Questions technique:

1. How to optimize sql query:

* Logic
* n'utiliser pas select \*
* WHERE is used instead of Having
* INNER JOIN is used instead of OUTER JOIN if possible
* User VIEW (for the security), Procedure
* Éviter utiliser DISTINCT
* Use EXIST, IN => normally use Exist
* INDEX
* JOIN au lieu de WHERE
* Optimise sub query, minimize the number of subquery

Interface : déclaration une collections de méthodes abstrait et static final variable (constant)

Use interface ou classe abstraite:

Classe abstrait :

1. You want to share code among several closely related classes.
2. You want to declare non-static or non-final fields
3. More detail, constructor

Interface:

1. Si on veut l'interface utilisé par plusieurs classes différentes
2. Si on veut profiter multi-héritage

Sérialisation :

La sérialisation: une façon d'écrire/transformer des objets en mémoire vers un flux de données binaires, ce procédé va donc nous permettre de rendre nos objets persistants.

SerialVersionUID: utilisé pour la désérialisation, vérifier que l’expéditeur and récepteur envoie et reçoit des objets de la même classe, si non => InvalidClassExceptions

* l'interface *Serializable*
* la classe *ObjectOutputStream : sérialiser et écrire : writeObject*
* la classe *ObjectInputStream : lire et déserialiser: readObject*

Une autre interface pour sérialiser: Externalizable

Extends Thread vs Implements Runable:

* Implementing Runnable should be preferred, flexible, can implements autre interface et extens une classe. Extend Thread is closed, stricted.
* Imp : can be use many time
* Imp: ne change pas, modifie pas le comportement de Thread
* Imp: les composant sont séparés, bien pour OOP. HAS au lieu ISA
* ExtendTh : classe hérite tous les methodes elle ne veut pas.

Index type:

1. B-tree: Indexe contenant des valeurs de clés dans une structure arborescente. <http://users.informatik.uni-halle.de/~brass/dbi13w/c4_index.pdf>.

Il effectue un parcours des différents blocs suivant un algorithme afin de trouver la feuille contenant le rowID pointant vers l'enregistrement souhaité.

1. - index Bitmap: index correspondant à un masque de bits pour les valeurs de colonnes indexées. Des AND et OR binaires permettent de faire des tests d'égalité.

Les index Bitmap sont adaptés à la manipulation de colonnes contenant des valeurs peu distinctes et aux requêtes contenant des prédicats d'égalité. Ils gèrent également mieux les valeurs NULL que les index B\*Tree.

Spring :

<bean id="petStore" class="org.springframework.samples.jpetstore.services.PetStoreServiceImpl">  
 <property name="accountDao" ref="accountDao"/>  
 <property name="itemDao" ref="itemDao"/>  
</bean>

ApplicationContext context = **new** ClassPathXmlApplicationContext("services.xml");  
PetStoreService service = context.getBean("petStore", PetStoreService.**class**);  
List<String> userList = service.getUsernameList();

* Maven
* Application.properties : datasource, view.prefix, view.suffix
* Application : SpringBootWebApplication (anotation @SpringBootWebApplication) avec main
* Entity Product
* ->interface ProductRepository extends CrudRepository<Product, Integer>
* ->interface ProductService
* -> ProductServiceImpl (anotation @Service et ProductRepository @Autowired)
* → ProductController (@Controller, ProductService @Autowired)
  + public ModelAndView showProduct(@PathVariable Integer id, Model model)
  + public ModelAndView saveProduct(Product product)
    - return new ModelAndView("redirect:/index");
  + public ModelAndView edit(@PathVariable Integer id, Model model)

@Autowired : cherche bean et l’injecter dans la classe ( dans la méthode setter, constructor)

J’ai fait pas mal des développement des bash shell pour lancer des cron job, le traitement des données, envoyer des emails aux utilisateurs, vérification du statut du serveur, redémarrer le serveur.

|  |  |
| --- | --- |
| * [**HashSet**](http://download.oracle.com/javase/6/docs/api/java/util/HashSet.html)   + Implements [Collection](http://download.oracle.com/javase/6/docs/api/java/util/Collection.html)<E>, [Set](http://download.oracle.com/javase/6/docs/api/java/util/Set.html)<E> | * This class permits the null element. * Fast access, assures no duplicates, provides no ordering. |
| * [**LinkedHashSet**](http://download.oracle.com/javase/6/docs/api/java/util/LinkedHashSet.html)   + Implements [Collection](http://download.oracle.com/javase/6/docs/api/java/util/Collection.html)<E>, [Set](http://download.oracle.com/javase/6/docs/api/java/util/Set.html)<E> | * Runs nearly as fast as HashSet. * No duplicates; iterates by insertion order. |
| * [**TreeSet**](http://download.oracle.com/javase/6/docs/api/java/util/TreeSet.html) - Implements [Collection](http://download.oracle.com/javase/6/docs/api/java/util/Collection.html)<E>, [NavigableSet](http://download.oracle.com/javase/6/docs/api/java/util/NavigableSet.html" \o "interface in java.util)<E>, [Set](http://download.oracle.com/javase/6/docs/api/java/util/Set.html)<E>, [SortedSet](http://download.oracle.com/javase/6/docs/api/java/util/SortedSet.html" \o "interface in java.util)<E> | * not syncronized * No duplicates; iterates in sorted order. |
| * [**ArrayList**](http://download.oracle.com/javase/6/docs/api/java/util/ArrayList.html)   + Implements [Collection](http://download.oracle.com/javase/6/docs/api/java/util/Collection.html)<E>, [List](http://download.oracle.com/javase/6/docs/api/java/util/List.html)<E> | * Fast iteration and fast random access |
| * [**Vector**](http://download.oracle.com/javase/6/docs/api/index.html?java/util/LinkedHashMap.html)   + Implements  [Collection](http://download.oracle.com/javase/6/docs/api/java/util/Collection.html)<E>, [List](http://download.oracle.com/javase/6/docs/api/java/util/List.html)<E> | * synchronised * It's like a slower ArrayList, but it has synchronized methods. |
| * [**LinkedList**](http://download.oracle.com/javase/6/docs/api/java/util/LinkedList.html)   + Implements  [Collection](http://download.oracle.com/javase/6/docs/api/java/util/Collection.html)<E>, [Deque](http://download.oracle.com/javase/6/docs/api/java/util/Deque.html" \o "interface in java.util)<E>, [List](http://download.oracle.com/javase/6/docs/api/java/util/List.html)<E>, [Queue](http://download.oracle.com/javase/6/docs/api/java/util/Queue.html)<E> | * May provide better performance than the ArrayListimplementation if elements are frequently inserted or deleted within the list. * Good for adding elements to the ends, i.e., stacks and queues. |
| * [**PriorityQueue**](http://download.oracle.com/javase/6/docs/api/java/util/PriorityQueue.html)   + Implements [Collection](http://download.oracle.com/javase/6/docs/api/java/util/Collection.html)<E>, [Queue](http://download.oracle.com/javase/6/docs/api/java/util/Queue.html)<E> | * The elements of the priority queue are ordered according to their natural ordering, or by a Comparator provided at queue construction time * unbounded. its capacity grows automatically. * does not permit null elements * The *head* of this queue is the *least* element * The Iterator provided in method iterator() is not guaranteed to traverse the elements of the priority queue in any particular order. |
| * [**Stack**](http://download.oracle.com/javase/6/docs/api/java/util/Stack.html) - Extends [Collection](http://download.oracle.com/javase/6/docs/api/java/util/Collection.html)<E>, [List](http://download.oracle.com/javase/6/docs/api/java/util/List.html)<E>   + [Deque](http://download.oracle.com/javase/6/docs/api/java/util/Deque.html) interface should be used in preference to the legacy [**Stack**](http://download.oracle.com/javase/6/docs/api/java/util/Stack.html) class | * E [pop](http://download.oracle.com/javase/6/docs/api/java/util/Stack.html#pop%28%29)() * E [push](http://download.oracle.com/javase/6/docs/api/java/util/Stack.html#push%28E%29)([E](http://download.oracle.com/javase/6/docs/api/java/util/Stack.html) item) * E [peek](http://download.oracle.com/javase/6/docs/api/java/util/Stack.html#peek%28%29)() |
| * [**HashMap**](http://download.oracle.com/javase/6/docs/api/java/util/HashMap.html)   + Implements [Map](http://download.oracle.com/javase/6/docs/api/java/util/Map.html)<K,V> | * Essentially an unsynchronized Hashtable that supports nullkeys and values. * Fastest updates (key/values); allows one null key, many null values. |
| * [**Hastable**](http://download.oracle.com/javase/6/docs/api/java/util/Hashtable.html)   + Implements [Map](http://download.oracle.com/javase/6/docs/api/java/util/Map.html)<K,V> | * Like a slower HashMap (as with Vector, due to its synchronized methods). * No null values or null keys allowed |
| * [**LinkedHashMap**](http://download.oracle.com/javase/6/docs/api/java/util/LinkedHashMap.html)   + Implements [Map](http://download.oracle.com/javase/6/docs/api/java/util/Map.html)<K,V> | * Predictable iteration order. * Runs nearly as fast as HashMap. * Allows one null key, many null values. |
| * [**TreeMap**](http://download.oracle.com/javase/6/docs/api/java/util/TreeMap.html)   + Implements [Map](http://download.oracle.com/javase/6/docs/api/java/util/Map.html)<K,V>, [NavigableMap](http://download.oracle.com/javase/6/docs/api/java/util/NavigableMap.html" \o "interface in java.util)<K,V>, [SortedMap](http://download.oracle.com/javase/6/docs/api/java/util/SortedMap.html" \o "interface in java.util)<K,V> | * A sorted map * Ascending element order |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Add** | **Remove** | **Get** | **Contains** | **Data  Structure** |
| **ArrayList** | O(1) | O(n) | O(1) | O(n) | Array |
| **LinkedList** | O(1) | O(1) | O(n) | O(n) | Linked List |
| **CopyonWriteArrayList** | O(n) | O(n) | O(1) | O(n) | Array |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Add** | **Contains** | **Next** | **Data Structure** |
| **HashSet** | O(1) | O(1) | O(h/n) | Hash Table |
| **LinkedHashSet** | O(1) | O(1) | O(1) | Hash Table + Linked List |
| **EnumSet** | O(1) | O(1) | O(1) | Bit Vector |
| **TreeSet** | O(log n) | O(log n) | O(log n) | Red-black tree |
| **CopyonWriteArraySet** | O(n) | O(n) | O(1) | Array |
| **ConcurrentSkipList** | O(log n) | O(log n) | O(1) | Skip List |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Offer** | **Peak** | **Poll** | **Size** | **Data Structure** |
| **PriorityQueue** | O(log n ) | O(1) | O(log n) | O(1) | Priority Heap |
| **LinkedList** | O(1) | O(1) | O(1) | O(1) | Array |
| **ArrayDequeue** | O(1) | O(1) | O(1) | O(1) | Linked List |
| **ConcurrentLinkedQueue** | O(1) | O(1) | O(1) | O(n) | Linked List |
| **ArrayBlockingQueue** | O(1) | O(1) | O(1) | O(1) | Array |
| **PriorirityBlockingQueue** | O(log n) | O(1) | O(log n) | O(1) | Priority Heap |
| **SynchronousQueue** | O(1) | O(1) | O(1) | O(1) | None! |
| **DelayQueue** | O(log n) | O(1) | O(log n) | O(1) | Priority Heap |
| **LinkedBlockingQueue** | O(1) | O(1) | O(1) | O(1) | Linked List |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Get** | **ContainsKey** | **Next** | **Data Structure** |
| **HashMap** | O(1) | O(1) | O(h / n) | Hash Table |
| **LinkedHashMap** | O(1) | O(1) | O(1) | Hash Table + Linked List |
| **IdentityHashMap** | O(1) | O(1) | O(h / n) | Array |
| **WeakHashMap** | O(1) | O(1) | O(h / n) | Hash Table |
| **EnumMap** | O(1) | O(1) | O(1) | Array |
| **TreeMap** | O(log n) | O(log n) | O(log n) | Red-black tree |
| **ConcurrentHashMap** | O(1) | O(1) | O(h / n) | Hash Tables |
| **ConcurrentSkipListMap** | O(log n) | O(log n) | O(1) | Skip List |

Apache nifi est un platform de gestion de flux de donnée en temp réel.

Mission : conception + développement de composant de platform Apache Nifi en Java pour plusieurs projets spécifique, d’intégration et traitement de données. Gérer les opérations de la platform, support technique.

Développement des webservice en utilisant Spring boot (client + serveur)

Dév de procédure pl sql (oracle exadata)

Collaborer avec les autres équipes.

Assuré : la qualité de donnée, la performance de la platform assuré que ça marche bien 24/24.

Recherche + mise en place des solutions pour que ça marche mieux : monitoring, sécurité de webservice, Oauth2, multithreading

Monitoring : développement des outils qui gère les opérations, récupéré des infor des traitements : comme le volume, le temps, le frequant arrivee de données. à partir de ça , on peux analyser, configurer, équilibrer les travaux de la platform.

Les points j’aime plus dans la mission : les technologies utilisés, les problématiques qu’on a rencontré.

\* ApacheNifi : un platform de gestion de flux de données, installé + développé la base de la platform, intégré avec autre techno.

\* Problématiques : traitement de gros fichiers, sécurité, synchronized, faire calcul en parallèle…