

# MOOCEBFR4 : Ingénierie des Données du Big Data : SGBD NoSql et Lacs de Données avec Big Data SQL par la pratique

Gabriel MOPOLO-MOKE  
Professeur chargé d'enseignements  
Université Côte d'Azur (UCA)

2023 / 2024

## Plan Général

### ➤ Plan

- *Module M4.1 : Rappel sur les concepts du Big Data et des SGBD NOSQL*
- **Module M4.2 : Introduction à Oracle NOSQL**
- **Module M4.3 : Oracle NoSql et le Modèle Key/Document**
- *Module M4.4 : INTRODUCTION A MONGODB ET LE MONGO SHELL*
- *Module M4.5 : INTRODUCTION A MONGODB ET SON API JAVA*
- *Module M4.6 : Architectures Big data et construction de lacs de Données avec Big Data SQL par la pratique*

# Module M4.2 : Introduction à Oracle NOSQL

G. Mopolo-Moké  
Professeur chargé d'enseignements  
Université Côte d'Azur (UCA)

2022 / 2023

## Module M4.2 : Introduction à Oracle NOSQL

### ➤ Plan

- **Module M4.2, section 1, partie 1 : Les concepts d'Oracle NOSQL**
- **Module M4.2, section 1, partie 2 : Les concepts d'Oracle NOSQL**
- **Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface**
- **Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java**

## Module M4.2, section 1, partie 1 : Les concepts d'Oracle NOSQL

## ■Module M4.2, section 1, partie 1 : Les concepts d'Oracle NOSQL

### ➤ Plan

- Objectifs
- Principales caractéristiques
- Bénéfices en cas d'utilisation d'Oracle NoSQL
- Types de données supportés par Oracle NoSQL
- Comment Oracle NoSQL Database fonctionne

# Module M4.2, section 1, partie 1 : Les concepts d'Oracle NOSQL

## ➤ Objectifs

- A la fin de la partie 1 de la section 1, vous devez être capable de :
  - D'identifier les **principales caractéristiques** d'Oracle NoSQL DB
  - **D'Expliquer l'architecture** d'Oracle NoSQL Database
  - D'identifier les **composants** d'Oracle NoSQL Database

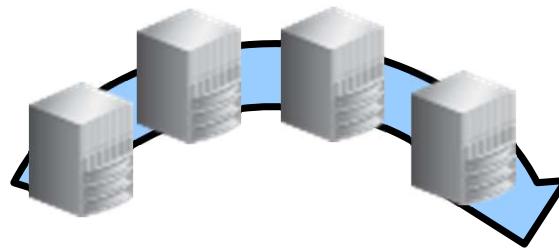
# Module M4.2, section 1, partie 1 : Les concepts d'Oracle NOSQL

## ➤ Principales caractéristiques

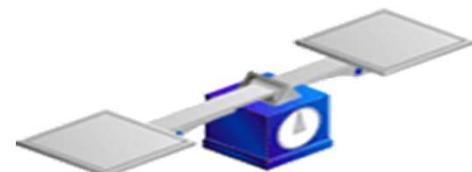
- Base de données **Clé-Valeur** à l'origine et maintenant aussi **Clé-Document**
- Base de données **écrites en Java**
- Accessible **via les API** : JAVA, C/C++, Python, NodeJS, C#/.NET, Go
- Construit à partir du SGBD Oracle **Berkeley DB**
- Fait partie de la solution Oracle pour la gestion du BIG DATA

# Module M4.2, section 1, partie 1 : Les concepts d'Oracle NOSQL

## ➤ Principales caractéristiques



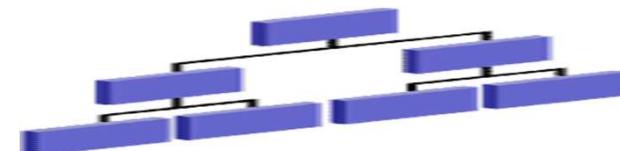
Scalabilité



Load Balancing /  
Equilibrage de charge



Availability /  
Disponibilité



Un modèle de données Simple (Key/Value)  
et complexe (Key/Document)

## Module M4.2, section 1, partie 1 : Les concepts d'Oracle NoSQL

### ➤ Bénéfices en cas d'utilisation d'Oracle NoSQL

- SGBD **Simple à installer et à configurer** : c'est sûr pour la version KVLITE
- SGBD **Universel** (tout type)
- SGBD avec Haute fiabilité
- SGBD permettant la Montée en charge
- SGBD avec Attente prédictibles

## Module M4.2, section 1, partie 1 : Les concepts d'Oracle NoSQL

### ➤ Types de données supportés par Oracle NoSQL



Text



Numérique



Vidéo



Image



Voix



Spatial



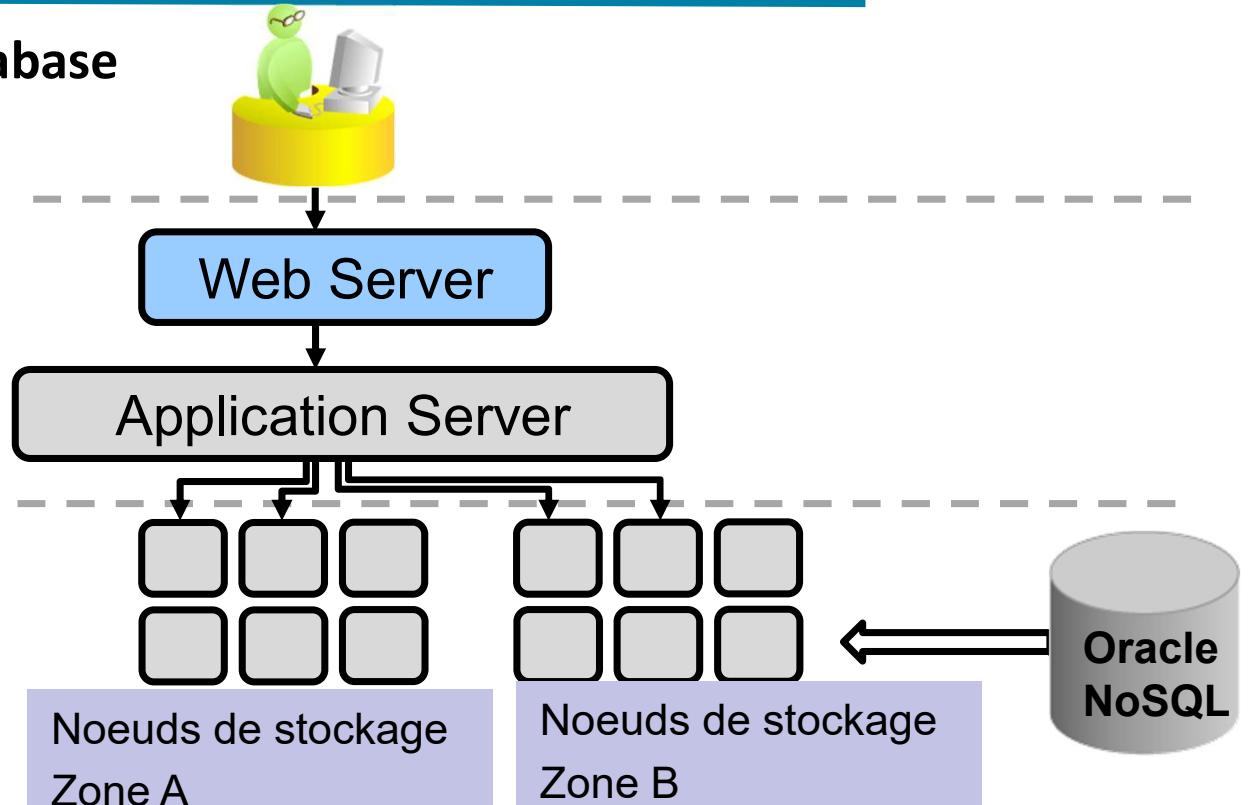
XML



Document

# Module M4.2, section 1, partie 1 : Les concepts d'Oracle NOSQL

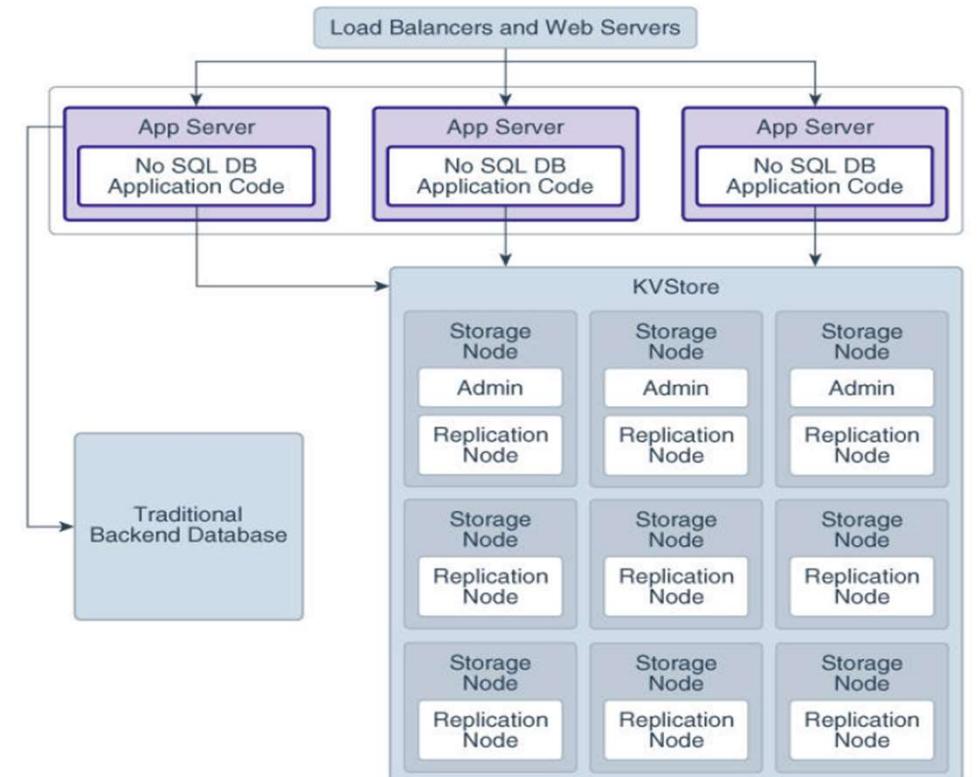
➤ Comment Oracle NoSQL Database fonctionne



# Module M4.2, section 1, partie 1 : Les concepts d'Oracle NOSQL

## ➤ Comment Oracle NoSQL Database fonctionne

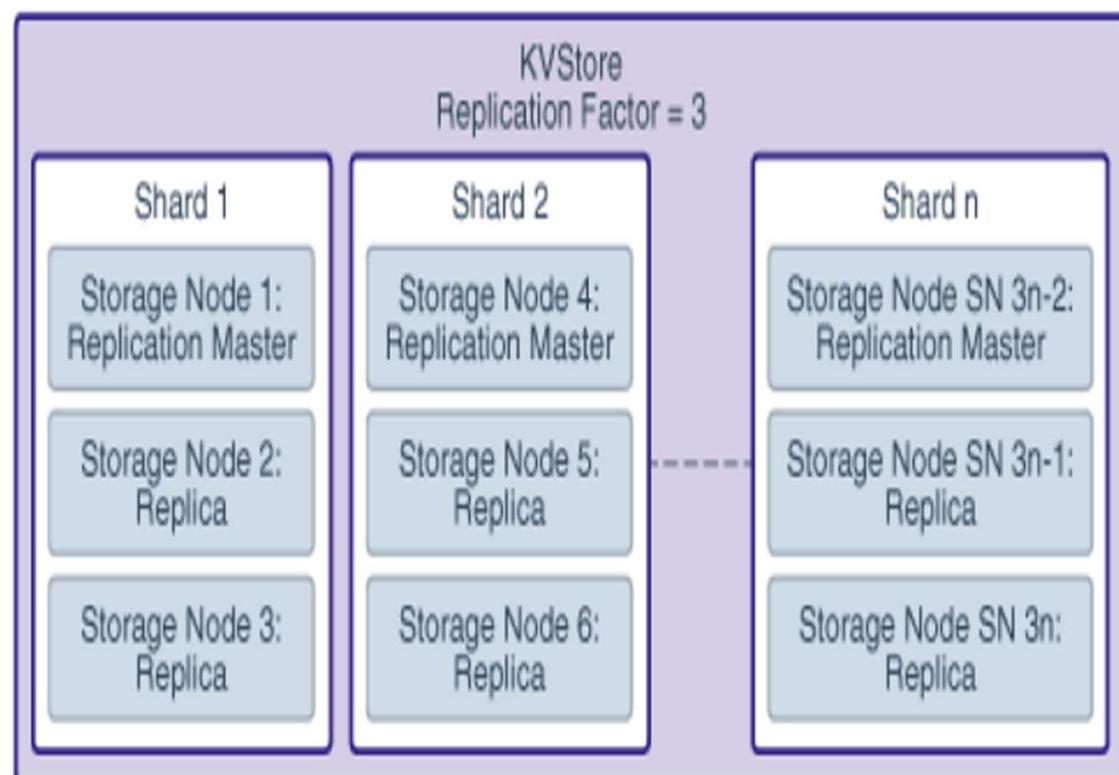
- Un nœud de stockage est une machine physique ou virtuelle
- Il est possible de créer des applications impliquant Oracle NOSQL et les SGBD traditionnels SQL



## Module M4.2, section 1, partie 1 : Les concepts d'Oracle NoSQL

### ➤ Comment Oracle NoSQL Database fonctionne

- Un **Shard** (ensemble de nœuds de réPLICATION dont l'un est maître) :
  - Les **maj sont faites dans le nœud maître (Master Node)**. Il est partitionné
  - Le **nœud maître réplique** sur les nœuds de réPLICATION
  - Les nœuds de réPLICATION **sont Read Only (lecture seule)**



## Module M4.2, section 1, partie 1 : Les concepts d'Oracle NOSQL

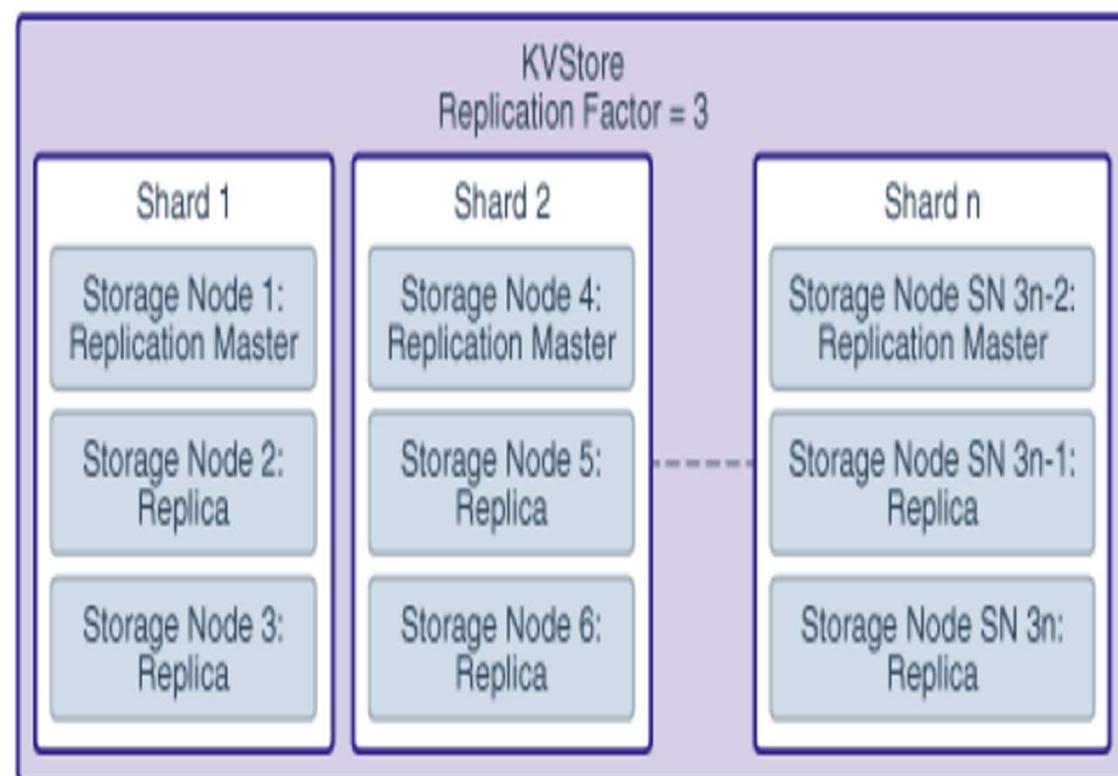
### ➤ Comment Oracle NoSQL Database fonctionne

#### ■ La réPLICATION dans un SHARD

- Se fait de façon **Asynchrone**
- Le nombre de **nœud réPLICATION n'est pas théoriquement limité**
- Oracle NOSQL supporte **le AP** du théorème CAP

#### ■ Replication factor :

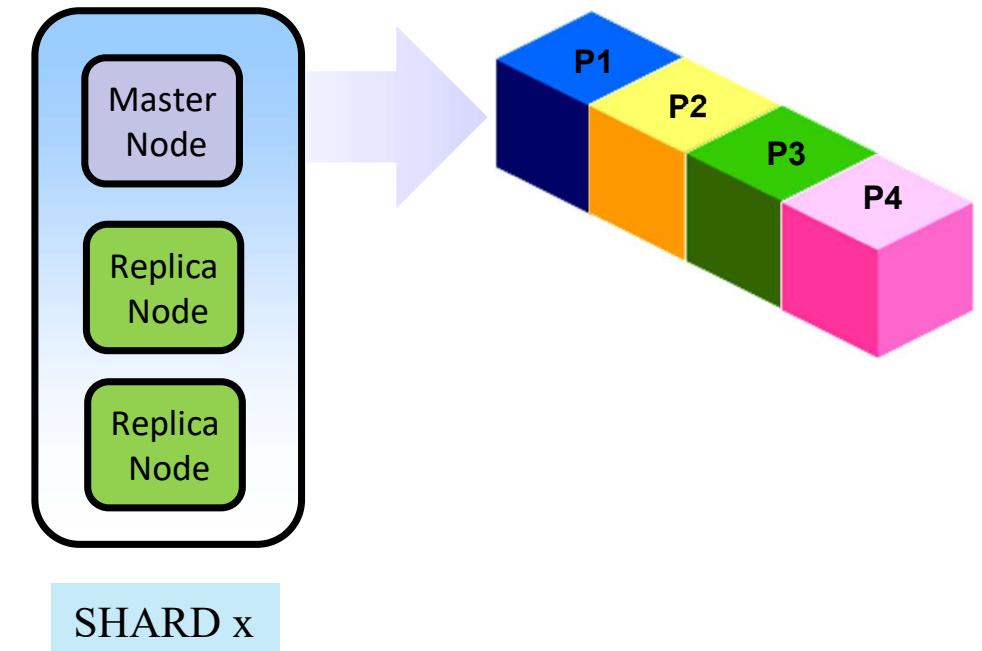
- Donne le nombre de nœud de réPLICATION dans un shard
- Plus le facteur de réPLICATION est grand, plus l'accès aux données sera rapide. Par contre l'écriture sera lente



## Module M4.2, section 1, partie 2 : Les concepts d'Oracle NOSQL

### ➤ Comment Oracle NoSQL Database fonctionne

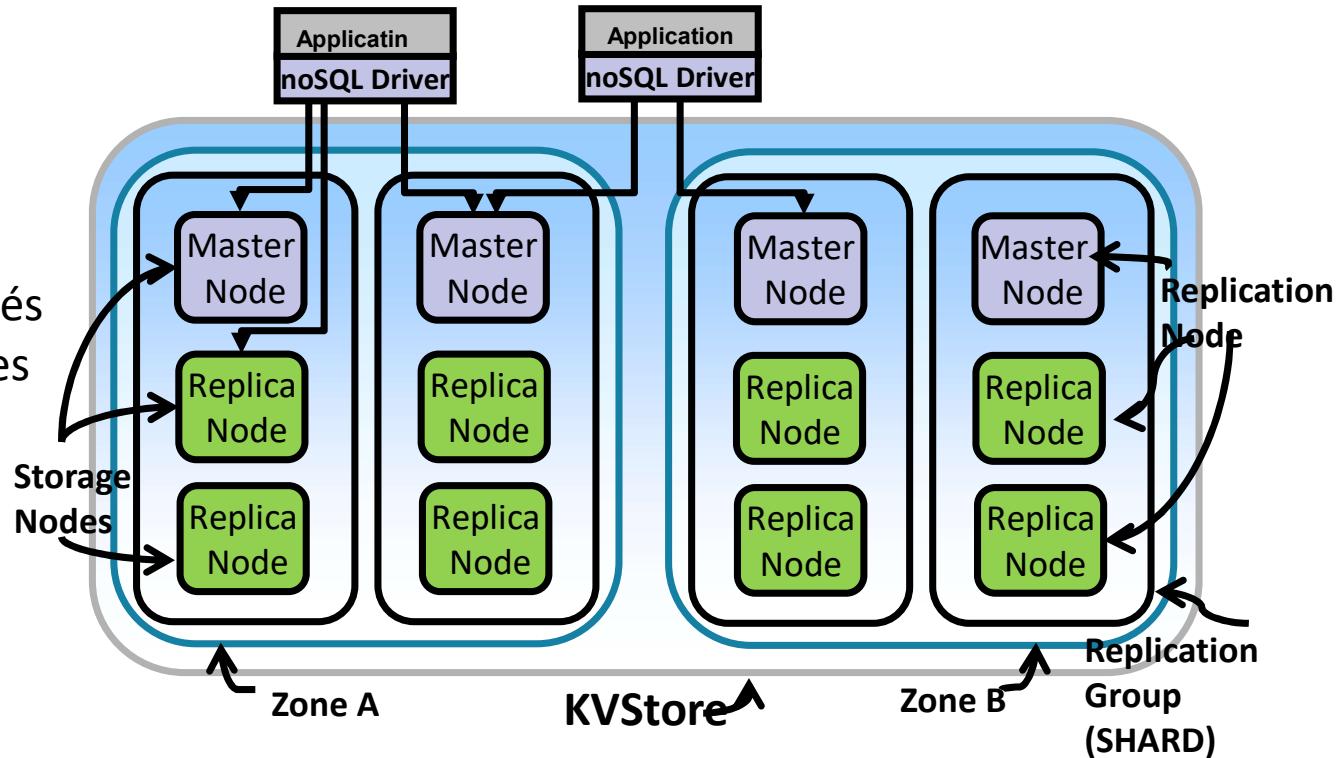
- Partitions dans un SHARD :
  - Chaque nœud de réPLICATION maître contient une ou plusieurs partitions
  - Chaque paire Clé/valeur atterrit dans une partition



## Module M4.2, section 1, partie 2 : Les concepts d'Oracle NOSQL

### ➤ Comment Oracle NoSQL Database fonctionne

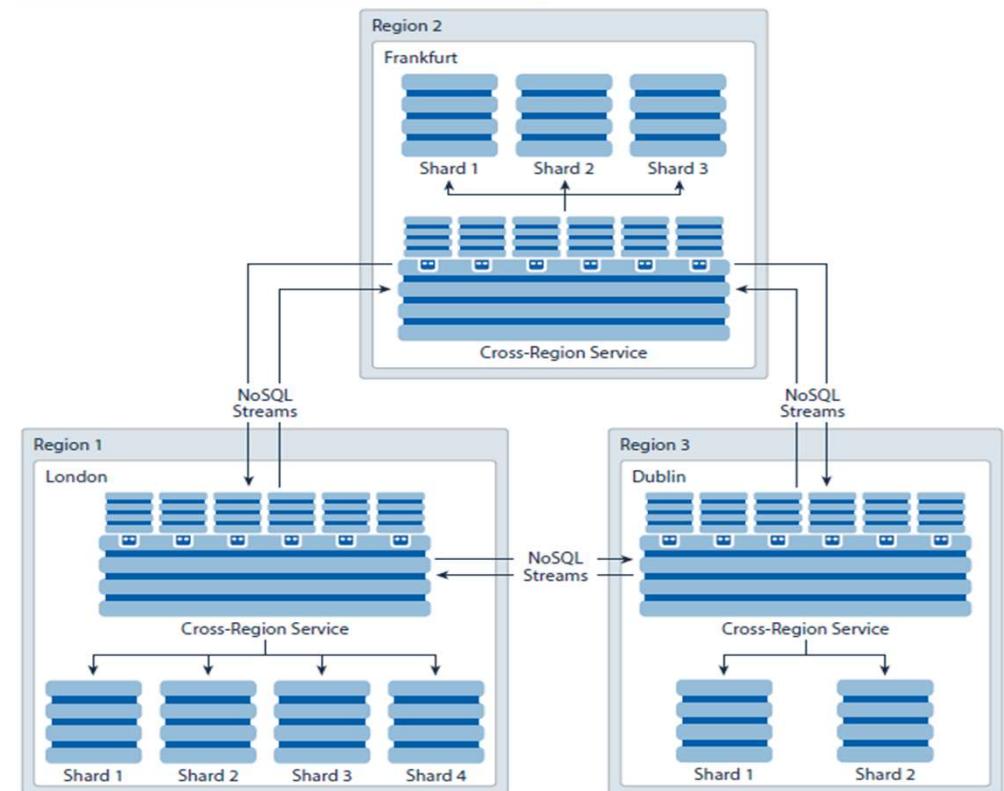
- Les nœuds de stockages sont regroupés dans des **ZONES** reliés entre eux par des bus physiques rapides
- **Equivalent** à peu près de la notion de **DATA CENTER**



## Module M4.2, section 1, partie 1 : Les concepts d'Oracle NOSQL

### ➤ Comment Oracle NoSQL Database fonctionne

- Architecture **MULTI-REGION** possible
- Chaque **REGION** à sa base NOSQL
- Une **REPLICATION** entre **REGION** peut être organisée pour recopier les données créés dans une région vers une autre (**CROSS REGION SERVICE**)

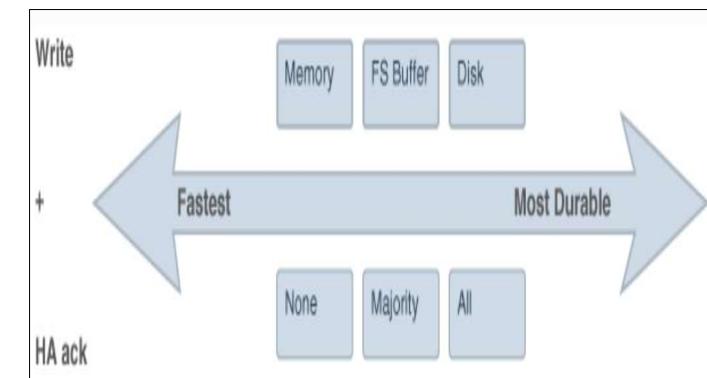


# Module M4.2, section 1, partie 1 : Les concepts d'Oracle NoSQL

## ➤ Comment Oracle NoSQL Database fonctionne

### ■ Durabilité

- Oracle NoSQL DB supporte **différentes approches de durabilité** :
  - ✓ **aucun** : écrire en mémoire sans garantir la persistance
    - Durability.**COMMIT\_NO\_SYNC**
  - ✓ **La majorité** : une majorité de nœuds garantissent la persistance
    - Durability.**COMMIT\_WRITE\_NO\_SYN**
  - ✓ **Tous** : tous les nœuds garantissent la persistance
    - Durability.**COMMIT\_SYNC**



**KVStore.put(Key key,  
Value value,  
ReturnValueVersion prevValue,  
Durability durability,  
long timeout,  
TimeUnit timeoutUnit)**

# Module M4.2, section 1, partie 1 : Les concepts d'Oracle NOSQL

## ➤ Comment Oracle NoSQL Database fonctionne

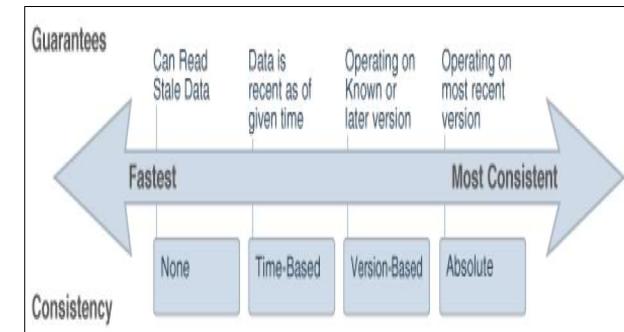
### ■ Quorum

- Les opérations qui modifient les données dans un **SHARD** requiert qu'une majorité de nœuds qui stockent les clés soient disponible afin de **garantir la cohérence et surtout la durabilité**
- Calcul du nombre de nœuds quorum
  - ✓ Pour un système à **majorité simple**, le quorum  $(N\text{nœud}/2)+1$
  - ✓ Pour le système à **majorité Absolut**, le quorum  $(N\text{nœud}-1)$
- **Note :** **Nnœud** est le nombre de nœud primaire de tous les **shard** dans une **zone** par exemple

# Module M4.2, section 1, partie 1 : Les concepts d'Oracle NoSQL

## ➤ Comment Oracle NoSQL Database fonctionne

- **Consistance** : Oracle NoSQL DB supporte différentes approches de consistance :
  - **aucune** : Lecture dans un **REPLICA** : aucune garantie en terme de consistance (rapide)  
✓ **Consistency.NONE**
  - **Basée sur le temps** : La lecture dans un nœud de réPLICATION n'est plus autorisée (exception) si le délai de recopie d'un objet du maître vers le réplica est dépassé. **Un timeout peut être accordé pour attendre la réconciliation**  
✓ **Consistency.Time**: Classe imbriquée dans **Consistency**
  - **Basée sur les versions** : La lecture dans un nœud de réPLICATION est autorisé si la version de l'objet dans le nœud de réPLICATION est identique à celle de l'objet avec la même clé dans le nœud maître. Un timeout pour laisser la réconciliation peut être accordé  
✓ **Consistency.Version**: Classe imbriquée dans **Consistency**
  - **Consistance absolue**: Lecture uniquement dans le nœud maître. Elle garantie la lecture de la dernière mise à jour (lent si toutes les transactions le font)  
✓ **Consistency.ABSOLUTE**



KVStore.get(Key key,  
**Consistency** consistency, ...)

## Module M4.2, section 1, partie 1 : QUIZ

### ➤ Question 1 : Cochez ce qui caractérise le mieux Oracle NOSQL

- A: Oracle NOSQL a été construit en s'inspirant du moteur Key / Value Berkeley DB
- B: Oracle NOSQL est une extension de Berkeley DB
- C: Oracle NOSQL écrit en Java
- D: Oracle NOSQL est un SGBD NOSQL multi-modèle

### ➤ Question 2 : Quels sont les modèles de données supportés par Oracle NOSQL

- A: Modèle Key / Value
- B: Modèle Key/ Document
- C: Modèle relationnel de CODD
- D: Modèle Objet Relationnel
- E: Modèle time série

### ➤ Question 3 : Pour gérer les documents, Oracle NOSQL s'appuie sur la notion de TABLE

- A: Les tables sont équivalentes aux tables des SGBD relationnels
- B: Les tables sont équivalentes aux collections MongoDB
- C: Les lignes des de ces tables sont des documents JSON
- D: L'insertion d'une ligne implique le contrôle des types à l'image des SGBD relationnels
- E: Non, non, à l'instar des SGBD NOSQL, chaque ligne a sa structure
- F: On dit aussi que Oracle NOSQL est un SGBD semi-structuré
- G: Non, non, Oracle NoSQL est un SGBD fortement structuré

## Module M4.2, section 1, partie 1 : QUIZ

### ➤ Question 4 : Cochez ce qui caractérise le mieux le partitionnement dans Oracle NOSQL

- A: C'est un SGBD partitionné, la partition est appelée SHARD
- B: C'est un SGBD partitionné, la partition est appelée NODE
- C: C'est un SGBD partitionné, dont le nombre de partition est illimité
- D: C'est un SGBD partitionné, il s'agit d'un partitionnement horizontal
- E: Les données d'une même table sont réparties dans plusieurs partitions en fonction de la SHARD KEY
- F: Les partitions Oracle permettent d'augmenter la puissance de mise à jour

### ➤ Question 5 : Cochez ce qui caractérise le mieux la réPLICATION dans Oracle NOSQL

- A: Une partition Oracle possède un nœud maître (mise à jour) et plusieurs nœuds esclaves (lectures)
- B: La réPLICATION dans Oracle NOSQL se fait en mode synchrone
- C: La réPLICATION dans Oracle NOSQL se fait en mode asynchrone
- D: Le nombre de nœud de réPLICATION est forcément limité
- E: Le nombre de nœud de réPLICATION n'est pas à priori limité

## Module M4.2, section 1, partie 2 : Les concepts d'Oracle NOSQL

# Module M4.2, section 1, partie 2 : Les concepts d'Oracle NOSQL

## ➤ Plan

- Positionnement d'Oracle Nosql
- Accès au KVStore : deux possibilités
- Rôle de KVLite
- Démarrage et arrêt de KVLite
- Vérifier que le KVLite est en fonctionnement
- Redémarrage et réexécution de KVLite
- Résumé du chapitre

## Module M4.2, section 1, partie 2 : Les concepts d'Oracle NOSQL

### ➤ Positionnement d'Oracle Nosql

Moteur NOSQL	Mode de représentation	Développé en	CAP	Application Communautaires	utilisation
Oracle NoSQL 12 C	Clé-Valeur , document	Java	AP	---	facilite le stockage efficace pour des quantités massives de données dans un format simple et flexible.
CouchDB	Document	Erlang	AP	CERN	Développement Web sans besoin de très bonnes performances
MongoDB	Document	C++	CP	---	Montée en charge facilitée sur les données semi structurées
Hbase	Colonne	Java	CP	Twitter, Netflix	Excellentes performances sur de très grands volumes
Cassandra	Colonne	Java	AP	Facebook, Twitter, Netflix	Montée en charge linéaire sur des grands volumes de données
Riak	Clé-Valeur	Erlang	AP		Excellent performance et montée en charge très aisée en ajoutant des nœuds
Redis	Clé-Valeur	ANSI C	CP	---	Entre le système de cache et la base de données. Assure d'Excellentes performances sur des traitements de données simples.

# Module M4.2, section 1, partie 2 : Les concepts d'Oracle NOSQL

## ➤ Positionnement d'Oracle Nosql

### ■ Comparaison MongoDB, Oracle NoSQL et Cassandra par DB ENGINE

- <https://db-engines.com/en/system/Cassandra%3BMongoDB%3BOracle+NoSQL>

### ■ Principaux Clients Oracle

- Large credit card company
- Airbus
- NTT Docomo
- Global rewards company

## Module M4.2, section 1, partie 2 : Les concepts d'Oracle NOSQL

### ➤ Accès au KVStore : deux possibilités

- **Utilisation des API (C/C++, C#, Go, Java, JavaScript (Node.js), Python)** Pour accéder aux données Clé-Valeur
- **Utilisation de l'interface ligne de commandes ou de l'interface graphique WEB (console)** pour les actions d'administrations et de manipulation des données

## Module M4.2, section 1, partie 2 : Les concepts d'Oracle NoSQL

### ➤ Rôle de KVLite

- **KVLite** est une **version simplifiée d'Oracle NoSQL Database**
  - Il n'y a qu'un seul **NOEUD**. Ce noeud n'est pas répliqué
  - KVLite est installé avec Oracle NoSQL Database
  - **Pas besoin de configuration**. Tourne comme un seul processus
  - N'est pas destiné pour des applications **en production**
  - Utile pour une **prise en main rapide** d'Oracle NoSQL DB
  - Utile pour apprendre à utiliser **les API**
  - Utile pour **créer et tester les applications**

## Module M4.2, section 1, partie 2 : Les concepts d'Oracle NOSQL

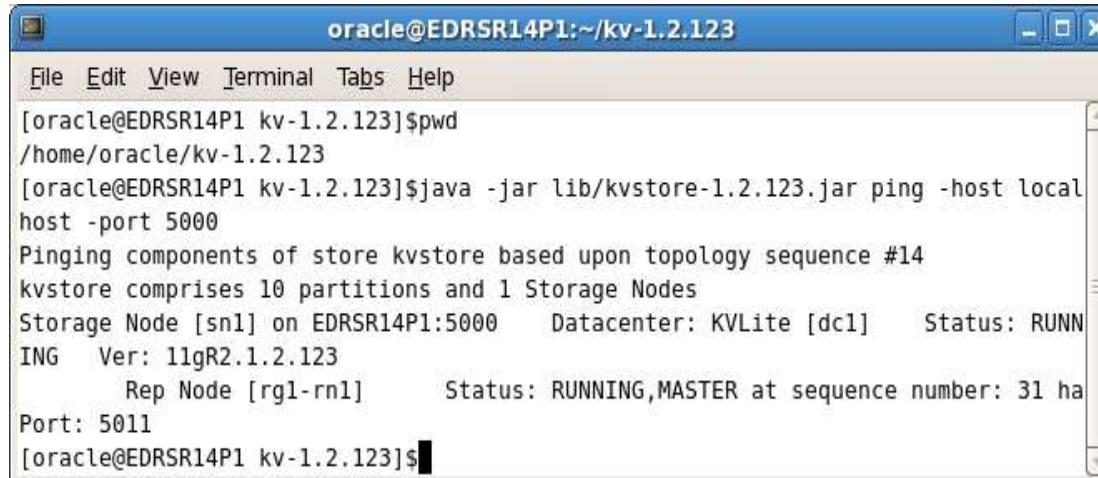
### ➤ Démarrage et arrêt de KVLite

```
> java -jar KVHOME/lib/kvstore-1.2.123.jar
      kvlite -host <hostname>
      -admin <port>
      -port <port>
      -store <storename>
      -root <path>
      -help
```

## Module M4.2, section 1, partie 2 : Les concepts d'Oracle NOSQL

### ➤ Vérifier que le KVLite est en fonctionnement

```
> java -jar KVHOME/lib/kvstore-1.2.123.jar ping  
      -host <hostname>  
      -port <port>
```



The screenshot shows a terminal window titled "oracle@EDRSR14P1:~/kv-1.2.123". The user has run the command "java -jar lib/kvstore-1.2.123.jar ping -host local host -port 5000". The output indicates that the store is pinging components based on topology sequence #14, which comprises 10 partitions and 1 Storage Node. It details the Storage Node [sn1] at EDRSR14P1:5000, Datacenter: KVLite [dc1], Status: RUNNING, Version: 11gR2.1.2.123, Rep Node [rg1-rn1], Status: RUNNING, MASTER at sequence number: 31, and Port: 5011.

## Module M4.2, section 1, partie 2 : Les concepts d'Oracle NOSQL

### ➤ Redémarrage et réexécution de KVLite

#### ■ Pour redémarrer KVLite

- Relancer l'utilitaire ou le fichier kvlite.sh

#### ■ Pour réexécuter KVLite

- Supprimer le répertoire KVRoot
- Exécuter KVLite à nouveau avec de nouvelles options si utile

## Module M4.2, section 1, partie 2 : QUIZ

➤ **Question 1 :** KVLite est une version simplifiée d'Oracle NoSQL Database. Cochez ce qui caractérise KVLite

- A: Il n'y a qu'un seul NŒUD. C'est le nœud master
- B: Peut être utilisé pour faire de la production
- C: Utile pour créer et tester des applications
- D: Très complexe dans sa prise en main

➤ **Question 2 :** Pour démarrer serveur KVLite il faut lancer la commande suivante

- A: java -Xmx256m -Xms256m -jar %KVHOME%/lib/kvstore.jar kvlite -secure-config disable
- B: java -Xmx256m -Xms256m -jar %KVHOME%\lib\kvclient.jar
- C: java -jar %KVHOME%\lib\kvstore.jar ping -host localhost -port 5000
- D: java -jar %KVHOME%/lib/kvcli.jar -host localhost -port 5000 -store kvstore
- E: java -jar %KVHOME%/lib/sql.jar -helper-hosts localhost:5000 -store kvstore

## Module M4.2, section 1, partie 2 : QUIZ

➤ **Question 3 :** Pour démarrer le client KVLite il faut lancer la commande suivante

- A: java -Xmx256m -Xms256m -jar %KVHOME%/lib/kvstore.jar kvlite -secure-config disable
- B: java -Xmx256m -Xms256m -jar %KVHOME%\lib\kvclient.jar
- C: java -jar %KVHOME%\lib\kvstore.jar ping -host localhost -port 5000
- D: java -jar %KVHOME%/lib/kvcli.jar -host localhost -port 5000 -store kvstore
- E: java -jar %KVHOME%/lib/sql.jar -helper-hosts localhost:5000 -store kvstore

➤ **Question 4 :** Pour envoyer un ping au serveur KVLite il faut lancer la commande suivante

- A: java -Xmx256m -Xms256m -jar %KVHOME%/lib/kvstore.jar kvlite -secure-config disable
- B: java -Xmx256m -Xms256m -jar %KVHOME%\lib\kvclient.jar
- C: java -jar %KVHOME%\lib\kvstore.jar ping -host localhost -port 5000
- D: java -jar %KVHOME%/lib/kvcli.jar -host localhost -port 5000 -store kvstore
- E: java -jar %KVHOME%/lib/sql.jar -helper-hosts localhost:5000 -store kvstore

## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

# Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

## ➤ Plan

- Généralités
- Activation de l'interface ligne de commandes
- Liste des commandes de l'interface ligne de commandes
- La commande SHOW
- Exemple de référence pour la mise en œuvre de certaine commandes
- Un premier pas dans la conception de la clé avec Oracle NOSQL
- La commande PUT
- La commande GET
- La commande DELETE

## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

### ➤ Généralités

- C'est l'outil principal pour l'administration et la manipulation d'un KVSTORE
- L'Interface ligne de commandes (CLI) est utilisée pour :
  - **Configurer, déployer et modifier** les composants du store
  - **Vérifier** le système, vérifier le status des services, vérifier la présence d'évènements critiques, vérifier les logs du KVSTORE
  - **LIRE, ECRIRE et SUPPRIMER** des enregistrements ou des tables dans le KVSTORE
  - Gérer des **SCHEMAS** si Key/Value de base
- **NOTES** : Un monitoring graphique SANS MISE A JOUR est possible la CONSOLE D'ADMINISTRATION GRAPHIQUE

## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

### ➤ Activation de la ligne de commandes

- Ouvrir un xterm (terminal)
- Attention : KVLITE doit avoir été démarré
- [oracle@bigdatalite nosql]\$ java -Xmx256m -Xms256m -jar \$KVHOME/lib/kvstore.jar runadmin -port 5000 -host localhost
- Si l'exécution est sans erreur, le prompt suivant apparaît :

kv->

# Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

## ➤ Liste des commandes de la ligne de commandes

- kv-> help

```
kv-> help
Oracle NoSQL Database Administrative Commands:
aggregate
await-consistent
change-policy
configure
connect
delete
execute
exit
get
help
history
load
logtail
namespace
page
ping
plan
pool
put
repair-admin-quorum
show
snapshot
table-size
timer
topology
verbose
verify

kv->
```

# Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

## ➤ Liste des commandes de la ligne de commandes

Commande	Rôle	Exemple
Change-policy	Les admin, Storage Node et replication node ont parfois des valeurs par défaut. Cette commande permet de les modifier	Change-policy -params [name=value]
configure	Configure un nouveau store	configure -name
connect	Se connecter à CLI-A&D active sur une autre machine	connect -host -port
ddl	Ajout, désaction ou activation de schéma	Voir le cours
exit	Sortie de CLI-A&D	
help	Liste des commandes / syntaxe	Help / help history ...
history	Historique de fonction du KVSTORE	history [-last] [-from] [-to]
load	Charger et exécuter un script	Voir le cours
logtail	Affiche les log du KVSTORE	
ping	Vérifie que KVSTORE est lancé	
plan	Modifie les paramètres du KVSTORE	
pool	Gère des pools de noeuds de stockage	pool create -name nomPool
show	Visualise la configuration d'un kvstore	Show / show topology
snapshot	Permet de gérer des sauvegardes	snapshot create -name Thursday ...
topology	Donne la topology du KVSTORE	verify configuration [-verbost]
verbose	Affiche des messages de vérification	
verify	Vérifie la topology du KVSTORE	

# Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

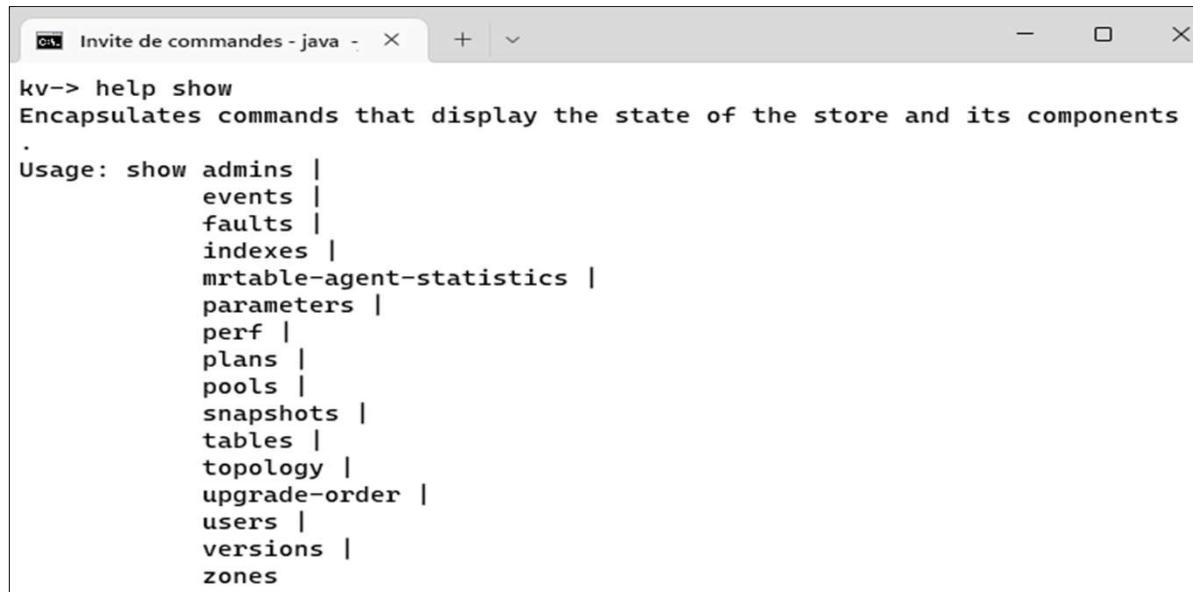
## ➤ Liste des commandes de la Command Line Interface

Commande	Rôle	Exemple
aggregate	Effectue des opérations de groupes sur les objets : count, sum, et average	aggregate -count Compte tous les enregistrements dans le STORE
connect	Se connecter à une machine	
delete	Supprimer un objet connaissant sa clé	Voir le cours
exit	Sortie de CLI	
get	Charger un objet connaissant sa clé	Voir le cours
help	Liste des commandes / syntaxe	Help / help load ...
load	Charger un script contenu dans un fichier	Voir le cours
put	Créer un objet avec sa clé	Voir le cours
show	Affiche les informations sur les schémas et les store	Voir le cours Show schemas
time	Compte le temps mis pour exécuter une commande	time get -all -file ./data.out

## 2.1.3 Command Line Interface pour l'Administration et les Données

### ➤ La commande SHOW

- Cette commande permet de visualiser les informations sur la configuration d'un STORE



```
Invité de commandes - java - X + - □ ×
kv-> help show
Encapsulates commands that display the state of the store and its components
.
Usage: show admins |
       events |
       faults |
       indexes |
       mrtable-agent-statistics |
       parameters |
       perf |
       plans |
       pools |
       snapshots |
       tables |
       topology |
       upgrade-order |
       users |
       versions |
       zones
```

## 2.1.3 Command Line Interface pour l'Administration et les Données

### ➤ La commande SHOW

kv->Show pools

kv-> Show admins

kv-> Show version

kv-> Show zone

kv-> Show fault

kv-> Show topology

```

kv-> Show pools
AllStorageNodes: sn1 zn:[id=zn1 name=KVLite]
KVLitePool: sn1 zn:[id=zn1 name=KVLite]

kv-> Show admins
admin1: Storage Node sn1 type=PRIMARY (RUNNING,MASTER)

kv-> Show version
Client version: 21.2.46
Server version: 21.2.46 2022-05-24 20:36:59 UTC Build id: 1b73ce65d872 Edition:
Community

kv-> Show zone
zn: id=zn1 name=KVLite repFactor=1 type=PRIMARY allowArbiters=false masterAffinity=false

kv-> Show fault

kv-> Show topology
store=kvstore numPartitions=10 sequence=14
  zn: id=zn1 name=KVLite repFactor=1 type=PRIMARY allowArbiters=false masterAffinity=false

  sn=[sn1] zn:[id=zn1 name=KVLite] PC-GABRIEL:5000 capacity=1 RUNNING
    [rg1-rn1] RUNNING
      single-op avg latency=0 ms  multi-op avg latency=0 ms

  numShards=1
  shard=[rg1] num partitions=10
    [rg1-rn1] sn=sn1

```

## 2.1.3 Command Line Interface pour l'Administration et les Données

### ➤ La commande SHOW

- Show tables

- Affiche la liste des tables

```
kv-> show tables
Tables in all namespaces:
  SYS$IndexStatsLease -- 1
  SYS$MRTableAgentStat -- 1
  SYS$MRTableInitCheckpoint -- 1
  SYS$PartitionStatsLease -- 1
  SYS$SGAttributesTable -- 1
  SYS$StreamRequest -- 1
  SYS$StreamResponse -- 1
  SYS$TableStatsIndex -- 2
  SYS$TableStatsPartition -- 2
  airbase2:pilote1
  airbase2:pilote2
  cpilote
  pilote2
    pilote2.checkup
    pilote2.training
  pilote3
    pilote3.checkup
```

# Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

## ➤ La commande SHOW

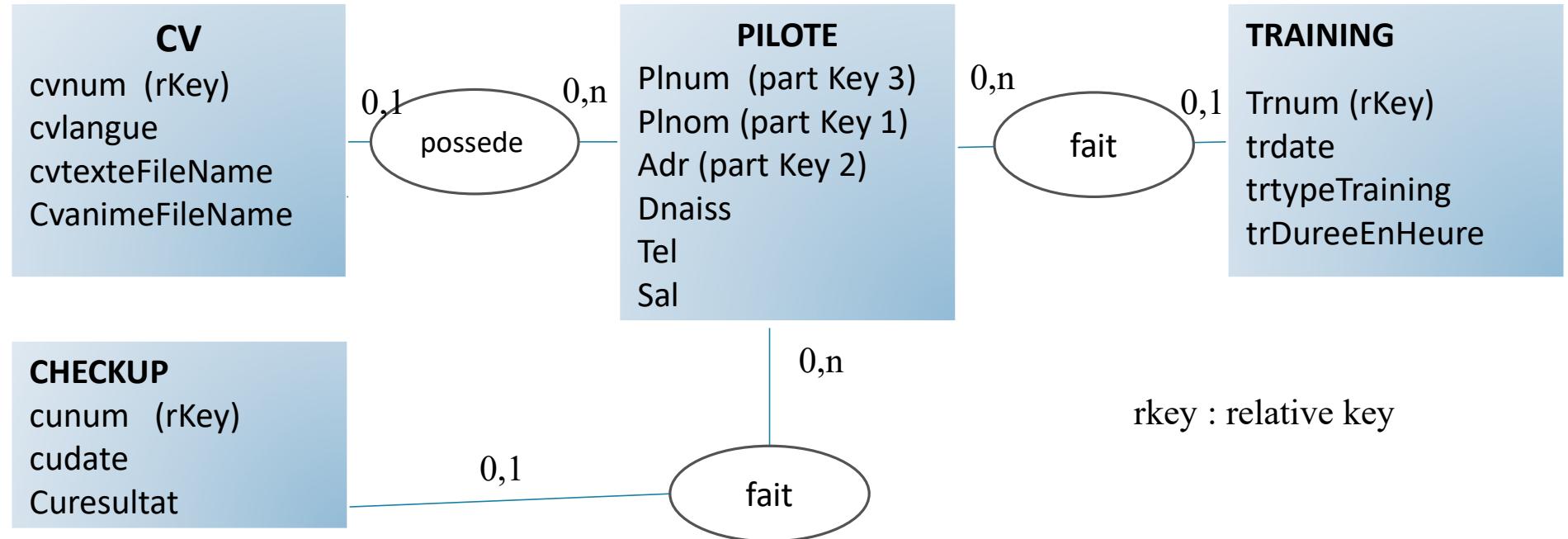
### ▪ Show plans

```
kv-> show plans
 66 DropTable pilote3      SUCCEEDED
 67 CreateTable pilote3    SUCCEEDED
 68 CreateTable pilote3.checkup SUCCEEDED
 69 DropTable pilote2.training SUCCEEDED
 70 DropTable pilote2.checkup SUCCEEDED
 71 DropTable pilote2      SUCCEEDED
 72 CreateTable pilote2    SUCCEEDED
 73 CreateTable pilote2.checkup SUCCEEDED
 74 CreateTable pilote2.training SUCCEEDED
 75 CreateIndex pilote2:idx_pilote2_adr SUCCEEDED
```

## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

➤ Exemple de référence pour la mise en œuvre de certaines commandes

- Considérons le schéma conceptuel suivant décrivant des pilotes



## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

### ➤ Un premier pas dans la conception de la clé avec Oracle NOSQL

- Une bonne conception de la clé sera un élément fondamental dans le cadre
  - du modèle **clé/Valeur**
  - Et même **clé/document**
- Une **clé à champs multiples** est essentielle pour augmenter les capacités de recherche surtout pour le modèle key/value
- Une clé à champs multiples peut être structurée en Major et Minor Key pour **éviter d'adresser de trop gros objets** avec une même clé: **/MajorKey1/MajorKey2/.../MinorKey1/MinorKey2/...**
- Une **discussion plus approfondie** sera faite dans les sections qui suivent sur la clé

# Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

## ➤ La commande PUT

- Permet de créer une valeur en lui associant une clé
- Put kv -key /MajorKey1/MajorKey2/...-/MinorKey1/MinorKey2/... -value "une Valeur non structure"
- put kv -key /Smith/Bob/-/phonenumbers -value "408 555 5555"

```
oracle@bigdatalite:u01/nosql/kv-ee/tpnosql/hello
File Edit View Search Terminal Help
kv->
kv->
kv->
kv-> put kv -key /Smith/Bob/-/phonenumbers -value "408 555 5555"
Operation successful, record inserted.
kv-> help put
The put command encapsulates commands that put key/value pairs into a store
and rows into a table.
Usage: put kv |
       table
kv-> help put kv
Usage: put kv -key <key> -value <valueString> [-file] [-hex | -json <schemaName>]
      [-if-absent | -if-present]
Puts the specified key, value pair into the store
-file indicates that the value parameter is a file that contains the
actual value
-hex indicates that the value is a BinHex encoded byte value with Base64
-json indicates that the value is a JSON string.
-json and -file can be used together.
-if-absent indicates to put a key/value pair only if no value for
the given key is present.
-if-present indicates to put a key/value pair only if a value for
the given key is present.
kv-> get kv -key /Smith/Bob/-/phonenumbers
408 555 5555
kv->
```

## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

### ➤ La commande PUT

- Ajout de **PILOTE** avec le modèle KEY/VALUE. Ici la clé /nom/adr/plnum, une clé à champ multiple est essentielle pour augmenter capacités de recherche. En effet, la valeur étant non structurée, elle ne peut pas servir pour rechercher.

```
put kv -key /Gagarin1/Klouchino1Russie/1 -value
'{"plnum":1,"plnom":"Gagarin1","dnaiss":"09/03/1934","adr":"Klouchino1, Russie",
"tel":"0071122334455", "sal":10000.75}';
put kv -key /Gagarin2/Klouchino2Russie/2 -value
'{"plnum":2,"plnom":"Gagarin2","dnaiss":"09/03/1935","adr":"Klouchino2, Russie",
"tel":"0071122334456", "sal":10050.75}';
put kv -key /Gagarin2/Moscou1Russie/3 -value
'{"plnum":3,"plnom":"Gagarin2","dnaiss":"09/03/1960","adr":"Moscou1, Russie", "tel":"0071122334470",
"sal":20050.75}';
```

## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

### ➤ La commande PUT

- Ajout de **CHECKUP** avec le modèle KEY/VALUE avec pour clé: /nom/adr/plnum/-/cunum/x

**put kv -key /Gagarin2/Klouchino2Russie/2/-/cunum/1 -value**

```
'{"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "cunum":1, "cudate":"12-12-2015",
"curest resultat": "BON"}';
```

**put kv -key /Gagarin2/Klouchino2Russie/2/-/cunum/2 -value**

```
'{"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "cunum":2, "cudate":"11-1-2016",
"curest resultat": "BON MAIS ATTENTION AU THE"}';
```

**put kv -key /Gagarin2/Moscou2Russie/5/-/cunum/1 -value**

```
'{"plnum":5, "plnom":"Gagarin2", "adr":"Moscou2, Russie", "cunum":3, "cudate":"12-12-2015",
"curest resultat": "BON"}';
```

## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

### ➤ La commande PUT

- Ajout de **TRAINING** avec le modèle KEY/VALUE avec pour clé: /nom/adr/plnum/-/trnum/x

```
put kv -key /Gagarin2/Klouchino2Russie/2/-/trnum/1 -value
'{"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "trnum":1, "trdate":"12-12-2015",
"trtypeTraining": "Simulateur", "trDureeEnHeure":5};
```

```
put kv -key /Gagarin2/Klouchino2Russie/2/-/trnum/2 -value
'{"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "trnum":2, "trdate":"11-1-2016",
"trtypeTraining": "Vol reel", "trDureeEnHeure":2};
```

```
put kv -key /Gagarin2/Klouchino2Russie/2/-/trnum/3 -value
'{"plnum":5, "plnom":"Gagarin2", "adr":"Moscou2, Russie", "trnum":3, "trdate":"12-12-2015",
"trtypeTraining": "Simulateur", "trDureeEnHeure":3};
```

# Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

## ➤ La commande GET

- Permet de lire un objet **connaissant sa clé**
- Permet aussi **de lire plusieurs objets connaissant une partie de la clé. D'où l'intérêt d'une clé multi-champs**
- Get kv –key /MajorKey1/ MajorKey2/.../-/MinorKey1/MinorKey2/...

```
oracle@bigdatalite:u01/nosql/kv-ee/ttnosql/hello
File Edit View Search Terminal Help
kv->
kv->
kv->
kv-> help get kv
Usage: get kv [-key <key>] [-json] [-file <output>] [-all] [-keyonly]
          [-valueonly] [-start <prefixString>] [-end <prefixString>]
Performs a simple get operation on the key in the store.
-key indicates the key (prefix) to use. Optional with -all.
-json should be specified if the record is JSON.
-all is specified for iteration starting at the key, or with
an empty key to iterate the entire store.
-start and -end flags can be used for restricting the range used
for iteration.
-keyonly works with -all and restricts information to keys.
-valueonly works with -all and restricts information to values.
-file is used to specify an output file, which is truncated.
kv->
```

## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

### ➤ La commande GET

- Afficher les informations sur **LE PILOTE** ayant clé suivante : /Gagarin2/Klouchino2Russie/2

**get kv -key /Gagarin2/Klouchino2Russie/2**

```
C:\Users\mondi>Set KVHOME=D:\tpbigdata\nosqlOracle\kv-21.2.46
C:\Users\mondi>java -jar %KVHOME%/lib/kvcli.jar -host localhost -port 5000 -store kvstore
kv-> get kv -key /Gagarin2/Klouchino2Russie/2
{"plnum":2,"plnom":"Gagarin2","dnaiss":"09/03/1935","adr":"Klouchino2, Russie",
 "tel":"0071122334456", "sal":10050.75}
kv->
```

# Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

## ➤ La commande GET

- Afficher les informations sur tous les pilotes de nom Gagarin2

```
get kv -key /Gagarin2 -all
```

```
/Gagarin2/Moscou2Russie/5/-/cunum/1
{"plnum":5, "plnom":"Gagarin2", "adr":"Moscou2, Russie", "cunum":3, "cudate":"12-12-2015", "curestatal": "BON"}
/Gagarin2/Klouchino2Russie/2
{"plnum":2,"plnom":"Gagarin2","dnaiss":"09/03/1935","adr":"Klouchino2, Russie", "tel":"0071122334456", "sal":10050.75}
/Gagarin2/Klouchino2Russie/2/-/cunum/1
 {"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "cunum":1, "cudate":"12-12-2015", "curestatal": "BON"}
/Gagarin2/Klouchino2Russie/2/-/cunum/2
 {"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "cunum":2, "cudate":"11-1-2016",
 "curestatal": "BON MAIS ATTENTION AU THE"}
/Gagarin2/Klouchino2Russie/2/-/trnum/1
 {"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "trnum":1, "trdate":"12-12-2015",
 "trtypeTraining": "Simulateur", "trDureeEnHeure":5}
/Gagarin2/Klouchino2Russie/2/-/trnum/2
 {"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "trnum":2, "trdate":"11-1-2016",
 "trtypeTraining": "Vol reel", "trDureeEnHeure":2}
/Gagarin2/Klouchino2Russie/2/-/trnum/3
 {"plnum":5, "plnom":"Gagarin2", "adr":"Moscou2, Russie", "trnum":3, "trdate":"12-12-2015",
 "trtypeTraining": "Simulateur", "trDureeEnHeure":3}
/Gagarin2/Moscou1Russie/3
 {"plnum":3,"plnom":"Gagarin2","dnaiss":"09/03/1960","adr":"Moscou1, Russie", "tel":"0071122334470", "sal":20050.75}
```

## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

### ➤ La commande GET

- Afficher les informations sur les pilotes de nom Gagarin2 habitant Klouchino2Russie

**get kv -key /Gagarin2/Klouchino2Russie -all**

```
/Gagarin2/Klouchino2Russie/2
{"plnum":2,"plnom":"Gagarin2","dnaiiss":"09/03/1935","adr":"Klouchino2, Russie", "tel":"0071122334456", "sal":10050.75}
/Gagarin2/Klouchino2Russie/2/-/cunum/1
{"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "cunum":1, "cudate":"12-12-2015", "curestutat": "BON"}
/Gagarin2/Klouchino2Russie/2/-/cunum/2
{"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "cunum":2, "cudate":"11-1-2016", "curestutat":
"BON MAIS ATTENTION AU THE"}
/Gagarin2/Klouchino2Russie/2/-/trnum/1
{"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "trnum":1, "trdate":"12-12-2015", "trtypeTraining": "Simulateur",
"trDureeEnHeure":5}
/Gagarin2/Klouchino2Russie/2/-/trnum/2
{"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "trnum":2, "trdate":"11-1-2016", "trtypeTraining": "Vol reel",
"trDureeEnHeure":2}
/Gagarin2/Klouchino2Russie/2/-/trnum/3
{"plnum":5, "plnom":"Gagarin2", "adr":"Moscou2, Russie", "trnum":3, "trdate":"12-12-2015", "trtypeTraining": "Simulateur",
"trDureeEnHeure":3}
```

## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

### ➤ La commande GET

- Afficher les informations sur les **checkup** du pilote avec la clé major /Gagarin2/Klouchino2Russie/2

```
get kv -key /Gagarin2/Klouchino2Russie/2/-/cunum -all
```

```
/Gagarin2/Klouchino2Russie/2/-/cunum/1
{"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "cunum":1, "cudate":"12-12-2015",
"cure resultat": "BON"}
/Gagarin2/Klouchino2Russie/2/-/cunum/2
{"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "cunum":2, "cudate":"11-1-2016",
"cure resultat": "BON MAIS ATTENTION AU THE"}
```

## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

### ➤ La commande GET

- Lecture de tous les enregistrements de la base  
Kv-> get kv -all

# Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

## ➤ La commande DELETE

- Permet de **supprimer** un objet connaissant sa clé
- Permet aussi de **supprimer plusieurs objets** connaissant une partie de la clé
- **delete kv -key /MajorKey1/ MajorKey2/.../- /MinorKey1/MinorKey2/...**

```
oracle@bigdatalite:/u01/nosql/kv-ee/tpnosql/hello
File Edit View Search Terminal Help
kv->
kv->
kv-> help delete kv
Usage: delete kv [-key <key>] [-start <prefixString>] [-end <prefixString>] [-all]
      Deletes one or more keys. If -all is specified, deletes all
      keys starting at the specified key. If no key is specified
      delete all keys in the store.
      -start and -end flags can be used for restricting the range used
      for deletion.

kv->
kv->
kv-> delete kv -key /hello
Key deleted: /hello
kv->
kv-> get kv -key /hello
Key not found in store: /hello
kv->
```

## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

### ➤ La commande DELETE

- Supprimer le pilote ayant la clé suivante : /Gagarin2/Moscou1Russie/3

**kv-> delete kv –key /Gagarin2/Moscou1Russie/3**

**Key deleted: /Gagarin2/Moscou1Russie/3**

## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

### ➤ La commande DELETE

- Supprimer les checkup du pilote ayant la major key suivante : /Gagarin2/Klouchino2Russie/2

```
kv-> delete kv -key /Gagarin2/Klouchino2Russie/2/-/cunum -all
```

2 Keys deleted starting at /Gagarin2/Klouchino2Russie/2/-/cunum

## Module M4.2, section 2, partie 1 : Le modèle Key/Value de base, Command Line Interface

### ➤ La commande DELETE

- Supprimer les objets dont la major key (major key partielle) commence par : /Gagarin2

kv-> delete kv –key /Gagarin2 –all

8 Keys deleted starting at /Gagarin2

## Module M4.2, section 2, partie 1 : QUIZ

### ➤ Question 1 : Que fait la commande SHOW TOPOLOGY

- A: Cette commande affiche les tables créées dans la base
- B: Cette commande affiche les indexes créés
- C: Cette commande affiche les dernières commandes d'administration exécutées sur la base
- D: Cette commande affiche les informations sur la structure de la base (nœud, ...)

### ➤ Question 2 : Que fait la commande SHOW TABLES

- A: Cette commande affiche les tables créées dans la base
- B: Cette commande affiche les indexes créés
- C: Cette commande affiche les dernières commandes d'administration exécutées sur la base
- D: Cette commande affiche les informations sur la structure de la base (nœud, ...)

## Module M4.2, section 2, partie 1 : QUIZ

➤ **Question 3 :** Vous avez créé dans la base de données la clé et la valeur suivante

Key : /Smith/bob Valeur: « Informations sur Smith bob »

Cochez les actions qui fonctionnent

- A: put kv –key /Smith/bob –Value "Nouvelle informations sur Smith bob"
- B: put kv –key /Smith/bob –Value "Nouvelle informations sur Smith bob" -if-present
- C: put kv –key /Smith/bob –Value "Nouvelle informations sur Smith bob" -if-absent
- D: put kv –key /Smith/bob –Value "Nouvelle informations sur Smith bob" -all

## Module M4.2, section 2, partie 1 : QUIZ

Considérons les informations

### Clé /Valeur de PILOTE avec pour clé /nom/adr/plnum

Key: /Gagarin1/Klouchino1Russie/1	value: 'InfoGagarin11'
Key: /Gagarin2/Klouchino2Russie/2	value: 'InfoGagarin22'
Key: /Gagarin2/Moscou1Russie/3	value: 'InfoGagarin23'

### Clé/valeur de CHECKUP avec pour clé: /nom/adr/plnum/-/cunum/x

Key: /Gagarin2/Klouchino2Russie/2/-/cunum/1	value: 'InfoGagarin22c1'
Key: /Gagarin2/Klouchino2Russie/2/-/cunum/2	value: 'InfoGagarin22c2'
Key: /Gagarin2/Moscou2Russie/5/-/cunum/3	value: 'InfoGagarin25c3'

### Clé/valeur de TRAINING avec pour clé: /nom/adr/plnum/-/trnum/x

Key: /Gagarin2/Klouchino2Russie/2/-/trnum/1	value: 'InfoGagarin22t1'
Key: /Gagarin2/Klouchino2Russie/2/-/trnum/2	value: 'InfoGagarin22t2'
Key: /Gagarin2/Klouchino2Russie/2/-/trnum/3	value: 'InfoGagarin22t1'

## Module M4.2, section 2, partie 1 : QUIZ

➤ **Question 4 :** au vu des données précédentes cochez le résultat qui est bon en lançant la commande suivante

**Get kv –key /Gagarin2/Klouchino2Russie/2**

- A: 'InfoGagarin22'
- B: /Gagarin2/Klouchino2Russie/2 : 'InfoGagarin22'
- C: /Gagarin2/Klouchino2Russie/2
- D: pas de réponse

➤ **Question 5 :** au vu des données précédentes cochez le résultat qui est bon en lançant la commande suivante

**Get kv –key /Gagarin2**

- A: 'InfoGagarin22'
- B: /Gagarin2/Klouchino2Russie/2 : 'InfoGagarin22'
- C: Tous les enregistrements commençant par Gagarin2
- D: pas de réponse

## Module M4.2, section 2, partie 1 : QUIZ

➤ **Question 6 :** au vu des données précédentes cochez le résultat qui est bon en lançant la commande suivante

**Get kv –key /Gagarin2 -all**

- A: 'InfoGagarin22'
- B: /Gagarin2/Klouchino2Russie/2 : 'InfoGagarin22'
- C: Tous les enregistrements commençant par Gagarin2
- D: pas de réponse

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

# Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

## ➤ Plan

- Conception d'un record pour une BD Oracle NoSql
- Accès au KVSTORE
- Création, modification d'un Enregistrement
- Suppression d'enregistrements
- Lecture des enregistrements
- Un exemple complet de création et lecture d'enregistrements

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Conception d'un record pour une BD

#### Oracle NoSql

- Structure d'enregistrements pour une BD Oracle NoSQL modèle key/value
  - BD sans Schéma
  - Pas de table à priori
  - A interpréter comme une table à 2 colonnes KEY / VALUE

Key	Value					
010101010	...	...	...	...	...	...
010101011	...	...	...	...	...	...
...	...	...	...	...	...	...

Enregistrements

Structure de données opaques

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Conception d'un record pour une BD Oracle NoSql

- **Rôle d'une Clé :** La clé dans une BD Oracle NoSQL permet uniquement d'identifier un enregistrement
  - Il s'agit d'une **clé applicative** (pas de clé système)
  - Elle est toujours **de type Chaîne de caractères (String)**
  - Une **valeur lui est attachée**
  - Elle peut être composée d'une ou plusieurs **clés MAJEURES** et d'une ou plusieurs **clés MINEURES**

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Conception d'un record pour une BD Oracle NoSql

#### ■ Considération à prendre en compte lors de la Conception de la partie VALEUR

- Quel est le **taux de croissance** acceptable pour chaque enregistrement ?
- Quel est la **fréquence d'accès** à l'enregistrement ?
- Faut-il plutôt **beaucoup de petits** enregistrements ou plutôt **un petit nombre** d'enregistrements trop larges ?
- Une valeur ne peut pas dépasser **4 Gigabytes**

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

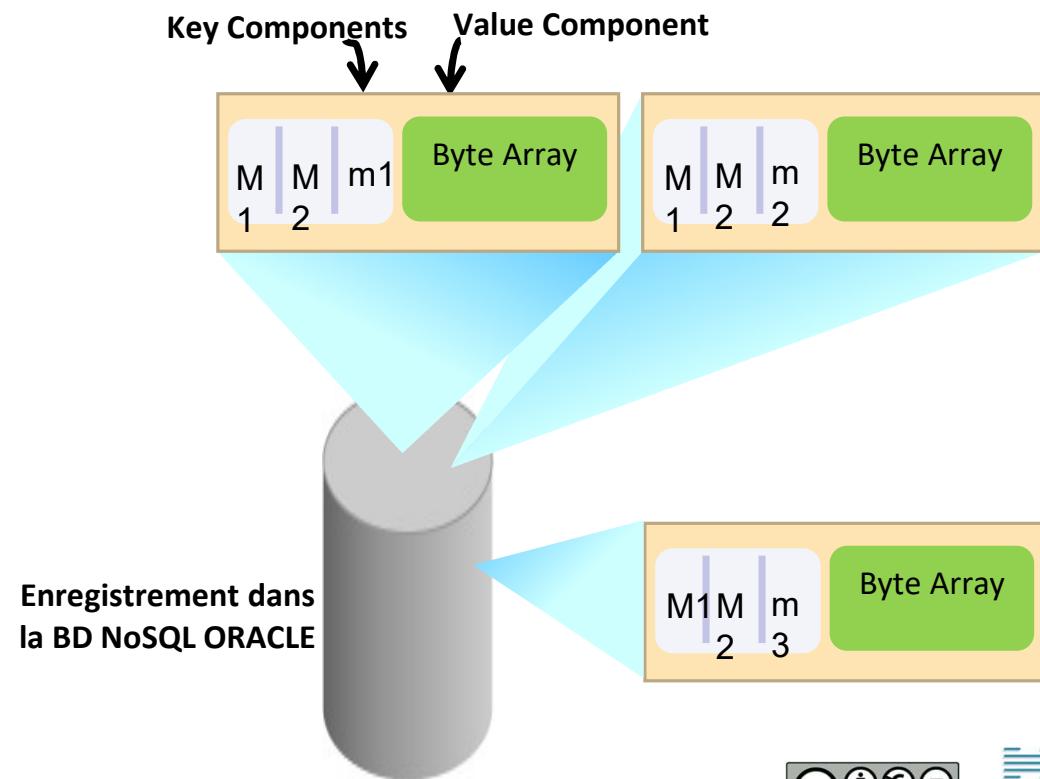
### ➤ Conception d'un record pour une BD Oracle NoSql

- Autres considérations à prendre en compte lors de la conception des clés
  - Quelles sont les **données (Value) à stocker** ?
  - Quelles sont les données qui ont besoins **d'être accédées ensemble** ?
  - Quelles sont les données qui peuvent **être accédées indépendamment** ?
  - Quelle est la **taille** des données (value) ?
  - Quelle est la **fréquence d'accès** aux données ?
  - Combien de partitions y a t-il dans le STORE ?

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Conception d'un record pour une BD Oracle NoSql

- Structure d'un enregistrement : rappel



## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Accès au KVSTORE

- Définition d'un KVStore Handle

- C'est une ressource qui permet de contrôler l'accès à la base de données Oracle NoSQL
- Il est utilisé pour ouvrir un STORE ou fermer un STORE déjà lancé
- Il est absolument nécessaire pour effectuer des opérations dans le STORE

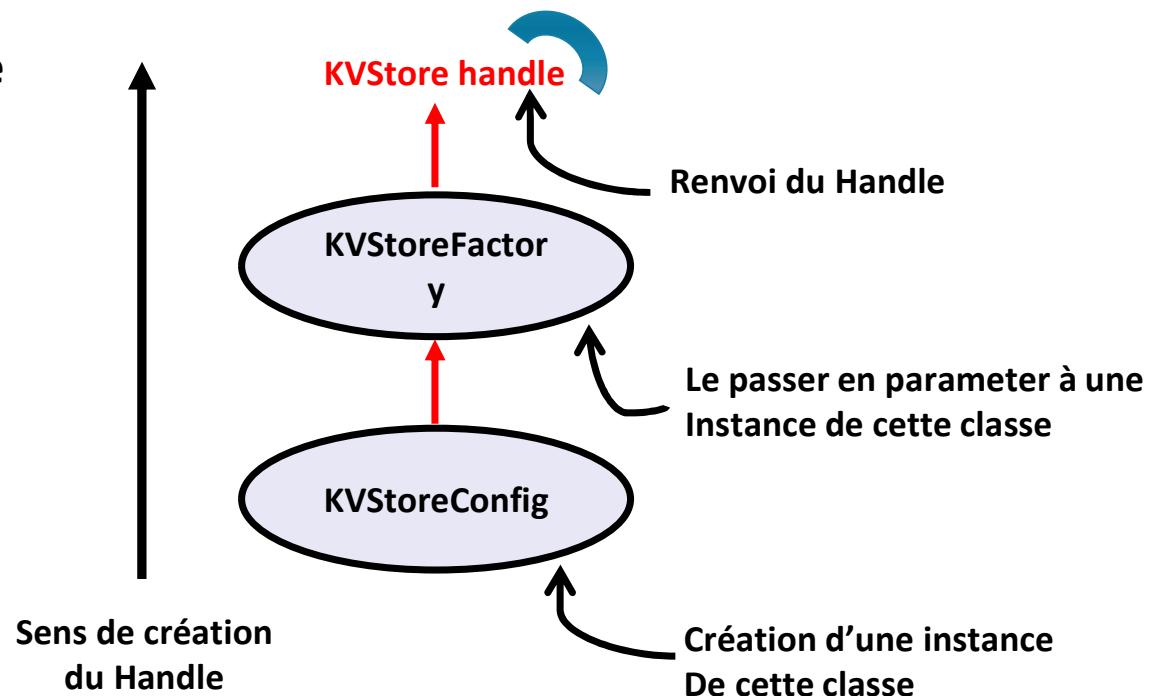
- Pour plus de détails se reporter à la Javadoc de l'API Java

<https://docs.oracle.com/en/database/other-databases/nosql-database/21.2/java-api/oracle/kv/package-summary.html>

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Accès au KVSTORE

- Crédit d'un KVStore Handle
  - Pointeur vers la base NOSQL



## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Accès au KVSTORE

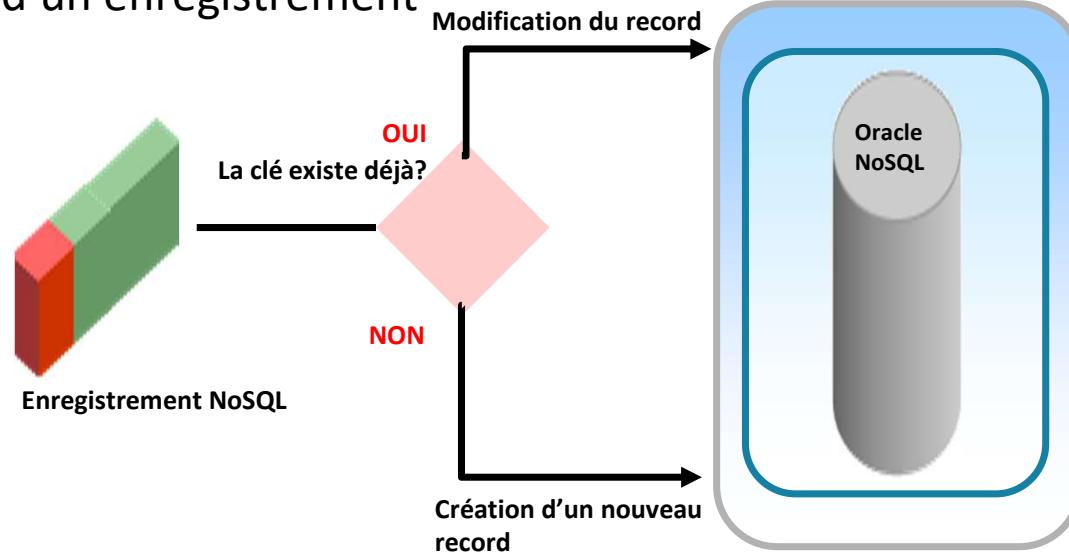
- Utilisation de la classe **KVStoreFactory** : EXAMPLE vers la base “kvstore” qui écoute sur le port 5000

```
KVStoreConfig kconfig = new  
KVStoreConfig("kvstore","localhost:5000");  
  
KVStore kvstore = KVStoreFactory.getStore(kconfig);
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤Création, modification d'enregistrements

- Processus de creation d'un enregistrement



## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤Création, modification d'enregistrements

- Les **méthodes de la classe KVStore** permettent de manipuler des enregistrements

- **put()** ; //permet de créer ou écraser un record existant
- **putIfAbsent()** ; //Créé un enregistrement si la clé n'est déjà associée à unrecord
- **putIfPresent();** //Effectue une mise à jour.
- **putIfVersion();** //Créé uniquement si la version de l'enregistrement existant matche avec les //arguments de version passés avec la fonction.

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤Création, modification d'un Enregistrement

#### ■ Nous souhaitons ajouter dans la base des PILOTE avec leur CHECKUP et leurs TRAINING

- Considérons la **clé et la Valeur du pilote Gagarin2**

**Clé :** /Gagarin2/Klouchino2Russie/2

**Valeur :** '{"plnum":2,"plnom":"Gagarin2","dnaiss":"09/03/1935","adr":"Klouchino2, Russie", "tel":"0071122334456", "sal":10050.75}';

- Considérons la **clé et la Valeur d'un checkup** du pilote Gagarin2

**clé :** /Gagarin2/Klouchino2Russie/2/-/cunum/1

**Valeur :** '{"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "cunum":1, "cudate":"12-12-2015", "curesultat": "BON"}';

- Considérons la **clé et la Valeur d'un training** du pilote Gagarin2

**clé :** /Gagarin2/Klouchino2Russie/2/-/trnum/1 -value

**Valeur :** '{"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "trnum":1, "trdate":"12-12-2015", "trtypeTraining": "Simulateur", "trDureeEnHeure":5}';

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤Création, modification d'un Enregistrement

#### ■ Crédit de la clé MAJORE

- Créer un ArrayList nommé par exemple **majorKeyGagarin2** avec les éléments de type String
- Ajouter les valeurs de la clé MAJORE dans la liste

```
List<String> majorKeyGagarin2 = new ArrayList<String>();  
...  
majorKeyGagarin2.add("Gagarin2");  
majorKeyGagarin2.add("Klouchino2Russie");  
majorKeyGagarin2.add("2");
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤Création, modification d'un Enregistrement

#### ■ Crédit des clés MINEURES

- Affecter les valeurs correspondantes aux clés MINEURES aux variables

```
// Clé mineure pour les checkup: Créer un ArrayList nommé par exemple minorCheckup
```

```
List<String> minorCheckup = new ArrayList<String>();  
minorCheckup.add("cunum");  
minorCheckup.add("1");
```

```
// Clé mineure pour les training: Créer un ArrayList nommé par exemple minorTraining
```

```
List<String> minorTraining = new ArrayList<String>();  
minorTraining.add("trnum");  
minorTraining.add("1");
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤Création, modification d'un Enregistrement

#### ■ Crédit des CLES des enregistrements

- Utiliser pour cela la méthode **createKey** de la classe **Key**

```
// Clé pour identifier le pilote Gagarin2
Key keyGagarin2 = Key.createKey(majorKeyGagarin2);

// Clé pour identifier le checkup du pilote Gagarin2
Key keyCheckupGagarin2 = Key.createKey(majorKeyGagarin2, minorCheckup);

// Clé pour identifier le training du pilote Gagarin2
Key keyTrainingGagarin2 = Key.createKey(majorKeyGagarin2, minorTraining);
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤Création, modification d'un Enregistrement

- Crédit des Valeurs des enregistrements
  - Utiliser pour cela la méthode **createValue** de la classe **Value**

```
// Creation de la Valeur pour le pilote Gagarin2
String gagarin2="{'plnum':2, 'plnom':'Gagarin2', 'dnaiss':'09/03/1935',"
+"adr':'Klouchino2, Russie', 'tel':'0071122334456', 'sal':10050.75}"
Value valueGagarin2 = Value.createValue(gagarin2.getBytes());

// Creation de la Valeur pour le checkup du pilote Gagarin2
String checkupGagarin2="{'plnum':2, 'plnom':'Gagarin2', 'dnaiss':'09/03/1935',"
+"adr':'Klouchino2, Russie', 'cunum':1, 'cudate':'12-12-2015', 'cure resultat':'BON'}";
Value valueGagarin2 = Value.createValue(checkupGagarin2.getBytes());

// Creation de la Valeur pour le training du pilote Gagarin2
String trainingGagarin2="{'plnum':2, 'plnom':'Gagarin2', 'dnaiss':'09/03/1935',"
+"adr':'Klouchino2, Russie', 'trnum':1, 'trdate':'12-12-2015', 'trtypeTraining':"
+"Simulateur', 'trDureeEnHeure':5}}";
Value valueTraingGagarin2 = Value.createValue(trainingGagarin2.getBytes());
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤Création, modification d'un Enregistrement

- Ecriture des enregistrements dans la base de données Oracle NOSQL

```
KVStoreConfig kconfig = new KVStoreConfig("kvstore","localhost:5000");
KVStore kvstore = KVStoreFactory.getStore(kconfig);

// Ecriture des informations sur le pilote Gagarin2 dans la base
kvstore.put(keyGagarin2, valueGagarin2);

//Ecriture des informations sur le checkup du pilote Gagarin2
kvstore.put(keyCheckupGagarin2, checkupGagarin2);

//Ecriture des informations sur le training du pilote Gagarin2
kvstore.put(keyTrainingGagarin2, valueTraingGagarin2);
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Suppression d'enregistrements

- API Java pour la suppression d'enregistrements. Il s'agit de fonctions de la classe **KVStore**
  - delete()
  - multiDelete()
  - deleteIfVersion()

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Suppression d'enregistrements

- Suppression d'un enregistrement unique

```
boolean delete(Key key)
throws DurabilityException, RequestTimeoutException,FaultException;
package hello;
import java.util.ArrayList;
import oracle.kv.Key;
import oracle.kv.Value;
import oracle.kv
public class DeleteRecord {
//Create Handle : myStore
//Create Key : myKey
myStore.delete(myKey);
}
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Suppression d'enregistrements

- Suppression d'enregistrements multiples

```
int multiDelete(Key parentKey, KeyRange subRange, Depth depth)
    Throws DurabilityException, RequestTimeoutException, FaultException;
package hello;
import java.util.ArrayList;
import oracle.kv.Key;
import oracle.kv.Value;
import oracle.kv
public class DeleteRecord {
//Create Handle : myStore
//Create Major Key: myMajorKey
myStore.multiDelete(myMajorKey, null, null);
}
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Lecture des enregistrements

#### ➤ Objectif de la lecture

- Identifier l'API Java utilisée pour lire un enregistrement
- Lire un enregistrement unique depuis la base de données Oracle NoSQL
- Lire plusieurs enregistrements depuis la base de données Oracle NoSQL
- Identifier les exceptions à traiter lors de la lecture des enregistrements

# Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

## ➤ Lecture des enregistrements

- API JAVA pour la lecture des enregistrements méthode de la classe **KVStore**

NOM DES METHODES DE LECTURE	DESCRIPTION DES METHODES DE LECTURE
<code>get(Key key),</code>	Charge un objet connaissant sa clé
<code>multiGet(Key parentKey, KeyRange subRange, Depth depth)</code>	Renvoie un tableau trié des paires clé/valeur descendantes d'une clé majeure entièrement renseignée. Le résultat complet réside en mémoire (risque d'erreur OutOfMemoryError)
<code>multiGetKeys(Key parentKey, KeyRange subRange, Depth depth)</code>	Renvoie un tableau trié des clés descendantes d'une clé majeur entièrement renseignée. (risque d'erreur OutOfMemoryError)
<code>multiGetIterator(Direction direction, int batchSize, Key parentKey, KeyRange s, Depth depth)</code>	Renvoie un itérateur trié des paires clé/valeur descendantes d'une clé majeure entièrement renseignée. A utiliser à la place de multiGet si risque d'erreur OutOfMemoryError
<code>multiGetKeysIterator(Direction d,int batchSize, Key parentKey, KeyRange subRange, Depth depth)</code>	Renvoie un itérateur trié des clés descendantes d'une clé majeur entièrement renseignée. . A utiliser à la place de multiGetKey si risque d'erreur OutOfMemoryError
<code>storeIterator(Direction direction,int batchSize, Key parentKey, KeyRange subRange, Depth depth)</code>	Renvoie un itérateur non trié des paires clé/valeur sur tout le store ou descendantes d'une clé majeure entièrement ou partiellement renseignée
<code>storeKeysIterator(Direction direction,int batchSize, Key parentKey, KeyRange subRange, Depth depth)</code>	Renvoie un itérateur non trié des clés sur tout le store ou descendantes d'une clé majeure entièrement ou partiellement renseignée

# Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

## ➤ Lecture des enregistrements

- Paramètres API JAVA pour la lecture des enregistrements

NOMS DES PARAMETRES	DESCRIPTION DES PARAMETRES
<b>Key</b> parentKey	MajorKey entièrement renseignée (multiGet, multiGetKey, multiGetIterator, multiGetKeyIterator) entièrement ou partiellement renseignée (storeIterator, storeKeyIterator)
<b>KeyRange</b> subRange	KeyRange(String startMinorKey, boolean StartInclusive, String endMinorKey, Boolean endInclusive)
<b>Depth</b> depth	Depth.CHILDREN_ONLY, Depth.DESCENDANTS_ONLY, Depth.PARENT_AND_CHILDREN, Depth.PARENT_AND_DESCENDANTS
<b>Direction</b> direction	Direction.UNORDERED, Direction.FORWARD, Direction.REVERSE
<b>Consistency</b> consistency	<b>Consistency.ABSOLUTE</b> (opération sur le noeud master), <b>Consistency.NONE_REQUIRED</b> (transaction sur le noeud de réplication indépendamment du noeud maître), <b>Consistency.Time</b> (consistance basée sur le temps), <b>Consistency.Version</b> (consistance basée sur la version). Note : Time et Version sont des <b>classes incluses</b> dans <b>Consistency</b>
long timeout	Temps maximum de traitement
TimeUnit timeoutUnit	unité de temps de timeout : TimeUnit.DAYS, TimeUnit.HOURS , TimeUnit.MICROSECONDS , TimeUnit.MILLISECONDS , TimeUnit.MINUTES , TimeUnit.NANOSECONDS, TimeUnit.SECONDS

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Lecture des enregistrements

- Package JAVA utile pour lire des enregistrements
  - Le package à importer pour lire des enregistrements est:

oracle.kv

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Lecture des enregistrements

#### ➤ Lecture d'un enregistrement connaissant sa clé : clé entièrement renseignée (1/2)

```
// Connexion à la base
KVStoreConfig kconfig = new KVStoreConfig("kvstore","localhost:5000");
KVStore kvstore = KVStoreFactory.getStore(kconfig);

// Construction de la clé pour la Lecture d'un checkup : clé entièrement renseignée

// Partie MajorKey du checkup
List<String> majorKeyGagarin2 = new ArrayList<String>();

majorKeyGagarin2.add("Gagarin2");
majorKeyGagarin2.add("Klouchino2Russie");
majorKeyGagarin2.add("2");
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Lecture des enregistrements

#### ➤ Lecture d'un enregistrement connaissant sa clé : clé entièrement renseignée (2/2)

```
// Partie minor key du checkup
List<String> minorCheckup = new ArrayList<String>();

minorCheckup.add("cunum");
minorCheckup.add("1");

// Clé checkup
Key keyCheckupGagarin2 = Key.createKey(majorKeyGagarin2, minorCheckup);

// Lectures des informations sur le checkup du pilote Gagarin2
final ValueVersion valueVersionCheckupGagarin2 = kvstore.get(keyCheckupGagarin2);

System.out.println(keyString + " " + new String(valueVersionCheckupGagarin2.getValue().getValue()));
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Lecture des enregistrements

#### ➤ Lecture des enregistrements ayant une même major key entièrement renseignée (1/2)

```
// Connexion à la base
KVStoreConfig kconfig = new KVStoreConfig("kvstore","localhost:5000");
KVStore kvstore = KVStoreFactory.getStore(kconfig);

// Construction de la clé pour la Lecture d'un pilote : clé majeure entièrement renseignée
List<String> majorKeyGagarin2 = new ArrayList<String>();

majorKeyGagarin2.add("Gagarin2");
majorKeyGagarin2.add("Klouchino2Russie");
majorKeyGagarin2.add("2");

Key keyGagarin2 = Key.createKey(majorKeyGagarin2);
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Lecture des enregistrements

#### ➤ Lecture des enregistrements ayant une même major key entièrement renseignée (2/2)

```
// Lectures des informations sur le pilote Gagarin2
SortedMap<Key, ValueVersion> gagarin2Records = null;

gagarin2Records = kvstore.multiGet(keyGagarin2, null, null);

for (Map.Entry<Key, ValueVersion> entry : gagarin2Records.entrySet()) {
    ValueVersion vv=entry.getValue();
    Value v=vv.getValue();
    System.out.println(entry.getKey() + " " + new String(v.getValue()));
}
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Lecture des enregistrements

#### ➤ Lecture des enregistrements ayant une même major key partiellement renseignée (1/2)

```
// Connexion à la base
KVStoreConfig kconfig = new KVStoreConfig("kvstore","localhost:5000");
KVStore kvstore = KVStoreFactory.getStore(kconfig);

// Construction de la clé pour la Lecture d'un pilote : clé majeure partiellement renseignée
List<String> partialMajorKeyGagarin2 = new ArrayList<String>();

// Seul le nom est renseigné
partialMajorKeyGagarin2.add("Gagarin2");

Key partialKeyGagarin2 = Key.createKey(partialMajorKeyGagarin2);
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Lecture des enregistrements

#### ➤ Lecture des enregistrements ayant une même major key partiellement renseignée (2/2)

```
final Iterator<KeyValueVersion> iter =
    kvstore.storeIterator(Direction.UNORDERED, 0, partialKeyGagarin2,
    null, null);

while (iter.hasNext()) {
    final KeyValueVersion keyValue = iter.next();
    System.out.println(keyValue.getKey() + " " +
    new String(keyValue.getValue().toByteArray()));
}
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Un exemple complet de création et lecture d'enregistrements

package hello;

```
import oracle.kv.KVStore;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;
import oracle.kv.Key;
import oracle.kv.Value;
import oracle.kv.ValueVersion;
/**
 * An extremely simple KVStore client application that writes and reads a single record. It can be used to validate an installation.
 *
 * Before running this example program, start a KVStore instance. The simplest way to do that is to run KV Lite as described in the INSTALL document. Use
 * the KVStore instance name, host and port for running this program, as follows:
 * <pre>
 * java schema.HelloBigDataWorld -store &lt;instance name&gt; \
 *           -host &lt;host name&gt; \
 *           -port &lt;port number&gt;
 * </pre>
 *
 * For all examples the default instance name is kvstore, the default host name
 * is localhost and the default port number is 5000. These defaults match the
 * defaults for running kvlite, so the simplest way to run the examples along
 * with kvlite is to omit all parameters.
 */
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Un exemple complet de création et lecture d'enregistrements

```
public class HelloBigDataWorld {  
    private final KVStore store;  
    /**  
     * Runs the HelloBigDataWorld command line program.  
     */  
    public static void main(String args[]) {  
        try {  
            HelloBigDataWorld example = new HelloBigDataWorld(args);  
            example.runExample();  
        } catch (RuntimeException e) {  
            e.printStackTrace();  
        }  
    }  
    /** * Parses command line args and opens the KVStore. */  
    HelloBigDataWorld(String[] argv) {  
        String storeName = "kvstore";  
        String hostName = "localhost";  
        String hostPort = "5000";  
        final int nArgs = argv.length;  
        int argc = 0;
```

## Module M4.2, section 2, partie 2 : Le modèle Key/Value de base, API Java

### ➤ Un exemple complet de création et lecture d'enregistrements

```
store = KVStoreFactorygetStore
    (new KVStoreConfig(storeName, hostName + ":" + hostPort));
}
/**
 * Performs example operations and closes the KVStore.
 */
void runExample() {

    final String keyString = "Hello";
    final String valueString = "Big Data World!";
    store.put(Key.createKey(keyString), Value.createValue(valueString.getBytes()));
    final ValueVersion valueVersion = store.get(Key.createKey(keyString));
    System.out.println(keyString + " " + new String(valueVersion.getValue().getValue()));
    store.close();
}
}
```

## Module M4.2, section 2, partie 2 : QUIZ

➤ **Question 1 : Accès au KVSTORE** : Quelle est parmi les appels ci-dessous celui qui renvoi un KVStore handle?

- A: KVStoreConfig.getStore()
- B: KVStoreConfig.getStoreName()
- C: KVStoreFactory.getStore()
- D: KVStoreFactory.getStoreName()

➤ **Question 2 : Conception d'un record pour une BD Oracle NoSql** : Si le nombre de clés Majeures et de clés mineures croît, les performances de l'application accroissent également

- A: VRAI
- B: FAUX

## Module M4.2, section 2, partie 2 : QUIZ

➤ **Question 3 : Lecture des enregistrements.** Lors de l'utilisation de l'API Java pour lire un enregistrement unique, qu'est ce qui est renvoyé par la méthode get()

- A: La partie valeur du record
- B: La valeur et la clé du record
- C: Le numéro de version du record
- D: La valeur et le numéro de version du record

➤ **Question 4 : Crédit, modification d'un Enregistrement :** quelles sont les instructions VALIDES pour une BD NoSQL Oracle

- A: Les clés MAJEURES et les clés MINEURES sont stockées dans un seul ArrayList dont les éléments sont des String
- B: Les clés mineures ne peuvent être stockées dans des ArrayList
- C: Le stockage des clés depend totalement de l'application et du développeur
- D: Les clés ne peuvent être que des données de types String

## Module M4.2, section 2, partie 2 : QUIZ

➤ Question 5 : Lecture d'enregistrements: que renvoie la fonction `get(...)` de la classe `KVStore`

- A: Elle renvoie une instance de la classe `SortedMap<Key,ValueVersion>`
- B: Elle renvoie une instance de la classe `Iterator<KeyValueVersion>`
- C: Elle renvoie une instance de la classe `ValueVersion`
- D: Elle renvoie une instance de la classe `SortedSet<Key>`

## Module M4.2, section 2, partie 2 : QUIZ

Considérons les informations écrites dans la BD Oracle NOSQL

**Clé/Valeur de PILOTE avec pour clé : /nom/adr/plnum**

Key: /Gagarin1/Klouchino1Russie/1	value: 'InfoGagarin11'
Key: /Gagarin2/Klouchino2Russie/2	value: 'InfoGagarin22'
Key: /Gagarin2/Moscou1Russie/3	value: 'InfoGagarin23'

**Clé/valeur de CHECKUP avec pour clé: /nom/adr/plnum/-/cunum/x**

Key: /Gagarin2/Klouchino2Russie/2/-/cunum/1	value: 'InfoGagarin22c1'
Key: /Gagarin2/Klouchino2Russie/2/-/cunum/2	value: 'InfoGagarin22c2'
Key: /Gagarin2/Moscou2Russie/5/-/cunum/3	value: 'InfoGagarin25c3'

**Clé/valeur de TRAINING avec pour clé: /nom/adr/plnum/-/trnum/x**

Key: /Gagarin2/Klouchino2Russie/2/-/trnum/1	value: 'InfoGagarin22t1'
Key: /Gagarin2/Klouchino2Russie/2/-/trnum/2	value: 'InfoGagarin22t2'
Key: /Gagarin2/Klouchino2Russie/2/-/trnum/3	value: 'InfoGagarin22t1'

## Module M4.2, section 2, partie 2 : QUIZ

➤ **Question 6 :** Au vu des données précédentes cochez ce que ramène la fonction **get** de **KVStore** en exécutant le code java suivant :

```
Key keyCheckupGagarin2 = Key.createKey(NSArray.asList("Gagarin2", "Klouchino2Russie", "2"), NSArray.asList("cunum", "1"));
kvstore.get(keyCheckupGagarin2); // kvstore handle vers la base
```

- A: 'InfoGagarin2'
- B: /Gagarin2/Klouchino2Russie/2 : 'InfoGagarin2'
- C: /Gagarin2/Klouchino2Russie/2
- D: pas de réponse
- E: Tous les objets de la base

➤ **Question 7 :** au vu des données précédentes cochez ce que ramène la fonction **get** de **KVStore** en exécutant le code java suivant :

```
Key partialGagarin2 = Key.createKey(NSArray.asList("Gagarin2"));
kvstore.get(partialGagarin2); // kvstore handle vers la base
```

- A: 'InfoGagarin2'
- B: Les informations sur le pilote Gagarin2, ses checkups et ses trainings
- C: Tous les enregistrements commençant par Gagarin2
- D: pas de réponse
- E: Tous les objets de la base

## Module M4.2, section 2, partie 2 : QUIZ

➤ **Question 8** : au vu des données précédentes cochez ce que ramène la fonction **multiGet** de **KVStore** en exécutant le code java suivant :

```
Key keyGagarin2 = Key.createKey(new Arrays.asList("Gagarin2", "Klouchino2Russie", "2"));
kvstore.multiGet(keyGagarin2, null, null);
```

- A: 'InfoGagarin2'
- B: Les informations sur le pilote Gagarin2, ses checkups et ses trainings
- C: Tous les enregistrements commençant par Gagarin2
- D: pas de réponse
- E: Tous les objets de la base

➤ **Question 9** : au vu des données précédentes cochez ce que ramène la fonction **multiGet** de **KVStore** en exécutant le code java suivant :

```
Key keyGagarin2 = Key.createKey(new Arrays.asList("Gagarin2"));
kvstore.multiGet(keyGagarin2, null, null);
```

- A: 'InfoGagarin2'
- B: Les informations sur le pilote Gagarin2, ses checkups et ses trainings
- C: Tous les enregistrements commençant par Gagarin2
- D: pas de réponse
- E: Tous les objets de la base

## Module M4.2, section 2, partie 2 : QUIZ

➤ **Question 10 :** au vu des données précédentes cochez ce que ramène la fonction **storeIterator** de **KVStore** en exécutant le code java suivant :

```
Key partialKeyGagarin2 = Key.createKey(Arrays.asList("Gagarin2"));
kvstore.storeIterator(Direction.UNORDERED, 0, partialKeyGagarin2, null, null);
```

- A: 'InfoGagarin2'
- B: Les informations sur le pilote Gagarin2, ses checkups et ses trainings
- C: Tous les enregistrements commençant par Gagarin2
- D: pas de réponse
- E: Tous les objets de la base

➤ **Question 11 :** au vu des données précédentes cochez ce que ramène la fonction **storeIterator** de **KVStore** en exécutant le code java suivant :

```
kvstore.storeIterator(Direction.UNORDERED, 0, null, null, null);
```

- A: 'InfoGagarin2'
- B: Les informations sur le pilote Gagarin2, ses checkups et ses trainings
- C: Tous les enregistrements commençant par Gagarin2
- D: pas de réponse
- E: Tous les objets de la base

## Module M4.2 : Introduction à Oracle NOSQL

➤ Bilan

➤ Exercices

# Module M4.3 : Oracle NoSql et le Modèle Key/Document

G. Mopolo-Moké  
Professeur chargé d'enseignements  
Université Côte d'Azur (UCA)

2022 / 2023

## Module M4.3 : Oracle NoSql et le Modèle Key/Document

### ➤ Plan

- **Module M4.3, section 1, partie 1 : Modèle Key/Document via l'interface ligne de commandes**
- **Module M4.3, section 1, partie 2 : Modèle Key/Document via l'interface ligne de commandes**
- **Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java des tables**
- **Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java des tables**
- **Module M4.3, section 3 : Modèle Key/Document via le langage SQL**

## Module M4.3, section 1, partie 1 : Modèle Key/Document via l'interface ligne de commandes

# Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Plan

- Généralités
- Gestion des Tables et d'indexes via l'interface ligne de commandes
- Les contraintes d'intégrité via l'interface ligne de commandes

# Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Généralités

### ■ Positionnement sur l'utilisation des tables dans Oracle NoSQL

- Oracle NoSQL est à l'origine une base de données qui supporte le modèle **CLE/VALEUR**
- Une API Java **CLE/VALEUR** est disponible. Les objets peuvent être stockées dans la base de données SANS SCHEMA ou avec un schéma AVRO (JSON)
- A l'instar de d'autre BD NOSQL (Cassandra, Hbase, ..), **Oracle va privilégier d'organiser les données sous forme de TABLE** supportant la gestion de tout type de données
- Permet à Oracle de proposer **sa nouvelle stratégie d'accès aux données de toutes sources** (Nosql, Hadoop, SQL) via le langage SQL appelé BIGDATA SQL

## Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Généralités

#### ➤ Les types de données supportés dans la BD Oracle NoSQL pour les colonnes

- **Array** : tableau de valeur d'un même type
- **Binary** : tableau de bytes sans taille prédéfinie
- **Binary(length)** : tableau de bytes avec de longueur fixe
- **Boolean**
- **Double**
- **Enum** : Enumération (tableau de chaînes)
- **Float**
- **Integer**
- **Number** : Nombre de n'importe quelle taille et précision
- **Long**
- **Map** : map dont tous les éléments sont d'un même type
- **Records** : permet de définir une structure imbriquée
- **String**
- **TIMESTAMP(<precision>)**

# Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

signifie qu'aucune fraction de seconde n'est stockée

## ➤ Généralités

### ■ Les types de données supportés dans la BD Oracle NoSQL pour les colonnes

- **TIMESTAMP(<precision>)** : représente une date et l'heure. La précision prend des valeurs de 0 à 9
  - 0 : Signifie qu'aucune fraction de seconde n'est stockée
  - 3: Signifie que l'horodatage stocke des millisecondes
  - 9: Signifie une précision en nanosecondes

**La saisie d'une date et heure doit être un String de format suivant :**

"<yyyy>-<mm>-<dd>[T<HH>:<mm>:<ss>[.<SS>]] «  
 Exemple : "1934-03-09T9:10:10"

**Les fonctions suivantes peuvent être utilisées avec TIMESTAMP :**

year(<timestamp>) : renvoie l'année  
 month(<timestamp>) : renvoie le mois  
 day(<timestamp>) : renvoie le jour  
 hour(<timestamp>) : renvoie l'heure  
 minute(<timestamp>) : renvoie les minutes  
 second(<timestamp>) : renvoie les secondes  
 millisecond(<timestamp>) : renvoie les millisecondes  
 microsecond(<timestamp>) : renvoie les microsecondes  
 extract(<unit> from <expr>) : expr est un timestamp. Unit peut être : year, month, day, hour, minute, second, millisecond, microsecond, nanosecond, week, isoweek

# Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Généralités

### ➤ Syntaxe de création tables et indexes

```
CREATE TABLE [IF NOT EXISTS] table-name (
    field-definition,
    [field-definition-2 ...],
    PRIMARY KEY ([shard(shard-fielname1, [shard-fieldname2...])], other-field-name,
    [other-field-name-2...]),
    [COMMENT "comment string"]
)
```

**Field-definition::=**field-name field-type [constraint1, [constraint2...]]

[...] : facultatif

**Shard:** permet de manipuler (get, delete) toutes les lignes ayant une même shard en une seule opération atomique (rapide). En effet, les lignes ayant une même SHARD KEY sont stockées dans un même nœud de stockage (SHARD)  
**other-field-name-x:** permet de spécifier des champs supplémentaires dans la clé

# Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Généralités

### ▪ Contraintes de nommage des tables, colonnes et indexes

- **Nom d'une table ou d'un indexe** : <= à 32 caractères
- **Nom d'une colonne** : <= 64 caractères
- **Mots clés** : ne peuvent être des noms de tables, d'indexes ou de colonnes
- **Noms de tables, indexes et colonnes** : ne sont pas CASE SENSITIVE
- **Caractères autorisés pour les noms des tables, des indexes et des colonnes** : Alpha numérique et \_ (underscore). Ils doivent obligatoirement commencer par une lettre.

## Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Gestion des Tables et d'indexes via l'interface ligne de commandes

#### ▪ Crédit de tables et indexes via l'interface ligne de commandes

- La commande **Execute** pour créer une table sous la Command Line Interface

kv-> help execute

**Usage: execute <'statement'>**

Executes the specified statement synchronously. The statement must be enclosed in single or double quotes.

**Commandes possibles avec EXECUTE :**

{ALTER, CREATE, DECLARE, DESCRIBE, DROP, GRANT, REVOKE, SELECT, SHOW, UPDATE, NAMESPACE, INSERT, ...}

## Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Gestion des Tables et d'indexes via l'interface ligne de commandes

- Crédit de tables et indexes via l'interface ligne de commandes
  - Exemple de création d'une table via l'interface ligne de commandes

Démarrer la BD NoSQL dans KVLite

```
[oracle@bigdatalite ~]$ java -Xmx256m -Xms256m -jar $KVHOME/lib/kvstore.jar kvlite
```

Opened existing kvlite store with config:

```
-root ./kvroot -store kvstore -host bigdatalite.localdomain -port 5000 -admin 5001
```

Démarrer l'interface ligne de commandes dans KVLite

```
[oracle@bigdatalite ~]$ java -jar $KVHOME/lib/kvstore.jar runadmin -port 5000 -host  
bigdatalite.localdomain
```

kv-> connect store -name kvstore

```
Connected to kvstore at bigdatalite.localdomain:5000.
```

## Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Gestion des Tables et d'indexes via l'interface ligne de commandes

- Crédit de tables et indexes via l'interface ligne de commandes
  - Exemple de création d'une table via l'interface ligne de commandes

```
Kv->execute 'create table pilote(
plnum  INTEGER,
plnom  STRING,
dnaiss STRING,
adr   STRING,
tel   STRING,
sal   FLOAT,
PRIMARY KEY (shard(plnom, adr), plnum))';

Statement completed successfully
```

**NOTE:** Grâce à la SHARD key. Les pilotes ayant une même clé SHARD seront stockées dans une même partition.

## Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Gestion des Tables et d'indexes via l'interface ligne de commandes

- Crédit de tables et indexes via l'interface ligne de commandes
  - Exemple de création d'une table via l'interface ligne de commandes

**NOTA :** **plnum** : numéro pilote est une clé relative. Il s'agit de distinguer deux pilotes ayant un même nom et une même adresse :

Plnom=Gagarin adr=klouchenko plnum=1

Plnom=Gagarin adr=klouchenko plnum=2

Plnom=Tintin adr=Bruxelles plnum=1

Plnom=Tintin adr=Bruxelles plnum=2

# Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Gestion des Tables et d'indexes via l'interface ligne de commandes

- Crédit de tables et indexes via l'interface ligne de commandes
  - Syntaxe générale de création d'un index

`CREATE INDEX [IF NOT EXISTS] index-name ON table-name (field-name)`

Pour les colonnes de type Array

`CREATE INDEX [IF NOT EXISTS] index-name ON table-name (ELEMENTOF(field-name))`

or

`CREATE INDEX [IF NOT EXISTS] index-name ON table-name (KEYOF(field-name),ELEMENTOF(field-name))`

# Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Gestion des Tables et d'indexes via l'interface ligne de commandes

### ■ Crédation de tables et indexes via l'interface ligne de commandes

- Exemple de création d'un index via KV

```
Kv->execute 'CREATE INDEX idx_pilote_plnom ON pilote (plnom)'
```

Statement completed successfully

## Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Gestion des Tables et d'indexes via l'interface ligne de commandes

#### ■ Modification de tables et indexes via l'interface ligne de commandes

- Syntaxe de modification des tables

- ✓ Ajout de colonnes

ALTER TABLE table-name (ADD field-definition)

- ✓ Suppression de colonnes

ALTER TABLE table-name (DROP field-name)

## Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Gestion des Tables et d'indexes via l'interface ligne de commandes

#### ▪ Modification de tables et indexes via l'interface ligne de commandes

- Exemple de modification des tables

- ✓ Ajout de colonnes

kv-> execute 'ALTER TABLE PILOTE (ADD age INTEGER)'

- ✓ Suppression de colonnes

Kv-> execute 'ALTER TABLE PILOTE (DROP age);'

## Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Gestion des Tables et d'indexes via l'interface ligne de commandes

#### ▪ Suppression de tables et indexes via l'interface ligne de commandes

- Syntaxe de suppression de tables (ligne de commandes , API)

Drop table table-name

- Exemple de suppression de tables via l'interface ligne de commandes

Kv->execute 'drop table pilote';

## Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Gestion des Tables et d'indexes via l'interface ligne de commandes

- Suppression de tables et indexes via l'interface ligne de commandes
  - Syntaxe de suppression d'indexes(ligne de commandes , API)

DROP INDEX [IF EXISTS] index-name ON table-name

- Exemple de suppression d'un index via l'interface ligne de commandes

Kv->execute 'drop index idx\_pilote\_plnom on pilote';

# Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Gestion des Tables et d'indexes via l'interface ligne de commandes

### ■ Crédit de tables filles (Nested table) via l'interface ligne de commandes

- Proche de la logique **Majeur Key** et **Minor Key** du modèle KEY/VALUE
- Ceci dans le but de :

Découper un objet de grande taille en plusieurs morceaux afin d'en accélérer l'accès  
De bâtir une stratégie d'accès performante

## Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Gestion des Tables et d'indexes via l'interface ligne de commandes

- Crédit de tables filles (Nested table) via l'interface ligne de commandes

```
CREATE TABLE [IF NOT EXISTS] Parent-table-name.child-table-name (
  field-definition,
  [field-definition-2 ...],
  PRIMARY KEY (child-pk-fieldname1, [child-pk-fieldname2, ...])
  [COMMENT "comment string"]
)
```

**Field-definition::=**field-name field-type [constraint1, [constraint2...]]

[...] : facultatif

**Parent-table-name.child-table-name** : Nom de la table fille

La clé primaire est : (Parent PK + Child PK)

Hérite de la Parent PK

Proche du concept de MinorKey

# Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Gestion des Tables et d'indexes via l'interface ligne de commandes

- Création de tables filles (Nested table) via l'interface ligne de commandes

- Exemple de création d'une table fille via l'interface ligne de commandes

```
Kv->execute 'create table pilote(
plnum INTEGER,
plnom STRING,
dnaiss STRING,
adr STRING,
tel STRING,
sal FLOAT,
PRIMARY KEY (shard(plnom, adr), plnum))';
Statement completed successfully
```

```
Kv->execute 'create table pilote.CV(
cvnum INTEGER,
cylangue STRING,
cvtexte binary,
Cvanimebinary,
PRIMARY KEY (cvnum))';
Statement completed successfully
```

# Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Gestion des Tables et d'indexes via l'interface ligne de commandes

- Création de tables filles (Nested table) via l'interface ligne de commandes
  - Exemple de création d'une table fille via l'interface ligne de commandes

```
Kv->execute 'create table pilote.checkup(  
cunum      INTEGER,  
cudate     STRING,  
curestatal STRING,  
PRIMARY KEY (cunum))';
```

Statement completed successfully

**NOTA :** **cunum** : numéro checkup est clé relative. Il s'agit de numérotter les checkups d'un même PILOTE

Plnom=Gagarin adr=klouchenko plnum=1 cunum=1

Plnom=Gagarin adr=klouchenko plnum=1 cunum=2

Plnom=Tintin adr=Bruxelles plnum=2 cunum=1

Plnom=Tintin adr=Bruxelles plnum=2 cunum=2

# Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Les contraintes d'intégrité via l'interface ligne de commandes

- Le support des contraintes d'intégrité est très sommaire. Les contraintes supportées sont :
  - La contrainte **primary Key**
  - La contrainte **NOT NULL**. Le champ ayant cette contrainte, ne doit pas faire partie de la primary key et **doit avoir une valeur par défaut**.
  - La possibilité de définir une valeur par **DEFAULT**
  - La possibilité de Restreindre un **INTEGER** lors de la définition d'une Primary key

```
create table myId (id integer, primary key(id(3)))
```

Number Bytes	Allowed Integer Values
1	-63 to 63
2	-8191 to 8191
3	-1048575 to 1048575
4	-134217727 to 134217727
5	Any integer value

- **Pas de contraintes CHECK** (introduite avec l'API des tables mais abandonnées depuis). Pour des listes de valeurs Utiliser **ENUM** pour une liste de type chaîne de caractère
- La possibilité de poser un commentaire : **COMMENT**
- **Note** : La gestion des contraintes est encore embryonnaire

# Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Les contraintes d'intégrité via l'interface ligne de commandes

### ▪ Exemple de création de table avec des contraintes

- Exemple de création d'une table via l'interface ligne de commandes

```
Kv->execute 'create table cpilote(  
plnum      INTEGER,  
plnom      STRING,  
dnaiss     STRING DEFAULT "01/01/1900",  
adr        ENUM (PARIS, NICE, LYON),  
tel        STRING default "Pas de tél",  
sal        FLOAT NOT NULL DEFAULT 0,  
PRIMARY KEY (shard(plnum)))';
```

Statement completed successfully

## Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Les contraintes d'intégrité via l'interface ligne de commandes

- Exemple de contraintes

- Insertion de lignes

```
kv-> put table -name cpilote -json '{"plnum":1,"plnom":"Gagarin1","dnaiss":"09/03/1934","adr":"PARIS", "tel":"0071122334455", "sal":10000.75};
```

-- avec valeur par défaut sur SAL

```
kv-> put table -name cpilote -json '{"plnum":2,"plnom":"Gagarin2","dnaiss":"09/03/1935","adr":"NICE", "tel":"0071122334455"};
```

-- Avec valeur nulle sur TEL

```
kv-> put table -name cpilote -json '{"plnum":3,"plnom":"Gagarin3","dnaiss":"09/03/1936","adr":"NICE", "tel":null};
```

-- Avec valeur nulle interdite sur SAL

```
kv-> put table -name cpilote -json '{"plnum":4,"plnom":"Gagarin4","dnaiss":"09/03/1937","adr":"PARIS", "tel":"0071122334455", "sal":null};
```

**Error handling command** put table -name cpilote -json '{"plnum":4,"plnom":"Gagarin4","dnaiss":"09/03/1937","adr":"PARIS", "tel":"0071122334455", "sal":null}': Invalid null value in JSON input for field sal

# Module M4.3, section 1, partie 1 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Les contraintes d'intégrité via l'interface ligne de commandes

### ■ Exemple de contraintes

- Insertion de lignes

-- Avec une adresse MARSEILLE absente de l'énuméré

```
kv-> put table -name cpilote -json '{"plnum":5,"plnom":"Gagarin5","dnaiss":"09/03/1938","adr":"MARSEILLE",  
"tel":"0071122334455", "sal":10000.75};
```

Error handling command put table -name cpilote -json

```
'{"plnum":5,"plnom":"Gagarin5","dnaiss":"09/03/1938","adr":"MARSEILLE", "tel":"0071122334455", "sal":10000.75}':  
Invalid enumeration value 'MARSEILLE', must be in values: [PARIS, NICE, LYON]
```

-- Avec valeur par défaut sur SAL

```
kv-> put table -name cpilote -json '{"plnum":4,"plnom":"Gagarin4","dnaiss":"09/03/1937","adr":"PARIS",  
"tel":"0071122334455"}';
```

## Module M4.3, section 1, partie 1 : QUIZ

➤ **Question 1 :** Nous souhaitons effectuer un certain nombre d'actions dans l'environnement ligne de commande **kv->** avec la commande **EXECUTE**. Cochez ce qui fonctionne

- A: Nous pouvons créer des TOPOLOGY (execute 'create topology ...')
- B: Nous pouvons créer des NAMESPACE (execute 'create namespace ...')
- C: Nous pouvons créer des indexes (execute 'create index ...')
- D: Nous pouvons créer des schémas AVRO (execute 'create schema avro ...')

➤ **Question 2 :** Nous souhaitons effectuer un certain nombre d'actions dans l'environnement ligne de commande **kv->** avec la commande **EXECUTE**. Cochez ce qui fonctionne

- A: Nous pouvons insérer des lignes dans une table (execute 'insert into ...')
- B: Nous pouvons consulter des tables (execute 'select ...')
- C: Nous pouvons supprimer des tables (execute 'drop table ...')
- D: La réponse en A n'est pas possible

## Module M4.3, section 1, partie 1 : QUIZ

➤ **Question 3 :** Lors de la création d'une table, cochez les types éligibles pour déclarer ses colonnes

- A: VARCHAR2
- B: CHAR
- C: STRING
- D: ARRAY
- E: RECORD
- F: TIMESTAMP
- G: INT

➤ **Question 4 :** Nous souhaitons créer une table et lui associer une clé primaire. Cochez ce qui est juste

- A: CREATE TABLE nomTable ( c1 typeDeC1 **primary key**, c2 typeDeC2, ...)
- B: CREATE TABLE nomTable ( c1 typeDeC1, c2 typeDeC2, ..., **primary key(c1)**)
- C: La primary key n'est pas obligatoire comme dans les SGBDR. On peut faire: CREATE TABLE nomTable ( c1 typeDeC1, c2 typeDeC2, ...)

## Module M4.3, section 1, partie 1 : QUIZ

➤ **Question 5 :** Nous souhaitons créer une table et lui associer une clé primaire comme suit :

```
execute 'create table pilote(plnum  INTEGER, plnom STRING, adr  STRING, ...  
PRIMARY KEY (shard(plnom, adr), plnum))';
```

Supposons une BD Oracle NOSQL partitionnée. Cochez ce qui est juste

- A: Le mot **SHARD** sert à répartir les données sur plusieurs partitions
- B: Les pilotes ayant un même nom et une même adresse résident dans une même partition
- C: Les requêtes faisant des restrictions sur la clé **SHARD** sont les plus rapides
- D: Le mot clé **SHARD** indique la clé de partitionnement

➤ **Question 6 :** Nous souhaitons créer une table, cochez les contraintes d'intégrité pouvant être posées:

- A: contrainte Foreign key
- B: contrainte Primary Key
- C: Contraintes CHECK
- D: contrainte ENUM
- E: contrainte DEFAULT

## Module M4.3, section 1, partie 1 : QUIZ

➤ **Question 7 :** Nous souhaitons créer une table fille, appelez **PILOTE.ENFANT**. Cochez l'explication qui décrit le mieux la notion de table fille (**create table PILOTE.ENFANT ...**):

- A: C'est une forme de mise en œuvre du lien d'héritage
- B: C'est une une forme de mise en œuvre du lien de de composition
- C: C'est une une forme de mise en œuvre des clés étrangères sans contrainte FOREIGN KEY
- D: C'est une une forme de mise en œuvre des clés étrangères avec contrainte FOREIGN KEY
- E: Toutes les affirmations précédentes sont fausses

# Module M4.3, section 1, partie 2 : Modèle Key/Document via l'interface ligne de commandes

# Module M4.3, section 1, partie 2 : Modèle Key/Document via l'interface ligne de commandes

## ➤ Plan

- Insertion, suppression de lignes via l'interface ligne de commandes
- Lecture de lignes via l'interface ligne de commandes
- Les namespaces

## Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Insertion, suppression de lignes via l'interface ligne de commandes

#### ▪ Insertion des lignes via l'interface ligne de commandes

- Syntaxe

kv-> **help put table**

**Usage:**

**put table -name <name> [-if-absent | -if-present] [-json <string>] [-file <file>][-exact] [-update]**

- Put a row into the named table. The table name is a dot-separated name with the format **tableName[.childTableName]+**.
- **-if-absent** indicates to put a row only if the row does not exist.
- **-if-present** indicates to put a row only if the row already exists.
- **-json** indicates that the value is a JSON string.
- **-file** can be used to load JSON strings from a file.
- **-exact** indicates that the input json string or file must contain values for all columns in the table, and cannot contain extraneous fields.
- **-update** can be used to partially update the existing record.

**Note :** Une alternative à **PUT TABLE** est d'utiliser SQL : **EXECUTE "INSERT INTO nomTable ... "**

## Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Insertion, suppression de lignes via l'interface ligne de commandes

- Insertion de lignes via l'interface ligne de commandes
  - Exemple de tables pour les insertions

```
execute 'create table pilote2(
plnum INTEGER ,
plnom STRING,
dnaiss STRING,
adr STRING,
tel STRING,
sal FLOAT,
PRIMARY KEY (shard(plnom, adr), plnum))';

execute 'create table pilote2.checkup(
cunum      INTEGER,
cudate     STRING,
cure resultat STRING,
PRIMARY KEY (cunum))';
```

# Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Insertion, suppression de lignes via l'interface ligne de commandes

### ■ Insertion de lignes via l'interface ligne de commandes

- Exemple

```
put table -name pilote2 -json
```

```
{"plnum":1,"plnom":"Gagarin1","dnaiss":"09/03/1934","adr":"Klouchino1, Russie", "tel":"0071122334455", "sal":10000.75};
```

```
put table -name pilote2 -json
```

```
{"plnum":2,"plnom":"Gagarin2","dnaiss":"09/03/1935","adr":"Klouchino2, Russie", "tel":"0071122334456", "sal":10050.75};
```

```
put table -name pilote2 -json
```

```
{"plnum":3,"plnom":"Gagarin2","dnaiss":"09/03/1960","adr":"Moscou1, Russie", "tel":"0071122334470", "sal":20050.75};
```

```
put table -name pilote2 -json
```

```
{"plnum":4,"plnom":"Gagarin3","dnaiss":"09/03/1935","adr":"Klouchino3, Russie", "tel":"00711223344606", "sal":11050.75};
```

## Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Insertion, suppression de lignes via l'interface ligne de commandes

- Insertion de lignes dans des tables filles via l'interface ligne de commandes

- Exemple

```
put table -name pilote2.checkup -json
```

```
{"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "cunum":1, "cudate":"12-12-2015", "curesultat":  
"BON"};
```

```
put table -name pilote2.checkup -json
```

```
{"plnum":2,"plnom":"Gagarin2","adr":"Klouchino2, Russie", "cunum":2, "cudate":"11-1-2016", "curesultat":  
"BON MAIS ATTENTION AU THE"};
```

```
put table -name pilote2.checkup -if-absent -json '{"plnum":5, "plnom":"Gagarin2", "adr":"Moscou2, Russie",  
"cunum":1, "cudate":"12-12-2015", "curesultat": "BON"}';
```

## Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Insertion, suppression de lignes via l'interface ligne de commandes

#### ▪ Suppression de lignes via l'interface ligne de commandes

- Syntaxe

kv-> Help delete table

Usage:

**delete table -name <name>**

[**-field <name> -value <value>**] [**-field <name> [-start <value>] [-end <value>]**] [**-ancestor <name>**] [**-child <name>**]+  
 [**-json <string>**] [**-delete-all**]

Deletes one or more rows from the named table. The table name is a dot-separated name with the format tableName[.childTableName]+.

**-field and -value** pairs are used to specify a primary key to use for the deletion.

**-field, -start and -end flags** can be used to specify a range of keys to be deleted.

**-ancestor and -child flags** can be used to delete rows from specified ancestor and/or descendant tables as well as the target table.

**-json** indicates that the key field values are in JSON format.

**-delete-all** is used to delete all rows in a table.

**Note :** Une alternative à **DELETE TABLE** est d'utiliser SQL : **EXECUTE « DELETE FROM nomTable ... »**

## Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Insertion, suppression de lignes via l'interface ligne de commandes

- Suppression de lignes via l'interface ligne de commandes
  - Exemple 1: suppression d'une ligne avec la clé entièrement renseignée

```
kv-> delete table -name pilote2 -field plnom -value "Gagarin" -field adr -value "Klouchino, Russie" -field plnum -value 1
```

1 row deleted.

**Note :** Pour supprimer une ligne et une seule, il faut s'appuyer sur la clé primaire entièrement renseignée.

## Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Insertion, suppression de lignes via l'interface ligne de commandes

#### ■ Suppression de lignes via l'interface ligne de commandes

- **Exemple 2:** suppression de plusieurs lignes avec la clé primaire partiellement renseignée

```
kv-> delete table -name pilote2 -field plnom -value "Gagarin2"
```

2 rows deleted.

**Note :** Pour supprimer plusieurs lignes partageant une partie de la clé il faut renseigné juste la partie de la clé concernée.

# Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Lecture de lignes via l'interface ligne de commandes

### ■ Syntaxe

Kv->Help get table

#### Note :

Une alternative à **GET TABLE** est d'utiliser SQL : **EXECUTE "SELECT ... FROM nomTable ... "**

```
kv-> help get table
Usage: get table -name <name> [-index <name>]
      [-field <name> -value <value>]+
      [-field <name> [-start <value>] [-end <value>]]
      [-ancestor <name>]+ [-child <name>]+
      [-json <string>] [-file <output>] [-keyonly]
      [-pretty] [-report-size]
Performs a get operation to retrieve one or more rows from a named table.
The table name is a dot-separated name with the format
tableName[.childTableName]+.
-field and -value pairs are used to specify fields of the
primary key or index key used for the operation. If no fields are
specified an iteration of the entire table or index is performed
-field,-start and -end flags can be used to define a value range for
the last field specified.
-ancestor and -child flags can be used to return results from
specified ancestor and/or descendant tables as well as the target
table.
-json indicates that the key field values are in JSON format.
-file is used to specify an output file, which is truncated.
-keyonly is used to restrict information to keys only.
-pretty is used for a nicely formatted JSON string with indentation
and carriage returns.
-report-size is used to show key and data size information for primary
keys, data values, and index keys for matching records. When
-report-size is specified no data is displayed.
```

## Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Lecture de lignes via l'interface ligne de commandes

#### ■ Lecture d'une ligne connaissant sa clé primaire complète

```
Kv-> get table -name pilote2 -field plnom -value "Icare" -field plnum -value 4 -field adr -value "Olympe,  
Grèce"
```

```
{"plnum":4,"plnom":"Icare","dnaiss":"13/02/1000","adr":"Olympe,  
Grèce","tel":"00300623344577","sal":8500.5}
```

## Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Lecture de lignes via l'interface ligne de commandes

#### ■ Lecture de plusieurs lignes connaissant une partie de la clé

- Plnom, plnum et adr forment la clé primaire complète
- Plnom est une partie de clé primaire (racine)

Kv-> **get table -name pilote2 -field plnom -value "Erzulie"**

```
{"plnum":6,"plnom":"Erzulie","dnaiss":"13/02/1804", "adr":"Port-au-Prince, Haïti", "tel":"005090623344557", "sal":13000.5}  
 {"plnum":7,"plnom":"Erzulie","dnaiss":"14/04/1804", "adr":"Les Cayes, Haïti", "tel":"0050906233445607", "sal":14000.5}
```

## Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Lecture de lignes via l'interface ligne de commandes

- Lecture de toutes les lignes d'une table
  - Clé primaire non renseignée

Kv-> **get table -name pilote2**

```
{"plnum":1,"plnom":"Gagarin1","dnaiss":"09/03/1934","adr":"Klouchino1,  
Russie","tel":"0071122334455","sal":10000.75}  
{"plnum":2,"plnom":"Gagarin2","dnaiss":"09/03/1935","adr":"Klouchino2,  
Russie","tel":"0071122334456","sal":10050.75}  
{"plnum":3,"plnom":"Gagarin2","dnaiss":"09/03/1960","adr":"Moscou1,  
Russie","tel":"0071122334470","sal":20050.75}  
{"plnum":4,"plnom":"Gagarin3","dnaiss":"09/03/1935","adr":"Klouchino3,  
Russie","tel":"00711223344606","sal":11050.75}
```

4 rows returned

## Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Lecture de lignes via l'interface ligne de commandes

#### ■ Lecture de lignes d'une table fille et/ou mère

- Clé primaire partiellement renseignée

Renvoie tous les checkup des pilotes de nom Gagarin2

```
kv-> get table -name pilote2.checkup -field plnom -value "Gagarin2"
```

```
{"cunum":1,"cudate":"12-12-2015","curesultat":"BON","plnum":5,"adr":"Moscou2, Russie","plnom":"Gagarin2"}  
{"cunum":1,"cudate":"12-12-2015","curesultat":"BON","plnum":2,"adr":"Klouchino2, Russie","plnom":"Gagarin2"}  
{"cunum":2,"cudate":"11-1-2016","curesultat":"BON MAIS ATTENTION AU THE","plnum":2,"adr":"Klouchino2,  
Russie","plnom":"Gagarin2"}
```

3 rows returned

# Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Lecture de lignes via l'interface ligne de commandes

### ■ Lecture de lignes d'une table fille et/ou mère

#### • Clé primaire partiellement renseignée

Renvoie tous les checkup des pilotes de nom Gagarin2 pour ceux qui ont un ancêtre, ce dernier est aussi renvoyé.

**kv-> get table -name pilote2.checkup -field plnom -value "Gagarin2" -ancestor pilote2**

```
{"cunum":1,"cudate":"12-12-2015","curesultat":"BON","plnum":5,"adr":"Moscou2, Russie","plnom":"Gagarin2"}  
{"plnum":2,"plnom":"Gagarin2","dnaiiss":"09/03/1935","adr":"Klouchino2, Russie","tel":"0071122334456","sal":10050.75}  
{"cunum":1,"cudate":"12-12-2015","curesultat":"BON","plnum":2,"adr":"Klouchino2, Russie","plnom":"Gagarin2"}  
{"cunum":2,"cudate":"11-1-2016","curesultat":"BON MAIS ATTENTION AU THE","plnum":2,"adr":"Klouchino2, Russie","plnom":"Gagarin2"}
```

4 rows returned

## Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Lecture de lignes via l'interface ligne de commandes

- Lecture de lignes d'une table fille et/ou mère

- Clé primaire **partiellement** renseignée

Renvoie tous les pilotes ayant de nom Gagarin2. Pour ceux qui ont des checkup ces derniers sont aussi retournés. Renvois les checkup de nom Gagarin2 qui ne sont liés à un pilote.

```
kv-> get table -name pilote2 -field plnom -value "Gagarin2" -child pilote2.checkup
```

```
{"plnum":3,"plnom":"Gagarin2","dnaiss":"09/03/1960","adr":"Moscou1, Russie","tel":"0071122334470","sal":20050.75}
{"cunum":1,"cudate":"12-12-2015","curesttat":"BON","plnum":5,"adr":"Moscou2, Russie","plnom":"Gagarin2"}
{"plnum":2,"plnom":"Gagarin2","dnaiss":"09/03/1935","adr":"Klouchino2, Russie","tel":"0071122334456","sal":10050.75}
{"cunum":1,"cudate":"12-12-2015","curesttat":"BON","plnum":2,"adr":"Klouchino2, Russie","plnom":"Gagarin2"}
 {"cunum":2,"cudate":"11-1-2016","curesttat":"BON MAIS ATTENTION AU THE","plnum":2,"adr":"Klouchino2,
 Russie","plnom":"Gagarin2"}
```

5 rows returned

## Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

### ➤ Lecture de lignes via l'interface ligne de commandes

- Lecture de lignes d'une table fille et/ou mère

- Clé primaire partiellement renseignée

```
kv-> get table -name pilote2
```

```
{"plnum":1,"plnom":"Gagarin1","dnaiss":"09/03/1934","adr":"Klouchino1, Russie","tel":"0071122334455","sal":10000.75}
 {"plnum":3,"plnom":"Gagarin2","dnaiss":"09/03/1960","adr":"Moscou1, Russie","tel":"0071122334470","sal":20050.75}
 {"plnum":2,"plnom":"Gagarin2","dnaiss":"09/03/1935","adr":"Klouchino2, Russie","tel":"0071122334456","sal":10050.75}
 {"plnum":4,"plnom":"Gagarin3","dnaiss":"09/03/1935","adr":"Klouchino3, Russie","tel":"00711223344606","sal":11050.75}
```

4 rows returned

-- il n'y pas le **pilote nr 5**

# Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Les namespaces

- Disponible à partir de la **version 19c** d'Oracle NOSQL
- Il est maintenant possible de créer des espaces de nom
- Cela permet d'avoir dans la même base de données 2 tables ayant le même nom
- Un espace de nom par défaut existe. Il s'appelle **sysdefault**
- **Création d'un espace de nom**

**CREATE NAMESPACE** nomEspaceDeNom

Exemple : **CREATE NAMESPACE** airbase

- **Changement** d'un espace de nom  
**NAMESPACE** sysdefault

# Module M4.3, section 1, partie 2 : Modèle Key/ Document via l'interface ligne de commandes

## ➤ Les namespaces

### ■ Avec Oracle NoSql version 19c

- Crédit d'une table dans un espace de nom

Avec changement de namespace

Kv-> Namespace **airbase**;

Kv-> Execute 'create table **pilote1**(plnum integer, plnom string, Primary key (plnum))';

Sans changement de namespace

Kv-> Execute 'create table **airbase:pilote2**(plnum integer, plnom string, Primary key (plnum))';

## Module M4.3, section 1, partie 2 : QUIZ

### Considérons les tables suivantes

Create table **MAISON**(manom string, manumero integer, maadresse string, **PRIMARY KEY**(manom , manumero));

Create table **MAISON.PORTES**(ponumero integer, pohauteur integer, polargeur integer , **PRIMARY KEY**(ponumero));

Create table **MAISON.FENETRES**(fenumero integer, fehauteur integer, felargeur integer , **PRIMARY KEY**(fenumero));

➤ **Question 1 :** Nous souhaitons ajouter des lignes dans la table **MAISON** dans l'interface ligne de commande **kv->**. Cochez ce qui fonctionne

- A: put table -name maison -json '{"manom":"Grand chateau", "manumero":1, "maadresse":"28, avenue de Valrose 06000 Nice"}'
- B: put table -name maison -json '{"manom":"Petit chateau", "manumero":2}'
- C: put table -name maison -json '{"maadresse":"66 rue des mimosas 06000 NICE"}'
- D: put table -name maison -json '{"maadresse":"67 rue des mimosas 06000 NICE", "manumero":3, "manom":"Grand chateau"}'
- E: put table -name maison -json '{"manom" :"Vaux-le-Vicomte", "manumero":4"}'
- F: put table -name maison -json '{"manom":"Grand chateau", "manumero":6, "maadresse":"28 BIS, avenue de Valrose 06000 Nice"}'

➤ **Question 2 :** Nous souhaitons ajouter des lignes dans la table fille **MAISON.PORTES** dans l'interface ligne de commande **kv->**. Cochez ce qui fonctionne

- A: put table -name maison.portes -json '{"**manom**":"Grand chateau", "manumero":1, "ponumero":1, "pohauteur":2, "polargeur":2 }'
- B: put table -name maison.portes -json '{"**manom**":"Petit chateau", "**manumero:2, "ponumero":1, "pohauteur":2, "polargeur":2 }'**
- C: put table -name maison.portes -json '{"ponumero":1, "pohauteur":2, "polargeur":2 }'
- D: put table -name maison.portes -json '{"**manumero:3, "ponumero":1, "pohauteur":2, "polargeur":2, "**manom**":"Grand chateau" }'**
- E: put table -name maison.portes -json '{"**manom** ":"Vaux-le-Vicomte ", "**manumero**":4", "ponumero":1, "pohauteur":2, "polargeur":2 }'
- F: put table -name maison.portes -json '{"manom":"Grand chateau", "manumero":5, "ponumero":1, "pohauteur":2, "polargeur":2 }'

## Module M4.3, section 1, partie 2 : QUIZ

➤ **Question 3 :** Cochez ce que fait la requête suivante

**get table -name maison.portes -field manom -value "Grand chateau" -ancestor maison**

- A: Cette requête renvoie les informations sur les portes des maisons de nom : Grand château
- B: Il s'agit d'une **jointure interne**, elle renvoie les infos sur les maisons «Grand Château» et les infos sur les portes de ces maisons
- C: Il s'agit d'une **jointure externe**, elle renvoie les infos sur les maisons «Grand Château» et les infos sur les portes de ces maisons. Mais aussi les infos sur les portes qui ne sont liées à aucune maison de nom «Grand Château»
- D: Il s'agit d'une **jointure full**, elle renvoie les infos sur les maisons «Grand Château» et les infos sur les portes de ces maisons. Mais aussi les infos sur les portes qui ne sont liées à aucune maison de nom «Grand Château» et aussi des informations sur les maisons de nom «Grand Château» qui n'ont pas encore de portes

➤ **Question 4 :** Cochez ce que fait la requête suivante

**get table -name maison -field manom -value "Grand chateau" -child maison.portes**

- A: Cette requête renvoie les informations sur les portes des maisons de nom : Grand château
- B: Il s'agit d'une **jointure interne**, elle renvoie les infos sur les maisons «Grand Château» et les infos sur les portes de ces maisons
- C: Il s'agit d'une **jointure externe**, elle renvoie les infos sur les maisons «Grand Château» et les infos sur les portes de ces maisons. Mais aussi les infos sur les portes qui ne sont liées à aucune maison de nom «Grand Château»
- D: Il s'agit d'une **jointure full**, elle renvoie les infos sur les maisons «Grand Château» et les infos sur les portes de ces maisons. Mais aussi les infos sur les portes qui ne sont liées à aucune maison de nom «Grand Château» et aussi des informations sur les maisons de nom «Grand Château» qui n'ont pas encore de portes

## Module M4.3, section 1, partie 2 : QUIZ

➤ Question 5 : Cochez ce que fait la requête suivante

`delete table -name maison -field manom -value "Grand chateau"`

- A: Cette requête permet de supprimer toutes les maisons de nom «Grand château» ainsi que leurs portes
- B: Cette requête permet de supprimer toutes les maisons de nom «Grand château». Si elles avaient des portes, ces dernières ne sont pas concernées par la suppression
- C: Cette requête supprime la première maison de nom «Grand château» ainsi que ses portes
- D: Cette requête supprime toutes les portes de nom «Grand château» et pour celles qui sont liées à des maisons, ces maisons sont aussi supprimées
- E: Cette requête supprime toutes les maisons de nom «Grand château»
- F: Cette requête supprime toutes les maisons de nom «Grand château» et supprime toutes les portes de nom «Grand château» y compris celles qui ne sont pas liées à des maisons
- G: Cette requête supprime toutes les maisons
- H: G: Cette requête supprime toutes les maisons et leurs portes

➤ Question 6 : Cochez ce que fait la requête suivante

`delete table -name maison -field manom -value "Grand chateau" -child maison.portes`

- A: Cette requête permet de supprimer toutes les maisons de nom «Grand château» ainsi que leurs portes
- B: Cette requête permet de supprimer toutes les maisons de nom «Grand château». Si elles avaient des portes, ces dernières ne sont pas concernées par la suppression
- C: Cette requête supprime la première maison de nom «Grand château» ainsi que ses portes
- D: Cette requête supprime toutes les portes de nom «Grand château» et pour celles qui sont liées à des maisons, ces maisons sont aussi supprimées
- E: Cette requête supprime toutes les maisons de nom «Grand château» et pour celles qui sont liées à des portes, ces maisons sont aussi supprimées
- F: Cette requête supprime toutes les maisons de nom «Grand château» et supprime toutes les portes de nom «Grand château» y compris celles qui ne sont pas liées à des maisons
- G: Cette requête supprime toutes les maisons
- H: G: Cette requête supprime toutes les maisons et leurs portes

## Module M4.3, section 1, partie 2 : QUIZ

➤ **Question 7 :** Cochez ce que fait la requête suivante

**delete table -name maison -delete-all**

- A: Cette requête permet de supprimer toutes les maisons de nom «Grand château» ainsi que leurs portes
- B: Cette requête permet de supprimer toutes les maisons de nom «Grand château». Si elles avaient des portes, ces dernières ne sont pas concernées par la suppression
- C: Cette requête supprime la première maison de nom «Grand château» ainsi que ses portes
- D: Cette requête supprime toutes les portes de nom «Grand château» et pour celles qui sont liées à des maisons, ces maisons sont aussi supprimées
- E: Cette requête supprime toutes les maisons de nom «Grand château» et pour celles qui sont liées à des portes, ces maisons sont aussi supprimées
- F: Cette requête supprime toutes les maisons de nom «Grand château» et supprime toutes les portes de nom «Grand château» y compris celles qui ne sont pas liées à des maisons
- G: Cette requête supprime toutes les maisons
- H: G: Cette requête supprime toutes les maisons et leurs portes

# Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

# Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

## ➤ Plan

- Tables et Indexes : Création de tables et indexes via l'API Java par étapes
- Tables et Indexes : Classes de l'API Java utiles
- Tables et Indexes : Modification de tables et indexes via l'API Java
- Tables et Indexes : Suppression de tables et indexes via l'API Java
- Mise à jour : Les principales fonctions d'insertion et suppression de lignes
- Mise à jour : Les classes à importer
- Mise à jour : Insertion des lignes via l'API JAVA
- Mise à jour : Suppression des lignes via l'interface ligne de commandes

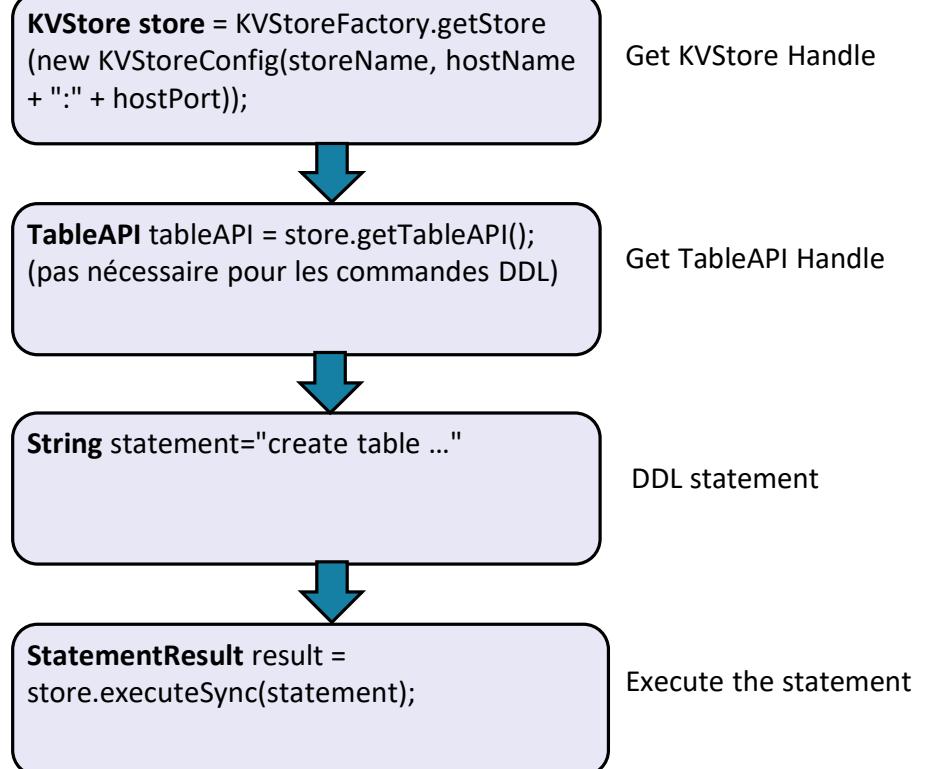
# Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

## ➤ Tables et Indexes : Classes de l'API Java à importer

```
import oracle.kv.KVStore;
import java.util.List;
import java.util.Iterator;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;
import oracle.kv.FaultException;
import oracle.kv.StatementResult;
import oracle.kv.table.TableAPI;
import oracle.kv.table.Table;
import oracle.kv.table.Row;
import oracle.kv.table.PrimaryKey;
import oracle.kv.ConsistencyException;
import oracle.kv.RequestTimeoutException;
import java.lang.Integer;
import oracle.kv.table.TableIterator;
```

## Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

➤ **Tables et Indexes** : Création de tables et indexes via l'API Java par étapes



# Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

## ➤ Tables et Indexes : Crédation de tables et indexes via l'API Java

- Get KVStore Handle

```
import oracle.kv.KVStore;
import java.util.List;
import java.util.Iterator;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;
import oracle.kv.FaultException;
import oracle.kv.StatementResult;
import oracle.kv.table.TableAPI;
import oracle.kv.table.Table;
import oracle.kv.table.Row;
import oracle.kv.table.PrimaryKey;
import oracle.kv.ConsistencyException;
import oracle.kv.RequestTimeoutException;
import java.lang.Integer;
import oracle.kv.table.TableIterator;
```

```
/**  
 * Cette classe fournit les méthodes nécessaires pour gérer les pilotes. Il s'agit des fonctions suivantes:  
 * createTablePilote : créé la table pilote  
 * alterTablePiloteAddColumnAge : ajouter une colonne dans la table pilote  
 * dropColumnAgeFromPiloteTable: suppression de la colonne age  
 * dropTablePilote : supprime la table pilote  
 * createIndexOnPilote : crée un index sur la colonne plnom de la table pilote  
 * dropIndexOnPilote : supprime l'index créé précédemment  
 * insertPiloteRows : Insère de nouvelles lignes dans la table pilote  
 * deletePiloteRow : supprime le pilote inséré connaissant le clé primaire  
 * multiDeletePiloteRows: supprime les pilotes ayant une même partie de la clé primaire  
 * getPiloteByKey: charge un pilote connaissant sa clé primaire (pk)  
 * multiGetPiloteByPartialKey  
 * multiGetTableIteratorOnPiloteRows : accès aux pilotes avec une même shard key  
 */
```

# Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

## ➤ Tables et Indexes : Création de tables et indexes via l'API Java

### ▪ Get KVStore Handle

```
public class Pilote {  
    private final KVStore store;  
  
    /**  
     * Runs the Pilote command line program.  
     */  
    public static void main(String args[]) {  
        try {  
            Pilote unPilote= new Pilote(args);  
            Pilote.testDdlCommandOnPilote(unPilote);  
            Pilote.testMajPilote(unPilote);  
            Pilote.testLectureLignesPilote(unPilote, "Baron",8,"Milo, Haïti");  
        } catch (RuntimeException e) {  
            e.printStackTrace();  
        }  
    }  
}
```

```
/**  
 * Parses command line args and opens the KVStore  
 */  
Pilote(String[] argv) {  
    String storeName = "kvstore";  
    String hostName = "localhost";  
    String hostPort = "5000";  
    final int nArgs = argv.length;  
    int argc = 0;  
    store = KVStoreFactory.getStore  
    (new KVStoreConfig(storeName, hostName + ":" +  
    hostPort));  
}
```

## Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

### ➤ Tables et Indexes : Création de tables et indexes via l'API Java

#### ■ Méthode d'affichage des résultats des commandes ddl

```
/*
 * Affichage du résultat pour les commandes DDL (CREATE, ALTER, DROP)
 */
private void displayResult(StatementResult result, String statement) {
    System.out.println("=====");
    if (result.isSuccessful()) {
        System.out.println("Statement was successful:\n\t" + statement);
        System.out.println("Results:\n\t" + result.getInfo());
    } else if (result.isCancelled()) {
        System.out.println("Statement was cancelled:\n\t" + statement);
    }
}
```

```
else {
    /**
     * Statement was not successful: may be in error, or may still
     * be in progress.
     */
    if (result.isDone()) {
        System.out.println("Statement failed:\n\t" + statement);
        System.out.println("Problem:\n\t" + result.getErrorMessage());
    } else {
        System.out.println("Statement in progress:\n\t" + statement);
        System.out.println("Status:\n\t" + result.getInfo());
    }
}
```

## Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

### ➤ Tables et Indexes : Création de tables et indexes via l'API Java

#### ■ Crédit de la table PILOTE via l'API

```
public void createPilote() {  
    StatementResult result = null;  
    String statement = null;  
    System.out.println("***** Dans : createPilote *****");  
    try {  
        /*  
         * Add a table to the database.  
         * Execute this statement asynchronously.  
         */  
        statement ="create table pilote("+"plnum  INTEGER,"+  
        "plnom STRING,"+"dname STRING,"+"adr  STRING,"+  
        "tel  STRING,"+"sal  FLOAT,"+  
        "PRIMARY KEY (shard(plnom, adr), plnum)";
```

```
        result = store.executeSync(statement);  
        displayResult(result, statement);  
    } catch (IllegalArgumentException e) {  
        System.out.println("Invalid statement:\n" +  
        e.getMessage());  
    } catch (FaultException e) {  
        System.out.println("Statement couldn't be executed,  
        please retry: " + e);  
    }  
}
```

## Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

### ➤ Tables et Indexes : Création de tables et indexes via l'API Java

- Création d'index via l'API

```
/**
 * createIndexOnPilote : crée un index sur la colonne plnom de la table pilote
 */
public void createIndexOnPilote() {
    StatementResult result = null;
    String statement = null;
    try { /*
        * Création d'un index sur la colonne PLNOM de la table pilote.
        * Execute this statement asynchronously.
     */
        statement = "create index idx_pilote_plnom on pilote(plnom)";
        result = store.executeSync(statement);
        displayResult(result, statement);
    } catch (IllegalArgumentException e) {
        System.out.println("Invalid statement:\n" + e.getMessage());
    } catch (FaultException e) {
        System.out.println("Statement couldn't be executed, please retry: " + e);
    }
}
```

# Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

## ➤ Tables et Indexes : Modification de tables et indexes via l'API Java

### ➤ Ajout d'une colonne à la table Pilote via l'API

```
/**  
 * alterTablePiloteAddColumnAge : ajouter une colonne dans la table pilote  
 */  
public void alterTablePiloteAddColumnAge() {  
    StatementResult result = null;  
    String statement = null;  
    try { /*  
        * Ajout d'une colonne à la table pilote.  
        * Execute this statement asynchronously.  
    */  
        statement ="alter table pilote (add age INTEGER)";  
        result = store.executeSync(statement);  
        displayResult(result, statement);  
    } catch (IllegalArgumentException e) { System.out.println("Invalid statement:\n" + e.getMessage());}  
    } catch (FaultException e) { System.out.println("Statement couldn't be executed, please retry: " + e);  
    }  
}
```

# Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

## ➤Tables et Indexes : Modification de tables et indexes via l'API Java

### ▪ Suppression d'une colonne à la table Pilote via l'API

```
/**  
 * dropColumnAgeFromPiloteTable: suppression de la colonne age  
 */  
public void dropColumnAgeFromPiloteTable() {  
    StatementResult result = null;  
    String statement = null;  
    try {  
        /*  
         * Suppression d'une colonne à la table pilote.  
         * Execute this statement asynchronously.  
         */  
        statement ="alter table pilote (drop age);  
  
        result = store.executeSync(statement);  
        displayResult(result, statement);  
    } catch (IllegalArgumentException e) {  
        System.out.println("Invalid statement:\n" + e.getMessage());  
    } catch (FaultException e) {  
        System.out.println("Statement couldn't be executed, please retry: " + e);  
    }  
}
```

# Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

## ➤Tables et Indexes : Suppression de tables et indexes via l'API Java

### ▪ Suppression d'une index via l'API

```
/**  
 * dropIndexOnPilte : supprime l'index créé précédemment  
 */  
public void dropIndexOnPilte() {  
    StatementResult result = null;  
    String statement = null;  
    try {  
        /*  
         * Suppression d'un index sur la colonne PLNOM de la table pilote.  
         * Execute this statement asynchronously.  
         */  
        statement ="drop index idx_pilote_plnom on pilote";  
  
        result = store.executeSync(statement);  
        displayResult(result, statement);  
    } catch (IllegalArgumentException e) {System.out.println("Invalid statement:\n" + e.getMessage());}  
    } catch (FaultException e) {System.out.println("Statement couldn't be executed, please retry: " + e);  
    }  
}
```

# Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

## ➤Tables et Indexes : Suppression de tables et indexes via l'API Java

### ▪ Suppression d'une table via l'API

```
/**  
 * dropTablePilote : suprime la table pilote  
 */  
  
public void dropTablePilote() {  
    StatementResult result = null;  
    String statement = null;  
    try {  
        /*  
         * Suppression de la table pilote.  
         */  
        statement ="drop table pilote";  
  
        result = store.executeSync(statement);  
        displayResult(result, statement);  
    } catch (IllegalArgumentException e) {System.out.println("Invalid statement:\n" + e.getMessage());}  
    } catch (FaultException e) {System.out.println("Statement couldn't be executed, please retry: " + e);  
    }  
}
```

## Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

### ➤ Mise à jour : Les principales fonctions d'insertion et suppression de lignes

#### ■ Les principales fonctions utiles pour insérer

- 1. **KVStore.getTableAPI()** : Récupérer le pointeur l'interface qui fournit l'API des tables
- 2. **TableAPI.getTable()**: Récupérer le pointeur vers la table à mettre à jour
- 3. **Table.createRow()** : Permet de créer une ligne vide dans la table à mettre à jour
- 4. **Row.put()** : Permet d'ajouter les valeurs de chaque champs dans ligne créer
- 5. **TableAPI.put()** : Ecriture de la ligne dans la base NoSQL

## Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

### ➤ **Mise à jour : Les principales fonctions d'insertion et suppression de lignes**

#### ■ **Les principales fonctions utiles pour supprimer (UNE LIGNE)**

- 1. **KVStore.getTableAPI()** : Récupérer le pointeur l'interface qui fournit l'API des tables
- 2. **TableAPI.getTable()**: Récupérer le pointeur vers la table à mettre à jour
- 3. **Table.createPrimaryKey()**: Permet de créer une ligne vide dans la table à mettre à jour
- 4. **PrimaryKey.put()** : Permet de construire la clé primaire de la ligne à supprimer
- 5. **TableAPI.delete() ou TableAPI.deleteIfVersion()**: Suppression de la ligne dans la base NoSQL connaissant la clé. deleteIfVersion supprime la ligne si la version n'a pas changée.

## Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

### ➤ Mise à jour : Les principales fonctions d'insertion et suppression de lignes

#### ■ Les principales fonctions utiles pour supprimer (PLUSIEURS LIGNES, clé partielle)

- 1. **KVStore.getTableAPI()** : Récupérer le pointeur l'interface qui fournit l'API des tables
- 2. **TableAPI.getTable()**: Récupérer le pointeur vers la table à mettre à jour
- 3. **Table.createPrimaryKey()**: Permet de créer une clé primaire vide afin d'y ajouter les valeurs des champs de la clé
- 4. **PrimaryKey.put()** : Permet de construire une partie de la clé primaire commune à plusieurs lignes à supprimer
- 5. **TableAPI.multiDelete()** : Suppression des en une seule opération des lignes qui matchent la clé partielle

# Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

## ➤ Mise à jour : Les classes à importer

```
import oracle.kv.KVStore;
import java.util.List;
import java.util.Iterator;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;
import oracle.kv.FaultException;
import oracle.kv.StatementResult;
import oracle.kv.table.TableAPI;
import oracle.kv.table.Table;
import oracle.kv.table.Row;
import oracle.kv.table.PrimaryKey;
import oracle.kv.ConsistencyException;
import oracle.kv.RequestTimeoutException;
import java.lang.Integer;
import oracle.kv.table.TableIterator;
```

- Pour plus de détails se reporter à la Javadoc de l'API Java

<https://docs.oracle.com/en/database/other-databases/nosql-database/21.2/java-api/oracle/kv/package-summary.html>

# Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

## ➤ Mise à jour : Insertion des lignes via l'API Java

```
/**  
 * insertAPiloteRow : Insère une nouvelle ligne dans la table pilote  
 */  
  
private void insertAPiloteRow (int plnum, String plnom, String dnaiss, String adr, String tel, float sal){  
    StatementResult result = null;  
    String statement = null;  
    try {  
        TableAPI tableH = store.getTableAPI();  
  
        // The name you give to getTable() must be identical to the name that you gave the table when you created the table using  
        // the CREATE TABLE DDL statement.  
        Table tablePilote = tableH.getTable("pilote");  
  
        // Get a Row instance  
        Row piloteRow = tablePilote.createRow();  
  
        // Now put all of the cells in the row. This does NOT actually write the data to the store. Create one row  
        piloteRow.put("plnum", plnum);  
        piloteRow.put("plnom", plnom);  
        piloteRow.put("dnaiss", dnaiss);  
        piloteRow.put("adr", adr);  
        piloteRow.put("tel", tel);  
        piloteRow.put("sal", sal);
```

## Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

### ➤ Mise à jour : Insertion des lignes via l'API Java

```
// Now write the table to the store.  
// "item" is the row's primary key. If we had not set that value,  
// this operation will throw an IllegalArgumentException.  
    tableH.put(piloteRow, null, null);  
}  
catch (IllegalArgumentException e) {  
    System.out.println("Invalid statement:\n" + e.getMessage());  
}  
catch (FaultException e) {  
    System.out.println("Statement couldn't be executed, please retry: " + e);  
}  
}
```

# Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

## ➤ Mise à jour : Insertion des lignes via l'API Java

```
/**  
 * insertPiloteRows : Insère de nouvelles lignes dans la table pilote en appelant la fonction insertAPiloteRow  
 */  
  
public void insertPiloteRows() {  
    try {  
        this.insertAPiloteRow(1, "Gagarin", "09/03/1934", "Klouchino, Russie", "0071122334455", 10000.75F);  
        this.insertAPiloteRow(2, "Jähn Sigmund", "13/02/1937", "Morgenröthe-Rautenkranz, Allemagne", "0049199999999", 12000.5F);  
        this.insertAPiloteRow(3, "Icare", "13/02/1947", "Athène, Grèce", "00300623344556", 8000.5F);  
        this.insertAPiloteRow(3, "Icare", "13/02/1000", "Olympe, Grèce", "00300623344577", 8500.5F);  
        this.insertAPiloteRow(3, "Icare", "14/02/1900", "Le Pirée, Grèce", "00300623344588", 8600.5F);  
        this.insertAPiloteRow(4, "Zorro", "14/02/1880", "Los Angeles, USA", "0010623344588", 8600.5F);  
        this.insertAPiloteRow(4, "Pagnol", "14/02/1980", "Marseille, France", "00330623344589", 8700.5F);  
        this.insertAPiloteRow(5, "Bleck", "14/02/2010", "Marseille, France", "00330623344589", 8700.5F);  
        this.insertAPiloteRow(6, "Erzulie", "13/02/1804", "Port-au-Prince, Haïti", "005090623344557", 13000.5F);  
        this.insertAPiloteRow(7, "Erzulie", "14/04/1804", "Les Cayes, Haïti", "0050906233445607", 14000.5F);  
        this.insertAPiloteRow(8, "Baron", "15/01/1809", "Milo, Haïti", "0050906233445607", 14000.5F);  
        this.insertAPiloteRow(8, "Baron", "16/04/1812", "Cap, Haïti", "0050906233445607", 14000.5F);  
    } catch (Exception e) {  
        e.printStackTrace();  
    }  
}
```

## Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

### ➤ Mise à jour : Insertion des lignes via l'API Java

```
this.insertAPiloteRow(9, "Baron", "14/04/1890", "Port-à-Piment, Haïti", "0050906233445607", 14000.5F);
} catch (IllegalArgumentException e) {
    System.out.println("Invalid statement:\n" + e.getMessage());
} catch (FaultException e) {
    System.out.println("Statement couldn't be executed, please retry: " + e);
}
```

# Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

## ➤ Mise à jour : Suppression de lignes via l'API Java

### ▪ Suppression d'une ligne connaissant la clé

```
/**  
 * deletePiloteRow : supprime un pilote connaissant la clé  
 * entièrement renseignée  
 */  
  
public void deletePiloteRow(String plnom, int plnum, String adr){  
    StatementResult result = null;  
    String statement = null;  
    try {  
        TableAPI tableH = store.getTableAPI();  
        // The name you give to getTable() must be identical  
        // to the name that you gave the table when you  
        // created  
        // the table using the CREATE TABLE DDL statement.  
        Table tablePilote = tableH.getTable("pilote");
```

```
// Get the primary key for the row that we want to delete  
PrimaryKey pilotePrimaryKey = tablePilote.createPrimaryKey();  
pilotePrimaryKey.put("plnom", plnom);  
pilotePrimaryKey.put("plnum", plnum);  
pilotePrimaryKey.put("adr", adr);  
  
// Now delete rows with the same name.  
// this operation will throw an IllegalArgumentException.  
tableH.delete(pilotePrimaryKey, null, null);  
} catch (IllegalArgumentException e) {  
    System.out.println("Invalid statement:\n" + e.getMessage());  
} catch (FaultException e) {  
    System.out.println("Statement couldn't be executed, please retry: " + e);  
}  
}
```

# Module M4.3, section 2, partie 1 : Modèle Key/Document via l'API Java

- **Mise à jour : Suppression de lignes via l'API Java**
  - **Suppression de pilotes avec Primary Key (PK) partielle**

```
public void multiDeletePiloteRows(String plnom, String adr) {  
    StatementResult result = null;  
    String statement = null;  
    try {  
        TableAPI tableH = store.getTableAPI();  
  
        // The name you give to getTable() must be identical  
        // to the name that you gave the table when you created  
        // the table using the CREATE TABLE DDL statement.  
        Table tablePilote = tableH.getTable("pilote");  
  
        // Get the partial primary key for the row that we want to delete  
        PrimaryKey pilotePrimaryKey = tablePilote.createPrimaryKey();  
    }  
}
```

```
// Supprimer tous les pilotes ayant le nom Icare  
// La PK composé (plnom, plnum, adr) sera renseigné  
// partiellement(plnom et plnum uniquement).  
pilotePrimaryKey.put("plnom", plnom);  
pilotePrimaryKey.put("adr", adr);  
  
// Now delete the table rows in the store.  
// "item" is the row's primary key. If we had not set that value,  
// this operation will throw an IllegalArgumentException.  
tableH.multiDelete(pilotePrimaryKey, null, null);  
} catch (IllegalArgumentException e) {  
    System.out.println("Invalid statement:\n" + e.getMessage());  
} catch (FaultException e) {  
    System.out.println("Statement couldn't be executed, please  
retry: " + e);  
}  
}
```

## Module M4.3, section 2, partie 1 : QUIZ

➤ **Question 1 :** Nous souhaitons effectuer des actions DDL (créer, supprimer et/ou modifier) sur les tables, index, ... cochez ce qui est juste

- A: Il faut appeler les méthodes : drop, alter, create de la classe KVStore
- B: Il appeler la fonction executeSync de la classe KVStore
- C: Il faut appeler la méthode createtable, createIndex, ... de la classe TableAPI
- D: Il faut appeler la méthode createtable, createIndex, ... de la classe Table
- E: Il faut appeler la méthode createtable, createIndex, ... de la classe PrimaryKey

➤ **Question 2 :** Pour insérer des lignes dans une table via l'API Java, cochez ce qui est juste. Une instance de KVStore kvs=...; existe et un pointeur vers l'API des tables existe TableAPI tapi=ksv.getTableAPI();

- A: String s= "Insert into nomTable values(...)" ; tapi.put(s);
- B: Table t=tapi.getTable(nomtable); Row r=t.createRow("Insert into nomTable values(...)"); tapi.put(r);
- C: Table t=tapi.getTable(nomtable); Row r=t.createRow(); r.put(col1, val1); r.put(col2, val2); ...; tapi.put(r);
- D: L'action en B est fausse
- E: La succession d'appels suivant est juste: TableAPI.getTable(); Row.put(); TableAPI.put()

## Module M4.3, section 2, partie 1 : QUIZ

➤ **Question 3 :** Nous souhaitons supprimer une ligne dans une table connaissant sa clé. Nous prenons l'hypothèse que la clé primaire de la table contient plusieurs colonnes. Cochez ce qui est juste

- A: KVStore.getTableAPI(); TableAPI.getTable(); Row.put(); TableAPI.delete()
- B: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey(); PrimaryKey.put(); TableAPI.delete()
- C: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey(); PrimaryKey.put(); TableAPI.multiDelete()
- D: La clé primaire doit être vide ou partiellement renseignée
- E: La clé primaire doit être entièrement renseignée

➤ **Question 4 :** Nous souhaitons supprimer plusieurs lignes dans une table. Nous prenons l'hypothèse que la clé primaire de la table contient plusieurs colonnes. Cochez ce qui est juste

- A: KVStore.getTableAPI(); TableAPI.getTable(); Row.put(); TableAPI.delete()
- B: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey(); PrimaryKey.put(); TableAPI.delete()
- C: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey(); PrimaryKey.put(); TableAPI.multiDelete()
- D: La clé primaire doit être vide ou partiellement renseignée
- E: La clé primaire doit être entièrement renseignée

## Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

# Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

## ➤ Plan

- Les principales fonctions de lectures de lignes
- Classes à importer
- Lecture d'une ligne connaissant sa clé primaire complète
- Lecture de plusieurs lignes connaissant une partie de la clé
- Lecture de toutes les lignes d'une table
- Lecture de lignes dans les tables filles
- Lecture de lignes via un index

## Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

### ➤ Les principales méthodes de lectures de lignes

- 1. **TableAPI.get(PrimaryKey key, ...)**: permettent de lire une ligne connaissant sa clé primaire
- **multiGet**
- 2. **TableAPI. (PrimaryKey key, ...)** ou **TableAPI.multiGetKeys(PrimaryKey key, ...)** : permettent de charger un ensemble de lignes partageant **une partie commune de la clé** primaire appelée
- 3. **TableAPI.tableIterator(PrimaryKey key or IndexKey key , ... )** ou **TableAPI.tableKeysIterator( PrimaryKey key or IndexKey key , ... )** : permet d'itérer sur les lignes d'une table partageant une même key ou une même valeur de clé d'index **partiellement renseignée**
- 4. **TableAPI.tableIterator(Iterator< PrimaryKey > primaryKeyIterator, ...)** ou **TableAPI.tableKeysIterator(Iterator< PrimaryKey > primaryKeyIterator ...)** : recherche parallèle de lignes dans chaque shard avec une séquence de clé comme critère de recherche.

# Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

## ➤ Classes à importer

```
import oracle.kv.KVStore;
import java.util.List;
import java.util.Iterator;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;
import oracle.kv.FaultException;
import oracle.kv.StatementResult;
import oracle.kv.table.TableAPI;
import oracle.kv.table.Table;
import oracle.kv.table.Row;
import oracle.kv.table.PrimaryKey;
import oracle.kv.ConsistencyException;
import oracle.kv.RequestTimeoutException;
import java.lang.Integer;
import oracle.kv.table.TableIterator;
import oracle.kv.table.FieldRange;
import oracle.kv.table.MultiRowOptions;
import java.util.Arrays;
import oracle.kv.table.Index;
import oracle.kv.table.IndexKey;
```

## Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

### ➤ Lecture d'une ligne connaissant sa clé primaire complète : TableAPI.get()

#### ▪ Méthode d'affichage des résultats des lectures

```
/**  
 * displayPiloteRow : Affichage d'une ligne de la table PILOTE  
 */  
  
private void displayPiloteRow (Row piloteRow) {  
    System.out.println("=====");  
    Integer plnum1 = piloteRow.get("plnum").asInteger().get();  
    String plnom1 = piloteRow.get("plnom").asString().get();  
    String dnaiss1 = piloteRow.get("dnaiss").asString().get();  
    String adr1 = piloteRow.get("adr").asString().get();  
    String tel1= piloteRow.get("tel").asString().get();  
    Float sal1= piloteRow.get("sal").asFloat().get();  
    System.out.println(" Pilote row :{ plnum=" + plnum1 + " plnom=" + plnom1 + " dnaiss=" + dnaiss1 + " adr=" + adr1 +  
    " tel=" + tel1 + " salaire=" + sal1 + "});  
}
```

## Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

### ➤ Lecture d'une ligne connaissant sa clé primaire complète : TableAPI.get()

```
/**  
 * getPiloteByKey : Lecture d'un pilote connaissant sa clé primaire  
 * entièrement renseignée  
 */  
  
private void getPiloteByKey(String plnom, int plnum, String adr){  
    StatementResult result = null;  
    String statement = null;  
    try {  
        TableAPI tableH = store.getTableAPI();  
  
        // The name you give to getTable() must be identical  
        // to the name that you gave the table when you created  
        // the table using the CREATE TABLE DDL statement.  
        Table tablePilote = tableH.getTable("pilote");  
  
        PrimaryKey key=tablePilote.createPrimaryKey();  
        key.put("plnom", plnom);  
        key.put("adr", adr);  
        key.put("plnum", plnum);
```

```
// Retrieve the row. This performs a store read operation.  
// Exception handling is skipped for this trivial example.  
Row row = tableH.get(key, null);  
  
// Now retrieve the individual fields from the row.  
displayPiloteRow(row);  
}  
catch (IllegalArgumentException e) {  
    System.out.println("Invalid statement:\n" + e.getMessage());  
}  
catch (FaultException e) {  
    System.out.println("Statement couldn't be executed, please retry: " + e);  
}
```

## Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

### ➤ Lecture de plusieurs lignes connaissant une partie de la clé : TableAPI.multiGet()

```
/**  
 * multiGetPiloteByPartialKey : Lecture des pilotes ayant une même shard key(même nom et même adresse). La PK partiellement renseignée  
 */  
  
private void multiGetPiloteByPartialKey(String plnom, String adr){  
    StatementResult result = null;  
    String statement = null;  
    try {  
        TableAPI tableH = store.getTableAPI();  
  
        // The name you give to getTable() must be identical  
        // to the name that you gave the table when you created  
        // the table using the CREATE TABLE DDL statement.  
        Table tablePilote = tableH.getTable("pilote");
```

```
PrimaryKey key = tablePilote.createPrimaryKey();  
key.put("plnom", plnom);  
key.put("adr", adr);  
  
// Retrieve the rows. This performs a store read operation.  
// Exception handling is skipped for this trivial example.  
List<Row> myPiloteRows = null;  
try {  
    myPiloteRows = tableH.multiGet(key, null, null);  
} catch (ConsistencyException ce) {  
    // The consistency guarantee was not met  
} catch (RequestTimeoutException re) {  
    // The operation was not completed within the  
    // timeout value  
}
```

## Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

### ➤ Lecture de plusieurs lignes connaissant une partie de la clé : TableAPI.multiGet()

```
for (Row piloteRow: myPiloteRows) {  
    displayPiloteRow(piloteRow);  
}  
}  
catch (IllegalArgumentException e) {  
    System.out.println("Invalid statement:\n" + e.getMessage());  
}  
catch (FaultException e) {  
    System.out.println("Statement couldn't be executed, please retry: " + e);  
}  
}
```

## Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

### ➤ Lecture de plusieurs lignes connaissant une partie de la clé :TableAPI.tableIterator()

```
/**  
 * multiGetIteratorOnPiloteRows : Lecture des pilotes ayant une même  
 shard key(même nom et même adresse). La PK partiellement  
 renseignée  
 */  
  
private void multiGetTableIteratorOnPiloteRows(String plnom, String adr){  
    StatementResult result = null;  
    String statement = null;  
    try {  
        TableAPI tableH = store.getTableAPI();  
  
        // The name you give to getTable() must be identical  
        // to the name that you gave the table when you created  
        // the table using the CREATE TABLE DDL statement.  
  
        Table tablePilote = tableH.getTable("pilote");  
  
        PrimaryKey key = tablePilote.createPrimaryKey();  
        key.put("plnom", plnom);  
        key.put("adr", adr);  
    }
```

```
// Exception handling is omitted, but in production code  
// ConsistencyException, RequestTimeException, and FaultException  
// would have to be handled.  
  
TableIterator<Row> iter = tableH.tableIterator(key, null, null);  
  
try {  
    while (iter.hasNext()) {  
        Row piloteRow = iter.next();  
        displayPiloteRow(piloteRow);  
    }  
    finally {  
        if (iter != null) { iter.close();}  
    }  
}  
catch (IllegalArgumentException e) {  
    System.out.println("Invalid statement:\n" + e.getMessage());  
}  
catch (FaultException e) {  
    System.out.println("Statement couldn't be executed, please retry: " + e);  
}
```

## Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

### ➤ Lecture de lignes dans les tables filles et ou mères

#### ■ Méthode d'affichage des résultats des tables filles

```
/**  
 * displayCheckupRow : Affichage d'une ligne de la table CHECKUP  
 */  
private void displayCheckupRow (Row checkupRow) {  
    // Now retrieve the individual fields from the row.  
    Integer plnum1 = checkupRow.get("plnum").asInteger().get();  
    String plnom1 = checkupRow.get("plnom").asString().get();  
    String adr1 = checkupRow.get("adr").asString().get();  
    Integer cunum1= checkupRow.get("cunum").asInteger().get();  
    String cudate1= checkupRow.get("cudate").asString().get();  
    String curesultat1= checkupRow.get("curesultat").asString().get();  
    System.out.println(" checkup row :{ plnum="+ plnum1 + "plnom"+plnom1 + " adr="+adr1+" cunum =" + cunum1 + " cudate1  
    =" + cudate1+"curesultat=" +curesultat1+"});  
}
```

# Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

## ➤ Lecture de lignes dans les tables filles

- Clé primaire entièrement renseignée

```
/**  
 * getCheckupByKey : Lecture du checkup d'un pilote connaissant sa clé primaire  
entièrement renseignée  
*/  
  
private void getCheckupByKey(String plnom, int plnum, String adr, int cunum) {  
StatementResult result = null;  
String statement = null;  
try {  
TableAPI tableH = store.getTableAPI();  
  
// The name you give to getTable() must be identical  
// to the name that you gave the table when you created  
// the table using the CREATE TABLE DDL statement.  
Table tableCheckup = tableH.getTable("pilote.checkup");  
  
PrimaryKey key=tableCheckup.createPrimaryKey();  
key.put("plnom", plnom);  
key.put("adr", adr);  
key.put("plnum", plnum);  
key.put("cunum", cunum);
```

```
// Retrieve the row. This performs a store read operation.  
// Exception handling is skipped for this trivial example.  
Row row = tableH.get(key, null);  
  
// Now retrieve the individual fields from the row.  
displayCheckupRow (row);  
}  
  
catch (ConsistencyException ce) {  
System.out.println(" Consistency exception:\n" + ce.getMessage());  
}  
catch (RequestTimeoutException re) {  
System.out.println(" Request timeout exception:\n" + re.getMessage());  
}  
catch (Exception e) {  
System.out.println(" Toutes Exceptions:\n" + e.getMessage());  
}
```

# Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

## ➤ Lecture de lignes dans les tables filles

- Clé primaire partiellement renseignée : parents seuls et parents avec filles

```
/**  
 * multiGetCheckupAncestorsOrChild : Lecture des pilotes ou  
 * des checkup de pilotes ayant une même shard key(même  
 * nom et même adresse). La PK partiellement renseignée  
 */  
private void multiGetCheckupAncestorsWithOrWithOutChild(  
    String plnom, String adr){  
    StatementResult result = null;  
    String statement = null;  
    try {  
        TableAPI tableH = store.getTableAPI();  
  
        // The name you give to getTable() must be identical  
        // to the name that you gave the table when you created  
        // the table using the CREATE TABLE DDL statement.  
        Table tablePilote = tableH.getTable("pilote");  
        Table tableCheckup = tableH.getTable("pilote.checkup");
```

```
PrimaryKey key = tablePilote.createPrimaryKey();  
key.put("plnom", plnom);  
key.put("adr", adr);  
  
// Get a MultiRowOptions and tell it to look at both the child tables  
MultiRowOptions mro = new MultiRowOptions(null, null,  
    Arrays.asList(tableCheckup ));  
  
// Retrieve the rows.  
TableIterator<Row> iter=tableH.tableIterator(key, mro, null);  
  
while (iter.hasNext()) {  
    Row row = iter.next();  
  
    // display parent rows  
    if (row.getTable().equals(tablePilote)) displayPiloteRow (row);  
  
    // display children rows  
    else if (row.getTable().equals(tableCheckup))  
        displayCheckupRow (row);  
}
```

## Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

### ➤ Lecture de lignes dans les tables filles

```
    }  
  
    catch (ConsistencyException ce) {  
        System.out.println(" Consistency exception:\n" + ce.getMessage());  
    }  
  
    catch (RequestTimeoutException re) {  
        System.out.println(" Request timeout exception:\n" + re.getMessage());  
    }  
  
    catch (Exception e) {  
        System.out.println(" Toutes Exceptions:\n" + e.getMessage());  
    }  
}
```

# Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

## ➤ Lecture de lignes dans les tables filles

- Clé primaire partiellement renseignée : filles seules et filles avec parents

```
/**  
 *multiGetCheckupChildAndAncestors: Lecture des checkup de pilotes et des  
 *ancêtres liés aux checkup ayant une même shard key(même nom et même  
 *adresse). La PK partiellement renseignée  
 */  
private void multiGetCheckupChildAndAncestors (String plnom, String adr){  
    TableAPI tableAPI = store.getTableAPI();  
    StatementResult result = null;  
    String statement = null;  
    try {  
        TableAPI tableH = store.getTableAPI();  
        Table tablePilote = tableH.getTable("pilote");  
        Table tableCheckup = tableH.getTable("pilote.checkup");  
        PrimaryKey key = tableCheckup.createPrimaryKey();  
        key.put("plnom", plnom);  
        key.put("adr", adr);  
        // Get a MultiRowOptions and tell it to look at both the child tables  
        MultiRowOptions mro = new MultiRowOptions(null,  
                                         Arrays.asList(tablePilote ), null);
```

```
// Retrieve the rows.  
TableIterator<Row> iter = tableH.tableIterator(key, mro, null);  
while (iter.hasNext()) {  
    Row row = iter.next();  
    // display parent rows  
    if (row.getTable().equals(tablePilote)) displayPiloteRow (row);  
    // display children rows  
    else if (row.getTable().equals(tableCheckup)) displayCheckupRow (row);  
}  
} catch (ConsistencyException ce) {  
    System.out.println(" Consistency exception:\n" +ce.getMessage());  
} catch (RequestTimeoutException re) {  
    System.out.println(" Request timeout exception:\n" + re.getMessage());  
} catch (IllegalArgumentException e) {  
    System.out.println("Invalid statement:\n" + e.getMessage());  
} catch (FaultException e) {  
    System.out.println("Statement couldn't be executed, please retry: " + e);  
}
```

# Module M4.3, section 2, partie 2 : Modèle Key/Document via l'API Java

## ➤ Lecture de lignes via un index

- Il est nécessaire de construire un index Key

```
/**  
 * multiGetOverIndex: Lecture via index  
 */  
  
private void multiGetOverIndexIdxPilotePlnom(String plnom){  
    StatementResult result = null;  
    String statement = null;  
    try {  
        TableAPI tableH = store.getTableAPI();  
        Table tablePilote = tableH.getTable("pilote");  
  
        // Construct the IndexKey. The name we gave our index when  
        // we created it was 'idx_pilote_plnom'.  
        Index idxPilotePlnom = tablePilote.getIndex("idx_pilote_plnom ");  
        IndexKey idxPilotePlnomKey = idxPilotePlnom.createIndexKey();  
        idxPilotePlnomKey.put("plnom", plnom);  
    }
```

```
// Retrieve the rows.  
TableIterator<Row> iter = tableH.tableIterator(idxPilotePlnomKey , null, null);  
while (iter.hasNext()) {  
    Row row = iter.next();  
    // display pilote rows  
    displayPiloteRow (row);  
}  
} catch (ConsistencyException ce) {  
    System.out.println(" Consistency exception:\n" + ce.getMessage());  
} catch (RequestTimeoutException re) {  
    System.out.println(" Request timeout exception:\n" + re.getMessage());  
} catch (IllegalArgumentException e) {  
    System.out.println("Invalid statement:\n" + e.getMessage());  
} catch (FaultException e) {  
    System.out.println("Statement couldn't be executed, please retry: " + e);  
}
```

## Module M4.3, section 2, partie 2 : QUIZ

➤ **Question 1 :** Nous souhaitons recherche une ligne dans une table connaissant sa clé. Nous prenons l'hypothèse que la clé primaire de la table contient plusieurs colonnes. Cochez ce qui est juste

- A: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey();PrimaryKey.put(); TableAPI.get()
- B: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey();PrimaryKey.put(); TableAPI.multiGet()
- C: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey();PrimaryKey.put(); TableAPI.tableIterator()
- D: La primary key doit être entièrement renseignée
- E: La primary key doit être partiellement renseigné
- F: La primary key peut être vide
- G: On peut ignorer le premier champ de la primary key

➤ **Question 2 :** Nous souhaitons recherche plusieurs lignes dans une table connaissant une partie de la clé. Nous prenons l'hypothèse que la clé primaire de la table contient plusieurs colonnes. Cochez ce qui est juste

- A: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey();PrimaryKey.put(); TableAPI.get()
- B: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey();PrimaryKey.put(); TableAPI.multiGet()
- C: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey();PrimaryKey.put(); TableAPI.tableIterator()
- D: La primary key doit être entièrement renseignée
- E: La primary key doit être partiellement renseigné
- F: La primary key peut être vide
- G: On peut ignorer le premier champ de la primary key

## Module M4.3, section 2, partie 2 : QUIZ

➤ **Question 3 :** Nous souhaitons rechercher une ligne dans une table connaissant un champ ou plusieurs champs qui n'ont rien à voir avec la clé. Cochez ce qui est juste

- A: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey();PrimaryKey.put(); TableAPI.get()
- B: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey();PrimaryKey.put(); TableAPI.multiGet()
- C: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey();PrimaryKey.put(); TableAPI.tableIterator()
- D: KVStore.getTableAPI(); TableAPI.getTable(); Table.getIndexes(); Index.createIndexKey();TableAPI.tableIterator()
- E: Les actions en A, B et C ne sont pas possibles
- G: Pour rechercher sur les champs qui n'ont rien à voir avec la clé, il faut qu'un index ai été créé sur eux

➤ **Question 4 :** Nous souhaitons rechercher une ligne dans une table connaissant un champ ou plusieurs champs qui n'ont rien à voir avec la clé. Un index a été créé sur ces champs. Cochez ce qui est juste

- A: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey();PrimaryKey.put(); TableAPI.get()
- B: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey();PrimaryKey.put(); TableAPI.multiGet()
- C: KVStore.getTableAPI(); TableAPI.getTable(); Table.createPrimaryKey();PrimaryKey.put(); TableAPI.tableIterator()
- D: KVStore.getTableAPI(); TableAPI.getTable(); Table.getIndexes(); Index.createIndexKey();TableAPI.tableIterator()
- E: Les actions en A, B et C ne sont pas possibles
- G: Pour rechercher sur les champs qui n'ont rien à voir avec la clé, il faut qu'un index ai été créé sur eux

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

# Module M4.3, section 3 : Modèle Key/Document via le langage SQL

## ➤ Plan

- Consultation de données : Syntaxe
- Consultation de données : Consultation de toutes les lignes d'une table
- Consultation de données : La clause Order By
- Consultation de données : La clause WHERE
- Consultation de données : La clause GROUP BY
- Mise à jour de données
- Le client sql pour Oracle NoSQL

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Consultation de données :

#### Syntaxe

■ Note :

- Les requêtes **n'impliquent qu'une seule table**
- Ou **plusieurs tables mais unique si table hiérarchiques** : Nested tables, left join)
- Les clauses WHERE, GROUP BY, ORDER BY sont possible

```

SELECT
[DISTINCT] [hints] (STAR | (expression AS id (
" expression AS id)*))
FROM
(table_name | SYSTEM_TABLE_NAME) [[AS] table_alias]
| NESTED TABLES "(" single_from_table
[ANCESTORS "(" aliased_table_name [ON or_expression]
(," aliased_table_name [ON or_expression])* ")"]
[DESCENDANTS "(" aliased_table_name [ON or_expression]
(," aliased_table_name [ON or_expression])* ")"]
")
| left_outer_join_table LEFT OUTER JOIN
single_from_table ON expression
| unnest_syntax)
[WHERE expression]
[GROUP BY expression (," expression)*]
[ORDER BY expression [ASC|DESC] [NULLS (FIRST|LAST)]
(," expression [ASC|DESC] [NULLS (FIRST|LAST)])*]
[LIMIT add_expression]
[OFFSET add_expression];
  
```

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Consultation de données : Consultation de toutes les lignes d'une table

```
Kv-> execute 'select * from pilote2';
```

```
{"plnum":4,"plnom":"Gagarin3","dnaiss":"09/03/1935","adr":"Klouchino3, Russie","tel":"00711223344606","sal":11050.75}  
{"plnum":3,"plnom":"Gagarin2","dnaiss":"09/03/1960","adr":"Moscou1, Russie","tel":"0071122334470","sal":20050.75}  
{"plnum":1,"plnom":"Gagarin1","dnaiss":"09/03/1934","adr":"Klouchino1, Russie","tel":"0071122334455","sal":10000.75}  
{"plnum":2,"plnom":"Gagarin2","dnaiss":"09/03/1935","adr":"Klouchino2, Russie","tel":"0071122334456","sal":10050.75}  
  
4 rows returned
```

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Consultation de données : La clause Order By

- Le tri n'est possible que sur **les colonnes** de la primary key ou des colonnes **indexées** en fonction des versions !!!
- Kv->**execute** 'select \* from pilote2 order by plnom';

```
{"plnum":1,"plnom":"Gagarin1","dnaiss":"09/03/1934","adr":"Klouchino1, Russie","tel":"0071122334455","sal":10000.75}  
{"plnum":2,"plnom":"Gagarin2","dnaiss":"09/03/1935","adr":"Klouchino2, Russie","tel":"0071122334456","sal":10050.75}  
{"plnum":3,"plnom":"Gagarin2","dnaiss":"09/03/1960","adr":"Moscou1, Russie","tel":"0071122334470","sal":20050.75}  
{"plnum":4,"plnom":"Gagarin3","dnaiss":"09/03/1935","adr":"Klouchino3, Russie","tel":"00711223344606","sal":11050.75}  
  
4 rows returned
```

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Consultation de données : La clause WHERE

- Les restrictions ne sont possible que **sur la primary key ou les colonnes indexes en fonction des versions !!!**
- Kv->**execute** 'select \* from pilote2 where plnom="Gagarin2"';

```
{"plnum":2,"plnom":"Gagarin2","dnaiss":"09/03/1935","adr":"Klouchino2, Russie","tel":"0071122334456","sal":10050.75}  
{ "plnum":3,"plnom":"Gagarin2","dnaiss":"09/03/1960","adr":"Moscou1, Russie","tel":"0071122334470","sal":20050.75}
```

2 rows returned

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Consultation de données : La clause GROUP BY

- Compter le nombre de pilotes ayant un même nom
- Kv->**execute** 'SELECT plnom, count(\*) FROM PIOTE2 GROUP BY plnom';

```
{"plnom": "Gagarin1", "Column_2": 1}  
{"plnom": "Gagarin2", "Column_2": 2}  
{"plnom": "Gagarin3", "Column_2": 1}
```

3 rows returned

## **Module M4.3, section 3 : Modèle Key/Document via le langage SQL**

## ➤ Mise à jour de données

- Kv-> execute 'describe table pilote2';

```
kv-> execute 'describe table pilote2';
== Information ==
+-----+-----+-----+-----+-----+-----+-----+-----+
| name | ttl | owner | sysTable | r2compat | parent | children | indexes      | description
+-----+-----+-----+-----+-----+-----+-----+-----+
| pilote2 |     |       | N        | N        |         | checkup    | idx_pilote2_adr |
+-----+-----+-----+-----+-----+-----+-----+-----+
== Fields ==
+-----+-----+-----+-----+-----+-----+-----+-----+
| id | name | type   | nullable | default | shardKey | primaryKey | identity |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1  | plnum | Integer | N        | NULL    |          | Y          |           |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 2  | plnom | String  | N        | NULL    | Y        | Y          |           |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 3  | dnaiss | String  | Y        | NULL    |          |           |           |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 4  | adr   | String  | N        | NULL    | Y        | Y          |           |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 5  | tel   | String  | Y        | NULL    |          |           |           |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 6  | sal   | Float   | Y        | NULL    |          |           |           |
+-----+-----+-----+-----+-----+-----+-----+-----+
```

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Mise à jour de données

- Kv-> execute '**INSERT INTO PILOTE2 values(5,"Gagarin5", "09/03/1935", "Klouchino5, Russie", "007112233466", 13000.75)**';

```
kv-> execute 'insert into PILOTE2 values(5,"Gagarin5", "09/03/1935", "Klouchino5, Russie", "007112233466", 13000.75)';  
{"NumRowsInserted":1}  
  
1 row returned
```

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Mise à jour de données

- Mise à jour d'une ligne connaissant sa clé

Kv-> execute 'UPDATE PILOTE2 p SET p.sal=15000 WHERE p.plnom="Gagarin5" and p.adr="Klouchino5, Russie" and p.plnum=5';

```
kv-> execute 'UPDATE PILOTE2 p SET p.sal=15000 WHERE p.plnom="Gagarin5" and p.adr="Klouchino5, Russie" and p.plnum=5';
>{"NumRowsUpdated":1}

1 row returned
```

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Mise à jour de données

- Suppression d'une ligne connaissant sa clé

Kv-> execute 'DELETE FROM PILOTE2 p WHERE p.plnom="Gagarin5" and p.adr="Klouchino5, Russie" and p.plnum=5';

```
kv-> execute 'DELETE FROM PILOTE2 p WHERE p.plnom="Gagarin5" and p.adr="Klouchino5, Russie" and p.plnum=5';
{" numRowsDeleted":1}

1 row returned
```

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Le client sql pour Oracle NoSQL

- Activation

```
d:\> java -jar %KVHOME%/lib/sql.jar -helper-hosts localhost:5000 -store kvstore
```

Le prompt suivant apparait

Sql->

- Utilisation

```
▪ Sql-> select * from pilote2  
      where plnom="Gagarin2";
```

```
{"plnum":2,"plnom":"Gagarin2","dnaiss":"09/03/1935","adr":"Klouchino2, Russie","tel":"0071122334456","sal":10050.75}  
{"plnum":3,"plnom":"Gagarin2","dnaiss":"09/03/1960","adr":"Moscou1, Russie","tel":"0071122334470","sal":20050.75}
```

2 rows returned

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Le client sql pour Oracle NoSQL

#### ▪ Crédation de la table mère

```
Sql-> create table pilote3(
plnum INTEGER ,
plnom STRING,
dnaiss STRING,
adr STRING,
tel STRING,
sal FLOAT,
PRIMARY KEY (shard(plnom, adr), plnum));
```

Statement completed successfully

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Le client sql pour Oracle NoSQL

#### ▪ Crédit de la table fille

```
Sql->create table pilote3.checkup(  
cunum    INTEGER,  
cudate      STRING,  
curestatal STRING,  
PRIMARY KEY (cunum));
```

Statement completed successfully

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Le client sql pour Oracle NoSQL

#### ▪ Insertion de lignes dans la table PILOTE3

```
Sql->insert into pilote3 values (1,"Gagarin1","09/03/1934","Klouchino1, Russie", "0071122334455", 10000.75);  
Sql->insert into pilote3 values (2,"Gagarin2","09/03/1935","Klouchino2, Russie", "0071122334456", 10050.75);  
Sql->insert into pilote3 values (3,"Gagarin2","09/03/1960","Moscou1, Russie", "0071122334470", 20050.75);  
Sql->insert into pilote3 values (4,"Gagarin3","09/03/1935","Klouchino3, Russie", "00711223344606", 11050.75);
```

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Le client sql pour Oracle NoSQL

#### ▪ Insertion de lignes dans la table fille PIOTE3.CHECKUP

```
sql->insert into pilote3.checkup values ("Gagarin2","Klouchino2, Russie", 2, 1, "12-12-2015", "BON");
sql-> insert into pilote3.checkup values ("Gagarin2","Klouchino2, Russie", 2, 2, "11-1-2016", "BON MAIS ATTENTION AU THE");
sql-> insert into pilote3.checkup values ("Gagarin2", "Moscou2, Russie",5, 1, "12-12-2015", "BON");
```

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Le client sql pour Oracle NoSQL

- Augmentation d'un pilote de 10%

```
Sql->UPDATE PILOTE3 p SET p.sal=p.sal*1.1 WHERE p.plnom="Gagarin2" AND p.adr="Klouchino2, Russie" AND p.plnum=2 ;
>{"NumRowsUpdated":1}
1 row returned
```

```
Sql->UPDATE PILOTE3 p SET p.sal=p.sal*1.1 WHERE p.plnom="Gagarin2" AND p.adr="Klouchino2, Russie" ;
```

```
Error handling command UPDATE PILOTE3 p SET p.sal=p.sal*1.1
WHERE p.plnom="Gagarin2" AND p.adr="Klouchino2, Russie":
Error: Multi-row update is not supported. A complete and
exact primary key must be specified in the WHERE clause.
```

NOTE : La mise à jour ne doit concerner qu'une ligne connaissant sa clé

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Le client sql pour Oracle NoSQL

#### ▪ Modification d'un checkup

```
Sql->UPDATE PIOTE3.CHECKUP p SET p.cudate= "12-12-2012" WHERE p.plnom="Gagarin2" AND p.adr="Klouchino2, Russie" AND p.plnum=2 and cunum=1;
```

```
{"NumRowsUpdated":1}
```

```
1 row returned
```

**NOTE :** La mise à jour ne doit concerner qu'une ligne connaissant sa clé

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Le client sql pour Oracle NoSQL

- Suppression de tous les pilote

Sql->**DELETE FROM PIOTE3 p;**

{"numRowsDeleted":4}

1 row returned

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Le client sql pour Oracle NoSQL

- Suppression de tous les checkup

Sql->**DELETE FROM PILOTE3.checkup p;**

{"numRowsDeleted":3}

1 row returned

# Module M4.3, section 3 : Modèle Key/Document via le langage SQL

## ➤ Le client sql pour Oracle NoSQL

### ▪ Consultation de la table pilote3

Sql->**SELECT \* FROM Pilote3 p;**

```
sql-> SELECT * FROM Pilote3 p;
{"plnum":1,"plnom":"Gagarin1","dnaiss":"09/03/1934","adr":"Klouchino1, Russie","tel":"0071122334455","sal":10000.75}
{"plnum":4,"plnom":"Gagarin3","dnaiss":"09/03/1935","adr":"Klouchino3, Russie","tel":"00711223344606","sal":11050.75}
{"plnum":3,"plnom":"Gagarin2","dnaiss":"09/03/1960","adr":"Moscou1, Russie","tel":"0071122334470","sal":20050.75}
{"plnum":2,"plnom":"Gagarin2","dnaiss":"09/03/1935","adr":"Klouchino2, Russie","tel":"0071122334456","sal":10050.75}

4 rows returned
sql->
```

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Le client sql pour Oracle NoSQL

#### ▪ Consultation de la table pilote3.CHECKUP

```
Sql->SELECT * FROM Pilote3.CHECKUP p;
```

```
sql-> SELECT * FROM Pilote3.CHECKUP p;
>{"plnom":"Gagarin2","adr":"Moscou2, Russie","plnum":5,"cunum":1,"cudate":"12-12-2015","curesttat":"BON"}
>{"plnom":"Gagarin2","adr":"Klouchino2, Russie","plnum":2,"cunum":1,"cudate":"12-12-2015","curesttat":"BON"}
>{"plnom":"Gagarin2","adr":"Klouchino2, Russie","plnum":2,"cunum":2,"cudate":"11-1-2016","curesttat":"BON MAIS ATTENTION AU THE"}
```

3 rows returned

# Module M4.3, section 3 : Modèle Key/Document via le langage SQL

## ➤ Le client sql pour Oracle NoSQL

### ▪ Consultation des tables PILOTE3 et pilote3.CHECKUP : Jointure externe gauche

```
Sql->SELECT * FROM pilote3 p LEFT OUTER JOIN pilote3.checkup pc  
ON p.plnom = pc.plnom AND p.adr = pc.adr AND p.plnum = pc.plnum ORDER BY p.plnom;
```

```
sql-> SELECT * FROM pilote3 p LEFT OUTER JOIN pilote3.checkup pc  
ON p.plnom = pc.plnom  
AND p.adr = pc.adr  
AND p.plnum = pc.plnum  
ORDER BY p.plnom;  
      ->      ->      -> {"p": {"plnum": 1, "plnom": "Gagarin1", "dnaiss": "09/03/1934", "adr": "Klouchino1, Russie", "tel": "0071122334455", "sal": 10000.75}, "pc": null}  
{"p": {"plnum": 3, "plnom": "Gagarin2", "dnaiss": "09/03/1960", "adr": "Moscou1, Russie", "tel": "0071122334470", "sal": 20050.75}, "pc": null}  
{"p": {"plnum": 2, "plnom": "Gagarin2", "dnaiss": "09/03/1935", "adr": "Klouchino2, Russie", "tel": "0071122334456", "sal": 10050.75}, "pc": {"plnom": "Gagarin  
2", "adr": "Klouchino2, Russie", "plnum": 2, "cunum": 1, "cudate": "12-12-2015", "curesultat": "BON"}}  
{"p": {"plnum": 2, "plnom": "Gagarin2", "dnaiss": "09/03/1935", "adr": "Klouchino2, Russie", "tel": "0071122334456", "sal": 10050.75}, "pc": {"plnom": "Gagarin  
2", "adr": "Klouchino2, Russie", "plnum": 2, "cunum": 2, "cudate": "11-1-2016", "curesultat": "BON MAIS ATTENTION AU THE"}}  
{"p": {"plnum": 4, "plnom": "Gagarin3", "dnaiss": "09/03/1935", "adr": "Klouchino3, Russie", "tel": "00711223344606", "sal": 11050.75}, "pc": null}  
  
5 rows returned
```

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Le client sql pour Oracle NoSQL

- Consultation des tables PILOTE3 et pilote3.CHECKUP : Jointure externe gauche

```
Sql->SELECT * FROM pilote3.CHECKUP pc LEFT OUTER JOIN pilote3 p
ON p.plnom = pc.plnom AND p.adr = pc.adr AND p.plnum = pc.plnum ORDER BY p.plnom;
```

```
sql-> SELECT * FROM pilote3.CHECKUP pc LEFT OUTER JOIN pilote3 p
ON p.plnom = pc.plnom AND p.adr = pc.adr AND p.plnum = pc.plnum ORDER BY p.plnom;
   -> {"p":{"plnum":2,"plnom":"Gagarin2","dhaiss":"09/03/1935","adr":"Klouchino2, Russie","tel":"0071122334456","sal":10050.75}, "pc":{"plnom": "Gagarin2", "adr": "Klouchino2, Russie", "plnum": 2, "cunum": 1, "cudate": "12-12-2015", "curesultat": "BON"}}
   {"p":{"plnum":2,"plnom":"Gagarin2","dhaiss":"09/03/1935","adr":"Klouchino2, Russie","tel":"0071122334456","sal":10050.75}, "pc":{"plnom": "Gagarin2", "adr": "Klouchino2, Russie", "plnum": 2, "cunum": 2, "cudate": "11-1-2016", "curesultat": "BON MAIS ATTENTION AU THE"}}
   {"p":null, "pc":{"plnom": "Gagarin2", "adr": "Moscou2, Russie", "plnum": 5, "cunum": 1, "cudate": "12-12-2015", "curesultat": "BON"}}

3 rows returned
```

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Le client sql pour Oracle NoSQL

#### ▪ Consultation des tables PILOTE3 et pilote3.CHECKUP : Clause NESTED TABLES

- Compter le nombre de training des pilotes de nom Gagarin2

Sql->SELECT count(pt.trnum) FROM **NESTED TABLES**(pilote3 p descendants(pilote3.training pt **ON** p.plnom = pt.plnom and pt.plnom = "Gagarin2"));

```
sql-> SELECT count(pt.trnum)
  FROM NESTED TABLES(pilote3 p descendants(pilote3.training pt ON
    ->      -> p.plnom = pt.plnom and pt.plnom = "Gagarin2"));
 {"Column_1":2}

1 row returned
```

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Le client sql pour Oracle NoSQL

#### ▪ Consultation des tables PILOTE3 et pilote3.CHECKUP : Clause NESTED TABLES

- Renvoie le nom et l'adresse d'un checkup et le nom du pilote parent du checkup

```
Sql-> SELECT pt.plnom, pt.adr, pt.plnum, p.plnom
  FROM NESTED TABLES(pilote3.training pt ANCESTORS(pilote3 p));
```

```
sql-> SELECT pt.plnom, pt.adr, pt.plnum, p.plnom
      -> FROM NESTED TABLES(pilote3.training pt ancestors(pilote3 p));
{"plnom":"Gagarin2","adr":"Moscou2, Russie","plnum":5,"Column_4":null}
 {"plnom":"Gagarin2","adr":"Klouchino2, Russie","plnum":2,"Column_4":"Gagarin2"}
 {"plnom":"Gagarin2","adr":"Klouchino2, Russie","plnum":2,"Column_4":"Gagarin2"}

 3 rows returned
```

## Module M4.3, section 3 : Modèle Key/Document via le langage SQL

### ➤ Le client sql pour Oracle NoSQL

- En cas d'erreur faire : show fault

```
sql-> SELECT pt.plnom, pt.adr, pt.plnum, p.plnom
      -> FROM NESTED TABLES(pilote4.training pt ancestors(pilote3 p));
Error handling command SELECT pt.plnom, pt.adr, pt.plnum, p.plnom
FROM NESTED TABLES(pilote4.training pt ancestors(pilote3 p));
Error: at (2, 19) Table pilote4.training does not exist
```

## Module M4.3, section 3 : QUIZ

### Considérons les tables suivantes

```
Create table MAISON(manom string, manumero integer, maadresse string, PRIMARY KEY(manom , manumero));
```

```
Create table MAISON.PORTES(ponumero integer, pohauteur integer, polargeur integer , PRIMARY KEY(ponumero));
```

```
Create table MAISON.FENETRES(fenumero integer, fehauteur integer, felargeur integer , PRIMARY KEY(fenumero));
```

➤ **Question 1 :** Nous souhaitons ajouter des lignes dans la table **MAISON** dans l'interface ligne de commande **sql->**. Cochez ce qui fonctionne

- A: insert into maison values("Grand chateau", 1, "28, avenue de Valrose 06000 Nice");
- B: insert into maison(manom, manumero) values("Petit chateau 2", 2);
- C: insert into maison(maadresse) values("66 rue des mimosas 06000 NICE");
- D: insert into maison (maadresse, manumero, manom) values("67 rue des mimosas 06000 NICE", 3, "Grand chateau");
- E: insert into maison (manom, manumero) values("Vaux-le-Vicomte", "4");
- F: insert into maison values("Grand chateau", 6, "28 BIS, avenue de Valrose 06000 Nice");
- G: insert into maison values("Villa Carole de Roumanie", 7);

➤ **Question 2 :** Nous souhaitons ajouter des lignes dans la table fille **MAISON.PORTES** dans l'interface ligne de commande **sql->**. Cochez ce qui fonctionne

- A: insert into maison.portes values("Grand chateau", 1, 1, 2, 2);
- B: insert into maison.portes values("Petit chateau", 2, 1, 2, 2);
- C: insert into maison.portes (ponumero, pohauteur, polargeur) values(1, 2, 2 );
- D: insert into maison.portes (manumero,ponumero, pohauteur, polargeur, manom) values(3, 1, 2, 2, "Grand chateau");
- E: insert into maison.portes values("Vaux-le-Vicomte", "4", 1, 2, 2);
- F: insert into maison.portes values("Grand chateau", 5, 1, 2, "2");

## Module M4.3, section 3 : QUIZ

➤ Question 3 : Cochez ce que fait la requête suivante

```
SELECT * FROM NESTED TABLES(maison.portes mp ANCESTORS(maison m)); ou bien
SELECT * FROM maison.portes mp LEFT OUTER JOIN maison m ON m.manom = mp.manom
AND m.manumero = mp.manumero ORDER BY m.manom;
```

- A: Cette requête renvoie les informations sur les portes des maisons de nom : Grand château
- B: Il s'agit d'une **jointure interne**, elle renvoie les infos sur les maisons «Grand Château» et les infos sur les portes de ces maisons
- C: Il s'agit d'une **jointure externe**, elle renvoie les infos sur les maisons «Grand Château» et les infos sur les portes de ces maisons. Mais aussi les infos sur les portes qui ne sont liées à aucune maison de nom «Grand Château»
- D: Il s'agit d'une **jointure full**, elle renvoie les infos sur les maisons «Grand Château» et les infos sur les portes de ces maisons. Mais aussi les infos sur les portes qui ne sont liées à aucune maison de nom «Grand Château» et aussi des informations sur les maisons de nom «Grand Château» qui n'ont pas encore de portes

➤ Question 4 : Cochez ce que fait la requête suivante

```
SELECT * FROM NESTED TABLES( maison m DESCENDANTS( maison.portes mp))ORDER BY m.manom; ou bien
SELECT * FROM maison m LEFT OUTER JOIN maison.portes mp
ON m.manom = mp.manom AND m.manumero = mp.manumero ORDER BY m.manom;
```

- A: Cette requête renvoie les informations sur les portes des maisons de nom : Grand château
- B: Il s'agit d'une **jointure interne**, elle renvoie les infos sur les maisons «Grand Château» et les infos sur les portes de ces maisons
- C: Il s'agit d'une **jointure externe**, elle renvoie les infos sur les maisons «Grand Château» et les infos sur les portes de ces maisons. Mais aussi les infos sur les portes qui ne sont liées à aucune maison de nom «Grand Château»
- D: Il s'agit d'une **jointure full**, elle renvoie les infos sur les maisons «Grand Château» et les infos sur les portes de ces maisons. Mais aussi les infos sur les portes qui ne sont liées à aucune maison de nom «Grand Château» et aussi des informations sur les maisons de nom «Grand Château» qui n'ont pas encore de portes

## Module M4.3, section 3 : QUIZ

➤ **Question 5 :** Cochez ce que fait la requête suivante

**delete from maison where manom = "Grand chateau"**

- A: Cette requête permet de supprimer toutes les maisons de nom «Grand château» ainsi que leurs portes
- B: Cette requête permet de supprimer toutes les maisons de nom «Grand château». Si elles avaient des portes, ces dernières ne sont pas concernées par la suppression
- C: Cette requête supprime la première maison de nom «Grand château» ainsi que ses portes
- D: Cette requête supprime toutes les portes de nom «Grand château» et pour celles qui sont liées à des maisons, ces maisons sont aussi supprimées
- E: Cette requête supprime toutes les maisons de nom «Grand château»
- F: Cette requête supprime toutes les maisons de nom «Grand château» et supprime toutes les portes de nom «Grand château» y compris celles qui ne sont pas liées à des maisons
- G: Cette requête supprime toutes les maisons
- H: G: Cette requête supprime toutes les maisons et leurs portes

## Module M4.3, section 3 : QUIZ

➤ **Question 6 :** Cochez ce que fait la requête suivante

**delete FROM maison all\_deletes;**

- A: Cette requête permet de supprimer toutes les maisons de nom «Grand château» ainsi que leurs portes
- B: Cette requête permet de supprimer toutes les maisons de nom «Grand château». Si elles avaient des portes, ces dernières ne sont pas concernées par la suppression
- C: Cette requête supprime la première maison de nom «Grand château» ainsi que ses portes
- D: Cette requête supprime toutes les portes de nom «Grand château» et pour celles qui sont liées à des maisons, ces maisons sont aussi supprimées
- E: Cette requête supprime toutes les maisons de nom «Grand château» et pour celles qui sont liées à des portes, ces maisons sont aussi supprimées
- F: Cette requête supprime toutes les maisons de nom «Grand château» et supprime toutes les portes de nom «Grand château» y compris celles qui ne sont pas liées à des maisons
- G: Cette requête supprime toutes les maisons
- H: Cette requête supprime toutes les maisons et leurs portes
- I : Aucune des actions précédentes n'est possible

## Module M4.3 : Oracle NoSql et le Modèle Key/Document

➤ Bilan

➤ Exercices

# Module M4.2 et M4.3 : Bibliographie

[1] Documentation Bigdata Oracle

<https://docs.oracle.com/en/bigdata/index.html>

[2] Oracle Big Data SQL Installation Guide

<https://docs.oracle.com/en/bigdata/big-data-sql/4.1.1/bdsig/oracle-big-data-sql-installation-guide.pdf>

[3] Oracle® Big Data SQL User's Guide

<https://docs.oracle.com/en/bigdata/big-data-sql/4.1.1/bdsug/oracle-big-data-sql-users-guide.pdf>

[4] Livres / Documentation NoSQL oracle

<https://docs.oracle.com/en/database/other-databases/nosql-database/21.2/books.html>

[5] Oracle NoSQL Database Administrator's Guide

<https://docs.oracle.com/en/database/other-databases/nosql-database/21.2/admin/administrators-guide.pdf>

[6] Oracle NoSQL Database Concepts Manual 12c Release

<https://docs.oracle.com/en/database/other-databases/nosql-database/21.2/concepts/concepts-guide.pdf>

[7] Oracle® NoSQL Database, SQL Beginner's Guide

<https://docs.oracle.com/en/database/other-databases/nosql-database/21.2/sqlfornosql/sql-beginners-guide.pdf>

## Module M4.2 et M4.3 : Bibliographie

[8] Oracle® NoSQL Database, SQL Reference Guide

<https://docs.oracle.com/en/database/other-databases/nosql-database/21.2/sqlreferencefornosql/sql-reference-guide.pdf>

[9] Oracle® NoSQL Database  
Integration with SQL Developer

<https://docs.oracle.com/en/database/other-databases/nosql-database/21.2/sqldeveloper-integration/integration-sql-developer.pdf>

[10] Oracle® NoSQL Database  
Java Direct Driver Developer's Guide

<https://docs.oracle.com/en/database/other-databases/nosql-database/21.2/java-driver-table/java-direct-driver-developers-guide.pdf>

[11] Getting Started with Oracle NoSQL Database Key/Value API

<https://docs.oracle.com/en/database/other-databases/nosql-database/18.3/java-driver-kv/getting-started-oracle-nosql-database-key-value-api.pdf>