Avisé

Table of Contents

Chapter 1. Introduction

Chapter 2. Couche de persistance

Bonne pratiques et utilisation de la couche de persistance de l'application.
La couche de persistance est assuré par `Hibernate` via `SpringData`.

2.1. Script SQL

La création de la base, les modifications sont piloté par les script `SQL` situés (pour le moment) ici : http://svnhm/svn/grc_src/avise-ng/avise-sql.

Les scripts sont localisé dans le repertoire de `tags` car ceux-ci sont propre à une version, définie par le nom du tag.
Ils doivent pouvoir être (re)joués de manière idempotente ! (nous verons plus loin comment le faire sans problème).

Ils sont au nombre de 3 :

- `00_Patch_AVISE_FUNCTION.sql`
- `10_Patch_AVISE_DB.sql`
- `20_Patch_AVISE_DATA.sql`

2.1.1. Nommage

Concernant le nommage des colonnes, table et index, peu de règles.

- Le nom de tables et de colonne doivent être en anglais.
- Les colonne de clé primaire doivent respecter ce pattern : `[NOM TABLE]_id_pk`
- Les séquences doivent être préfixé par `SEQ_`
- les contrainte et index doivent être préfixé par :

Prefixe	Description
`PK_`	Contrainte de clé primaire
`FK_`	Contrainte de clé étrangère
`UX_`	Index unique
`IX_`	Index simple

2.1.2. Les fonctions de base

script : `00_Patch_AVISE_FUNCTION.sql`

Les fonctions de base regroupe les actions disponible pour la modification de la structure de la base.

Pour ajouter une fonction, il faut faire un `drop` si elle existe et après seulement faire le `create` afin de pouvoir la jouer de manière idempotente.

Exemple :

```
DROP FUNCTION IF EXISTS CREATE_TABLE(table_name text, create_stmt text);
CREATE OR REPLACE FUNCTION CREATE_TABLE(table_name text, create_stmt text)
RETURNS text AS $_$
BEGIN
  ...
END;
$_$ LANGUAGE plpgsql;
```

2.1.3. Structure de base

script : `10_Patch_AVISE_DB.sql`

Ce fichier permet de définir la structure de la base. *Il est important d'utiliser les fonctions de base afin de garantir l'idempotence de l'execution de cette création de structure*.

Exemple pour la création de la table utilisateur :

```
select CREATE_SEQUENCE('SEQ_WEB_USER', 'CREATE SEQUENCE SEQ_WEB_USER START WITH 1
INCREMENT 1 CACHE 1');
select create_table('web_user',
    'CREATE TABLE web_user (
        web_user_id_pk         BIGINT       NOT NULL CONSTRAINT PK_web_user_id_pk
PRIMARY KEY DEFAULT nextval(''SEQ_WEB_USER''),
        login                  VARCHAR(100) NOT NULL,
        first_name             VARCHAR(250),
        last_name              VARCHAR(255),
        administrator          BOOLEAN,
        last_connection_date   TIMESTAMP,
        token                  VARCHAR(255),
        token_generation_date TIMESTAMP
    )'
);
select create_index('WEBUSER_LOGIN', 'WEB_USER', 'LOGIN', null, true);
SELECT create_index('WEBUSER_FIRSTNAME_LASTNAME', 'WEB_USER', 'FIRST_NAME, LAST_NAME',
null, false);
```

Nous preconisons de réaliser la création de table selon cet ordre :

- Création de la séquence
- Déclaration de la création de la table
- Ajout de contraintes
- Ajout d'index

NOTE	En ce qui concerne la clé primaire, nous preferons en faire la déclaration, ***quand celle-ci est basé sur une colonne unique***, sur la ligne même de la déclaration de la colonne. Et en affectant la valeur de la séquence comme valeur par défaut.

2.1.4. Les données

script : `20_Patch_AVISE_DATA.sql`

Ce script permet l'insertion des données de l'application, données d'initialisation, de référence, etc. Mais aussi la modification de celles-ci.

Comme pour les autres, ces insertions/modifications, *doivent pouvoir être joué de manière imdempotente*.

Ce fichiers dont être conclu par une mise à jour du numéro de version :

```
select update_version('AVISE_SQL_18.01.00_01');
```

2.2. Les entités

Toutes les entités (comme les dto) sont outillé par `lombok`, avec les annotations `@Data`, `@NoArgsConstructor` et `@AllArgsConstructor` (quoique la définition des constructeur ne soit necessaire que si vous avez besoin des deux constructeur).
Ce chapitre ne presente aucune difficulté, mais permet l'introduction des élément suivant

Petit exemple simple :

```
@Entity @Table(name = "web_user")
@SequenceGenerator(name = "SEQ_WEB_USER", sequenceName = "SEQ_WEB_USER",
allocationSize = 1)
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {

    @Id @Column(name = "web_user_id_pk")
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "SEQ_WEB_USER")
    private Long id;

    @Column(name = "login", length = 100)
    private String login;

    ...

}
```

2.3. CMD

C'est à partir de la couche de persistance que nous commançons à mettre en place la notion CMD, CMD pour ***C**reate*, ***M**odification* & ***D**eletion*.
Système qui a été mis en place pour gérer la mémorisation de la création, (*dernière*) modification et suppression d'un élément de la base de donnée. En indiquant pour chaque information, le timestamp et l'utilisateur responsable de l'action.
Sauf pour la suppression qui permet une gestion de la suppression logique. Car dès qu'un élément à une information de deletion non null, il doit être concidéré comme supprimé de manière logique.

2.3.1. CMD & SQL

Pour être stocké au niveau de la base, les information CMD requière 5 colonnes :

- `creation TIMESTAMP NOT NULL`
- `creator VARCHAR(100) NOT NULL`
- `modification TIMESTAMP`
- `modificator VARCHAR(100)`
- `deletionTIMESTAMP`

Cepedant, il ne faut les ajouter à la main, mais executer une fonction SQL mis à votre disposition :

user.sql

```
select create_table('web_user',
  'CREATE TABLE web_user (
      web_user_id_pk BIGINT NOT NULL CONSTRAINT PK_web_user_id_pk PRIMARY KEY DEFAULT
nextval(''SEQ_WEB_USER''),
      ...
  )'
);
...
select ADD_CMD_INFORMATION('web_user');
```

L'appel de la fonction `ADD_CMD_INFORMATION` permet l'ajout des 5 colonnes si celles-ci n'existe pas et rempli par défaut les valeur creation et creator avec les valeurs par défaut [`dete/heure du jour`] & [`patch-db`].

2.3.2. CMD & Entité

Au niveau des entités, l'inclusion des informations CMD va permettre de :

- Inclure les information de création / modification
- Exclure les éléments supprimés logiquement

Pour les inclure les colonne au niveau d'une entité. Il faudra faire heriter l'entité d'une super classe abstraite `FollowedEntity` qui permet d'exposer la composition `FollowUp` comme portion embaquée.

Et il faudra inclure une clause de selection pour exclure les enregistrement ayant une valeur pour la colonne `deletion`.

Exemple :

user.java

```
@Entity @Table(name = "web_user")
@SequenceGenerator(name = "SEQ_WEB_USER", sequenceName = "SEQ_WEB_USER",
allocationSize = 1)
@Data
@NoArgsConstructor
@AllArgsConstructor
@Where(clause = "deletion is null") ①
public class User extends FollowedEntity { ②
    ...
}
```

① Ajout de la clause d'exclusion

② Heritage de la super classe exposant la composition

Chapter 3. Les services

Chapter 4. Gestion du cache

L'application utilise ehCache (version 3).
La gestion du cache est défini dans les fichier ehCache-[`env-classifier`].xml qui sont transmis avec la configuration globale de l'application.
Pour les env autre que DEV & INT, les variables et leur valeurs seront à transmettre aux équipe en charge de mettre en place les env.

Voici les caches actuels :

Cache	**unité**	**valeur**
ContractIdentity	Cache des informations d'identité d'un contrat	
	CONTRACT_ID_TTL_UNIT	CONTRACT_ID_TTL_VALUE
	`hours`	`1`
	CONTRACT_ID_HEAP_UNIT	CONTRACT_ID_HEAP_VALUE
	`entries`	`1000`
InfiniteContract	Cache des informations de contrat (INFINITE)	
	INFINITE_CONTRACT_TTL_UNIT	INFINITE_CONTRACT_TTL_VALUE
	`minutes`	`15`
	INFINITE_CONTRACT_HEAP_UNIT	INFINITE_CONTRACT_HEAP_VALUE
	`entries`	`500`
InfiniteClient	Cache des informations client (INFINITE)	
	INFINITE_CLIENT_TTL_UNIT	INFINITE_CLIENT_TTL_VALUE
	`minutes`	`15`
	INFINITE_CLIENT_HEAP_UNIT	INFINITE_CLIENT_HEAP_VALUE
	`entries`	`500`
ContratAssureCouverture	Cache des informations (INFINITE) de couverture assuré	
	INFINITE_COUV_TTL_UNIT	INFINITE_COUV_TTL_VALUE
	`minutes`	`15`
	INFINITE_COUV_HEAP_UNIT	INFINITE_COUV_HEAP_VALUE
	`entries`	`500`
ContratAssureCouvertureDetaillee	Cache des informations (INFINITE) détaille de couverture assuré	
	INFINITE_COUV_DET_TTL_UNIT	INFINITE_COUV_DET_TTL_VALUE
	`minutes`	`15`
	INFINITE_COUV_DET_HEAP_UNIT	INFINITE_COUV_DET_HEAP_VALUE
	`entries`	`500`
AssureContrat	Informations (INFINITE) assuré lié à un contrat	
	INFINITE_ASS_TTL_UNIT	INFINITE_ASS_TTL_VALUE
	`minutes`	`15`
	INFINITE_ASS_HEAP_UNIT	INFINITE_ASS_HEAP_VALUE
	`entries`	`500`

Cache	unité	valeur
Stage	Informations (INFINITE) de "*stage*"	
	INFINITE_STAGE_TTL_UNIT	INFINITE_STAGE_TTL_VALUE
	minutes	15
	INFINITE_STAGE_HEAP_UNIT	INFINITE_STAGE_HEAP_VALUE
	entries	500
RattachementRoBeneficiaire	Rattachement RO des bénéficiaires (INFINITE)	
	INFINITE_RATT_RO_BEN_TTL_UNIT	INFINITE_RATT_RO_BEN_TTL_VALUE
	minutes	15
	INFINITE_RATT_RO_BEN_HEAP_UNIT	INFINITE_RATT_RO_BEN_HEAP_VALUE
	entries	500
AdminUrlApi	Informations (INFINITE) de "*AdminurlApi*"	
	ADMIN_URL_API_TTL_UNIT	ADMIN_URL_API_TTL_VALUE
	days	1
	ADMIN_URL_API_HEAP_UNIT	ADMIN_URL_API_HEAP_VALUE
	entries	500
AdminTranscoding	Gestion des transcoding	
	INFINITE_ADMIN_TRANSCO_TTL_UNIT	INFINITE_ADMIN_TRANSCO_TTL_VALUE
	minutes	15
	INFINITE_ADMIN_TRANSCO_HEAP_UNIT	INFINITE_ADMIN_TRANSCO_HEAP_VALUE
	entries	500
GroupNameUser	Liste des groupes pour les utilisateurs	
	GROUP_NAME_USER_TTL_UNIT	GROUP_NAME_USER_TTL_VALUE
	days	10
	GROUP_NAME_USER_HEAP_UNIT	GROUP_NAME_USER_HEAP_VALUE
	entries	1500
ErrorDescription	Liste des descriptif d’erreur	
	ERROR_DESCRIPTION_TTL_UNIT	[ERROR_DESCRIPTION_TTL_VALUE]
	days	10
	ERROR_DESCRIPTION_HEAP_UNIT	ERROR_DESCRIPTION_HEAP_VALUE
	entries	1500

Cache	unité	valeur
messageInformation	Message d’information	
	MESSAGE_INFORMATION_TTL_UNIT	[MESSAGE_INFORMATION_TTL_VALUE]
	`days`	`10`
	MESSAGE_INFORMATION_HEAP_UNIT	MESSAGE_INFORMATION_HEAP_VALUE
	`entries`	`1`

Les valeurs indiquées, sont les valeurs actuelle prise en compte en intégration.

Chapter 5. Les API REST

Chapter 6. Les services INFINITE

Listes des services INFINITE actuellement appelé par AVISé

6.1. consulterClient

- lireContratClient
- lireRattachementRoBeneficiaire

6.2. paiementPrestationSante

- lireDecompteContrat
- lireDetailDecompte
- lireConsommation

6.3. contratSante

- lireOperationsContrat
- lireAssureEtDroitsContrat

Chapter 7. Appendix - Fonctions SQL (structure)

`CREATE_TABLE`

Création d'une table si elle n'existe pas

- Nom de la table
- requête `SQL` de création de la table

`DROP_TABLE`

Suppression de la table

- Nom de la table

`CREATE_INDEX`

Création d'un index

- Nom de l'index
- Nom de la table
- Liste des colonnes de l'index séparé par une virgule sous forme de texte
- Contrainte d'index additionnelle
- Index unique ou non

exemple

```
select CREATE_INDEX('TEST_ID', 'TEST', 'ID', '', false);
select CREATE_INDEX('TEST_ID', 'TEST', 'ID, LABEL', '', true);
```

Les contrainte d'index additionnelle peuvent être necessaire si votre index embarque une colonne pouvant être null.
Car sous `postgrSQL` concidère que la valeur `NULL` n'en est pas vraiment une et ne peut pas être indexé.
Si c'est le cas, vous devrez déclarer deux index, un pour les valeur nulle et un pour les valeur non nulle.

```
select CREATE_INDEX('TEST_ID_NULL', 'TEST', 'ID, LABEL', 'LABEL is NULL', true);
select CREATE_INDEX('TEST_ID_NOTNULL', 'TEST', 'ID, LABEL', 'LABEL is not NULL', true
);
```

`DROP_INDEX`

Suppression de l'index

- Nom de la table

`ADD_COLUMN`

Ajout d'une colonne

- Nom de la table
- Nom de la colonne
- Type de la colonne

`CREATE_SEQUENCE`

Création d'une sequence si elle n'existe pas

- Nom de la séquence
- Requête SQL de création de la requête

`ADD_CMD_INFORMATION`

Ajout des informations CMD si elle n'existent pas

- Nom de la table