NO SQL, Not Only SQL – partie 1

NO SQL
Not Only SQL

NO SQL - INTRODUCTION

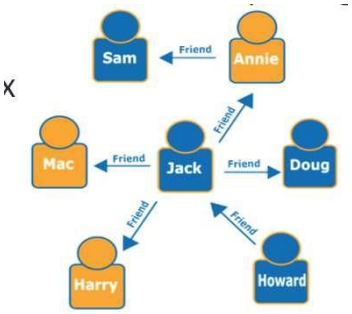
- bases de données non relationnelles
- stockage des données dans un format différent
- **1998 par Carl Strozz :** base de données relationnelle légère et open source
- concept adopté par Google, Facebook ou Amazon
- interrogation avec de multiples types de langage (déclaratif, requête, API)
  ➔ considérées comme des bases de données « pas seulement SQL ».
- évolutivité + disponibilité ➔ applications Web, big data en temps réel
- adaptation exigences ➔ développement agile
- "stockage intuitif" ➔ moins de transformations requises lors du stockage.

Différence entre le SQL et le NoSQL

BdD SQL : relationnelle ➔ langage SQL

BdD NoSQL : NON relationnelle ➔ syntaxe d'accès suivant structure :

- paire clé/valeur : tableau, clé unique, valeur est généralement JSON ou autre
- orientée colonne (BigTable de Google) : table partiellement remplie, ligne indexée par une clé
- orientée graphe : utilisée par les réseaux sociaux



- orientée document (mongoDB : SGBD NoSQL le plus utilisé) : paire clé-valeur stockée sous forme de **document au format JSON ou XML**

Différences entre les BdD relationnelles et les BdD NoSQL ?

SGBDR

- tables, colonnes/lignes/clé
- préalable : colonnes et types de données associés connus
- stockage liaison de tables par des clés

Bases de données NoSQL

- pas de schéma de données ➔ avancement rapide
- diversité, rapidité et volume ➔ gestion trafic élevé

Quand choisir une base de données NoSQL ?

- grands volumes de données et temps de réponse faibles (jeux en ligne, ecommerce)
-

Quand ne pas choisir une base de données NoSQL ?

- données normalisées obligatoirement : finance, comptabilité, planification des ressources (doublon interdit)
- requêtes complexes : jointures complexes, requêtes imbriquées
- **approche hybride** : (BdD SQL et Bd NoSQL combinées) flexibilité avec cohérence sans dégrader les performances.

Apports des BdD NoSQL?

- pour des requêtes rapides et simples,
- pour des données volumineuses
- pour des modifications fréquentes des applications.
- processus appelé « sharding »
- sharding : fragmentation d'une BdD en unités plus faciles à gérer ➔ ajout de machines ➔ gestion sur plusieurs serveurs ➔ meilleure accessibilité.
- ajout structurel ou données pour BdD SQL ➔ ajout puissance et mémoire à la machine existante ➔ limite dans la durée

➔ **BdD NoSQL : efficacité même en cas d'augmentation du volume de quantités de données importantes.**

Les avantages d'une base de données NoSQL

Constat : consommation de données, flux ininterrompu de contenus

➔ approche plus moderne et fluide du stockage

➔ adaptation rapide des BdD ➔ émergence NoSQL

- **flexibilité :** stockage libre, gestion tout format de données
  ➔ innovation et développement rapide d'applications
- **évolutivité :** volume et puissance
- **hautes performances :** temps de réponse rapides
- **disponibilité :** réplication automatiquement des données sur plusieurs serveurs
  ➔ latence (durée d'attente entre une demande et une réponse) réduite
- **fonctionnalité à un haut niveau :** conçues pour des répertoires de données distribués
  ➔ stockage volumineux (big date, e-commerce, jeux en ligne, réseaux sociaux)

Limitation du cours

Cours traitant les documents de nature textuelle (exclusion des documents multimédias) traitant :

- documents structurés au format JSON
- SGBD : MongoDB

Installation mongoDB et IDE/interface graphique

Contrainte : utilisation de son ordinateur personnel

Suivre mode opératoire décrit dans le document
Installation, utilisation mongoDB

Modélisation NoSQL

document : structure très simple à très complexe

collection : ensemble de documents

représentation document adaptée aux échanges

- XML : complet, très lourd ➔ complexe
- JSON : moins riche, très léger

Choix pour ce cours : JSON

BDR : modélisation essentielle

NoSQL : modélisation négligée ➔ **ERREUR**

➔ structuration collection de documents essentielle

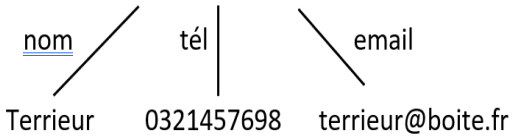
Documents

représenter des informations + ou - complexes :

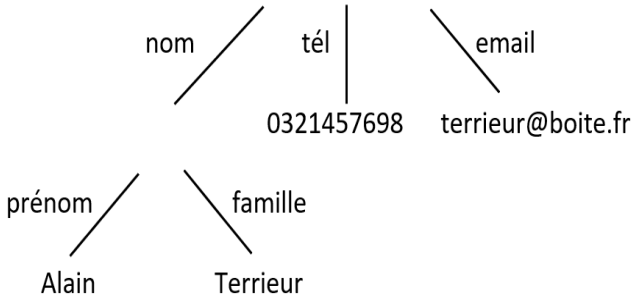
- flexibilité : structure adaptative à des variations plus ou moins importantes
- richesse de la structure : imbrication de tableaux, des liste, d'agrégats ➔ structures complexes.
- autonomie : inclusion de toutes les informations dans la représentation lors de l'échange de documents entre deux systèmes
- sérialisation : codage d'un document pour "voyager" sans dégradation sur le réseau, propriété essentielle pour un système distribué.

Documents structurés = graphes

Les noms sont sur les arêtes, les valeurs sur les feuilles.



Arbre représentant un agrégat



Arbre représentant une composition d'agrégats

JSON : ensemble minimal et suffisant de structures

Les paires (clé, valeur)	"nom" : "Terrieur"
Les agrégats	{"nom" : "Alain Terrieur", "tél" : 0321457698, "email" : "terrieur@boite.fr"}
Imbrication d'agrégats	{ "nom" : { "prénom" : "Alain", "famille" : "Terrieur }, "tél" : 0321457698, "email" : " terrieur@boite.fr " }
(clé, liste)	{"nom" : "Terrieur" , "téls" : [0321457698, 0624256531] }

NO SQL – MODÉLISATION JSON

Syntaxe JSON: JavaScript Object Notation

Créée pour la sérialisation, échange d'objets (archi REST en web)

Indépendant du langage de programmation

Modèle de données natif : mongoDB, couchDB

Structure de base : paire { clé, valeur }

Valeurs simples

chaînes de caractères : { "nom" : "Albert" }

nombres (entiers, flottants) : { "year" : 2019 }

valeurs booléennes (true ou false) : { "actif" : true }

Valeurs complexes

Objet (agrégat) :

ensemble de paires clé-valeur ; chaque clé n'apparaît qu'une fois.

{"last_name" : "Fincher", "first_name" : "David", "actif" : true}

Liste (array) : séquence de valeurs

"actors" : ["Eisenberg", "Mara", "Garfield", "Timberlake"]
"bricabrac" : ["Eisenberg", 1948, {"prenom", "Albert", "nom" : "Rigaux"}, true, [1, 2, 3]]

Imbrication sans limite

{ "title" : "The Social network", "summary" : "On a fall night in 2003, Harvard undergrad and programming genius Mark Zuckerberg sits down at his computer and heatedly begins working on a new idea. (...)", "year" : 2010, "director" : { "last_name" : "Fincher", "first_name" : "David" }, "actors" : [{ "first_name" : "Jesse", "last_name" : "Eisenberg"}, {"first_name" : "Rooney", "last_name" : "Mara"}] }

Exercice : validation d'un document JSON

Le document suivant contient des erreurs, à vous de les corriger Aidez-vous du validateur (http : //jsonlint.com/)

```
{
        "title" : "Taxi driver",
        "year" : 1976,
        "genre" : "drama",
        "summary" :  'Vétéran de la Guerre du Vietnam, Travis Bickle est chauffeur de taxi dans la ville de New York.
La violence quotidienne l'affecte peu à peu.',
        "country" :  "USA",
        "director" : {
        "last_name" :  "Scorcese",
        first_name :  "Martin",
        "birth_date" :  "1962"
        },
        "actors" :  [ {
                first_name :  "Jodie",
                "last_name" :  "Foster",
                "birth_date" :  "1962",
                "role" :  null
        }, {
                first_name :  "Robert",
                "last_name" :  "De Niro",
                "birth_date" :  "1943",
                "role" :  "Travis Bickle ",
        }
}
```

NoSQL :

- pas de méthodologie équivalente au relationnel
- pas de modèle normalisé

➔ extrapoler modèle relationnel vers modèle de documents

Exemple : schéma relationnel ➔ base NoSQL

soit le schéma E/A "films"

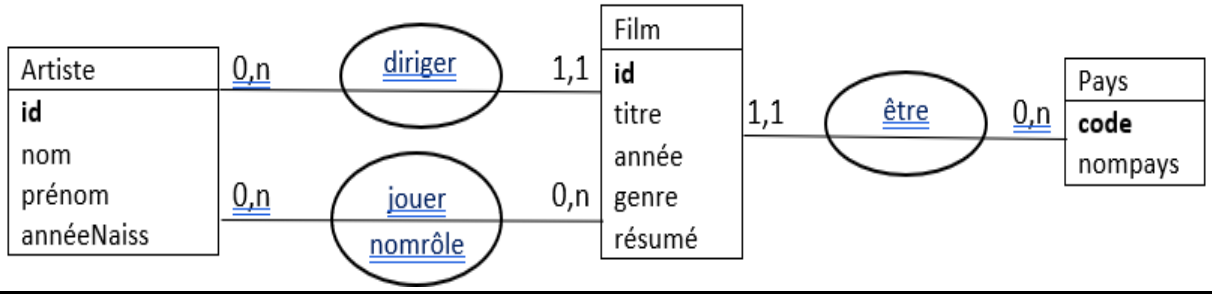


schéma relationnel

Artiste (**idArtiste** integer, nom varchar (30), prenom varchar (30)
,anneeNaiss integer) *tel que* (nom, prenom) est unique

Film (**idFilm** integer, titre varchar (50), annee integer
, idRealisateur# integer, genre varchar (20), resume varchar(255),
codePays varchar(4),

Role (**idFilm# integer, idActeur# integer**, nomRole varchar(30)

Avantages du schéma relationnel

- contrainte forte clé étrangère/clé primaire : vérification par le système relationnel et garantie d'une base saine
- pas de redondance
- pas d'anomalie de mise à jour

Inconvénients

- distribution des données dans plusieurs tables
  ➔ contenu de chacune incomplet.
- l'identité des acteurs d'un film n'est pas immédiat ➔ jointure
- la représentation des informations d'un film nécessite plusieurs lectures/écritures

Document structuré > structure tabulaire

exemple : ajouter les genres pour un film

➔ relationnel : table genre + table associant "film" et "genre"

➔ document structuré : ajout tableau "genre" dans document "film"

```
{
  "title" :  "Pulp fiction",
  "year" :  "1994",
  "genre" :  ["Action", "Policier", "Comédie"]
  "country" :  "USA"
}
```

NoSQL par identification : proche structure relationnel

exemple : réalisateur du film "pulp fiction"

modèle relationnel : tables "films", "artistes"

NoSQL :

deux collections

"artistes" { id, nom, prenom, datenaiss }

"films" {id, title, year, genre, country, idrealisateur, [idacteur,rôle] }

tels que

idrealisateur : id artiste réalisateur du film

[idacteur, rôle] : tableau des artistes acteurs du film avec le rôle de chacun

NoSQL par imbrication des structures

exemple : réalisateur du film "pulp fiction"

modèle relationnel : tables "films", "artistes"

document structuré :

```
{ "title" :  "Pulp fiction",
  "year" : "1994",
  "genre" : "Action",
  "country" : "USA",
  "director": { "last_name" : "Tarantino", "first_name" : "Quentin"
                ,"birth_date" : "1963" }
}
```

exemple : film "pulp fiction"

➔ relationnel : jointures

➔ document structuré : {

"title" : "Pulp fiction", "year" : "1994", "genre" : "Action", "country" : "USA"
,"director" : { "last_name" : "Tarantino", "first_name" : "Quentin"
,"birth_date" : "1963" }
,"actors" : [{ "first_name" : "John", "last_name" : "Travolta"
,"birth_date" : "1954", "role" : "Vincent Vega" }
, { "first_name" : "Bruce", "last_name" : "Willis"
,"birth_date" : "1955", "role" : "Butch Coolidge" }
, { "first_name" : "Quentin", "last_name" : "Tarantino"
,"birth_date" : "1963", "role" : "Jimmy Dimmick"}
] }

Avantages du modèle de documents par imbrication

- plus besoin de jointure
- création d'un document = 1 écriture
- lecture: une seule pour récupérer l'ensemble des informations.
- facilité de déplacement d'un document

Inconvénients

- hiérarchisation des accès :
  exemple des films : accès aux artistes en passant par les films
- perte d'autonomie des entités :
  exemple : supprimer un film ➔ risque suppression définitive des données d'un artiste
- redondance : *exemple* : un artiste sera représenté plusieurs fois.

NoSQL < BDR

- représentations asymétriques des informations
- pas de schéma, pas de contrôle des données

➔contrôle cohérence pris en charge par les applications

NoSQL > BDR

- pour des données peu ou faiblement structurés, graphes, données textuelles et multimédia
- peu de mises à jour, beaucoup de lectures (type analytique : écrire une fois, lire et relire pour analyser)
- pour les très gros volumes de données (quelques TéraOctets)
  exemple des films : accès aux artistes en passant par les films
- entrepôts de données (insertions, pas de mises à jour, lectures)

Exercice 1

Donner un document structuré donnant toutes les informations disponibles sur Quentin Tarantino. On veut donc représenter un document centré sur les artistes et pas sur les films.

Extrait de la base des films

Film

id	titre	année	idRéalisateur
1	Alien	1979	101
2	Vertigo	1958	102
3	Psychose	1960	102
4	Kagemusha	1980	103
5	Volte-face	1997	104
6	Pulp Fiction	1995	105
7	Titanic	1997	106
8	Sacrifice	1986	107

Role

idFilm	idArtiste	nomrole
6	11	Vincent Vega
6	27	Butch Coolidge
6	105	Jimmy Dimmick

Artiste

id	nom	prénom	année
11	Travolta	John	1954
27	Willis	Bruce	1955
101	Scott	Ridley	1943
102	Hitchcock	Alfred	1899
103	Kurosawa	Akira	1910
104	Woo	John	1946
105	Tarantino	Quentin	1963
106	Cameron	James	1954
107	Tarkovski	Andrei	1932

Exercice 2

Voici un exemple de documents centrés sur les étudiants et incluant les Unités d'Enseignement (UE) suivies par chacun.

```
{
  "_id" :  978,
  "nom" :  "Barnabé Dubois",
  "UE" :  [ {"id" : "ue : 11", "titre" :  "Java", "note" :  12},
       {"id" : "ue : 27", "titre" :  "Bases de données", "note" :  17},
       {"id" : "ue : 37",  "titre" :  "Réseaux", "note" :  14}
       ]
}
{
  "_id" :  476,
   "nom" :  "Vanessa Prado",
   "UE" :  [ {"id" :   "ue : 13",  "titre" :  "Méthodologie", "note" :  17,
        {"id" :  "ue : 27",  "titre" :  "Bases de données", "note" :  10},
        ]
}
```

Proposez une représentation des mêmes données centrée sur les UEs.

Serveur LINUX ➔ respect de la casse

Terminologie

base de données ➔ collection ➔ document ➔ champ

clé primaire : champ "_id" créé automatiquement si non présent lors de la création

Opérateurs de définition

db.createCollection("people") ; *Création de la collection « people »*

db.people.insertOne({ user_id: "casimir", age: 55, status: "A",niveau : 2 })

Ajout d'un seul document dans la collection "people"

résultat

```
{   "_id" : ObjectId("6231fb48839568e434207032"), "user_id" : "casimir", "age" : 55.0, "status" : "A", "niveau" : 2.0  }
```


Ajout de plusieurs documents

```
db.people.insertMany( [
        { _id: 11, user_id: "isidore", age: 40, status: "A" },
        { _id: 12, user_id: "dorian", age: 25, status: "B" }
        ] );
```

avec clé "_id"

ou

sans clé "_id" : la clé « _id » est automatiquement créée si elle n'est pas spécifiée dans la requête

```
db.people.insertMany( [
        { user_id: "isidore", age: 40, status: "A" },
        { user_id: "dorian", age: 25, status: "B" }
        ] );
```

NO SQL – MONGODB – OPÉRATEURS DE DEFINITION

Génération "_id" unique : ObjectId

{ "_id" : ObjectId("51 | 97 | c6 | b4 | 53 | cc | e2 | ec | 3a | 74 | 38 | 11") }

- timestamp courant
- identifiant de la machine
- identifiant du processus
- compteur qui démarre à un numéro aléatoire

➔ ObjectId contient automatiquement la date de création du document.

NO SQL – MONGODB – OPÉRATEURS DE DEFINITION

db.people.drop ( ) *suppression de la collection people*

db.people.createIndex ({ user_id: 1 }) *Index croissant sur « user_id »*

db. people.createIndex ({ user_id: 1, age: -1 })

Index sur (« user_id » croissant et « age » décroissant)

db.people.deleteMany ({ })

Suppression de tous les documents de la collection « people »

db.people.deleteMany ({ status: "D" }) *Suppression des documents de statut D*

NO SQL – MONGODB – OPÉRATEURS DE DEFINITION

*Ajout d'un champ dans un document déjà existant : opérateur **$set***

```
db.people.updateMany(  { }, { $set: { date_naiss: new Date() } } )
```

*Suppression d'un champ dans une collection : opérateur **$unset***

```
db.people.updateMany( { }, { $unset: { "date_naiss": "" } } )
```

Modifie le statut des documents si age > 25

```
db.people.updateMany( { age: { $gt: 25 } } , { $set: { status: "C" } } )
```

Ajoute 3 ans à l'âge dans chaque document si status = A documents si age > 25

```
db.people.updateMany( { status: "A" } , { $inc: { age: 3 } } ).
```

NO SQL – MONGODB – OPÉRATEURS DE DEFINITION

Opérateurs sur les champs dans "update"

$currentDate	date du jour
$inc, **$min**, **$max**, **$mul**	
$**rename**	renomme un champ

db.collection.find({conditions} , {projections})

db.people.find ( )

db.people.find({ }, { user_id: 1, status: 1 })

$eq, $gt, $gte, $in, $lt, $lte, $ne, $nin

$and, $not, $nor, $or

$nor : aucune condition vérifiée

opérateurs ensemblistes

$**exists** : existence d'un champ dans le document

$**type** : sélectionne un document si un champ est du type spécifié

opérateurs sur un champ tableau

$**all** : sélectionne le document si tous les éléments spécifiés dans la requête existe dans le tableau

$**elemMatch** : sélectionne le document si l'élément du tableau correspond à toutes les conditions spécifiées dans l'opérateur

$**size** : sélectionne le document si le tableau à la taille spécifiée

opérateurs d'évaluation

$**mod** : { nbetudiant : {$mod : [3,1] } }
retourne vrai si la division du nombre d'étudiants par 3 a pour reste 1

création d'un index obligatoire

$**text** : { $search :"étudiant" , $language :"fr"
, $caseSensitive :false, $diacriticSensitive : false }
cherche si le mot étudiant apparaît dans le document article, spécifié comme étant en français, la recherche sera ni sensible à la casse, ni diacritique (Etudiant, étudiant et etudiant sont acceptés)

{ champ : { $**where** : expression javascript } }
quand les opérateurs natifs de MongoDB ne suffisent plus,
on peut introduire du code javascript simple ou une fonction (plus lent)

NO SQL – MONGODB – PROJECTION

application de méthodes aux collections

```
db.people.find( … ).count( )

db.people.find( … ).sort( {user_id :1, age :-1 } )


db.people.find( … ).foreach(function(doc){ print(doc.user_id, doc.age) } )
```

Attention : l'utilisation d'une fonction personnalisée ralentit sensiblement la requête

```
db.people.find( … , {user_id :1, age : 1} )
```

est 400 fois plus rapide que la requête précédente pour résultat identique

Exercices: requête SQL / correspondance MongoDB

SELECT user_id, status FROM people	
SELECT * FROM people WHERE status = "A"	
SELECT user_id, status FROM people WHERE status = "A"	
SELECT * FROM people WHERE status != "A"	
SELECT * FROM people WHERE status = "A" AND age = 50	
SELECT * FROM people WHERE status = "A" OR age = 50	
SELECT * FROM people WHERE age > 25	
SELECT * FROM people WHERE age < 25	
SELECT * FROM people WHERE age > 25 AND age <= 50	
SELECT * FROM people WHERE user_id like "%dor%"	
SELECT * FROM people WHERE user_id like "do%"	
SELECT * FROM people WHERE user_id like "%mir"	
SELECT * FROM people WHERE status = "A" ORDER BY user_id ASC	
SELECT * FROM people WHERE status = "A" ORDER BY user_id DESC	
SELECT COUNT(*) FROM people	
SELECT COUNT(niveau) FROM people	
SELECT COUNT(*) FROM people WHERE age > 30	
SELECT DISTINCT(status) FROM people	
SELECT COUNT(DISTINCT status)) FROM people	

Cas pratique

Installation de la collection « publis »

Fichier JSON : publis.json

Exemples de document :


```
{
        "_id" : ObjectId("5b87df2c3b6c3a27dcc951ff"), "type" : "Article", "title" : "SmartKom-Public.",
        "pages" : { "start" : 471,"end" : 492 },
        "year" : 2006, "booktitle" : "SmartKom",
        "url" : "db/series/cogtech/54023732.html#HorndaschRR06",
        "authors" : [
                "Axel Horndasch",
                "Horst Rapp",
                "Hans Rittger"
                ]
}
```


Les chemins sur plusieurs niveaux doivent être obligatoirement entre guillemets : "pages.start"

Exprimez des requêtes simples pour les recherches suivantes :

1. Liste de tous les livres (type “Book”) ;
2. Liste des publications depuis 2011 ;
3. Liste des livres depuis 2014 ;
4. Liste des publications de l’auteur “Toru Ishida” ;
5. Liste de tous les éditeurs (type “publisher”), distincts ;
6. Liste de tous les auteurs distincts ;