Le SGBD java

Olivier Caron

SGBD - GIS4

Polytech'Lille

Les possibilités de Java

- De la conception à l'implémentation (penser UML avec Java)
  - entités, associations, cardinalités, CIR, ...
- La persistance des données
- L'aspect transactionnel, concurrence, accès à distance
- L'introspection
- Les vues, la sécurité (JDK 1.2)
- La gestion des extensions

Est-ce que Java peut être considéré comme un SGBD ?

Polytech'Lille

Les possibilités de Java

- De la conception à l'implémentation (penser UML avec Java)
  - entités, associations, cardinalités, CIR, ...
- La persistance des données
- L'aspect transactionnel, concurrence, accès à distance
- L'introspection
- Les vues, la sécurité (JDK 1.2)
- La gestion des extensions

Est-ce que Java peut être considéré comme un SGBD ?

Polytech'Lille

Les possibilités de Java

- De la conception à l'implémentation (penser UML avec Java)
  - entités, associations, cardinalités, CIR, ...
- La persistance des données
- L'aspect transactionnel, concurrence, accès à distance
- L'introspection
- Les vues, la sécurité (JDK 1.2)
- La gestion des extensions

Est-ce que Java peut être considéré comme un SGBD ?

Polytech'Lille

Les possibilités de Java

- De la conception à l'implémentation (penser UML avec Java)
  - entités, associations, cardinalités, CIR, ...
- La persistance des données
- L'aspect transactionnel, concurrence, accès à distance
- L'introspection
- Les vues, la sécurité (JDK 1.2)
- La gestion des extensions

Est-ce que Java peut être considéré comme un SGBD ?

Polytech'Lille

Les possibilités de Java

- De la conception à l'implémentation (penser UML avec Java)
  - entités, associations, cardinalités, CIR, ...
- La persistance des données
- L'aspect transactionnel, concurrence, accès à distance
- L'introspection
- Les vues, la sécurité (JDK 1.2)
- La gestion des extensions

Est-ce que Java peut être considéré comme un SGBD ?

Polytech'Lille

Les possibilités de Java

- De la conception à l'implémentation (penser UML avec Java)
  - entités, associations, cardinalités, CIR, ...
- La persistance des données
- L'aspect transactionnel, concurrence, accès à distance
- L'introspection
- Les vues, la sécurité (JDK 1.2)
- La gestion des extensions

Est-ce que Java peut être considéré comme un SGBD ?

Polytech'Lille

Les possibilités de Java

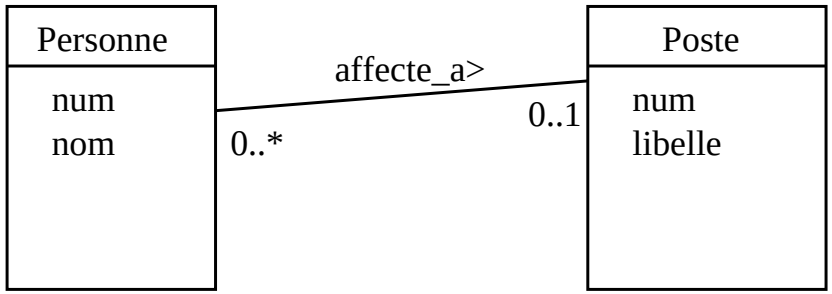
- De la conception à l'implémentation (penser UML avec Java)
  - entités, associations, cardinalités, CIR, ...
- La persistance des données
- L'aspect transactionnel, concurrence, accès à distance
- L'introspection
- Les vues, la sécurité (JDK 1.2)
- La gestion des extensions

Est-ce que Java peut être considéré comme un SGBD ?

Polytech'Lille

Conception UML

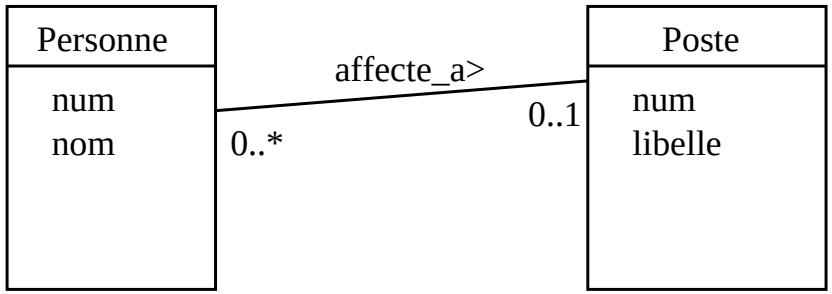
- Cardinalité 1 :



- Entité → Classe, propriété → attribut
- Association → attribut et méthodes (add, get et remove)
- Programmer les contraintes d'intégrité référentielles !

Conception UML

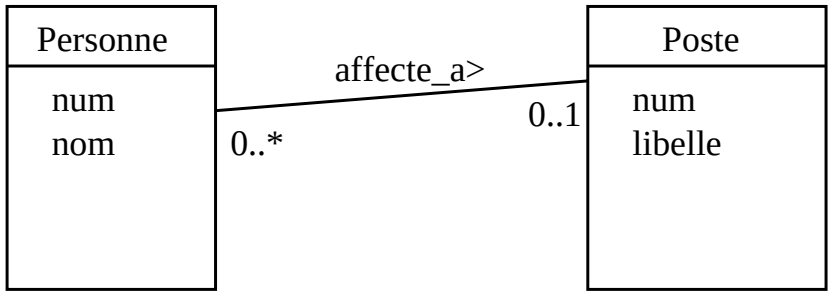
- Cardinalité 1 :



- Entité → Classe, propriété → attribut
- Association → attribut et méthodes (add, get et remove)
- Programmer les contraintes d'intégrité référentielles !

Conception UML

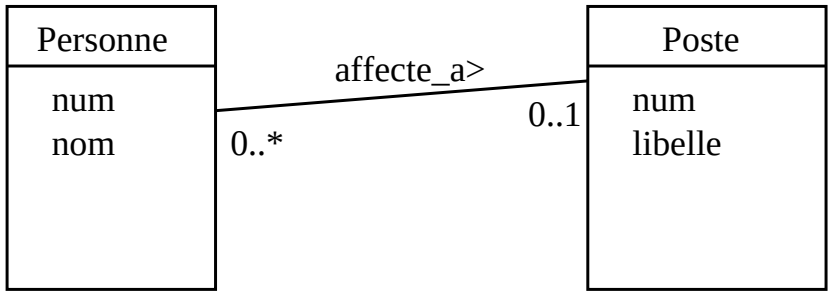
- Cardinalité 1 :



- Entité → Classe, propriété → attribut
- Association → attribut et méthodes (add, get et remove)
- Programmer les contraintes d'intégrité référentielles !

Conception UML

- Cardinalité 1 :



- Entité $\rightarrow$ Classe, propriété $\rightarrow$ attribut
- Association $\rightarrow$ attribut et méthodes (add, get et remove)
- Programmer les contraintes d'intégrité référentielles !

Avantages UML et JAVA

- UML : notation objet de conception
- Richesse de la notation : cardinalités, rôle des associations, encapsulation, interfaces, héritage...
- Les problèmes de traduction UML vers SGBD :
  - Pas de notion de clé primaire (identité de l'objet)
  - Difficile de gérer les CIR (à la norme ODMG), exemple cardinalité 1 :1

Avantages UML et JAVA

- UML : notation objet de conception
- Richesse de la notation : cardinalités, rôle des associations, encapsulation, interfaces, héritage...
- Les problèmes de traduction UML vers SGBD :
  - Pas de notion de clé primaire (identité de l'objet)
  - Difficile de gérer les CIR (à la norme ODMG), exemple cardinalité 1 :1

Avantages UML et JAVA

- UML : notation objet de conception
- Richesse de la notation : cardinalités, rôle des associations, encapsulation, interfaces, héritage...
- Les problèmes de traduction UML vers SGBD :
  - Pas de notion de clé primaire (identité de l'objet)
  - Difficile de gérer les CIR (à la norme ODMG), exemple cardinalité 1 :1

Polytech'Lille

Avantages UML et JAVA

- UML : notation objet de conception
- Richesse de la notation : cardinalités, rôle des associations, encapsulation, interfaces, héritage...
- Les problèmes de traduction UML vers SGBD :
  - Pas de notion de clé primaire (identité de l'objet)
  - Difficile de gérer les CIR (à la norme ODMG), exemple cardinalité 1 :1

Polytech'Lille

Avantages UML et JAVA

- UML : notation objet de conception
- Richesse de la notation : cardinalités, rôle des associations, encapsulation, interfaces, héritage...
- Les problèmes de traduction UML vers SGBD :
  - Pas de notion de clé primaire (identité de l'objet)
  - Difficile de gérer les CIR (à la norme ODMG), exemple cardinalité 1 :1

Polytech'Lille

La persistance des données (1/2)

- Mécanisme permettant de rendre persistant des objets Java :
  - Implémenter l'interface `java.io.Serializable`
  - Hériter d'une classe *sérialisable*
- Sauvegarder un objet :

```
FileOutputStream fic=new FileOutputStream("banque.ser") ;
ObjectOutputStream os=new ObjectOutputStream(fic)
os.writeObject(laBanque) ;
```

Polytech'Lille

La persistance des données (1/2)

- Restaurer un objet :

```
FileInputStream fic=new FileInputStream("banque.ser") ;
ObjectInputStream is=new ObjectInputStream(fic)
laBanque = (Banque) is.readObject() ;
```

- La clause `transient` (ex : une donnée calculée)
- Utilisée pour le téléchargement (applets)

Transactionnel, concurrence, accès à distance

- Un nouveau mot clé dans le langage `synchronized`
- Notion de verrou sur l'objet
- Verrou sur les méthodes ou des blocs de code au sein d'une méthode
- Accès concurrent : les threads ou objet distant
- Accès à distance : protocole Java RMI

Polytech'Lille

Transactionnel, concurrence, accès à distance

- Un nouveau mot clé dans le langage `synchronized`
- Notion de verrou sur l'objet
- Verrou sur les méthodes ou des blocs de code au sein d'une méthode
- Accès concurrent : les threads ou objet distant
- Accès à distance : protocole Java RMI

Polytech'Lille

Transactionnel, concurrence, accès à distance

- Un nouveau mot clé dans le langage `synchronized`
- Notion de verrou sur l'objet
- Verrou sur les méthodes ou des blocs de code au sein d'une méthode
- Accès concurrent : les threads ou objet distant
- Accès à distance : protocole Java RMI

Polytech'Lille

Transactionnel, concurrence, accès à distance

- Un nouveau mot clé dans le langage `synchronized`
- Notion de verrou sur l'objet
- Verrou sur les méthodes ou des blocs de code au sein d'une méthode
- Accès concurrent : les threads ou objet distant
- Accès à distance : protocole Java RMI

Polytech'Lille

Transactionnel, concurrence, accès à distance

- Un nouveau mot clé dans le langage `synchronized`
- Notion de verrou sur l'objet
- Verrou sur les méthodes ou des blocs de code au sein d'une méthode
- Accès concurrent : les threads ou objet distant
- Accès à distance : protocole Java RMI

Polytech'Lille

L'introspection (1/6)

- Tout SGBD doit fournir des moyens pour analyser dynamiquement la structure d'une table :
  Exemple postgres : tables pg_class, pg_user, ...
  Exemple JDBC : la classe `MetaBase`
- La reflexion Java permet d'analyser **dynamiquement** un composant et de le configurer dynamiquement !

Polytech'Lille

L'introspection (1/6)

- Tout SGBD doit fournir des moyens pour analyser dynamiquement la structure d'une table :
  Exemple postgres : tables pg_class, pg_user, ...
  Exemple JDBC : la classe `MetaBase`
- La reflexion Java permet d'analyser **dynamiquement** un composant et de le configurer dynamiquement !

Polytech'Lille

Analyse dynamique et invocation dynamique (1/5)

```
import java.lang.reflect.* ;
import java.beans.Beans ;
import java.util.Vector ;

public class Reflection {
  public static void main(String args[]) {
    Class laClasse ; Method lesMethodes[] ;
    Vector bools=new Vector() ;
    try {
        Object obj = Beans.instantiate(null,args[0]) ;
        laClasse = obj.getClass() ;
        lesMethodes = laClasse.getMethods() ;
        int i=0 ;
```

Polytech'Lille

Analyse dynamique et invocation dynamique (2/5)

```
while (i<lesMethodes.length) {
   if (lesMethodes[i].getName().startsWith("is")&&
      lesMethodes[i].getParameterTypes().length==0 &&
      lesMethodes[i].getReturnType().toString().equals("boolean")
      ) {
      String nomProp=lesMethodes[i].getName().substring(2) ;
      System.out.println(nomProp) ;
      bools.addElement(nomProp) ;
   }
   i++ ;
}
```

Analyse dynamique et invocation dynamique (3/5)

```
i=0 ; Object param[] =new Boolean[1] ;
param[0]=new Boolean(true) ;
while (i<lesMethodes.length) {
   if (lesMethodes[i].getName().startsWith("set") &&
      bools.contains(lesMethodes[i].getName().substring(2))) {
      System.out.println("invocation␣méthode") ;
      lesMethodes[i].invoke(obj,param) ;
   }
   i++ ;
}
```

Analyse dynamique et invocation dynamique (4/5)

```
} catch (ArrayIndexOutOfBoundsException e) {
    System.err.println("Erreur : java Reflection nomClasse")
} catch (SecurityException e) {
    System.err.println("exception raised...") ;
} catch(IllegalAccessException e) {
    System.err.println("Access...") ;
} catch(InvocationTargetException e) {
    System.err.println("Argument...") ;
} catch(IllegalArgumentException e) {
    System.err.println("Argument...") ;
} catch(java.io.IOException e) {
    System.err.println("IO...") ;
} catch(ClassNotFoundException e) {
    System.err.println("class...") ; } } }
```

Polytech'Lille

Analyse dynamique et invocation dynamique (5/5)

- On ne connait pas la structure de la classe au départ !
- Exécution de l'exemple : `java Reflection java.awt.Button`

```
Valid
Displayable
Visible...
```

Analyse dynamique et invocation dynamique (5/5)

- On ne connait pas la structure de la classe au départ !
- Exécution de l'exemple : `java Reflection java.awt.Button`

```
Valid
Displayable
Visible...
```

Les vues

- Utilisation des interfaces Java
- Java RMI utilise également ce mécanismes d'interfaces
- Difficile de modifier la structure du schéma (jointure)

La sécurité JAVA

- Des classes standards : `java.lang.SecurityManager` `java.security.Permission`
- Possibilité de définir des politiques de sécurité :

```
grant codeBase "file:/home/sysadmin/" {
    permission java.io.FilePermission "/tmp/abc", "read";
  };
```

- Possibilité de définir ses propres permissions
- Exécution du programme :

```
java -Djava.security.policy=fichier.police Main
```

Polytech'Lille

La gestion des extensions

- Richesse de l'API Java pour gérer des collections : `Vector, Hashtable, ...`
- cf Cours Bernard Carré

La gestion des extensions

- Richesse de l'API Java pour gérer des collections : `Vector, Hashtable, ...`
- cf Cours Bernard Carré

Polytech'Lille

Conclusion

- Les plus
  - L'approche objet (passage naturel de UML à Java)
  - L'API riche au niveau des collections
  - Une solution pour presque chaque aspect BD
  - simple !
- Les moins
  - L'aspect requête
  - Tout existe mais en version (trop) simplifiée
  - L'aspect performance ! !
- L'avenir : JDO, hibernate, EJB 3, ...

Polytech'Lille

Conclusion

- Les plus
  - L'approche objet (passage naturel de UML à Java)
  - L'API riche au niveau des collections
  - Une solution pour presque chaque aspect BD
  - simple !
- Les moins
  - L'aspect requête
  - Tout existe mais en version (trop) simplifiée
  - L'aspect performance ! !
- L'avenir : JDO, hibernate, EJB 3, ...

Polytech'Lille

Conclusion

- Les plus
  - L'approche objet (passage naturel de UML à Java)
  - L'API riche au niveau des collections
  - Une solution pour presque chaque aspect BD
  - simple !
- Les moins
  - L'aspect requête
  - Tout existe mais en version (trop) simplifiée
  - L'aspect performance ! !
- L'avenir : JDO, hibernate, EJB 3, ...

Polytech'Lille

Conclusion

- Les plus
  - L'approche objet (passage naturel de UML à Java)
  - L'API riche au niveau des collections
  - Une solution pour presque chaque aspect BD
  - simple !
- Les moins
  - L'aspect requête
  - Tout existe mais en version (trop) simplifiée
  - L'aspect performance ! !
- L'avenir : JDO, hibernate, EJB 3, ...

Polytech'Lille

Conclusion

- Les plus
  - L'approche objet (passage naturel de UML à Java)
  - L'API riche au niveau des collections
  - Une solution pour presque chaque aspect BD
  - simple !
- Les moins
  - L'aspect requête
  - Tout existe mais en version (trop) simplifiée
  - L'aspect performance ! !
- L'avenir : JDO, hibernate, EJB 3, ...

Polytech'Lille

Conclusion

- Les plus
  - L'approche objet (passage naturel de UML à Java)
  - L'API riche au niveau des collections
  - Une solution pour presque chaque aspect BD
  - simple !
- Les moins
  - L'aspect requête
  - Tout existe mais en version (trop) simplifiée
  - L'aspect performance ! !
- L'avenir : JDO, hibernate, EJB 3, ...

Polytech'Lille

Conclusion

- Les plus
  - L'approche objet (passage naturel de UML à Java)
  - L'API riche au niveau des collections
  - Une solution pour presque chaque aspect BD
  - simple !
- Les moins
  - L'aspect requête
  - Tout existe mais en version (trop) simplifiée
  - L'aspect performance ! !
- L'avenir : JDO, hibernate, EJB 3, ...

Polytech'Lille

Conclusion

- Les plus
  - L'approche objet (passage naturel de UML à Java)
  - L'API riche au niveau des collections
  - Une solution pour presque chaque aspect BD
  - simple !
- Les moins
  - L'aspect requête
  - Tout existe mais en version (trop) simplifiée
  - L'aspect performance ! !
- L'avenir : JDO, hibernate, EJB 3, ...

Polytech'Lille

Conclusion

- Les plus
  - L'approche objet (passage naturel de UML à Java)
  - L'API riche au niveau des collections
  - Une solution pour presque chaque aspect BD
  - simple !
- Les moins
  - L'aspect requête
  - Tout existe mais en version (trop) simplifiée
  - L'aspect performance ! !
- L'avenir : JDO, hibernate, EJB 3, ...

Polytech'Lille

Conclusion

- Les plus
  - L'approche objet (passage naturel de UML à Java)
  - L'API riche au niveau des collections
  - Une solution pour presque chaque aspect BD
  - simple !
- Les moins
  - L'aspect requête
  - Tout existe mais en version (trop) simplifiée
  - L'aspect performance ! !
- L'avenir : JDO, hibernate, EJB 3, ...

Polytech'Lille