**Oracle Berkeley Database**

**Document préparé par: Elie Bou Hanna**

**ISSAE CNAM - Liban**

**SMB214 – Mr. Pascal Fares - 2016**

**Berkeley DB** fait partie de la famille des bases de données **« clé-valeur »** et est sans doute le produit de cette famille le plus utilisé au monde avec plusieurs dizaines de millions de déploiements revendiqués par l'éditeur.

Berkeley DB (BDB) est un [moteur de base de données](http://www.wikiwand.com/fr/Moteur_de_base_de_donn%C3%A9es). Il est utilisé dans de nombreux environnements, sous [Unix](http://www.wikiwand.com/fr/Unix), [GNU/Linux](http://www.wikiwand.com/fr/Linux), [Microsoft Windows](http://www.wikiwand.com/fr/Microsoft_Windows) et dans des systèmes embarqués. Son éditeur revendique 200 millions de déploiements.

Depuis la version 2.0, Berkeley DB est disponible sous deux licences, une libre, certifiée par l'[OSI](http://www.wikiwand.com/fr/Open_Source_Initiative) et une licence commerciale. Les versions précédentes étaient sous [licence BSD](http://www.wikiwand.com/fr/Licence_BSD). Berkeley DB est développée par Sleepycat Software qui a été racheté en février 2006 par [Oracle Corporation](http://www.wikiwand.com/fr/Oracle_Corporation).

Elle se présente sous la forme d'une [bibliothèque](http://www.wikiwand.com/fr/Biblioth%C3%A8que_logicielle) écrite en [C](http://www.wikiwand.com/fr/C_%28langage%29) et offrant une interface publique ([API](http://www.wikiwand.com/fr/Application_programming_interface)). Des connecteurs existent pour de très nombreux langages de programmation : [C](http://www.wikiwand.com/fr/C_%28langage%29), [C++](http://www.wikiwand.com/fr/C%2B%2B), [Java](http://www.wikiwand.com/fr/Java_%28langage%29), [Perl](http://www.wikiwand.com/fr/Perl_%28langage%29), [Python](http://www.wikiwand.com/fr/Python_%28langage%29), [Tcl](http://www.wikiwand.com/fr/Tcl_%28langage%29), etc.

La base n'est composée que d'[enregistrements](http://www.wikiwand.com/fr/Enregistrement_%28informatique%29) dont le format est librement déterminé par le programme appelant. Il n'y a pas de notion de [table](http://www.wikiwand.com/fr/Table_%28BDD%29), et la base n'est pas interrogeable via un langage de manipulation de données comme [SQL](http://www.wikiwand.com/fr/Structured_Query_Language). Chaque enregistrement est composé d'une paire clé / valeur, la clé n'étant pas unique.

Les BD clé-valeur à leur tour sont membres de la famille NoSQL (Not only SQL en anglais) désigne une catégorie de [systèmes de gestion de base de données](https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_gestion_de_base_de_donn%C3%A9es) (SGBD) qui n'est plus fondée sur l'architecture classique des [bases relationnelles](https://fr.wikipedia.org/wiki/SGBDR). L'unité logique n'y est plus la [table](https://fr.wikipedia.org/wiki/Table_%28base_de_donn%C3%A9es%29), et les données ne sont en général pas manipulées avec [SQL](https://fr.wikipedia.org/wiki/Structured_Query_Language).

À l'origine, servant à manipuler des [bases de données](https://fr.wikipedia.org/wiki/Bases_de_donn%C3%A9es) géantes pour des sites web de très grande audience. Le NoSQL s'est aussi étendu par le bas après 2010. Il renonce aux fonctionnalités classiques des [SGBD relationnels](https://fr.wikipedia.org/wiki/SGBDR) au profit de la simplicité. Les performances restent bonnes avec la montée en charge ([scalabilit](https://fr.wikipedia.org/wiki/Scalabilit%C3%A9" \o "Scalabilité)y) en multipliant simplement le nombre de serveurs. Les systèmes géants sont les premiers concernés : **énorme quantité de données**, **structuration relationnelle faible** (ou de moindre importance que la capacité d'accès très rapide, quitte à multiplier les serveurs). Un modèle typique en NoSQL est le système clé-valeur, avec une base de données pouvant se résumer topologiquement à un simple [tableau associatif](https://fr.wikipedia.org/wiki/Tableau_associatif) unidimensionnel avec des millions — voire des milliards — d'entrées. Parmi les applications typiques, on retrouve des analyses temps-réel, statistiques, du stockage de logs (journaux), etc.

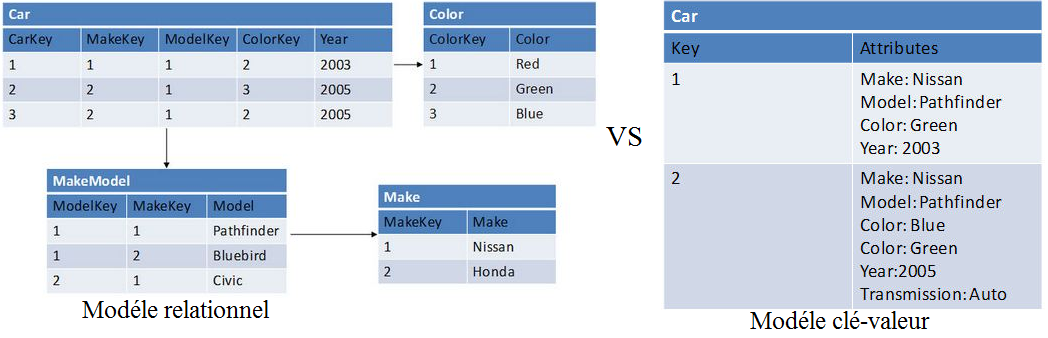
Un [système de gestion de base de données](https://fr.wikipedia.org/wiki/Syst%C3%A8me_de_gestion_de_base_de_donn%C3%A9es) (SGBD) relationnel permet de réaliser des [transactions](https://fr.wikipedia.org/wiki/Transaction_informatique) atomiques, cohérentes, isolées, et durables (ACID).

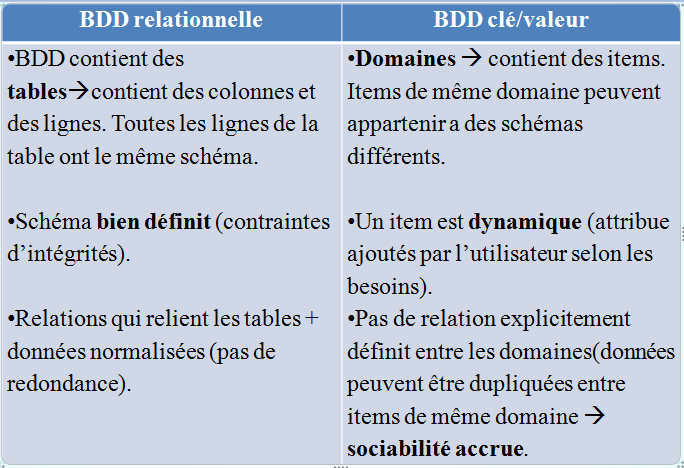
Les capacités ACID garantissent que si plusieurs utilisateurs font de manière simultanée des modifications des données, toutes les modifications vont être prises en compte, dans un ordre précis et maîtrisé de manière à avoir un résultat cohérent (intégrité des données) avec l'historique des modifications faites par chacun. La mise en œuvre stricte des capacités **ACID entraîne des coûts logiciels importants et un niveau de performance moindre à infrastructure matérielle équivalente.**

Les SGBD d'annuaires ont servi de modèle en permettant de lever certaines de ces contraintes en fonction de l'usage, en particulier dans les cas où la grande majorité des accès aux bases de données **consistent en lectures sans modification** (dans ce cas, seule la propriété de persistance importe).

Les solutions du marché implémentent ce protocole en ajoutant leurs techniques propres pour limiter les conséquences de l'impossibilité d'ACID lors des écritures et mises à jour de données.

* **Comparaison entre BD relationnelle et BD clé/valeur :**

****

****

* **La famille de Berkeley DB :**

1. Oracle BDB Core : Cette version de Berkeley DB est écrite en C ANSI et peut être utilisée comme une bibliothèque pour accéder aux données persistées. Oracle BDB fournit de nombreuses interfaces pour différents langages de programmation (C, Perl, PHP, Python…).
2. Oracle BDB XML Edition : Cette version est une surcouche de BDB. Cette version XML permet de stocker et récupérer facilement des documents XML. L'utilisation de XQuery permet d'accéder aux documents XML stockés.
3. BDB Java Edition : Cette version de Berkeley DB, 100 % Java est une base embarquée adoptant le format de stockage clé-valeur. De plus, BDB JE est très performante et flexible. Cette base supporte les transactions, permet de stocker des objets Java en utilisant seulement quelques annotations.

* La famille de Berkeley DB :

Chaque membre de la famille BDB possède différentes fonctionnalités. Dans toutes les familles de BDB, nous pouvons remarquer les fonctionnalités suivantes :

|  |  |
| --- | --- |
| **Feature Set** | **Description** |
| Data Store (DS) | 1 Writer and n reader |
| Concurrent Data Store (CDS) | n writers and n snapshot reader |
| Transactional Data Store (TD) | Full ACID support on top of CDS |
| High Availability (HA) | Replication for fault tolerence. Fail over recovery support |

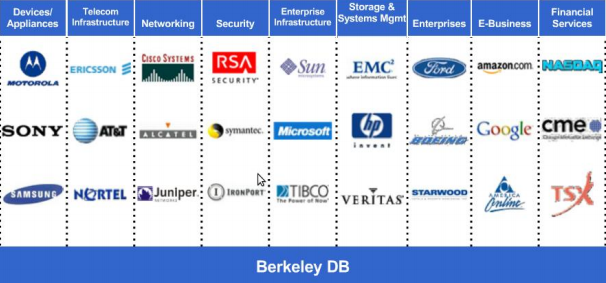
Le tableau suivant décrit les différences entre BDB "core" et BDB JE :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **DS** | **CDS** | **TS** | **HA** |
| BDB/BDB XML | X | X | X | X |
| BDD JE |  | X | X |  |

* **Quelques Fonctionnalités de Berkeley DB :**

Berkeley DB offre également des services de base de données basics pour les développeurs. Ces services comprennent :

* la possibilité de verrouiller des enregistrements ;
* une gestion simplifiée des sauvegardes et de la réplication. On peut effectuer des sauvegardes « à chaud », c.-à-d. sans arrêter la base ;
* la gestion d'un système de cache mémoire interne ;
* Elle supporte de grosses capacités de données
* Les données peuvent être chiffrées ou cryptées (algorithme AES) ;
* Support des transactions Xa.
* **Applications utilisant BDB**
* [OpenLDAP](http://www.wikiwand.com/fr/OpenLDAP), un serveur [LDAP](http://www.wikiwand.com/fr/Lightweight_Directory_Access_Protocol) libre.
* [Subversion](http://www.wikiwand.com/fr/Subversion_%28logiciel%29), un système de [gestion de versions](http://www.wikiwand.com/fr/Gestion_de_versions).
* [RPM](http://www.wikiwand.com/fr/RPM_Package_Manager).
* [Asterisk](http://www.wikiwand.com/fr/Asterisk_%28logiciel%29).
* Citadel/UX, une suite collaborative pour le courrier électronique.
* [SQLite](http://www.wikiwand.com/fr/SQLite) un moteur de base de données embarquées.



* **Introduction à Berkeley DB Java Edition :**

1. **Installation :**

Tout d'abord, débutons par le commencement. Pour pouvoir utiliser BDB Java Edition, vous devez l'installer à l'adresse suivante :

<http://www.oracle.com/technetwork/products/berkeleydb/downloads/index.html>. L'installation de Berkeley DB est assez simple, il suffit de décompresser l'archive que vous venez de télécharger ! Le seul fichier dont vous aurez besoin pour faire fonctionner les exemples que nous verrons plus tard est le : "je-\*.jar".

1. **JE offre les caractéristiques principales suivantes:**

-    Les Bases de données JE varient efficacement d'un à plusieurs millions d'enregistrements. **La taille des bases de données BDB JE** sont susceptibles d'être limités par des ressources matérielles que par toutes limites imposées par le logiciel JE.

**-    Les environnements de base de données** : Les Environnements de base de données dans JE fournissent une unité d'encapsulation et de gestion d'une ou plusieurs bases de données. Les environnements sont également l'unité de gestion des ressources internes telles que le cache en mémoire et les background threads. On utilise des environnements pour gérer la concurrence et les transactions. Notez que toutes les applications utilisant JE sont tenus d'utiliser les environnements de bases de données.

-    **Multiple thread and process support** : JE est conçu pour de multiples threads de contrôle. Lire et écrire les opérations peut être effectué par plusieurs threads. JE utilise le verrouillage des enregistrements pour une grande concurrence dans les applications filetées. En outre, les délais d'attente pour JE utilise la détection de blocage pour vous aider à vous assurer que deux fils de commande non pas des blocages indéfiniment. En outre, JE permet de multiples processus d'accéder aux mêmes bases de données. Toutefois, dans cette configuration JE exige qu'il n'y ait pas plus qu'un processus autorisé à écrire dans la base de données. les processus de lecture sont garantis de façon cohérente, une vue sur les données stockées dès le temps que l'environnement est ouvert.

-    **Transactions :** Les Transactions nous permettent de traiter une ou plusieurs opérations sur une ou plusieurs bases de données en une seule unité de travail. JE transactions offrent pour le développeur de l'application la recouvrabilité, l'atomicité, et l'isolement pour les opérations de base de données. A noter que la protection de la transaction est facultative.

**-    In-memory cache** : Le cache permet un accès de base de données à haute vitesse pour les opérations de lecture et d'écriture en évitant les I/O inutiles sur disque. La cache se développera sur demande jusqu'à une taille maximale pré-configuré. Pour améliorer les performances de l’application immédiatement après le temps de démarrage, On peut précharger le cache afin d'éviter les I/O sur disque pour les demandes des données de production.

**-    Indices :** JE vous permet de créer et de maintenir facilement des indices secondaires pour les données primaires. De cette façon, on peut obtenir un accès rapide à vos données grâce à l'utilisation d'une clé alternative, ou secondaire.

-    **Les fichiers journaux (Log Files) :** JE bases de données sont stockées dans une ou plusieurs fichiers journaux dans le répertoire environnement. Les fichiers journaux sont à écriture unique et sont sur toutes les plateformes portables avec différents endian-ness. Contrairement à d'autres implémentations de bases de données, il n'y a pas de distinction entre les fichiers de base de données (qui est, la "base de données de matériel») et les fichiers journaux. Au lieu de cela JE emploie un système de stockage basée sur le journal pour protéger les modifications de base de données. Avant que toute modification est apportée à une base de données, JE écrit des informations sur la modification sur les fichiers de log.

-    **Background threads** : JE fournit plusieurs threads qui gèrent les ressources internes. Le « checkpointer » est responsable du rinçage des données sur le disque qui a été écrit en cache comme le résultat d'une validation de la transaction de base de données (ce qui est fait dans le but de raccourcir le temps de récupération). Le  « compressor » supprime les sous-arbres de la base de données qui sont vides à cause d’une activité de suppression de données. Enfin, le thread de « nettoyage » est responsable du nettoyage et de la suppression des fichiers journaux inutiles, qui aide ainsi à économiser de l'espace disque.

-    **Sauvegarde et restauration**: Procédure de sauvegarde de JE consiste à copier simplement les fichiers journaux de JE à un endroit sûr pour le stockage. Pour récupérer d'un échec catastrophique, vous copiez vos fichiers journaux archivés vers votre site de production sur le disque et rouvrez l'environnement JE. Notez que JE effectue toujours la reprise normale quand il ouvre un environnement de base de données. Récupération normale apporte la base de données à un état cohérent basé sur des informations de changement trouvé dans les fichiers journaux de base de données.

1. **Les API d'accès aux données :**

Vous utilisez JE à travers une série d'API Java qui vous donnent la capacité de lire et d'écrire vos données, gérer votre base de données (s), et effectuer d'autres activités plus avancées telles que la gestion des transactions. Les API Java que vous utilisez pour interagir avec JE viennent dans deux saveurs de base principales. La première est une API de haut niveau qui vous permet de faire des classes Java persistante. La seconde est une API de niveau inférieur qui fournit une flexibilité supplémentaire lors de l'interaction avec les bases de données JE.

l'API de niveau inférieur est l'API traditionnelle que vous êtes probablement habitué à utiliser. Quel que soit le jeu d'API que vous choisissez d'utiliser, il y a une série de concepts et des API qui sont communs à travers le produit.

* **l'API de base** fournit un simple modèle clé-valeur pour stocker et récupérer les données.
* **l'API DPL** (Direct Persistence Layer) qui vous laisse persister n'importe quelle classe Java comportant un constructeur par défaut en base de données. Cette API vous permet de récupérer les données avec des fonctionnalités de recherche assez riches.
* **la Collection API qui étend la Java Collection API** avec le support de la persistance et des transactions en plus de l'accès aux données.

**L'API de base:**

Si vous utilisez **l’API de base**, alors les concepts et les caractéristiques suivantes sont susceptibles d'être d'intérêt pour vous:

• **Dossiers de base de données**. Tous les enregistrements de base de données sont organisés comme de simples paires clé/valeur de données. Les clés et les données peuvent être des types Java primitifs ou même des objets Java complexes.

**• Lecture et écriture direct de la base de données**. Vous pouvez utiliser les méthodes d'un objet de base de données à lire et à écrire des enregistrements de base de données.

• **Curseurs**: Les Curseurs vous donnent la possibilité de se déplacer successivement à travers une base de données. Avec l’utilisation de curseurs, vous pouvez demander à un point spécifique dans la base de données (en utilisant les critères de recherche appliquées à la clé et / ou de la partie de données d'un enregistrement de base de données) et ensuite, soit avancer ou revenir en arrière à travers la base de données.

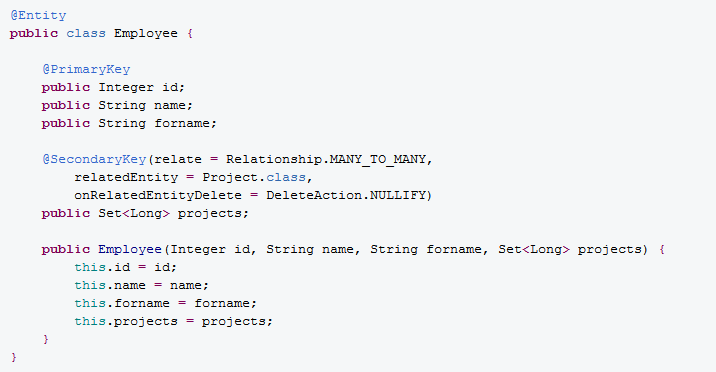
• JCA. JE fournit un support pour la « Java Connector Architecture ».

• JMX. JE fournit un support pour «Java Management Extensions ».

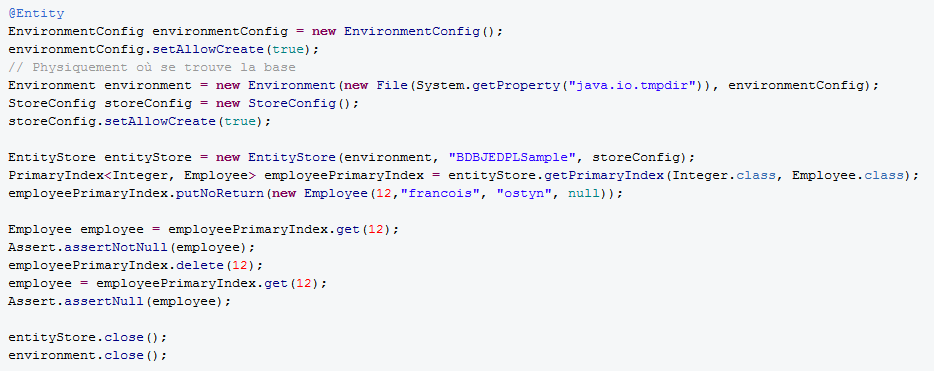
Tout d'abord, avant de commencer à vous parler de DPL, même si nous sommes dans l'univers « NoSQL », il est toujours possible de faire une analogie avec le SQL :

|  |  |
| --- | --- |
| **Terme SQL** | **Équivalence Oracle DPL** |
| Database | Environment |
| Table | Primary Index and its related secondary indices |
| Primary Index | Primary Index |
| Secondary Index | Secondary Index |
| Tuple/Row | Entity |

L'exemple que je vais vous présenter se décompose en deux parties. Premièrement, un entity bean et une classe de management qui gère le CRUD (Create-Replace-Update-Delete) au-dessus de l'entity bean.



Le code ci-dessous nous montre comment accéder aux données présentes dans le POJO Employee :

Dans l'exemple ci-dessus nous ne gérons ni les transactions, ni des relations complexes.

**Le DPL (Data Persistance Layer)** est l'un des deux API que JE fournit pour l'interaction avec les bases de données JE. Le DPL fournit la capacité de causer tout type Java pour être persistant sans mettre en œuvre des interfaces spéciales. La seule véritable exigence est que chaque classe persistante a un constructeur par défaut. Le DPL fournit toutes les fonctionnalités précédemment identifiés. En outre, le DPL offre:

• **Un moyen pratique pour accéder les objets persistants**.

• **Aucun codage manuel** des consolidations n’est nécessaire. Une liaison est une manière de transformer les types de données dans un format qui peut être stocké dans une base de données JE. Si vous n’utilisez pas le DPL, vous pouvez être amené à créer des liaisons personnalisées pour vos types de données. Notez que byte code Java amélioration est utilisé par l'API DPL pour fournir des liaisons entièrement optimisés qui ne utilisent pas la réflexion Java.

• **Pas de schéma externe nécessaire** pour définir les clés d'index primaires et secondaires. Des Java annotations sont utilisées pour définir toutes les métadonnées.

• **Interopérabilité avec des composants externes** est pris en charge en utilisant les collections Java. Tous les index peuvent être accessibles en utilisant une collection de java.util standard.

• **L'évolution des classes est explicitement soutenue**. Cela signifie que vous pouvez ajouter des champs ou d'élargir les types automatique et transparente. Vous pouvez également effectuer de nombreux changements de classe incompatibles, telles que renommer des champs ou refactoring d’une seule classe. Ceci est fait en utilisant un mécanisme appelé mutations intégrées DPL.

Les mutations sont automatiquement appliquées que les données sont accessibles de manière à éviter les temps d'arrêt pour convertir grandes bases de données lors d'une mise à niveau logicielle.

• Les champs de classe persistante peuvent être **private, package-private, protected ou public**. Le DPL peut accéder aux champs de persistance soit par l'amélioration de bytecode ou par réflexion.

• **La performance du moteur JE sous-jacent est sauvegardée**. Toutes les opérations DML sont mappés directement sur les API sous-jacents, les liaisons d'objets sont légers, et tous les paramètres de réglage du moteur sont disponibles.

• Java 1.5 types et annotations génériques sont prises en charge.

**Description de l'environnement de Berkeley DB Java Edition :**

Une base de données BDB JE est composée d'un ou plusieurs fichiers de logs qui sont situés dans le répertoire d'environnement.

Par défaut, leur taille est de 10 Mb et peut être modifiée (par exemple, augmenter la taille des logs permet de limiter le nombre d'I/O).

Le support des transactions :

Pour gérer les transactions avec DPL, il faut faire quelques modifications dans son code.

**Quel API Choisir ?**

Sur les deux API que JE met à votre disposition, Il est recommandé d'utiliser le DPL si tout ce que vous voulez faire est de rendre les classes avec un schéma relativement statique persistant.  
En outre, si vous ne portez pas une application entre Berkley DB et Berkeley DB Java Edition, alors vous ne devriez pas utiliser le DPL mais l'API de base qui est beaucoup plus proche de l'API Berkley DB Java. En outre, si votre application utilise un schéma très dynamique, le DPL est probablement un mauvais choix pour votre application, bien que l'utilisation des annotations Java puisse faire la DPL fonctionne un peu mieux pour vous dans cette situation.

**Conclusion :**

Berkeley diffère des autres SGBD dans:

* Sa flexibilisé et rapidité.
* Produit fiable et évolutive.
* Capacité de choisir les caractéristiques et fonctions à utiliser.