**NO SQL, Not Only SQL – partie 1**

|  |
| --- |
| **LP GL - NOSQL Pascal NITSCHKE** |
| **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. |

|  |
| --- |
| **NO SQL - INTRODUCTION** |
| **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 * Une image contenant texte, capture d’écran, Police, logo    Description générée automatiquementorientée 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** |

|  |
| --- |
| **NO SQL - INTRODUCTION** |
| |  | | --- | | **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 |

|  |
| --- |
| **NO SQL - INTRODUCTION** |
| |  | | --- | | **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. |

|  |
| --- |
| **NO SQL - INTRODUCTION** |
| |  | | --- | | **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.** |

|  |
| --- |
| **NO SQL - INTRODUCTION** |
| **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) |

|  |
| --- |
| **NO SQL - INTRODUCTION** |
| **Limitation du cours**  Cours traitant les documents de nature textuelle (exclusion des documents multimédias) traitant :   * documents structurés au format JSON * SGBD : MongoDB |

|  |
| --- |
| **NO SQL - INTRODUCTION** |
| |  | | --- | | **Installation mongoDB et IDE/interface graphique** |   **Contrainte** : utilisation de son ordinateur personnel  Suivre mode opératoire décrit dans le document  Installation, utilisation mongoDB |









Une image contenant texte

Description générée automatiquement



**Exercice : validation d’un document JSON**

Le document suivant contient des erreurs, à vous de les corriger Aidez-vous du validateur

([http : //jsonlint.com/](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 ",

}

}



















**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.

Une image contenant texte

Description générée automatiquement



















**Exercices: requête SQL / correspondance MongoDB**

|  |  |  |  |
| --- | --- | --- | --- |
| SELECT user\_id, status FROM people | db.people.find({} {user\_id: 1, status: 1, \_id: 0}) | | |
| SELECT \* FROM people WHERE status = "A" | db.people.find({status: “A”} {}) | | |
| SELECT user\_id, status FROM people  WHERE status = "A" | db.people.find({status :  « A »} { user\_id: 1, status: 1, \_id: 0}) | | |
| SELECT \* FROM people WHERE status != "A" | Db.people.find({status: {$ne: “A”}}) | | |
| SELECT \* FROM people  WHERE status = "A" AND age = 50 | Db.people.find({$and: [{status: “A”}, {age: 50}]}) ou  db.people.find({status: “A”, age: 50}) | | |
| SELECT \* FROM people  WHERE status = "A" OR age = 50 | Db.people.find({$or: [{status: “A”}, {age: 50}]} ) | | |
| SELECT \* FROM people WHERE age > 25 | Db.people.find({age: {$gt: 25}}) | | |
| SELECT \* FROM people WHERE age < 25 | Db.people.find({age: {$lt: 25}}) | | |
| SELECT \* FROM people  WHERE age > 25 AND age <= 50 | Db.people.find({$and: [{age: {$gt: 25}}, {age: {$lte: 50}}]}) | | |
| SELECT \* FROM people WHERE user\_id like "%dor%" | | | Db.people.find(user\_id: /dor/}) |
| SELECT \* FROM people WHERE user\_id like "do%" | | | Db.people.find({user\_id: /^do/}) |
| SELECT \* FROM people WHERE user\_id like "%mir" | | | Db.people.find({user\_id: /mir$/}) |
| SELECT \* FROM people  WHERE status = "A" ORDER BY user\_id ASC | | Db.people.find({status: “A”}).sort({user\_id:1}) | |
| SELECT \* FROM people  WHERE status = "A" ORDER BY user\_id DESC | | Db.people.find({status: “A”}).sort({user\_id:-1}) | |
| SELECT COUNT(\*) FROM people | | Db.people.find().count() ou db.people.count() | |
| SELECT COUNT(niveau) FROM people | | Db.people.find(niveau: {$exists: true}}).count() | |
| SELECT COUNT(\*) FROM people  WHERE age > 30 | | Db.people.find({age: {$gt: 30}}).count() | |
| SELECT DISTINCT(status) FROM people | | Db.people.find(status: {$exists: true}}).distinct() ou db.people.distinct(“status”) | |
| SELECT COUNT(DISTINCT status)) FROM people | | Db.people.find(status: {$exists: true}).distinct().count() ou db.people.distinct(“status”).length | |

**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”) ;

**db**.**publis**.**find({**type**:** "Book"**})**

1. Liste des publications depuis 2011 ;

**db**.**publis**.**find({**year**:** **{**$gte**:** 2011**}})**

1. Liste des livres depuis 2014 ;

**db**.**publis**.**find({**$and **:[{**year**:** **{**$gte**:** 2011**}},** **{**type**:** "Book"**}]})**

1. Liste des publications de l’auteur “Toru Ishida” ;

**db**.**publis**.**find({**authors**:**"Toru Ishida"**})**

**db**.**publis**.**find({**authors**:{$in: [**"Toru Ishida"**]} })**

1. Liste de tous les éditeurs (type “publisher”), distincts ;

db.publis.distinct**(**"publisher"**)**

1. Liste de tous les auteurs distincts ;

db.publis.distinct**(**"authors"**)**