exactement « FERARD »
  - traduction DML: `db.clients.find({NOM: {$eq: 'FERARD'}}, {})`
  - résultat: `{ "_id": "C400", "NOM": "FERARD", "ADRESSE": "23, a. Dumont", "LOCALITE": "Poitiers", "CAT": "B2", "COMPTE": 350 }`
- Exemple de requête `$gt(e)`, `$lt(e)`:
  - requête: Afficher le NOM des clients dont le COMPTE est strictement positif.
  - traduction DML: `db.clients.find({COMPTE: {$gt: 0}}, {NOM: true})`
  - résultat:

```
{ "_id": "B112", "NOM": "HANSENNEA" }
{ "_id": "K111", "NOM": "VANBIST" }
{ "_id": "C400", "NOM": "FERARD" }
{ "_id": "K112", "NOM": "EXO" }
{ "_id": "F401", "NOM": "1234" }
{ "_id": "B632", "NOM": "AROUNDI" }
```

# Extraction – Les opérateurs de recherche

## L'opérateur \$in


- Cet opérateur est utilisé dans la comparaison
- Il signifie que le champ doit être égal au moins à un élément du tableau passé en valeur.
- Structure générale de cet opérateur:  
`{ $in: [<expression1>, <expression2>, ...<expressionN> ] }`
- Exemple de requête:
  - requête: Afficher les clients qui habitent à Bruxelles ou à Poitiers.
  - traduction DML: `db.clients.find({LOCALITE: { $in: [ 'Bruxelles', 'Poitiers' ] } }, { })`



# Extraction – Les opérateurs de recherche

- Exercices

1. Afficher le NOM des clients dont le COMPTE est strictement négatif.
2. Afficher le NOM et la LOCALITE des clients dont le COMPTE est vide
3. Afficher la LOCALITE des clients dont celle-ci est plus grande que Bruxelles.
4. Afficher le NOM, la LOCALITE des clients dont la LOCALITE n'est pas 'Bruxelles' ou 'Poitiers'.



# Extraction simple: Les opérateurs conditionnels

Les opérateurs \$and,  
\$or

L'opérateur \$not

Exercices

# Extraction – Les opérateurs conditionnels

## L'opérateur \$and et \$or

- Ces opérateurs permettent d'ajouter des opérateurs logiques à la recherche.
- Structure générale de ces opérateurs:  
    {<opérateur>: [<critère1>, <critère2>, ...]}
- Exemple de requêtes:
  - requête: Afficher les produits dont le prix est 185 ou si le NCOM est CS424.
  - traduction DML: `db.produits.find({$or: [{NCOM: 'CS424'}, {PRIX: 185}]}, {})`

# Extraction – Les opérateurs conditionnels

## L'opérateur \$not

- Cet opérateur permet la négation des opérateurs logiques.
- Structure de cet opérateur:  
`{ $not: { <opérateur>: <expression> } }`
- Exemple de requêtes:
  - requête: Afficher le NOM des clients dont le COMPTE n'est pas strictement positif.
  - traduction: `db.clients.find({COMPTE: { $not: { $gt: 0 } } }, {})`

# Extraction – Les opérateurs conditionnels

- Exercices

1. Afficher le NOM des clients qui sont dans la CAT 'B1' et dont le compte est strictement positif.
2. Afficher le NOM des clients qui ont un COMPTE strictement négatif habitant à 'Namur'.
3. Afficher le NOM des clients dont le compte est positif ou négatif et qui habitent à 'Namur' ou à 'Lille'.



# Extraction simple: Les opérateurs d'élément

L'opérateur \$exists

L'opérateur \$type

Exercices

# Extraction – Les opérateurs d'éléments

## L'opérateur \$exists

- Cet opérateur est utilisé pour la recherche de champs présents dans le document ou non.
- Structure de cet opérateur:  
{\$exists: <true/false>}
- Exemple de requête:
  - requête: Afficher le NOM des clients ayant passé au moins une COMMANDES.
  - traduction DML: `db.clients.find({COMMANDES: {$exists: true}}, {NOM: true})`
  - résultat:

# Extraction – Les opérateurs d'éléments

## L'opérateur \$ type

- Cet opérateur permet de spécifier le type de l'élément recherché.
- Structure de cet opérateur:  
{\$type: '<type>'}
- Exemple de requête:
  - requête: Afficher le NOM du client dont le NOM est de type 'double'.
  - traduction DML: db.clients.fnd({NOM: {\$type: 'double'}},{NOM: true})
  - résultat:


```
{"_id":"F401","NOM":1234}
```



# Extraction – Les opérateurs d'éléments

- Exercices

1. Afficher tous les clients n'ayant pas passé de COMMANDES.
2. Afficher tous les clients n'ayant pas de COMMANDES, de CAT et dont le NOM est de type 'double'.
3. Afficher le NOM, l'ADRESSE et le COMPTE des clients n'ayant pas de CAT.



# Extraction simple: Les opérateurs d'évaluation

L'opérateur \$mod

L'opérateur \$regex

L'opérateur \$text

L'opérateur \$where

Exercices

# Extraction – Les opérateurs d'évaluation

## L'opérateur \$mod

- Cet opérateur permet d'effectuer le modulo (le reste d'une division euclidienne).
- Structure générale de cet opérateur:  
{\$mod: [<diviseur>, <reste>]}
- Exemple de requête:
  - requête: Afficher le NOM des clients dont le COMPTE est pair.
  - traduction DML: `db.clients.find({COMPTE: {$mod: [2,0]}},{NOM: true})`
  - résultat:

# Extraction – Les opérateurs d'évaluation

## L'opérateur \$regex

- Cet opérateur est extrêmement puissant, mais compliqué à mettre en place. Nous ne verrons ici que l'équivalent du « like » en SQL.
- Cet opérateur est utilisé afin de vérifier la correspondance entre la requête et le contenu de la valeur d'un champ.
- Structure de l'opérateur  
{\$regex: '.\*'+ <texte à trouver>+ '.\*', \$options: '<options>'}
- Pour plus d'informations sur le regex:  
[http://regexone.com/lesson/introduction\\_abcs](http://regexone.com/lesson/introduction_abcs)

# Extraction – Les opérateurs d'évaluation

- Exemple de requête avec l'opérateur \$regex.
  - Requête: Afficher le NOM des clients dont le NOM contient la lettre F.
  - Traduction DML:  
db.clients.find({NOM: {\$regex: '.\*'+ 'F'+ '.\*', \$options: 'i'}},{NOM: true})
  - Résultat:

```
{"_id":"B062","NOM":"GOFFIN"}
```

```
{"_id":"C400","NOM":"FERARD"}
```

```
{"_id":"L422","NOM":"FRANCK"}
```

# Extraction – Les opérateurs d'évaluation

## L'opérateur \$text

- Cet opérateur ne fonctionne que sur des champs indexés.
- Il permet de trouver des chaînes de caractères incluses dans une valeur. (Équivalent du « like » SQL mais nécessite d'avoir le champ indexé).
- Structure de l'opérateur:  
{\$text: <chaîne de caractère>}
- Exemple de requête:
  - requête: Afficher le NOM des clients dont le NOM contient 'F'.
  - traduction DML: `db.clients.find({NOM: {$text: 'F'}}, {NOM: true})`

# Extraction – Les opérateurs d'évaluation

## L'opérateur \$where


- Cet opérateur est utilisé pour évaluer une expression Javascript.
- Structure de cet opérateur  
`{ $where: '<expression Javascript>' }`
- Exemple de requête:
  - requête: Afficher le NOM des clients ayant effectué plusieurs COMMANDES.
  - traduction DML:  
`db.clients.find({COMMANDES: { $exists: true }, $where: 'this.COMMANDES.length > 1' }, {NOM: true})`
  - résultat:

# Extraction – Les opérateurs d'évaluation

- Exercices

1. Afficher le NOM des clients ayant effectué exactement une COMMANDES.
2. Afficher le NOM des clients dont le COMPTE est pair et dont le NOM contient « er » dans un champ non indexé.
3. Afficher le NOM, la LOCALITE des clients n'ayant jamais commandé dont le COMPTE est positif et que le NOM n'est pas une chaine de caractère.





# Extraction simple: Les opérateurs sur tableau

L'opérateur \$all

L'opérateur  
\$elemMatch

L'opérateur \$size

Exercices

# Extraction – Les opérateurs sur tableau

## L'opérateur \$all

- Cet opérateur est utilisé pour sélectionner les documents contenant tous les éléments demandés.
- Structure de cet opérateur:  
`{ $all: [<valeur1>, <valeur2>, ...] }`
- Exemple de requêtes:
  - requête: Afficher les expériences dont le champ « A » comporte tous les éléments « red » et « blue »
  - traduction DML: `db.experiments.find({A: { $all: ['red', 'blue'] } }, {})`

# Extraction – Les opérateurs sur tableau

## L'opérateur \$elemMatch

- Cet opérateur est utilisé pour sélectionner les documents contenant un ou plusieurs éléments demandés
- Structure de cet opérateur:  
`{ $elemMatch: { <expression1>, <expression2>, ... } }`
- Exemple de requêtes:
  - requête: Afficher les expériences dont le champ « A » comporte tous les éléments « red » et « blue »
  - traduction DML: `db.experiments.find({A: { $elemMatch: [ { $gt: 'blue' }, { $eq: 'red' } ] } }, { })`

# Extraction – Les opérateurs sur tableau

## L'opérateur \$size

- Cet opérateur est utilisé pour sélectionner les documents selon la taille du tableau d'un champ
- Structure de cet opérateur:  
{\$size: <nombre>}
- Exemple de requêtes:
  - requête: Afficher les expériences dont le champ « A » comporte deux éléments
  - traduction DML: `db.experiments.find({A: {$size: 2}},{})`

# Extraction – Les opérateurs sur tableau

- Exercices

1. Afficher l'id des expériences dont le champ « A » contient deux éléments et dont le contenu de « B » est « red » et « blue ».
2. Afficher le champ « A » des expériences dont le champ « B » ne contient pas de composant.
3. Afficher les clients ayant passé une COMMANDES



# Extraction simple: Les opérateurs de projection

L'opérateur \$

L'opérateur  
\$elemMatch

L'opérateur \$slice

# Extraction – Les opérateurs de projection

## L'opérateur \$ (projection)

- Cet opérateur est utilisé pour limiter les éléments retournés d'un tableau à afficher au premier élément.
- Structure:  
`{<tableau>.$: true}`
- Exemple de requête:
  - requête: Afficher la première COMMANDES effectuée par les clients.
  - traduction DML: `db.clients.find({COMMANDES: {$exists: true}},{"COMMANDES.$": true})`

# Extraction – Les opérateurs de projection

## L'opérateur \$elemMatch (projection)

- Cet opérateur permet l'affichage conditionné.
- Structure de cet opérateur:  
{\$elemMatch: {<expression>}}
- Exemple de requête:
  - requête: Afficher tous les \_id des clients, mais afficher en plus la commande dont le NCOM est 30182.
  - traduction DML: db.clients.find({}, {COMMANDES: {\$elemMatch: {NCOM: 30182}}})



# Extraction – Les opérateurs de projection

## L'opérateur \$slice

- Cet opérateur permet l'affichage d'un ensemble d'éléments d'un tableau.
- Structure de cet opérateur:  
{\$slice: [<index\_start>, <nombre\_element>]}  
L'index de départ est optionnel.
- Exemple de requête:
  - requête: Afficher le deuxième élément du tableau de COMMANDES des clients.
  - traduction DML: `db.client.find({COMMANDES: {$exists: true}}, {COMMANDES: {$slice: [1,1]}})`

# Extraction – Les opérateurs de projection

- Exercice
  1. Afficher le NCOM de la première COMMANDES effectuée par les clients.
  2. Afficher le premier PRODUIT du DETAIL d'une commandes des clients.
  3. Afficher la QCOM de tous les produits de la deuxième COMMANDES des clients.

# Extraction – Exercices globales

1. Afficher le NOM, le COMPTE et l'ADRESSE du client dont le NOM possède un « n » et dont le COMPTE n'est pas égale null/0 de toutes les manières possibles.

# L'agrégation

Introduction

Les opérateurs d'étapes et logiques

Les opérateurs d'ensemble

Les opérateurs d'arithmétiques

Les opérateurs sur les strings

Les opérateurs sur les dates

Les opérateurs conditionnels

Les opérateurs d'accumulation

# L'agrégation – Introduction

- L'agrégation est une méthode de la collection permettant de renvoyer un ou plusieurs document(s) selon un ou plusieurs critère(s), de n'afficher que les attributs nécessaires et d'effectuer des opérations agrégatives sur les données (arithmétique, groupement, ...).
- Structure générale d'une extraction:
  - `db.<collection>.aggregate([<expression1>,<expression2>], ...)`
- Exemple d'extraction sans critère et sans projection:
  - requête: Affichage de tous les clients.
  - traduction DML: `db.clients.aggregate([])`

# L'agrégation – Introduction

- L'agrégation fonctionne par étapes. Ainsi, l'exécution d'une étape s'appuiera sur les résultats obtenus antérieurement.
- Exemple d'agrégation simple:
  - Afficher le NOM des clients dont le NOM est 'MONTI'.  

```
db.clients.aggregate([{$match: {NOM: 'MONTI'}}, {$project: {NOM: true, _id: false}}])
```
  - Afficher la moyenne des comptes des clients ayant passé COMMANDES  

```
db.clients.aggregate([{$match: {COMMANDES: {$exists: true}}, {$group: {_id: null, AVG: {$avg: '$COMPTE'}}}])
```

# L'agrégation: Les opérateurs d'étapes

L'opérateur \$project

L'opérateur \$group

L'opérateur \$match

Les opérateurs \$sort  
et \$out

L'opérateur \$redact

Les opérateurs \$limit  
et \$skip

L'opérateur \$unwind

# L'agrégation – Les opérateurs d'étapes

## L'opérateur \$project

- Cet opérateur permet la projection des données.
- Structure de l'opérateur
  - {\$project: {<champ>: <true/false/expression>}}
- Exemple de requête
  - requête: Afficher le NOM des clients possédant un compte négatif.
  - traduction: db.clients.aggregate([{\$match: {COMPTE: {\$lt: 0}}}, {\$project: {NOM: true}}])
  - résultat:

```
{ "_id": "S127", "NOM": "VANDERKA" }  
{ "_id": "B062", "NOM": "GOFFIN" }  
{ "_id": "B512", "NOM": "GILLET" }  
{ "_id": "C123", "NOM": "MERCIER" }  
{ "_id": "C003", "NOM": "AVRON" }  
{ "_id": "D063", "NOM": "MERCIER" }
```



# L'agrégation – Les opérateurs d'étapes

## L'opérateur \$match

- Cet opérateur permet la création de critères de recherche.
- Structure de l'opérateur
  - {\$match: {<champ>: <valeur>}}
- Exemple de requête:
  - requête: Afficher tous les clients dont le NOM est 'MONTI'
  - traduction: `db.clients.aggregate([{$match: {NOM: 'MONTI'}}])`

# L'agrégation – Les opérateurs d'étapes

## L'opérateur \$redact

- Cet opérateur permet l'affichage de champs du document en fonction d'autres champs de ce même document.
- Structure de l'opérateur
  - {\$redac: <expression>}
    - Cette expression peut être une condition renvoyant \$\$DESCEND, \$\$PRUNE, \$\$KEEP
  - \$\$DESCEND permet d'afficher les champs
  - \$\$PRUNE permet de ne pas afficher les champs
  - \$\$KEEP permet de renvoyer ou de garder tous les champs.

# L'agrégation – Les opérateurs d'étapes

- Exemple de requêtes:

- requêtes: Afficher les comptes positifs et n'affiche pas le DETAIL des COMMANDES éventuelles.
- traduction DML:

```
db.clients.aggregate([{$redact: {$cond: [{$lte: ["$COMPTE", 0]}, "$$PRUNE", "$$DESCEND"]}}])
```

- résultat:

```
{ "_id": "B112", "NOM": "HANSENNEA", "ADRESSE": "23, a. Dumont", "LOCALITE": "Poitiers", "CAT": "C1", "COMPTE": 1250 }
{ "_id": "K111", "NOM": "VANBIST", "ADRESSE": "23, a. Dumont", "LOCALITE": "Lille", "CAT": "B1", "COMPTE": 720, "COMMANDES": [] }
{ "_id": "C400", "NOM": "FERARD", "ADRESSE": "23, a. Dumont", "LOCALITE": "Poitiers", "CAT": "B2", "COMPTE": 350, "COMMANDES": [] }
{ "_id": "K112", "NOM": "EXO", "ADRESSE": "4/105, r. Charles Dubois", "LOCALITE": "Namur", "COMPTE": 125 }
{ "_id": "F401", "NOM": 1234, "ADRESSE": "23, a. Dumont", "LOCALITE": "Bruxelles", "CAT": "C2", "COMPTE": 2523 }
{ "_id": "B632", "NOM": "AROUNDI", "ADRESSE": "23, a. Dumont", "LOCALITE": "Geneve", "CAT": "B2", "COMPTE": 0.95 }
```

# L'agrégation – Les opérateurs d'étapes

## Les opérateurs \$limit et \$skip

- Ces opérateurs permettent de renvoyer un ensemble de documents
- \$limit permet de limiter le nombre de documents
- \$skip permet de passer un nombre de documents
- Structure de ces opérateurs
  - {<opérateur>: <entier positif>}
- Exemple de requête:
  - requête: Afficher les 3 premiers documents ayant effectué une COMMANDES.
  - traduction DML: `db.clients.aggregate([{$match:{COMMANDES: {$exists: true}}}, {$limit: 3}])`

# L'agrégation – Les opérateurs d'étapes

## L'opérateur \$unwind

- Cet opérateur permet de répartir les éléments d'un tableau dans des documents différents.
- Structure de cet opérateur:
  - {\$unwind: <expression>}
- Exemple de requêtes:
  - requête: Afficher les COMMANDES et le NOM des clients. Un document par commandes.
  - traduction DML:

```
db.clients.aggregate([{$unwind: '$COMMANDES'}, {$project: {NOM: true, "COMMANDES.NCOM": true, "COMMANDES.DETAIL.produit_id": true}}])
```

# L'agrégation – Les opérateurs d'étapes

- Résultat:

```
{ "_id": "S127", "NOM": "VANDERKA", "COMMANDES": { "NCOM": 30182, "DETAIL": [ { "produit_id": "PA60" } ] } }  
{ "_id": "B512", "NOM": "GILLET", "COMMANDES": { "NCOM": 30188, "DETAIL": [ { "produit_id": "PA60" }, { "produit_id": "PH222" }, { "produit_id": "CS464" }, { "produit_id": "PA45" } ] } }  
{ "_id": "K111", "NOM": "VANBIST", "COMMANDES": { "NCOM": 30178, "DETAIL": [ { "produit_id": "CS464" } ] } }  
{ "_id": "C400", "NOM": "FERARD", "COMMANDES": { "NCOM": 30179, "DETAIL": [ { "produit_id": "CS262" }, { "produit_id": "PA60" } ] } }  
{ "_id": "C400", "NOM": "FERARD", "COMMANDES": { "NCOM": 30184, "DETAIL": [ { "produit_id": "CS464" }, { "produit_id": "PA45" } ] } }  
{ "_id": "C400", "NOM": "FERARD", "COMMANDES": { "NCOM": 30186, "DETAIL": [ { "produit_id": "PA45" }, { "produit_id": "PA60" } ] } }  
{ "_id": "F011", "NOM": "PONCELET", "COMMANDES": { "NCOM": 30185, "DETAIL": [ { "produit_id": "PS222" }, { "produit_id": "CS464" }, { "produit_id": "PA60" } ] } }
```

# L'agrégation – Les opérateurs d'étapes

## L'opérateur \$group

- Cet opérateur permet de grouper selon un ou plusieurs champs les documents. Ainsi, nous pouvons effectuer des opérations arithmétiques sur ces documents pour les compter, etc...
- La structure de cet opérateur:
  - {\$group: {\_id: {<champ>}, <créer champ>: <valeur>, ...}}
  - On peut réaliser un groupement sur plusieurs champs communs.
- Exemple de requête:
  - afficher le nombre de clients habitant chaque LOCALITE
  - traduction DML: `db.clients.aggregate([{$group: {_id: '$LOCALITE', nbClient: {$sum: 1}}])`

# L'agrégation – Les opérateurs d'étapes

## Les opérateurs \$sort, \$out

- Ces opérateurs permettent de trier, et d'injecter les résultats de l'agrégation dans une collection en base de données.
- \$sort sert aux tris alors que \$out sert d'export.
- Structure de ces requêtes:
  - {\$sort: {<champ>: <1/-1>}}
  - {\$out: '<nom\_collection>'}
- Exemple de requête:
  - Trier les clients par NOM ascendant, et enregistrer le résultat dans une collection.
  - `db.clients.aggregate([{$sort: {NOM: 1}}, {$out: 'clients_nom'}])`



# L'agrégation – Les opérateurs d'étapes

## L'opérateur \$lookup

- Lookup permet d'effectuer des jointures sur des champs dans une base de données non répartie.
- {\$lookup: {from: <collection>, localField: <champ>, foreignField: <champ>, as: <string>}}

# L'agrégation – Les opérateurs logiques

- Il s'agit des mêmes opérateurs et du même fonctionnement que les opérateurs logiques dans les critères de recherche simples.
- Il y a donc \$and, \$or et \$not

# L'agrégation – Les opérateurs d'étapes et logiques

- Exercices

1. Afficher le nombre de clients dont la LOCALITE est 'Namur'.
2. Afficher le nombre de clients par LOCALITE ayant effectué une et une seule COMMANDES
3. Afficher le nombre de clients par LOCALITE ayant un compte positif ou ayant effectué une COMMANDES
4. Afficher dans des documents séparés, le DETAIL des COMMANDES ainsi que le NOM des clients, et le NCOM.

# L'agrégation – Les opérateurs d'étapes et logiques

5. Afficher le NPROD, sont LIBELLE et le(s) CLIENTS l'ayant acheté

# L'agrégation: Les opérateurs d'ensemble

Les opérateurs `$setEquals` et `$setIsSubset`

Les opérateurs `$setIntersection`, `$setUnion`  
et `$setDifference`

Les opérateurs `$anyElementTrue` et  
`$allElementTrue`

Exercices

# L'agrégation – Les opérateurs d'ensembles

## Les opérateurs \$setEquals, \$setIsSubset

- Ces opérateurs permettent de déterminer si deux ensembles d'un même document ont exactement les mêmes composants ou si l'ensemble est un sous-ensemble du second.
- Structure de cet opérateur:
  - {<opérateur>: [<expression1>, <expression2>]}
- Exemple de requête
  - requête: Les composants des deux réactifs de l'expérience sont-ils les mêmes ?
  - traduction DML: `db.experiment.aggregate([{$project: {SAME: {$setEquals: ["$A", "$B"]}}])`

# L'agrégation – Les opérateurs d'ensembles

Les opérateurs `$setInteraction`, `$setUnion`,  
`$setDifference`.

- Ces opérateurs permettent de récupérer les différences, les ressemblances et la totalité des composants uniques aux deux ensembles.
- Structure de ces opérateurs
  - {<opérateur>: [<ensemble1>, <ensemble2>, ...]}
- Exemple de requête:
  - afficher les composants communs à l'expérience.
  - traduction: `db.experiment.aggregate([{$project: {<nom>: {$setInteraction: ['$A', '$B']}}})`

# L'agrégation – Les opérateurs d'ensembles

Les opérateurs \$anyElementTrue, \$allElementTrue.

- Ces opérateurs permettent de connaître l'état booléen d'un tableau.
- \$anyElementTrue renvoie true si chaque élément est true ou false.
- \$allElementTrue renvoie true si tous les éléments sont true.
- Structure de ces opérateurs:
  - {<opérateur>: [<ensemble>]}



# L'agrégation – Les opérateurs d'ensembles

- Exercices
  1. Afficher les éléments communs entre les composants A et B de chaque expérience.
  2. Afficher les éléments différents entre les composants de chaque expérience.

# L'agrégation – Les opérateurs de comparaison

- Encore une fois, ce sont les mêmes opérateurs que pour la comparaison des requêtes simples.
- Il y a donc  $\$gt(e)$ ,  $\$lt(e)$ ,  $\$ne$ ,  $\$eq$ .
- Un nouveau est  $\$cmp$ . Il permet d'effectuer une comparaison entre deux expressions (renvoie -1 si <, 0 si = et 1 sinon)
- Structure de ces opérateurs:
  - $\{<opérateur>: [<expression1>, <expression2>]\}$
- Exemple de requête:
  - requête: Afficher les NOM des clients et si leur COMPTE est positif.
  - traduction DML:
    - `db.clients.aggregate([{$project: {NOM: true, POSITIF: {$cond: [{$gt: ["$COMPTE", 0]}, "OUI", "NON"]}}}]])`

# L'agrégation: Les opérateurs d'arithmétique

Les opérateurs  
\$add, \$subtract et  
\$multiply

Les opérateurs  
\$divide et \$mod

Exercices

# L'agrégation – Les opérateurs d'arithmétique

## Les opérateurs \$add, \$subtract et \$multiply

- Ces opérateurs permettent d'effectuer une addition, une soustraction et une multiplication entre des champs et/ou des valeurs lors de la projection.
- Structure de ces opérateurs
  - {<opérateur>: [<champ1>, <champ2>]}
- Exemple de requête:
  - afficher la valeur des marchandises en stock.
  - traduction DML: `db.produits.aggregate({$project: {_id: true, Total: {$multiply: ['$PRIX', '$QSTOCK']}}})`

# L'agrégation – Les opérateurs d'arithmétique

## Les opérateurs \$divide et \$mod

- Ces opérateurs permettent d'effectuer des divisions et des modulus (reste d'une division euclidienne) entre deux champs et/ou valeurs dans une projection.
- Structure de ces opérateurs
  - {<opérateur>: [<dividende>, <diviseur>]}

# L'agrégation – Les opérateurs d'arithmétique

## Les opérateurs \$abs, \$exp, \$ln et \$log10

- Ces opérateurs permettent d'effectuer une valeur absolue, une exponentielle, un logarithme népérien et un logarithme en base 10.
- Structure de ces opérateurs
  - {<opérateur>: {<expression>}ou<nombre>}
- Exemple de requête:
  - requête: quelle est la valeur absolue de chacun des comptes.
  - traduction: `db.clients.aggregate([{$project: {COMPTE: {$abs: '$COMPTE'}}}])`

# L'agrégation – Les opérateurs arithmétiques

- Exercice

1. Afficher 1 si le montant du COMPTES est impair et 0 sinon, pour tous les clients ayant passé COMMANDES
2. Afficher le montant du COMPTES des clients ayant effectué une ou plusieurs COMMANDES divisé par le nombre clients.  
(Indice: Utiliser un curseur pour récupérer le nombre de clients)
3. Afficher le montant total de la valeur des produits en stock.

# L'agrégation: Les opérateurs sur les strings

L'opérateur  
\$concat

\$toLower et  
\$toUpper

L'opérateur  
\$substr

L'opérateur  
\$strcasecmp

L'opérateur

Exercices



# L'agrégation – Les opérateurs sur les strings

## L'opérateur \$concat

- Cet opérateur permet de réaliser une concaténation entre plusieurs champs.
- Structure de cet opérateur
  - {\$concat: [<champ1>, <champ2>]}
- Exemple de requêtes
  - Afficher l'ADRESSE et la LOCALITE d'un client dans un seul champ.
  - Traduction DML: `db.clients.aggregate([{$project: {ADRESSE: {$concat: ['$ADRESS', '$LOCALITE']}}})`

# L'agrégation – Les opérateurs sur les strings

## L'opérateur \$substr

- Cet opérateur permet de sélectionner des sous-éléments de la chaîne de caractère et de réaliser un « cast » entre les entiers et les chaînes de caractères.
- Structure de cet opérateur
  - {\$substr: [<string>, <entier\_depart>, <entier\_arriver/-1>]}
  - -1 permet la sélection de toute la chaîne jusqu'à sa fin.
- Exemple de requête
  - Afficher les trois premières lettres de la LOCALITE.
  - Traduction DML: `db.clients.aggregate([{$project: {LOCALITE: {$substr: ['$LOCALITE', 0, 3]}}})`

# L'agrégation – Les opérateurs sur les strings

## Les opérateurs \$toLower et \$toUpper

- Ces opérateurs permettent la mise en minuscule ou en majuscules d'une chaîne de caractère.
- Structure de ces opérateurs
  - {<opérateur>: <string>}
- Exemple de requêtes
  - Afficher le NOM des clients en minuscule
  - Traduction DML: `db.clients.aggregate([{$project: {NOM: {$toLower: '$NOM'}}}])`

# L'agrégation – Les opérateurs sur les strings

## L'opérateur \$strcasecmp

- Cet opérateur permet de comparer deux chaînes de caractères.
- Structure de cet opérateur
  - {\$strcasecmp : [<string1>, <string2>]}
  - Cet opérateur renvoie -1 si le premier est plus petit, 0 si les chaînes sont les mêmes et 1 si le premier est plus grand
- Exemple de requête:
  - comparer le champ NOM et LOCALITE d'un client et afficher si le NOM est avant dans l'ordre alphabétique.
  - traduction DML: `db.clients.aggregate([{$project: {ORDRE: {$strcasecmp : ['$NOM', '$LOCALITE']}}}]])`

# L'agrégation – Les opérateurs sur les strings

- Exercice

1. Afficher en minuscule et en majuscule le NOM des clients dans un champ unique appeler NOM\_MIN\_MAJ.
2. Afficher le montant du COMPTE en € de chaque client.
3. Afficher le PRODUIT , son QSTOCK, son PRIX et sa valeur en stock dans un seul champ.

# L'agrégation: Les opérateurs sur les dates

Les opérateurs  
\$year, \$month,  
\$week

\$hour, \$minute,  
\$second,  
\$milisecond

Les opérateurs  
\$dayOfYear,  
\$dayOfMonth et  
\$dayOfWeek

L'opérateur  
\$dateToString

Les opérateurs

# L'agrégation – Les opérateurs sur les dates

Les opérateurs \$year, \$month, \$week.

- Ces opérateurs permettent de traiter séparément l'année, le mois et la semaine d'une date.
- Structure de ces opérateurs
  - {<opérateur>: <date>}
- Exemple de requêtes
  - Afficher le numéro du mois et de la semaine de la date 21 Juillet 1922.
  - Traduction DML: `db.collection.aggregate([{$project: {MOIS: {$month: new Date(1922-07-21)}, WEEK: {$week: new Date(1922-07-21)}}})`

# L'agrégation – Les opérateurs sur les dates

Les opérateurs \$dayOfYear, \$dayOfMonth et \$dayOfWeek

- Ces opérateurs permettent de récupérer le numéro correspondant au jour de la date selon l'année, le mois ou la semaine.
- Structure de ces opérateurs
  - {<opérateur>: <date>}
- Exemple de requêtes
  - Afficher le numéro du jour du 29 février 2016 dans l'année, le mois et la semaine.



# L'agrégation – Les opérateurs sur les dates

Les opérateurs \$hour, \$minute, \$second, ...

- Ces opérateurs permettent la récupération des heures, minutes, secondes, ... à partir d'une date.
- Structure de ces opérateurs
  - {<opérateur>: <date>}

# L'agrégation – Les opérateurs sur les dates

## L'opérateur \$dateToString

- Cet opérateur permet de renvoyer une date en format chaîne de caractère.
- Structure de cet opérateur
  - {\$dateToString: <date>}
- Exemple de requête
  - Afficher la date des commandes en format texte.
  - Traduction DML:

# L'agrégation – Les opérateurs sur les dates

- Exercice

# L'agrégation:

## Les opérateurs conditionnels

L'opérateur \$cond

L'opérateur \$ifNull

Exercices

# L'agrégation – Les opérateurs conditionnels

## L'opérateur \$cond

- Cet opérateur permet de réaliser des conditions à toutes les étapes d'une agrégation. Ces conditions sont de la forme « si ... alors ... sinon ... ».
- Structure de ces opérateurs
  - {\$cond: [<condition booléenne si>, <alors>, <sinon>]}
- Exemple de requête
  - Afficher le NOM et le COMPTE des clients s'ils ont un COMPTE positif, sinon afficher COMPTE négatif dans le champ COMPTE.
  - Traduction DML: `db.clients.aggregate([{$project: {NOM: true, COMPTE: {$cond: [{$gte: ["$COMPTE", 0]}, '$COMPTE', 'COMPTE Négatif']}}}]])`

# L'agrégation – Les opérateurs conditionnels

## L'opérateur \$ifNull

- Cet opérateur permet de remplacer, lors de l'affichage, la valeur d'un champ évalué à NULL
- Structure de cet opérateur
  - {\$ifNull: [<sinon null>, <si null>]}
- Exemple de requête
  - Afficher les COMMANDES et le NOM des clients. S'ils n'ont pas effectué de COMMANDES, afficher « Pas de commandes »
  - Traduction DML: `db.clients.aggregate([{$project: {COMMANDES: {$ifNull: ['$COMMANDES', 'Pas de commandes']}, NOM: true}}])`

# L'agrégation – Les opérateurs conditionnels

- Exercice

1. Afficher le montant du COMPTE ou « compte à 0 » si le montant du COMPTE n'est pas égal à 0.
2. Afficher « pas de catégorie » pour les clients n'ayant pas de catégorie et qui n'ont jamais effectué de COMMANDES
3. Afficher si le COMPTE est positif ou négatif des clients n'ayant pas de catégorie.

# L'agrégation:

## Les opérateurs d'accumulation

Les opérateurs \$sum, \$avg, \$min et \$max  
Les opérateurs \$push et \$addToSet

Les opérateurs \$first et \$last



# L'agrégation – Les opérateurs d'accumulation

Les opérateurs \$sum, \$avg, \$min et \$max

- Ces opérateurs permettent d'effectuer des sommes, des moyennes, de retourner les valeurs minimales et maximales dans une étape de groupement.
- Structure de ces opérateurs
  - {<opérateur>: <expression>}

# L'agrégation – Les opérateurs d'accumulation

- Exemple d'utilisation de \$sum
  - Afficher le nombre de clients par LOCALITE
  - Traduction DML: `db.clients.aggregate([{$group: {_id: '$LOCALITE', SUM: {$sum: 1}}}]])`
  - Afficher le total des comptes des clients par LOCALITE
  - Traduction DML: `db.clients.aggregate([{$group: {_id: '$LOCALITE', TOT: {$sum: '$COMPTE'}}}]])`

# L'agrégation – Les opérateurs d'accumulation

## Les opérateurs \$first et \$last

- Ces opérateurs permettent d'afficher l'élément du premier ou dernier document d'un groupement.
- Structure de ces opérateurs.
  - {<opérateur>: <expression>}
- Exemple de requête
  - Afficher la dernière COMMANDES de chaque produit.
  - Traduction DML: `db.clients.aggregate([{$group: {_id: '$COMMANDES.DETAIL.produit_id', COMMANDES: {$last: '$DATE_COM'}}}])`

# L'agrégation – Les opérateurs d'accumulation

## L'opérateur \$push et \$addToSet

- Ces opérateurs retournent un tableau contenant tous les éléments ajoutés, ou tous les éléments ne se trouvant pas déjà dans le tableau.
- Structure de ces opérateurs
  - {<opérateur>: <expression>}
- Exemple de requête
  - Afficher tous les produits commandés selon les jours de l'année et l'année.
  - Traduction DML: `db.clients.aggregate([{$group: {_id: {JOUR: {$dayOfYear: "$COMMANDES.DATECOM"}, ANNEE: {$year: "$COMMANDES.DATECOM"}}, PRODUIT: {$addToSet: {'$COMMANDES.DETAIL.produit_id '}}}}])`

# L'agrégation – Les opérateurs d'accumulation

- Exercice

1. Afficher la moyenne des COMPTES des clients ayant passé au moins une COMMANDES par LOCALITE
2. Afficher le PRIX minimal et maximal des produits.
3. Afficher la liste des NOM des clients ayant passé COMMANDES

# L'agrégation – Exercices global

1. Réaliser une agrégation permettant d'afficher l' \_id, le PRIX, le LIBELLE, le NOM des clients, la quantité commandée par l'ensemble des clients et le gain total de tous les produits

# La mise à jour des données

# Le DML – La modification des données

- La modification est une méthode de la collection permettant de modifier un ou plusieurs champs de un ou plusieurs documents selon des critères.
- Structure générale d'une modification:
  - `db.<collection>.update({<critères>},{<champ>: <valeur>}, {<ACTION>})`
- Une modification simple remplace le document répondant aux critères par les nouveaux champs => perte de données. Il existe des opérateurs permettant la modification que des champs concernés.



# Le DML – La modification des données

## Les actions lors d'une modification

- {upsert: <true/false>}

Permet l'insertion de document lorsque aucun document ne correspond aux critères de recherche.

- {multi: <true/false>}

Permet d'effectuer la modification sur tous les documents répondants aux critères de recherche. Par défaut, cette action est à false et ne s'exécute pas.

# L'update: La modification des champs

L'opérateur  
\$set

L'opérateur  
\$unset

L'opérateur  
\$setOnInsert

L'opérateur  
\$currentDate

L'opérateur  
\$inc

Les  
opérateurs  
\$min et \$max

L'opérateur  
\$mul

Exercices

L'opérateur  
\$rename

# L'update – L'opérateur \$set

- Cet opérateur est responsable de la mise à jour d'un champ dans un document. Cette mise à jour est soit une création, soit une modification de la valeur d'un champ existant.
- Structure de cet opérateur:  
`{ $set: { <ensemble des champs à modifier> } }`
- Exemple de modification
  - Requête: Changer le NOM du client répondant au NOM de 'MONTI' par 'tUpdate'.
  - Traduction DML: `db.clients.update({NOM: 'MONTI'}, { $set: {NOM: 'tUpdate'}})`

# L'update – L'opérateur \$setOnInsert

- Cet opérateur a le même rôle que \$set, mais s'exécute lors d'une insertion.
- Structure de cet opérateur:  
`{ $setOnInsert: { <ensemble des champs à modifier> } }`
- Exemple de modification
  - Requête: Chercher le client dont le NOM est 'MONTI', s'il n'existe pas, créer un document.
  - Traduction DML: `db.clients.update({NOM: 'MONTI'}, { $setOnInsert: {NOM: 'MONTI'} }, {upsert: true})`

# L'update – L'opérateur \$inc

- Cet opérateur permet l'incrémentation d'une valeur numérique d'un champ.
- Structure de l'opérateur:  
`{ $inc: { <champ>: <nombre>, ... } }`
- Exemple de requête
  - Requête: Mettre à jour le stock de produits lors d'un achat.
  - Cette requête nécessite l'utilisation du code client.
  - Simulation: `db.produits.update({_id: 'CS464'}, { $inc: { QSTOCK: -10 } })`

# L'update – L'opérateur \$mul

- Cet opérateur permet de multiplier la valeur numérique d'un champ par un nombre.
- Structure de cet opérateur:
  - {\$mul: {<champ>: <nombre>}}
- Exemple de requête
  - Requête: Mettre à jour le prix de 25% avant les soldes.
  - Traduction: `db.produits.update({},{$mul: {PRIX: 1,25}}, {multi: true})`

# L'update – L'opérateur \$rename

- Cet opérateur permet la modification de l'intitulé d'un champ.
- Structure de cet opérateur:
  - {\$rename: {<champ1>: <nouveau\_nom1>, ...}}
- Exemple de requête
  - Requête: Modifier l'intitulé du NOM par NAME pour tous les clients.
  - Traduction DML: `db.clients.update({},{$rename: {NOM: 'NAME'}}, {$multi: true})`
  - Résultat:

avant =	
après =	<pre>{ "_id": "B112", "NOM": "HANSENNEA" }  { "_id": "B112", "NAME": "HANSENNEA" }</pre>

# L'update – L'opérateur \$unset

- Cet opérateur permet de supprimer un ou plusieurs champs d'un ou plusieurs documents.
- Structure de cet opérateur:
  - {\$unset: {<champ1>, ...}}
- Exemple de requête
  - Requête: Supprimer le champ ADRESSE de tous les documents de la collection « clients ».
  - Traduction: `db.clients.update({},{$unset: {ADRESSE}},{multi: true})`



# L'update – Les opérateurs \$min et \$max

- Cet opérateur permet de conditionner la modification d'une valeur numérique. Elle n'effectue la modification que si le champ possède une valeur supérieure/inférieure ou équivalente à celle donnée dans l'opérateur.
- Structure de l'opérateur
  - {<opérateur>: <nombre>}
- Exemple de requête
  - Requête: Mettre à jour QSTOCK si la quantité présente est supérieure ou égale à celle demandée.

# L'update – L'opérateur \$currentDate

- Cet opérateur permet d'attribuer la date actuelle sous différents formats à un champ.
- Structure de cet opérateur
  - {\$currentDate: {<champ\_applique>: <true ou {\$type: '<type>'}>, ...}}
- Exemple de requête
  - Requête: Mettre à jour la date de la dernière modification de QSTOCK du produit 'CS464'
  - Traduction DML: `db.produits.update({_id: 'CS464'},{$currentDate: {LastModified: true}})`

# L'update – La modification des champs

- Exercices

1. Ajouter à tous les produits une date(`LastStockMod`) permettant de savoir la dernière modification du stock.
2. Effectuer une modification du stock du produit dont l'\_id est 'CS262'. (ajout de 23 produits dans le stock)
3. Effectuer une modification de l'\_id, du LIBELLE (`NotFound404`), du PRIX (100) et du QSTOCK (404) du produit 'NF404'.

# L'update: La modification de tableaux

L'opérateur \$

Les opérateurs  
\$push et  
\$addToSet

L'opérateur  
\$pull

L'opérateur  
\$pullAll

# L'update – L'opérateur \$

- Cet opérateur permet de modifier un élément dont on ne connaît pas l'index.
- Structure de cet opérateur
  - {'<tableau>.\$.<champ>':value}
    - Le champ n'est pas obligatoire et permet de sélectionner un champ à mettre à jour.

# L'update – Les opérateurs \$push et \$addToSet

- Ces opérateurs ajoutent un élément à un tableau s'il est présent ou non.
- Structure de ces opérateurs
  - {<opérateur>: {<champ>: <valeur>, ...}}
- Exemple de requête
  - Ajouter au champ COMMANDES de clients de nouvelles commandes.
  - `db.clients.update({},{$push: {COMMANDES: {NCOM: '0000', DETAIL: {produit_id: 'CS672'}}}}, {multi: true})`

# L'update – L'opérateur \$pull

- Ces opérateurs permettent la suppression d'un des éléments d'un tableau répondant à un critère.
- Structure de cet opérateur
  - {\$pull: {<champ>:{<critère>}}, ...}
- Exemple de requête
  - Supprimer la COMMANDES dont le NCOM est '30182'
  - Traduction DML: `db.clients.update({COMMANDES: {$exists: true}},{$pull: {COMMANDES: {NCOM:30182}}},{})`

# L'update – L'opérateur \$pullAll

- Cet opérateur permet une suppression de tous les éléments d'un tableau correspondant à la liste de valeurs données.
- Structure de cet opérateur
  - {\$pullAll: {<champ>: [<valeur1>, <valeur2>, ...]}}



# L'update: Les modificateurs de \$push

Le modificateur  
\$each

Le modificateur  
\$slice

Le modificateur  
\$position

# L'update – Le modificateur \$each

- Ce modificateur permet de modifier le comportement de l'opérateur \$pull et \$addToSet en permettant l'ajout de chaque valeur d'un tableau.
- Structure de ce modificateur
  - {<champ>: {\$each: [<valeur1>, <valeur2>, ...]}}
- Exemple de requête:
  - requête: Mettre à jour le champ « A » en insérant deux nouveaux composants.
  - traduction: `db.experiment.update({},{$addToSet: {A: {$each: ['vert', 'jaune']}}}, {multi: true})`

# L'update – Le modificateur \$slice

- Ce modificateur permet de déclarer le nombre d'éléments maximum du tableau dans lequel se déroule l'insertion.
- Structure du modificateur
  - `{ $push: { <champ>: { $each: [<valeur>, ... ], $slice: <entier> } } }`
    - Un entier positif délimite à partir de la gauche
    - Un entier négatif à partir de la droite.
- Exemple de requête:
  - mettre à jour le tableau de résultat d'un étudiant.
  - traduction DML: `db.students.update( { _id: 2 }, { $push: { scores: { $each: [ 100, 20 ], $slice: 3 } } }, { upsert: true } )`

# L'update – Le modificateur \$sort

- Ce modificateur permet le tri des éléments d'un tableau durant l'action de l'opérateur \$push
- Structure de ce modificateur
  - {\$sort: {<champ>: <ordre>}}
- Exemple de modificateur:
  - trier les COMMANDES d'un client selon l'ordre décroissant du NCOM
  - traduction DML: `db.clients.update({$push: {COMMANDES: {$each: [{NCOM: 1}, {NCOM:2}], $sort: {NCOM: -1}}}})`

# L'update – Le modificateur \$position

- Ce modificateur de l'opérateur \$push permet de spécifier la position de départ de l'insertion d'élément dans le tableau.
- Structure de ce modificateur
  - {\$position: <entier>}
- Exemple de requête:
  - ajouter un 2<sup>ème</sup> composant au produit « A » de l'expérience dont l'\_id est 3. Ce composant est le vert.
  - traduction DML: `db.clients.update({_id: 3},{ $push: {A: { $each: ['vert'], $position: 1}}},{})`

# L'update – Isolation des opérations

## L'isolation des opérations d'écriture

- L'opérateur `$isolated` permet d'effectuer une opération d'écriture sur plusieurs documents en prévenant toutes opérations de lecture et d'écriture sur la durée de son opération d'écriture.
- Structure de cet opérateur
  - `{ $isolated: true }`
  - Se place dans l'étage de recherche.
- Exemple de requête:
  - requête: Mettre à jour les comptes positifs à 0
  - traduction: `db.clients.update({COMPTE: { $gt: 0 }, $isolated: true }, { $set: {COMPTE: 0 } }, { multi: true })`

NoSQL avancé avec MongoDB

# NoSQL avancé – Introduction

- Les moteurs NoSQL sont davantage performants lors de leurs utilisations avec un langage de programmation.
- Ces moteurs permettent par exemple de traiter les données et de les fragmenter en ensemble utile permettant l'extraction.





# Le DML – Les curseurs

Les méthodes  
map/forEach

La méthode count

# MongoDB – Les curseurs

- MongoDB s'utilisant dans une console de commandes, les curseurs ne posent pas de problème d'utilisation. Malheureusement, les interfaces utilisateur ne permettent pas toutes leurs utilisations.
- Exemple de GUI
  - Accepte: RobotMongo
  - Refuse: Mongo Management Studio
- Les méthodes associées au curseur peuvent être utilisées directement sur la requête et sur une variable contenant le retour d'une requête.

# MongoDB – Les curseurs

## Les méthodes map et forEach

- Ces deux méthodes permettent d'effectuer un traitement particulier sur les données récupérées par une requête.
- Structure de ces méthodes
  - `<curseur>.<methode>(function(doc){<traitement>;return <valeur>;})`

# MongoDB – Les curseurs

## La méthode count

- Cette méthode permet d'afficher le nombre d'éléments contenus dans le curseur.
- Structure de cette méthode
  - `<curseur>.count();`

# MongoDB – Les curseurs

## La méthode batchSize

- Cette méthode permet de limiter le nombre de documents renvoyés par la requête dans un message réseaux.
- Structure de cette méthode
  - <curseur>.batchSize(<nombre>)

# MongoDB – Les curseurs

## La méthode close

- Cette méthode permet la fermeture d'un curseur et la libération en mémoire de son contenu.
- Structure de cette méthode
  - `<curseur>.close()`

# MongoDB – Les curseurs

## Les méthodes comment, explain

- Ces méthodes permettent le débogage des requêtes en fournissant un moyen de commenter et d'observer ce que fait la requête.
- Structure de ces méthodes
  - `<curseur>.comment('<commentaire>');`
  - `<curseur>.explain();`

# MongoDB – Les curseurs

## Les méthodes limit, maxScan

- Ces méthodes permettent de limiter le nombre maximal de documents retournés ou lus par le moteur.
- La méthode « limit » spécifie la taille maximale du curseur.
- La méthode « maxScan » spécifie le nombre de documents maximum à lire.
- Structure de ces méthodes
  - `<curseur>.limit(<nombre>);`
  - `<curseur>.maxScan(<nombre>);`



# MongoDB – Les curseurs

## Les méthodes hasNext et next

- Ces méthodes permettent l'itération du curseur dans des boucles Javascript.
- La méthode « hasNext » renvoi un booléen.
- La méthode « next » renvoie le prochain document.
- Structure de ces méthodes
  - <curseur>.hasNext()
  - <curseur>.next()

# MongoDB – Les curseurs

## La méthode hint

- Cette méthode permet de forcer l'utilisation d'un index existant dans la collection.
- Structure de cette méthode
  - `<curseur>.hint('<nom_index>')`

# MongoDB – Les curseurs

- Exercice

1. Afficher en une phrase le nombre de clients ayant commandé chaque produit, le LIBELLE du produit, et la valeur du stock en €.
2. Afficher le LIBELLE et le montant du contenu dans le stock. Le montant doit respecter les règles de notations des nombres.
3. Expliquer le retour de la méthode explain.
4. Afficher le NOM, les PRODUIT des COMMANDES dont le LIBELLE contient le mot « SAPIN » des clients. (Attention, plusieurs requêtes sont nécessaires).
5. Afficher le nombre de personnes ayant effectué une COMMANDES d'un produit en ACIER.



# LE DML – Le Map-Reduce

# Le map-reduce – Introduction

- Le Map-Reduce est un paradigme de traitement des données consistant en la réduction de très grosses quantités de données en résultat d'agrégation utile.
- Dans MongoDB, ce paradigme est fourni par une méthode de la collection appelée « mapReduce ». Cette méthode fait appel à des connaissances de JavaScript afin de traiter la collection.
- Mongo Management Studio n'autorise pas le mapReduce

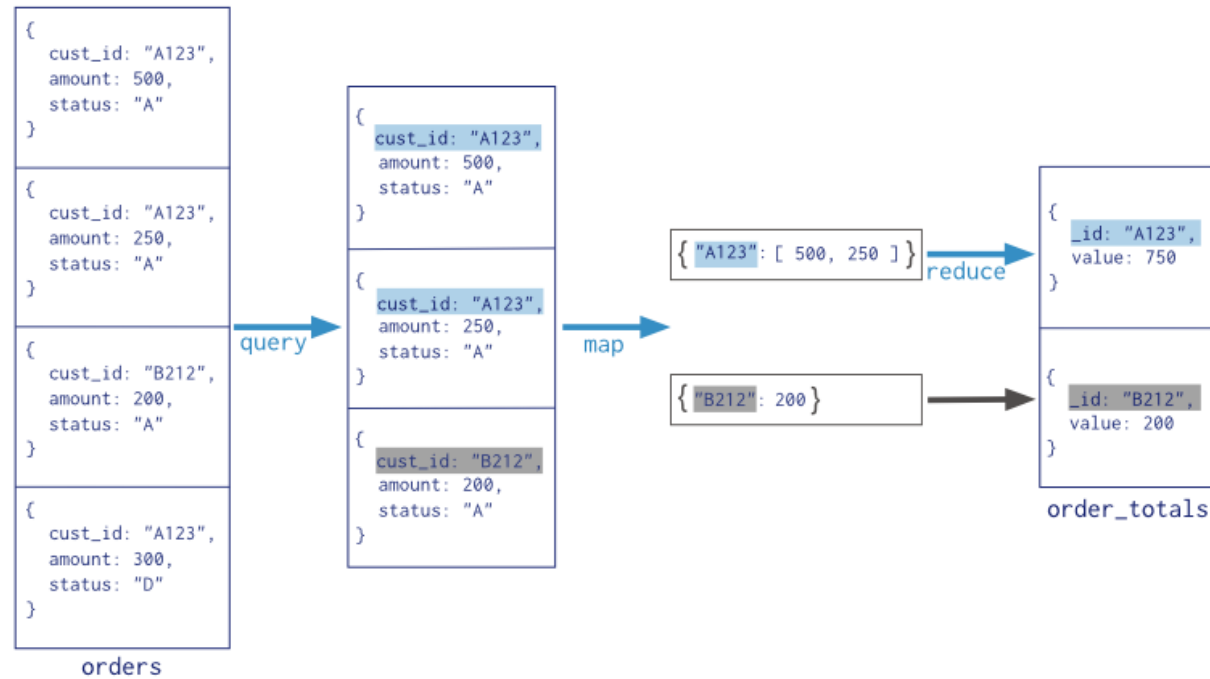
# Le Map-Reduce – Structure

- Structure d'un exemple simple

```
Collection
↓
db.orders.mapReduce(
  map    → function() { emit( this.cust_id, this.amount ); },
  reduce → function(key, values) { return Array.sum( values ) },
  {
    query: { status: "A" },
    out: "order_totals"
  }
)
```

# Le Map-Reduce – Structure

- Illustration de l'exemple et des étapes



# Le Map-Reduce – Le mapping

- La première fonction à insérer dans un mapReduce, est une fonction permettant de mapper les données sur lesquelles effectuer le traitement.
- Ainsi, cette fonction doit émettre les champs sur lesquels travailler.
- Structure générale
  - `function(){emit(this.<champ1>,...)}`
- Le retour de cette fonction est une paire de clé-valeurs dont le premier champ est la clé et le second, la valeur



# Le Map-Reduce – Exercices

1. Réaliser la réduction permettant de connaître le montant total des comptes des clients par LOCALITE et insérer le résultat dans une collection.
2. Réaliser la réduction permettant de connaître le montant total des comptes, la moyennes des comptes et le nombre de clients par LOCALITE.



# Le DML – La méthode group



# La méthode group - Introduction

- Group est une méthode de la collection permettant de grouper les données selon des champs spécifiés.
- Structure de cette méthode
  - `<collection>.group({  
 key: {<champ1>: true, <champ2>:true, ...},  
 reduce: <fonction JavaScript>,  
 initial: {},  
 <options>  
})`

# La méthode group – Options

- keyf

Fonction JavaScript permettant de créer un clé alternative.

- cond

Ensemble des critères de sélection sur les données à traiter.

- Finalize

Fonction JavaScript permettant de modifier la présentation de l'ensemble des données retourné.

# La méthode group – Exercices

1. Créer une requête affichant la LOCALITE, le nombre de client et le mois en abrégé de la date de la commande effectuée par les clients de ces localités.