Conventions de nommage

Projet E4: Agriotes

20/02/19

—

Abdellah, Khalil et Imed

Référence: http://www.loribel.com/java/normes/nommage.html & https://sqlpro.developpez.com/cours/standards

Table de matières:

1. conventions de java
    a. packages
    b. classes
    c. interfaces
    d. variables
    e. constants
    f. norms pour les noms de packages
        i. abstraction - Abstraction de data
        ii. exception - Classes d'exceptions
        iii. bo ou business - Objets métiers
        iv. datamgr - Data manager
        v. gui - Modules graphiques
        vi. generated - Classes générées
        vii. tools - Classes outils
        viii. images - Ressources graphiques
        ix. Exemple de structure
    g. norms pour les noms de classes
        i. introduction
        ii. XxxAbstract - Abstraction
        iii. XxxAdapter
        iv. XxxBO - Business Object
        v. XxxConfig - Configuration
        vi. XxxDefault - Implémentation par défaut
        vii. XxxConverter
        viii. XxxException
        ix. XxxFactory
        x. XxxFactoryBuilder
        xi. XxxImpl - Implémentation
        xii. XxxLaunch - Lancer d'application
        xiii. XxxLink - Link entre modules
        xiv. XxxMgr - XxxManager
        xv. XxxModel - Modèle = Données
        xvi. XxxOwner - Contient ...
        xvii. XxxPrototype
        xviii. XxxReader
        xix. XxxSingleton

xx. XxxSet
xxi. XxxSupport
xxii. XxxTemplate - Interface ou Classe template
xxiii. XxxTools - Classe utilitaire
xxiv. XxxView
xxv. XxxVM - XxxViewManager
xxvi. XxxWriter

h. Normes pour les noms de méthodes
i. Noms de variables couramment utilisées
j. Normes utilisées pour nommer les constantes

2. conventions de sql
   a. Nommage des objets
   b. Nom d'un serveur
   c. Nom d'une base de données
   d. Nom d'une entité
   e. Nom d'une relation (association)
   f. Nom d'une table, d'une vue
   g. Noms de colonnes
   h. Noms de contraintes d'une table
   i. Noms de index
   j. Noms de procédures stockées
   k. Documentation, ergonomie et écriture
      i. Introduction
      ii. Écriture des requêtes
      iii. Écriture de code
      iv. Cartouche
      v. Valeur de retour
      vi. Usages
      vii. Documentation

Conventions de JAVA

Packages

Le nom d'un package doit respecter les conventions suivantes

1. Tout en minuscule.
2. Utiliser seulement [a-z], [0-9] et le point '.': Ne pas utiliser de tiret '-', d'underscore '_', d'espace, ou d'autres caractères ($, *, accents, ...).
3. La convention de Sun indique que tout package doit avoir comme root un des packages suivant: com, edu, gov, mil, net, org ou les deux lettres identifiants un pays (ISO Standard 3166, 1981).

Exemples:

com.sun.eng
com.apple.quicktime.v2
edu.cmu.cs.bovik.cheese

Classes

Les noms des classes doivent respecter les conventions suivantes (d'après SUN):

1. 1ère lettre en majuscule
2. Mélange de minuscule, majuscule avec la première lettre de chaque mot en majuscule
3. Donner des noms simples et descriptifs
4. Eviter les acronymes : hormis ceux commum (XML, URL, HTML, ...)

De plus, d'une manière générale :

- N'utiliser que les lettres [a-z] et [A-Z] et [0-9] : Ne pas utiliser de tiret '-', d'underscore '_', ou d'autres caractères ($, *, accents, ...).

Exemples:

```
class Raster;
class ImageSprite;
```

Interfaces

Les mêmes conventions que les noms de classes s'appliquent.

D'une manière générale, je suis contre l'utilisation systématique de la lettre ' I ' pour préfixer les interfaces.

En effet, une classe peut devenir une interface sans pour autant changer toutes ses références.

```
interface RasterDelegate;
interface Storing;
interface StringConvertor;
```

Variables

Les noms des variables doivent respecter les conventions suivantes:

1. 1ère lettre en minuscule
2. Mélange de minuscule, majuscule avec la première lettre de chaque mot en majuscule
3. Donner des noms simples et descriptifs
4. Ne pas commencer les noms avec '$' ou '_' bien que ce soit possible.
5. Variable d'une seule lettre (pour un usage local)
    - int : i, j, k, m, et n
    - char : c, d, et e
    - boolean : b

De plus, d'une manière générale :

- N'utiliser que les lettres [a-z] et [A-Z] et [0-9] : Ne pas utiliser de tiret '-', d'underscore '_', ou d'autres caractères ($, *, accents, ...).

Exemples:

```
int     i;
char    c;
float   myWidth;
```

```
public int getPropertyLength(String a_name) {
    String l_property = getProperty(a_name);
    return l_property.length;
}
```

Constants

Les noms des variables doivent respecter les conventions suivantes (d'après SUN):

- Tout en majuscule
- Séparer les mots par underscore '_'
- Donner des noms simples et descriptifs

De plus, d'une manière générale :

- N'utiliser que les lettres [A-Z], [0-9] et '_' : Ne pas utiliser de tiret '-' ou d'autres caractères ($, *, accents, ...).

Exemples:

```
static final int MIN_WIDTH = 4;
static final int MAX_WIDTH = 999;
static final int GET_THE_CPU = 1;
```

Norms pour les noms de packages

Introduction

Ci dessous vous trouverez une liste de noms de packages qui peuvent être assez généraux:

Abstraction - Abstraction de data

Ce package ne contient que des interfaces. C'est bien de limiter ses liens avec le reste du système.

Exception - Classes d'exceptions

En Java, est indispensable de gérer sa propre hiérarchie d'exceptions. Etant données que des abstractions peuvent utilisé des exceptions, est intéressant de séparer ces classes dans un package à part d'autant plus que des classes d'exception doivent rester très simples et ne pas avoir de dépendances importantes.

Bo ou business - Objets métiers

Le coeur de l'application: le modèle objet de données.

En général, une partie des objets métier peuvent être générés. On mettra alors les classes généré dans le package bo.generated.

Datamgr - Data manager

Modules de traitement de données.

datamgr est le package racine de tous les modules non graphiques qui font du traitement de data.

Par exemples : convertisseur, parseur, logger, ...

GUI - Modules graphiques

GUI - Graphic User Interface - Modules d'interfaces graphiques.

gui est le package racine de tous les modules graphiques.

Afin de mieux séparer les modules graphiques, il est très adapté d'utiliser des interfaces pour les lier au reste du système.

Generated - Classes générées

Il est toujours bon de séparer les classes générées des autres classes.

Par exemple pour les classes générés d'objets métiers, on utilisera bo.generated.

Tools - Classes outils

Les classes d'outils peuvents être vues comme des bibliothèques de fonctions. Cette technique peut, à première vu, paraître rétrograde avec une orientation procédurale. En fait, elles s'intègrent parfaitement dans une méthodologie purement objet et permet une meilleure abstraction, simplifie les api, ...

En effet, dans beaucoup de cas ces classes travaillent sur des interfaces. Pour un exemple concret, il suffit de regarder java.util.Collections.

Images - Ressources graphiques

En général c'est bien de regrouper toutes les ressources graphiques d'un projet dans un package image:

Par exemple les images génériques (icon windows, ...) dans 2 sous packages x16 et x32 en fonction de la dimension des images, respectivement 16x16 et 32x32.

```
images
  |- x16
  |- x32
  |- ...
```

Exemple de structure

Voici un exemple de structure:

```
com.myproject
  |- abstraction
  |- business
  |    |- generated
  |- datamgr
  |    |- ...

  |    |- ...

  |- exception
  |- gui
  |    |- ...

  |    |- ...

  |- images
  |    |- x16
  |    |- x32
  |    |- ...

  |- tools
  |    |- comparator
```

Normes pour les noms de classes

Introduction

Ci dessous vous trouverez une liste de noms de classes qui peuvent être assez généraux:

XxxAbstract - Abstraction

Classe abstraite racine de différentes implémentations.

On préfèrera XxxAbstract à AbstractXxx qui est moins standard mais qui permet de garder les classes d'un même type ensemble (lorsque trier alphabétiquement).

XxxAdapter

Classe abstraite qui implémente les méthodes d'une interface sans rien faire.

ex: MouseAdapter, ...

XxxBO - Business Object

Classe Business Object.

La notion d'objet métier ou de 'business object'. Dans certains cas, pour identifier de tels objets on utilise la convention XxxBO.

Mais attention, d'une manière générale, il ne faut pas qu'un nom contiennent le type. Plus qu'un type, la notion de 'busines object' peut-êre vue de manière sémantique.

XxxConfig - Configuration

Classe ou interface de configuration.

Une classe qui contient une configuration (ex: ConsoleConfig).

Afin de rentrer les noms à la fois courts et compréhensible, on utilisera Config plutôt que Configuration. Quelque soit votre choix, soyez consistent et utiliser toujours la même règle pour une même contraction.

XxxDefault - Implémentation par défaut

Implémentation par défaut d'une interface ou d'une classe abstraite.

Dans de nombreux cas on favorisera les trois niveaux :

- Xxx : Abstraction sous forme d'interface
- XxxAbstract : Implementation abstraite partielle
- XxxDefault : Implémentation par défaut.

XxxConverter

Classe ou interface de conversion.

Classe qui est utilisée pour convertir des données, des logs...

ex:

- DateStringConverter : Convertit une date en String et réciproquement.
- LogConverter : Convertit des logs...

XxxException

Une bonne pratique consiste à créer ses propres instances d'exception. Toujours utiliser un nom XxxException pour bien les identifier.

XxxFactory

Interface ou Classe factory.

- Interface : En général ne contiendra qu'une méthode newXxx(--).
- Classe : Permet de créer des objets.

XxxFactoryBuilder

Builder de Factory.

Permet de créer une factory. Dans ce cas, la Factory sera définie comme une interface.

- static newFactory() : retourne l'instance d'une factory

Exemple : voir framework XML contenu dans le JDK.

XxxImpl - Implémentation

Implémentation par défaut.

Exemples : interface LabelIcon / class LabelIconImpl

XxxLaunch - Lancer d'application

Classe avec un main pour lancer une application.

Bien souvent les gens font une classe dérivant de classe Swing (par exemple JFrame) avec une méthode main pour lancer leur application.

Il est préférable de séparer la classe de lancement des classes graphique. Dans cette classe on retrouvera les appels aux initialisations, et l'appel à la classe dérivant de JFrame correspondant à notre application.

Cela permet par exemple de lancer dans la même VM (virtual machine) plusieurs applications, de modifier les initialisation et d'utiliser les même interface graphique, ...

XxxLink - Link entre modules

Interface permettant de lier des modules.

Quand on veut faire un module indépendant du reste du système, on pourra regrouper toutes ces interactions dans une interface afin de rendre ce module indépendant.

Les classes qui voudront utiliser ce module devront alors définir une implémentation de cette interface pour utiliser le module.

Dans le cas où c'est possible, on préférera l'usage d'événement plutôt que de définir une telle interface. (Ex : modification du contenu d'une liste, ...).

Voir aussi XxxVM - XxxViewManager.

Cas classique(s) d'utilisation :

- Module GUI : Un module ayant une interface graphique représentant des données et pouvant réagir avec le système (Extension du pattern MVC).
    - XxxViewManager : le manager du module
    - XxxView : la vue

- XxxModel : le modèle, contient toutes les données utiles
- XxxLink : les liens avec le système (de type action par exemple)

XxxMgr - XxxManager

Classe ou interface Manager.

Une classe qui manage des data, des log, ... (ex: DataMgr, LogMgr).

La même remarque que XxxConfig - XxxConfiguration : Utiliser plutôt XxxMgr et surtout soyez consistent.

XxxModel - Modèle = Données

Interface permettant d'accéder à des données.

On l'utilisera souvent quand on utilise le pattern MVC (Model-View-Controller).

Voir XxxVM - XxxViewManager pour plus de détails.

XxxOwner - Contient ...

Interface Owner.

- En général ne contiendra qu'une méthode getXxx() : Xxx

XxxPrototype

Interface Prototype.

- En général ne contiendra qu'une méthode newXxx() : Xxx

Un prototype est une sorte de factory, il peut instancier des objets. On utilisera un prototype pour instancier toujours le même type d'objet contrairement à une factory qui peut instancier des objets différents.

Un Classe qui implémente une interface de type prototype peut créer d'autres instances de cette même Classe.

Cas classique(s) d'utilisation :

- Factory : Permet de créer une factory qui crée des instances en fonction d'un nom ou autre paramètre.
    - Pour cela, créer un map de prototypes indexés par les noms.
    - On évite de faire un switch en utilisant la map.
    - Permet d'avoir une implémentation de la factory qui est indépendante des objets à créer. En effet on peut utiliser une methode addPrototype(name, prototype)

XxxReader

Classe ou interface Reader.

XxxSingleton

Classe Singleton.

- static getInstance() : retourne l'instance par défaut du Singleton.
- static getInstance(--) : Dans le cas où l'on veut gérer une liste de singletons de même type.
- constructeur private, ou protected si on veut prévoir plusieurs implémentations possibles (utilisation de sous classes).
- static initWith(XxxSingleton) : L'avantage important d'un singleton par rapport à l'utilisation de méthodes statiques réside dans le fait que l'on peut définir plusieurs implémentations. Dans le cas où je veux pouvoir définir ou paramétrer une instance de ce singleton sans passer par des méthodes publiques, j'utilise la méthode initWith pour initialiser le singleton avec une instance particulière.

Dans de nombreux cas, on pourra ne pas utiliser le suffixe Singleton. Par contre on indiquera dans les commentaires de la classe qu'il s'agit d'un singleton.

XxxSet

Interface de deuxième niveau avec 'setter'.

Exemple: LabelIcon LabelIconSet LabelIconImpl.

- interface LabelIcon
    - getLabel() : String
    - getIcon(int) : Icon
- interface LabelIconSet
    - implements LabelIcon
    - setLabel(String)
    - putImage(int, Icon)
- classe concrète LabelIconImpl
    - implémentation par défaut

XxxSupport

Classe qui permet de gérer les évènements par délégation.

Les méthode de cette classe sont en général :

- addXXXListener(XXXListener)
- removeXXXListener(XXXListener)
- fireXXX(XXXEvent)

ex: PropertyChangeSupport, ...

XxxTemplate - Interface ou Classe template

Une interface template définit les méthodes qui permettent par exemple à un objet de se représenter selon un template.

XxxTools - Classe utilitaire

Classe utilitaire.

En général, la classe sera abstraite et ne contiendra que des méthodes statiques.

Pour des questions de performance, on pourra mettre toutes les méthodes finales.

Si plusieurs implémentations sont possibles, on utilisera un singleton plutôt qu'une classe Tools avec des méthodes statiques.

XxxView

Classe View utilisée comme view dans le pattern MVC (Model-View-Controller).

Voir XxxVM - XxxViewManager pour plus de détails.

XxxVM - XxxViewManager

Classe ViewManager utilisé comme manager dans le pattern MVC (Model-View-Controller).

En général le viewManager recevra en paramètre le model de type XxxModel, et construira une vue de type XxxView.

Voir aussi XxxLink.

XxxWriter

Classe ou interface Writer.

Normes pour les noms de méthodes

getXxx() : Object

Getter.

setXxx(Object) : void

Setter.

isXxx() : boolean

Getter pour un valeur boolean.

getAllXxx()

Retourne une collection des éléments.

createXxx()

Crée un objet.

newXxx()

Crée un objet.

updateXxx()

Met à jour un valeur

addXxx()

Ajoute un valeur

addNewXxx()

Crée un nouveau valeur et l'ajoute

addAllXxx()

Ajoute des valeurs

paramString()

Retourne la liste des paramètres d'une classe pour mode debug.

Cette méthode est appelée par toString(). Voir le cas des composants swing.

Exemple de la méthode pour JComboBox.

```
protected String paramString() {
    return super.paramString() +
        ",isEditable=" + isEditableString +
        ",lightWeightPopupEnabled=" + lightWeightPopupEnabledString +
        ",maximumRowCount=" + maximumRowCount +
        ",selectedItemReminder=" + selectedItemReminderString;
}
```

removeXxx()

Supprime un valeur.

removeAllXxx()

Supprime des valeurs.

toXxx()

Fait une conversion vers un autre type.

checkXxx()

Méthode de validation.

Noms de variables couramment utilisées

len - Nombre d'éléments dans une liste ou un tableau

Pour la variable représentant la taille d'un tableau ou le nombre d'éléments d'une liste quand on veut boucler sur ses éléments.

Ceci est plus optimal que d'utiliser dans la condition la méthode size() ou la propriété length`.

```
int len = list.size();
Object l_item;
for int i=0; i<len; i++) {
    l_item = list.get(i);
    ...
}
```

retour - Utilisation de cette variable pour retourner la valeur d'une fonction

Beaucoup de méthodes retournent une valeur. Pour nous simplifier la vie, et normer notre code, on crée une variable nommée retour qui me permet de retourner la valeur de retour. Ainsi, on a une vision simple de ce qui est retourné, et nous n'avons pas besoin de chercher un autre nom.

```
public List buildMyList() {
    List retour = new ArrayList();
    retour.add(...);
    ...

    return retour;
}
```

e / ex - Paramètre pour une exception.

On utilise très souvent la variable e dans un bloc try catch. Dans le cas de plusieurs catch, on utilise e2, e3, ...

On utilise de plus en plus ex car c'est ce qu'utilisent certains générateurs comme JBuilder.

```
try {
    ...
} catch (Exception ex) {
    ex.printStackTrace();
}

try {
    ...
} catch (IOException ex) {
    ex.printStackTrace();
} catch (SAXException ex2) {
    ex2.printStackTrace();
}
```

it - Iterateur

```
Iterator it = l_list.iterator();
String l_str;
while (it.hasNext()) {
    l_str = (String) it.next();
    ...
}
```

Normes utilisées pour nommer les constantes

Ci dessous vous trouverez une liste de suggestions de noms de constants:

- **BUTTON_xxx - Label d'un bouton.**
- **CHECK_xxx - Label d'une case à cocher (checkBox) ou radio bouton (radioButton).**
- **COL_xxx - En Tête d'une colonne dans une table.**
- ERR_xxx - Message d'une exception, ou message d'erreur.
- EXCEPTION_xxx - Message d'une exception, ou message d'erreur.
- GROUP_xxx - Titre d'un groupe.
- ICON_xxx - Nom de fichier pour une icone ou image.
- IMG_xxx - Nom de fichier pour une image.
- INFO_xxx - Info pour les boîtes d'attente ou écran de démarrage.
- GROUP_xxx - Titre d'un groupe.
- ICON_xxx - Nom de fichier pour une icone ou image.
- IMG_xxx - Nom de fichier pour une image.
- INFO_xxx - Info pour les boîtes d'attente ou écran de démarrage.
- NODE_xxx - Label d'un noeud (TreeNode) dans un arbre.
- QUESTION_xxx - Question qui est affiché dans une boite de dialogue.
- STR_xxx - String utilisé à divers endroits.
- TAB_xxx - Nom d'un onglet.
- TITLE_xxx - Titre d'une fenêtre ou d'un panel.
- TOOLTIP_xxx - Description d'un tool tip.
- UNIT_xxx - Unité.
- ZONE_xxx - Titre d'une zone.

Conventions de SQL

Nommage des objets

Les noms des objets d'un modèle ou d'un schéma devront être significatifs et pertinents. Ils devront être constitué uniquement des caractères suivants :

[a .. z] + [A .. Z] + [0 .. 9] +[_]

Avec les restrictions suivantes :

- ne pas dépasser 128 caractères
- ne doit pas commencer par un chiffre
- ne peut avoir plusieurs caractères "blanc souligné" de suite
- la casse n'a pas d'importance
- le nom ne doit pas être un mot réservé de SQL (voir à ce sujet : LES MOTS CLEFS DU SQL)

ATTENTION : Autrement dit, les lettres accentuées (é à ù ï É ...), les "kanas" (ç œ ...), les caractères de ponctuation (, ; : ! ? ...) et autres caractères spéciaux, comme le blanc, sont proscrits.

Exemples :

_TOTO	Autorisé
123TOTO	Interdit
TITI__TATA	Interdit
_toto	Autorisé (mais identique au premier)
Vérité	Interdit

ATTENTION :

On réservera les noms en minuscule aux modèles logiques et les noms en majuscule aux modèles physique.

Exemple :

client	nom logique (entité par exemple)
T_CLIENT_CLI	nom physique (table par exemple)

et leur longueur devra tenir compte des limites du nombre de caractères utilisables dans le nom des objets du SGBDR et des variables du langage de programmation interne au SGBDR (triggers et procédures stockées).

Nom d'un serveur

Le nom d'un serveur doit commencer par le préfixe SRV_ suivi d'une indication pertinente.

Exemples :

SRV_GESTION	pour un serveur de base de données de gestion
SRV_FO	pour un serveur "front office"

Nom d'une base de données

Le nom d'une base de donnée doit commencer par le préfixe BD_ suivi d'une indication pertinente.

Exemples :

BD_GESCOM	Base de données de GEStion COMmerciale
BD_SUIVIPROD	Base de données de SUIVI de PRODuction

Nom d'une entité

Elle doit commencer par un préfixe :

e_	lorsqu'il s'agit d'entité fonctionnelle
er_	lorsqu'il s'agit d'entité de référence
es_	lorsqu'il s'agit d'entité "système"

ATTENTION : L'emploi du pluriel est à proscrire.

Exemples :

e_client	entité fonctionnelle des clients
er_pays	entité de référence des pays
es_user	entité système des utilisateurs

Nom d'une relation (association)

Elle doit commencer par le préfixe R :

Exemple :

r_loue	relation "loue" entre entité e_client et e_maison
r_achete	relation achète entre e_client et e_maison

Nom d'une table, d'une vue

Elle doit commencer par un préfixe :

T_	lorsqu'il s'agit d'une table fonctionnelle	V_
TR_	lorsqu'il s'agit d'une table de référence	VR_
TS_	lorsqu'il s'agit d'une table "système"	VS_
TJ_	lorsqu'il s'agit d'une table de jointure	VJ_
TG_	lorsqu'il s'agit d'une table générique (héritage)	VG_

Elle doit reprendre le corps du nom de l'entité, où à défaut (table de jointure) le nom de la relation si cette dernière est nommée.

Elle doit être suffixée par un **trigramme unique** au sein de la base de données, permattant l'identification rapide de la table.

Exemples :

T_CLIENT_CLI	table fonctionnelle des clients
TR_PAYS_PAY	table de référence des pays
TS_USER_USR	table système des utilisateurs
TJ_ACHETE_ACH	table de jointure relation "achete"

Une vue sera systématiquement préfixée par V_.

Noms de colonnes

Les noms de colonnes doivent être préfixée par le trigramme de la table d'origine et reprendre le nom d'attribut.

Exemple :

CLI_NOM	colonne nom de la table T_CLIENT
CLI_DATE_NAISSANCE	colonne date de dnaissance de la table T_CLIENT
FAC_DATE_EMISSION	colonne date d'émission de la table facture

Certaines abréviation d'usage courant devront être utilisées :

ID	identifiant (en général clef auto incrémentée)
NUM	numéro
REF	référence
CODE	code, codage, codification
LIB	libellé
ORD	ordre
CP	code postal
ADR	adresse
FAX	télécopie
MAIL	e-mail
TEL	téléphone
GSM	téléphone mobile
LOG	"login"
PWD	"password"
NBR	nombre
QTE	quantité
POS	position
NDX	index
MNT	montant
TX	taux

PCT	pourcentage
PUTC	prix unitaire toutes taxes
PUHT	prix unitaire hors taxes

Noms de contraintes d'une table

Les contraintes de table devront être préfixées C_ suivi d'un indicateur du type de contrainte. Elles seront suffixées par le trigramme de table.

Indicateur de type de contrainte :

C_PK_	contrainte de table "clé primaire"
C_FK_	contrainte de table "clef étrangère"
C_UNI_	contrainte de table "unicité"
C_CHK_	contrainte de table "validité"

- Les contraintes de clé primaire doivent reprendre le trigramme de la table d'origine
- Les contraintes de clé étrangère doivent reprendre le trigramme de la table d'origine et celui de la table dans laquelle elles figurent
- Les contraintes d'unicité et de validité doivent avoir un nom significatif

Exemples :

C_PK_CLI	contrainte de clef primaire de la table client
C_FK_FAC_CLI	contrainte de clef étrangère de la table facture dans la table client
C_UNI_LOGIN_PASSWORD_CLI	contrainte d'unicité pour les colonnes LOGIN et PASSWORD
C_CHK_CP_CLI	contrainte de validité d'un code postal

Noms de index

Les index doivent être préfixés X_ suivi d'un indicateur de la nature de l'index et d'un nom significatif. Ils seront suffixées par le trigramme de table.

On pourra choisir comme indicateur de nature, parmi les abréviations suivantes :

X_CSR_	index en cluster
X_BMP_	index "bitmap"
X_BTR_	index arbre équilibré
X_HCG_	index, "clef de hachage"

Exemples :

X_CSR_ID_CLI	index clusterisé de l'indentifiant du client
X_BTR_NOM_PRENOM_CLI	index arbre equilibré pour nom/prenom de client
X_HCG_TEL_CLI	index en clef de hachage pour téléphone client

Noms de procédures stockées

Les procédures stockées seront préfixées par SP_ suivi d'un nom significatif.

Exemple :

SP_CALC_TARIF	procédure stockée de calcul des tarifs

Documentation, ergonomie et écriture

Introduction

Les règles ci dessous établissent la manière dont les développeurs doivent écrire les requêtes et le code afférent aux objets d'une base de données.

Écriture des requêtes

chaque clause de requête devra être indentée :

MAUVAIS !!!

```
SELECT CLI_ID, CLI_NOM,
CLI_ENSEIGNE,
CLI_PRENOM  FROM T_CLIENT
WHERE CLI_ADR_PAYS = 'F'
AND CLI_ENSEIGNE IS NULL OR
CLI_ENSEIGNE =''
ORDER BY CLI_NOM
```

BON !!!

```
SELECT CLI_ID, CLI_NOM,
CLI_ENSEIGNE,
       CLI_PRENOM
FROM   T_CLIENT
WHERE  CLI_ADR_PAYS = 'F'
       AND CLI_ENSEIGNE IS
NULL OR CLI_ENSEIGNE =''
ORDER  BY CLI_NOM
```

Toutes les colonnes renvoyées devront être nommées :

MAUVAIS !!!

```
SELECT CLI_ID, CLI_PRENOM
|| ' ' || CLI_NOM,
       CLI_ENSEIGNE
FROM   T_CLIENT
```

BON !!!

```
SELECT CLI_ID,
       CLI_PRENOM || ' ' ||
CLI_NOM AS NOM_COMPLET_CLI,
       CLI_ENSEIGNE
FROM   T_CLIENT
```

Les noms des colonnes renvoyées ne doivent pas comporter de doublons (en particulier l'usage de l'étoile dans la clause SELECT est à proscrire) :

MAUVAIS !!!

```
SELECT *
FROM   T_CLIENT_CLI
CLI
       INNER JOIN
T_FACTURE_FAC FAC
             ON
CLI.CLI_ID =
FAC.CLI_ID
WHERE
CLI.CLI_ADR_PAYS = 'F'
```

BON !!!

```
SELECT CLI.CLI_ID, CLI.TIT_CODE,
CLI.CLI_NOM,
       CLI.CLI_PRENOM,
CLI.CLI_ENSEIGNE, FAC.FAC_ID,
       FAC.PMT_CODE, FAC.FAC_DATE,
FAC.FAC_PMT_DATE
FROM   T_CLIENT_CLI CLI
       INNER JOIN T_FACTURE_FAC FAC
             ON CLI.CLI_ID =
FAC.CLI_ID
WHERE  CLI.CLI_ADR_PAYS = 'F'
-- un seule fois la colonne CLI_ID
```

BON !!!

```
SELECT CLI.CLI_ID AS CLI_CLI_ID,
FAC.CLI_ID AS FAC_CLI_ID,
       CLI.TIT_CODE, CLI.CLI_NOM,
       CLI.CLI_PRENOM,
CLI.CLI_ENSEIGNE, FAC.FAC_ID,
       FAC.PMT_CODE, FAC.FAC_DATE,
FAC.FAC_PMT_DATE
FROM   T_CLIENT_CLI CLI
       LEFT OUTER JOIN
T_FACTURE_FAC FAC
             ON CLI.CLI_ID =
FAC.CLI_ID
WHERE  CLI.CLI_ADR_PAYS = 'F'
-- deux fois la colonne CLI_ID avec
deux noms différents
```

Écriture de code

Si l'éditeur texte du code en offre la possibilité, on utilisera toujours une police a espacement fixe telle que la police Courrier.

Le code devra être écrit exclusivement en majuscule, sauf contrainte particulière. Cette règle s'explique dans le sens ou il est important dans un langage hôte de pouvoir faire la distinction entre le code exécuté sur le poste client et le code envoyé et exécuté sur le serveur.

Le renommage des tables dans les requêtes se fera à l'aide du trigramme.

En cas de présence de plusieurs instances de la même table en ajoutera un numéro.

Les différentes clauses et partie de clauses des requêtes devront être indentées avec un retrait d'au moins 3 caractères par type d'item. De même si dans les requêtes figure un branchement CASE.

Les jointures de table devront toujours être réalisées avec l'opérateur JOIN lorsque celui ci est disponible.

Exemple :

```
SELECT CLI.CLI_ID, FAC1.FAC_ID,
       CASE FAC1.FAC_MODE_PAIEMENT
            WHEN 'B' THEN 'Chèque bancaire'
            WHEN 'E' THEN 'Espèces'
            WHEN 'C' THEN 'Carte de paiement'
ELSE ''
       END AS FAC_MODE_PAIEMENT
FROM   T_CLIENT_CLI CLI
       INNER JOIN T_FACTURE_FAC FAC1
             ON CLI.CLI_ID = FAC1.CLI_ID
WHERE  CLI.CLI_ADR_PAYS = 'F'
GROUP  BY FAC1.FAC_ID, CLI.CLI_ID
HAVING SUM(FAC1.FAC_MONTANT_TTC) > (SELECT MAX(FAC2.FAC_MONTANT_TTC)
                                    FROM   T_FACTURE_FAC FAC2
                                    WHERE  FAC2.CLI_ID = CLI.CLI_ID)
ORDER BY CLI.CLI_ID, FAC_MODE_PAIEMENT
```

Dans les procédures stockées, l'indentation sera faite :

- pour les requêtes comme indiqué précédemment
- pour les blocs de code, par un retrait pour chaque bloc d'au moins 3 caractères

On veillera en outre à placer des commentaires courts de manière judicieuse.

Exemple :

```
CREATE PROCEDURE SP_DEV_SUPPRESSION
       @id_element integer,
       @recursif   bit
AS

DECLARE @OK integer
DECLARE @bg_element integer
DECLARE @bd_element integer
DECLARE @intervalle integer

SET NOCOUNT ON

-- démarrage transaction
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE
BEGIN TRANSACTION DELETE_TREE

-- vérifie de l'existence de l'élément
SELECT @OK = count(*)
FROM   T_DEVELOPPEMENT_DEV
WHERE  DEV_ID = @id_element

-- si élément supprimé, alors retour sans insertion avec valeur -1
IF @OK = 0
BEGIN
   SELECT -1
   ROLLBACK TRANSACTION DELETE_TREE
   RETURN
END
...
```

Cartouche

Toute procédure stockée, trigger ou vue devra être pourvue d'un cartouche dans le code. Le cartouche devra contenir les éléments suivants :

- le nom du développeur ayant codé l'objet
- la date de réalisation
- Un commentaire succinct
- la référence de la nomenclature si une nomenclature des éléments de code a été établie
- la description des paramètres
- s'il y a lieu, les modifications successives apportées au code (identifiant du codeur, date et nature)

Exemples :

```
-------------------------------------------------------
-- ACTOR V 3 - ARBODEV - GA01
-------------------------------------------------------
-- Alfred DURAND - 2001-02-17
-------------------------------------------------------
-- Gestion de l'arborescence : insertion d'un fils ainé
-- retourne 0 en cas d'anomalie
-- PARAMETRES :
--    en entrée :
--       id du père : DEV_ID (integer)
--    en sortie :
--       id du fils inseré : DEV_ID (integer)
-------------------------------------------------------
-- MODIFICATION
-- 2001-05-11 #1 - Martine DUPONT [MDP] :
--    bug lors du calcul si dd = df
-- 2001-05-11 #2 - Martine DUPONT [MDP] :
--    bug lors du calcul si dd > df
-------------------------------------------------------
```

```
/* ================================== */
/* GESTION DE L'ARBRE DU DÉVELOPPEMENT */
/* suppression                        */
/* ================================== */
/* Frédéric BROUARD    20/11/2001     */
/* ================================== */
/* suppression d'un élément ou d'un   */
/* sous arbre                         */
/* ================================== */
/* paramètres :                       */
/*   id_element : integer             */
/*   recursif   : bit                 */
/* Si @recursif = 0 =>                */
/*    supression d'un élément         */
/* si @recursif = 1 =>                */
```

```
/*    supression d'un sous arbre       */
/* ================================== */
```

Valeur de retour

Dans la mesure du possible une procédure renverra toujours une valeur de retour sous la forme d'un entier, permettant de connaître l'état d'exécution de la procédure.

En cas de succès de la procédure, cette valeur de retour sera 0.

En cas de problème cette valeur sera :

- une valeur négative en cas d'exception (erreur)
- une valeur positive pour des valeur limites d'exécution et des conditions d'exécution imprévues

Usages

Voici quelques règles en usage dans l'écriture des requêtes permettant d'en optimiser l'exécution:

MAUVAIS	BON	Pourquoi?
`SELECT * ...`	`SELECT col1, col2 ...`	Le SGBDR doit faire un effort important pour rechercher les colonnes adéquates.
`... col>='val1' AND col=<'val2' ...`	`... col BETWEEN 'val1' AND 'val2' ...`	L'opérateur BETWEEN est optimisé (sinon à quoi servirait-il ?)

`COUNT(col) ...`	`COUNT(*) ...`	Préférez le COUNT(*), le moteur va piocher dans les statistiques, coût voisin de zéro !
`CASE col WHEN ... THEN ... ELSE ... END`	`COALESCE(...)` ou `UNION`	Remplacer les CASE par des COALESCE ou des opérations ensemblistes de type UNION, chaque fois que cela est possible, car la structure CASE est d'un coût exorbitant.
`SELECT DISTINCT ...`	`SELECT ...`	Évitez le mot clef DISTINCT lorsque cela n'est pas d'une absolue nécessité. Le distinct oblige à dédoublonner donc trier et si les résultats sont unique c'est du temps de perdu.
`... IN (1, 2, 3) ...`	`... BETWEEN 1 AND 3 ...`	Évitez le IN lorsque le BETWEEN suffit
`... WHERE col1+1 = col2+5 ...`	`... WHERE col1 = col2 + 4`	Simplifiez les expressions en ayant si possible une seule colonne indexée de part ou d'autre des opérateurs de comparaison, afin d'activer les index.
`... WHERE EXISTS (SELECT Col1, Col2 ...`	`... WHERE EXISTS (SELECT * ...`	Dans ce cas (requête imbriquée avec opérateurs EXISTS) l'optimiseur remplace le caractère * est remplacé par une constante appropriée

`... WHERE EXISTS (SELECT ...`	`... WHERE ... IN (SELECT ...`	Lorsque cela s'avère possible, remplacez l'opérateur [NOT] EXISTS par un opérateur [NOT] IN.
`... WHERE col1 IN (SELECT col1 ...`	`...FROM table1 t1 INNER JOIN table2 t2 ON t1.col1 = t2.col1 ...`	Lorsque cela est possible remplacer les sous requêtes avec opérateur IN par des jointures.
`VARCHAR`	`CHAR`	Préférez le CHAR Lorsque la colonne de la table est sollicité en recherche et/ou que l'on y ajoute un index.
`NCHAR, NVARCHAR`	`CHAR, VARCHAR`	Préférez le CHAR/VARCHAR au NCHAR/NVARCHAR lorsque l'application n'est pas multilange. Le coût de stockage est divisé par deux.
`FLOAT`	`DECIMAL`	Pour les calculs financiers qui ne doivent pas générer d'erreurs d'écarts d'arrondis.

Documentation

On veillera à implanter dans la base de données, une table permettant de décrire les objets de la base.

Une telle table, de nom TS_DESCRIPTION_DSC pourra prendre la forme suivante :

TS_OBJ_NAME_DSC	Nom de l'objet

TS_OBJ_TYPE_DSC	Nature de l'objet (table, vue, procédure, fonction, trigger...)
TS_ATB_NAME_DSC	Nom de l'attribut
TS_ATB_TYPE_DSC	Type d'attribut (colonne de table, paramètre de procédure ou de fonction)
TS_ATB_ORDER_DSC	Position ordinale de l'attribut
TS_ATB_LENGTH_DSC	Longueur de l'attribut
TS_ATB_REQUIRED_DSC	Obligatoire
TS_DESCRIPTION_DSC	Description de l'objet (à destination des utilisateurs)
TS_OBSERVATION_DSC	Annotation de l'objet (à destination des développeurs et administrateurs)
TS_ACCESS_RULE_DSC	Règle d'accès (par exemple, combinaison binaire pour : 1 : utilisateurs, 2 : administrateurs, 4 : développeurs, 8 : chefs de projet ...)
TS_APPLICATION_DSC	Règle d'utilisation par les applications clientes (par exemple, combinaison binaire pour : 1 : paye, 2 : comptabilité, 4 : gestion commerciale, 8 : ...)