RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Ifremer

Documentation **Historisation des données HR Access**

Détection de modifications
Affichage utilisateur

Hugo SALOU – 2024-08-21

Table des matières

I. Introduction.

L'objectif de ce projet est, dans un premier temps, d'*historiser* les données de HR Access. Cette historisation permet de connaître l'évolution du paramétrage HR Access, d'effectuer des traitements en cas de mise à jour de certains champs, *etc.*

Dans un second temps, l'objectif est de *visualiser* ces mouvements à l'aide d'une interface web en OpenUI5.

II. Détection de mouvements en PL/SQL.

L'historisation utilise quatre tables. Les tables historisation clé-valeur et historisation occurrences sont utilisées pour conserver l'historique des modifications. Les tables historisation occurrences veille et actuel servent à la détection de changements.

HISTORISATION CLÉ-VALEUR
→ **Date de traitement**
→ **Structure**
→ **Information**
→ **Numéro de dossier**
→ **Numéro de ligne**
→ Arguments de tri
→ **Clé (Rubrique)**
→ Valeur avant
→ Valeur après

HISTORISATION OCCURRENCES
→ **Date de traitement**
→ **Structure**
→ **Information**
→ **Numéro de dossier**
→ **Numéro de ligne**
→ Arguments de tri
→ Données en JSON

HISTORISATION OCCURRENCES VEILLE
→ **Structure**
→ **Information**
→ **Numéro de dossier**
→ **Numéro de ligne**
→ Arguments de tri
→ Données en JSON

HISTORISATION OCCURRENCES ACTUEL
→ **Structure**
→ **Information**
→ **Numéro de dossier**
→ **Numéro de ligne**
→ Arguments de tri
→ Données en JSON

Table 1 – *Tables utilisées pour l'historisation*

Deux vues matérialisées sont également utilisées pour optimiser la détection de modifications. Elles permettent de filtrer les occurrences dans les tables veille et actuel par structure.

HISTORISATION OCCURRENCES VEILLE VUE
→ **Information**
→ **Numéro de dossier**
→ **Numéro de ligne**
→ Arguments de tri
→ Données en JSON

HISTORISATION OCCURRENCES ACTUEL VUE
→ **Information**
→ **Numéro de dossier**
→ **Numéro de ligne**
→ Arguments de tri
→ Données en JSON

Table 2 – *Vues matérialisées utilisées pour l'historisation*

II. A. Processus de détection de mouvements.

Le processus de détection des mouvements se déroule en trois phases. Ces trois phases sont réalisées pour chaque structure (pour l'instant, ZE, ZD et ZY, mais d'autres peuvent être ajoutées par la suite).

1. *Préparation des tables veille et actuel*
   a. On vide la table veille
   b. On transfère les données de actuel dans veille
   c. On ré-initialise les données de actuel (*c.f.* Section II. B)
   d. On met à jour les vues matérialisées avec la structure donnée

2. *Comparaison global entre veille et actuel*
   a. On compare globalement le JSON (vu comme une chaîne de caractères)
   b. On en déduit le type de mouvement (insertion, suppression, mise à jour)
   c. On ajoute ces changements dans la table historisation occurrences

3. *Comparaison clé par clé entre veille et actuel*
   a. On récupère les clés à comparer
   b. Sur les occurrences modifiées, on calcule les différences pour une clé donnée
   c. On répète *b.* pour chacune des clés à comparer

La fonction `compare` réalise l'entièreté du processus. En cas de problème, un e-mail est envoyé automatiquement. Cette fonction appelle deux autres fonctions `prepare_comparaison` qui réalise l'étape *1.* (sauf *1. d.*) du processus ci-dessus, et `lance_comparaison` qui réalise les étapes *1. d.*, *2.* et *3.* du processus.

II. A. 1. Procédure `prepare_comparaison`

Pour l'étape *1. a.*, on supprimes toutes les occurrences d'une structure donnée (ce n'est qu'un **`DELETE`** filtré par structure).

Pour l'étape *1. b.*, on récupère et insère les données de occurrences dans veille. Pour le moment, cette étape est réalisée à l'aide d'une requête de la forme **`INSERT ... SELECT`**, mais une vue matérialisée pourrait amé-

liorer la performance de cette opération. Au moment de l'écriture de ce document, cette amélioration n'est que très minime, on ne gagne qu'environ 3 s avec une vue matérialisée. Ceci requiert cependant que la « préparation à la comparaison » soit réalisée pour toute les structures en une étape (on transfère toutes les données de actuel dans veille peu importe la structure).

Pour l'étape *1. c.*, on réalise une initialisation/sauvegarde des données. Ce processus est décrit dans la Section II. B.

II. A. 2. Procédure `lance_comparaison`

La procédure `lance_comparaison` commence par rafraîchir les deux vues matérialisées (procédure `update_vues_materialisees`), puis elle compare les données extraites du jour avec celles extraites la veille (procédure `compare_jour_veille`)

Cette procédure `compare_jour_veille` réalise les étapes *2.* (avec la procédure `compare_complet`) puis *3.* (extraction des clés puis comparaison clé par clé avec la procédure `compare_cles`).

Lors de l'appel de `compare_complet`, une seule requête est effectuée pour trouver les différences entre veille et actuel pour une structure donnée. Ces différences sont ensuite insérées dans la table occurrences. La comparaison des `clobs` contenant les données en JSON se réalise avec la fonction `dbms_lob.compare`. Afin de détecter les insertions et suppressions, la jointure entre occurrences veille et occurrences actuel est *externe*. Cette jointure est réalisée sur les informations, numéro de dossiers et sur les numéros de lignes.

Lors de l'appel de `compare_cles`, on compare, sur les données modifiées uniquement, les valeurs des champs avant (*i.e.* veille) et après (*i.e.* actuel) la modification. Ces valeurs peuvent être extraites du JSON à l'aide de la fonction `json_value` : l'appel

```
json_value(
    '{"cle1": "valeur1", "cle2": "valeur2"}', -- JSON
    '$.cle1' -- selecteur JSON au format '$.CLE'
)
```

renvoie `'valeur1'`.

Ces différences de valeurs de champs sont insérées dans la table clé-valeurs.

II. B. Processus d'initialisation (sauvegarde) des données.

Ce processus d'initialisation/de sauvegarde des données est réalisée par la procédure `remplir_actuel`. On procède répertoire par répertoire (procédure `traite_repertoire_actu`), puis information par information (table `DI35` et procédure `traite_table_actu`).

Dans ce processus, on peut choisir de traiter *tous* les répertoires (pour ZE par exemple), ou *une partie* uniquement (pour ZY par exemple, on n'historise que SAL). Également, certaines tables sont exclues de l'historisation (par exemple, pour ZD, les tables `ZD03`, `ZD09` et `ZD9R` sont ignorées). Ces informations (répertoires à traiter, tables ignorées) sont définies dans l'en-tête du *package* `P_HISTORISATION_DONNEES`.

Pour chaque information (et donc chaque table), on crée dynamiquement une requête SQL (fonction `cree_req_json_actu`) qui extrait les données de la table donnée (les colonnes/rubriques sont données par la table `DI60`, *via* la fonction `cree_json_colonnes`). Elle les insère directement dans la table occurrences actuel.

Ces requêtes utilisent la fonction `json_object` qui n'est disponible qu'après Oracle 12. De plus, au lieu de `varchar2`, nous utilisons des `clob` pour stocker les données JSON : ils permettent de stocker jusqu'à 4 Go en format texte. Ceci permet d'éviter les dépassements de taille lors de la sauvegarde de chaînes dont la taille est la taille maximale d'un `varchar2`.

Certains dossiers n'ont pas besoin d'être historisés (par exemple, les anciens employés de l'IFREMER partis depuis plusieurs années). Pour cela, il est plus efficace d'initialiser la totalité des données, puis de supprimer les dossiers non historisés. Réaliser le filtre au moment de l'initialisation implique de filtrer de multiples fois (nombre de répertoires multiplié par le nombre d'informations), il est donc plus rapide de ne filtrer qu'une fois.

II. C. Représentation des différentes fonctions du package.

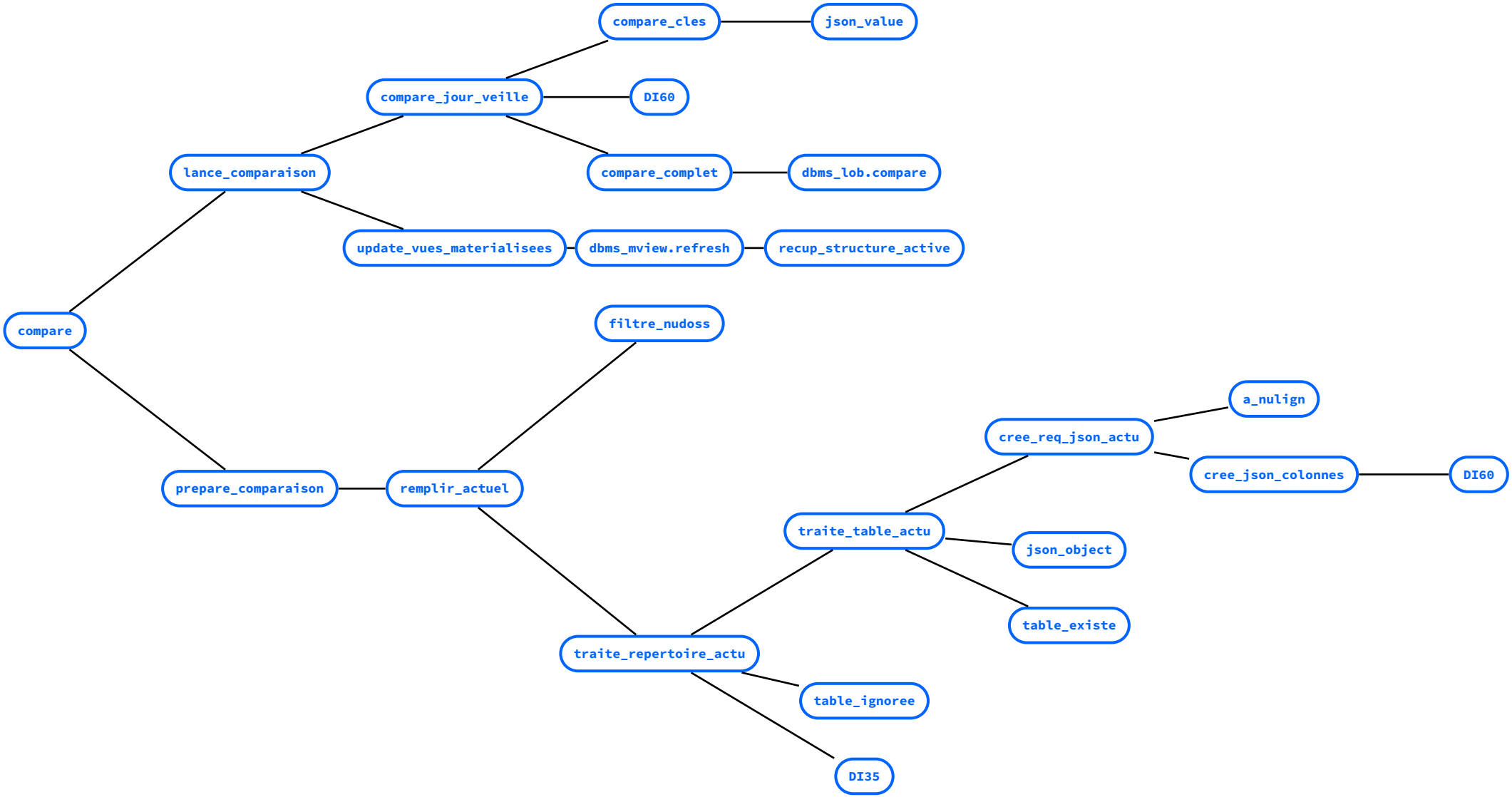


Figure 1 – *Fonctions et procédures définies dans le package et interactions entre elles*

II. D. Analyse du temps d'exécution.

Cette historisation prend un temps relativement important : en moyenne, elle prend près de 45 minutes pour ZE, ZD, et ZY. C'est principalement les données de ZY qui ralentissent le traitement (pour ZE et ZD uniquement, le processus prend un peu moins d'une minute).

La Table 3 rassemble les temps d'exécution pour chaque étape du processus d'historisation, structure par structure.

Il est important de remarquer que, par l'ordre de traitement des structures (ZE puis ZD et enfin ZY), le rafraîchissement des deux vues matérialisées pour ZE implique de vider les données de ZY (qui y sont depuis la dernière exécution). Ceci permet d'expliquer le délai anormalement long : il est même plus long que celui de ZD, bien que ZE contienne beaucoup moins de données à historiser que ZD.

		ZE		ZD		ZY			
Préparation	Suppression veille	0 min 01 s	± 0 min 01 s	0 min 12 s	± 0 min 05 s	4 min 58 s	± 1 min 21 s		
	Copie veille -> actuel	0 min 23 s	± 0 min 19 s	0 min 41 s	± 0 min 30 s	6 min 51 s	± 1 min 40 s		
	Suppression actuel	0 min 02 s	± 0 min 01 s	0 min 11 s	± 0 min 04 s	5 min 42 s	± 1 min 25 s		
	Initialisation actuel	0 min 00 s	± 0 min 00 s	0 min 11 s	± 0 min 04 s	6 min 36 s	± 1 min 49 s		
	Suppression des dossiers inutiles	0 min 00 s	± 0 min 00 s	0 min 00 s	± 0 min 00 s	2 min 11 s	± 0 min 49 s		
	TOTAL	0 min 26 s	± 0 min 19 s	1 min 15 s	± 0 min 31 s	26 min 18 s	± 3 min 15 s		
Comparaison	Update vues matérialisées	6 min 45 s	± 1 min 45 s	0 min 50 s	± 0 min 14 s	7 min 48 s	± 2 min 04 s		
	Comparaison globale	0 min 05 s	± 0 min 02 s	0 min 06 s	± 0 min 00 s	1 min 51 s	± 0 min 33 s		
	Comparaison clé-valeur	0 min 01 s	± 0 min 01 s	0 min 01 s	± 0 min 00 s	0 min 03 s	± 0 min 07 s		
	TOTAL	6 min 51 s	± 1 min 45 s	0 min 57 s	± 0 min 14 s	9 min 42 s	± 2 min 08 s		
TOTAL		**7 min 17 s**	**± 1 min 46 s**	**2 min 12 s**	**± 0 min 34 s**	**36 min 00 s**	**± 3 min 53 s**	**45 min 29 s**	**± 4 min 19 s**

Table 3 – *Temps d'exécution du processus d'historisation*

III. Visualisation des mouvements en OpenUI5.

L'interface Web se décompose en plusieurs pages, accessibles *via* différentes URLs :

- `/` Accueil
  - `/console` Page « recherche » dans la console
    - `/console/graphs` Pages « graphiques » dans la console
    - `/console/details` Pages « détails » dans la console
  - `/ds9` Interface constantes de paie
  - `/drc` Interface rubriques de paie

Chacune de ces pages a sa propre vue et son propre contrôleur.

III. A. Interface console.

Cette interface se décompose en une série d'écrans. Ces écrans ne sont pas toujours sur différentes pages, ils correspondent parfois à des « *panels* ».

III. A. 1. Écran « recherche ».

Cet écran permet premièrement de sélectionner la structure à visualiser (ZE, ZD ou ZY au moment de l'écriture de ce document). Chaque structure a ses propres filtres. Par exemple, sur ZD, on filtre sur la réglementation, le répertoire et le code ; mais, sur ZY, on filtre sur le nom ou sur le matricule.

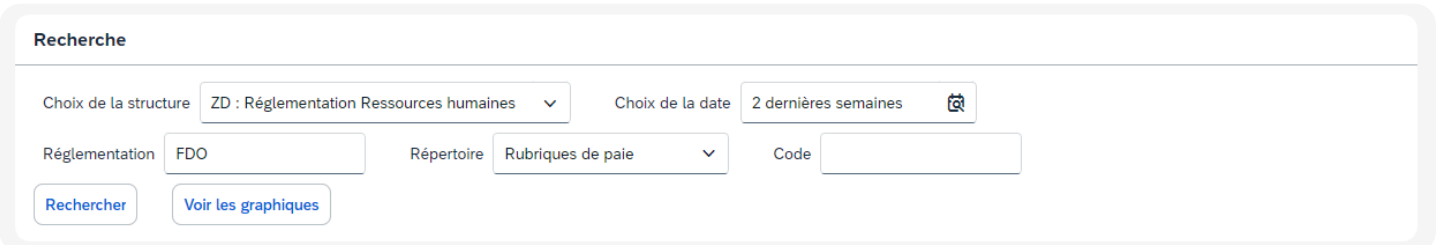


Figure 2 – *L'écran « recherche » (ici, la structure choisie est ZD)*

Les champs servant de filtres pour chaque structure sont définis dans le fichier « `structures.json` ». Ce fichier se situe sur le serveur, et il est chargé au début de l'exécution du serveur Node JS. Attention, en cas de modification de ce fichier « `structures.json` », il est **nécessaire** de relancer le serveur.

Certains champs peuvent être définis comme « *autocomplete* », ceux-ci utilisent donc une fonction SQL qui permet d'extraire les libellés associés au « code brut » (par exemple `di_rep_liblon` sur ZD).

En fonction du type de champs (*autocomplete* ou non), deux *vues imbriquées* différentes peuvent être utilisés pour le sélecteur : si le champ n'est pas *autocomplete*, la vue utilisée est `InputText`, si le champ est *autocomplete*, alors c'est la vue `InputComboBox` qui est utilisée.

Les champs *autocomplete* envoient une requête au serveur ayant la forme ci-dessous. Elle renvoie les codes et les libellés associés pour le champ sélectionné (ici, le champ répertoire).

*/structures/**ZD**/autocomplete/**CDSTCO***

Ceci permet de remplir la *ComboBox* avec les libellés des répertoires. **Attention**, au delà d'un certain nombre de codes (plus de 5 000), l'interface ralentit fortement.

Un champ `DynamicDateRange` est ajouté (depuis le contrôleur) afin de sélectionner la période de recherche.

Au lancement de la page, une requête est envoyé au serveur avec le chemin `/structures`. Cette requête récupère, sur le serveur, le contenu du fichier JSON « `structures.json` », ainsi que les libellés des structures historisées (*via* la `DI11`)

Au niveau modèle de données OpenUI5, le contrôleur définit un modèle, qui contient les paramètres de recherche, c'est à dire les valeurs des différents champs de recherche, pour toutes les structures. Ce modèle contient ainsi des données comme ci-après.

```
├── /selectedStructure = "ZD"
├── /ZD
│   ├── /ZD/CDREGL = "FDO"
│   ├── /ZD/CDSTCO = "DRC"
│   └── /ZD/CDCODE = ""
├── /ZY
│   ├── /ZY/NOMUSE = "FDO"
│   └── /ZY/MATCLE = ""
├── /dateStart = 2024-07-07 00:00:00
└── /dateEnd = 2024-07-21 23:59:59
```

Les noms des champs « filtres » (c'est à dire ceux sous `/ZD/` ou `/ZY/`) dans ce modèle correspondent aux noms des colonnes dans la base SQL, dans la table `[structure]00` (*i.e.* `ZD00` pour ZD, `ZY00` pour ZY, *etc*).

Lorsque le bouton « Rechercher » est appuyé, on appelle la fonction nommée `search` dans le contrôleur. Elle récupère les paramètres de recherche pour la bonne structure et les transmet au serveur par une requête de la forme de la *requête ci-dessous*.

```
/changes/codes?structure=ZD
              &CDREGL=FDO
              &CDSTCO=DRC
              &CDCODE=
              &dateSart=2024-07-07%2000:00:00
              &dateEnd=2024-07-21%2023:59:59
```

Sur le serveur, cette requête récupère les codes modifiés et leur libellé, en filtrant les résultats à l'aide des filtres dans la requête.

Une fois les résultats transmis au client web, le tableau « sélection du code » est rempli.

III. A. 2. Écran « sélection du code ».

Dans cet écran, il n'y a qu'un tableau à trois colonnes : code, libellé, et détails.

Figure 3 – *L'écran « sélection du code »*

Dans la colonne code, on place le « code » du dossier modifié dans la structure donnée. Ce « code » est composé d'une ou plusieurs valeurs. Par exemple, dans ZD, un « code » est un triplet réglementation, répertoire et code dans le répertoire. Mais, dans ZY, c'est un matricule.

Pour définir ce qu'est un code, dans le fichier « `structures.json` », on définit les colonnes à utiliser pour le code.

Le libellé associé est récupéré à l'aide de la fonction `recup_libelle`. Elle récupère, peu importe la structure, le libellé d'un dossier. Par exemple, pour ZD, c'est `LIBLON` dans `ZD01` ; mais, pour ZY, c'est `NOMUSE` + `PRENOM` dans `ZY00`. Cette fonction essaie, si plusieurs langues sont disponibles, d'avoir un libellé en français, mais sinon elle prendra un libellé en anglais, ou en italien, de telle sorte à ce qu'on puisse comprendre même si le libellé n'existe pas en français.

La colonne détails ne contient que des boutons qui permettent d'accéder à l'écran « informations » sur la page « détails ». On accède donc à l'URL `/console/details?{arguments de recherche}`. Ces arguments de recherches sont les mêmes que ceux de la *requête ci-avant*.

En haut de l'écran suivant, un *export* Excel est disponible. Ce processus est expliqué en Annexe B. Ce fichier Excel contient plusieurs feuilles, correspondant aux différentes informations. Sur chaque feuille, on retrouve les mouvements et les rubriques modifiées (et uniquement les rubriques qui ont été modifiées).

III. A. 3. Écran « sélection de l'information ».

Cet écran permet de filtrer l'information affichée lors de la visualisation des mouvements. Par « information », on l'entend au sens HR Access, c'est donc la table modifiée dans la base SQL.

Figure 4 – *L'écran « sélection de l'information »*

Sur cette page, deux mises en page (sur deux onglets différents) existent : la vue « console », et la vue « listing ». En vue « console », les informations modifiées sont affichées dans une colonne à gauche, et les détails des modifications sont à droite. On peut redimensionner ces colonnes. En vue « listing » (la vue par défaut), les informations modifiées sont en haut, et les détails des modifications en bas.

Afin de récupérer les codes et libellés des informations modifiées, une requête est effectuée au serveur ayant la forme de la *requête ci-dessous*.

```
/changes/infos/FDO-DS9-PFV?structure=ZD
                           &CDREGL=FDO
                           &CDSTCO=DRC
                           &CDCODE=
                           &dateSart=2024-07-07%2000:00:00
                           &dateEnd=2024-07-21%2023:59:59
```

Les paramètres de recherche sont encore dans la requête, même s'ils ne sont plus utiles (hormis les filtres `dateStart` et `dateEnd`).

Une fois l'information sélectionnée, elle apparaît comme active (le fond devient bleu). Les détails des modifications sont ensuite chargés dans l'écran « détails ».

III. A. 4. Écran « détails ».

Dans cet écran, les détails des modifications sont affichés : les insertions, suppressions et mise à jours de rubriques.

Chaque modification d'occurrence est affichée dans son « panel » dédié. Des étiquettes permettent de différencier les insertions, les suppressions et les mise à jour. **Attention**, si il n'y a qu'une seule ligne de l'occurrence à être modifiée, alors seule cette ligne sera affichée.

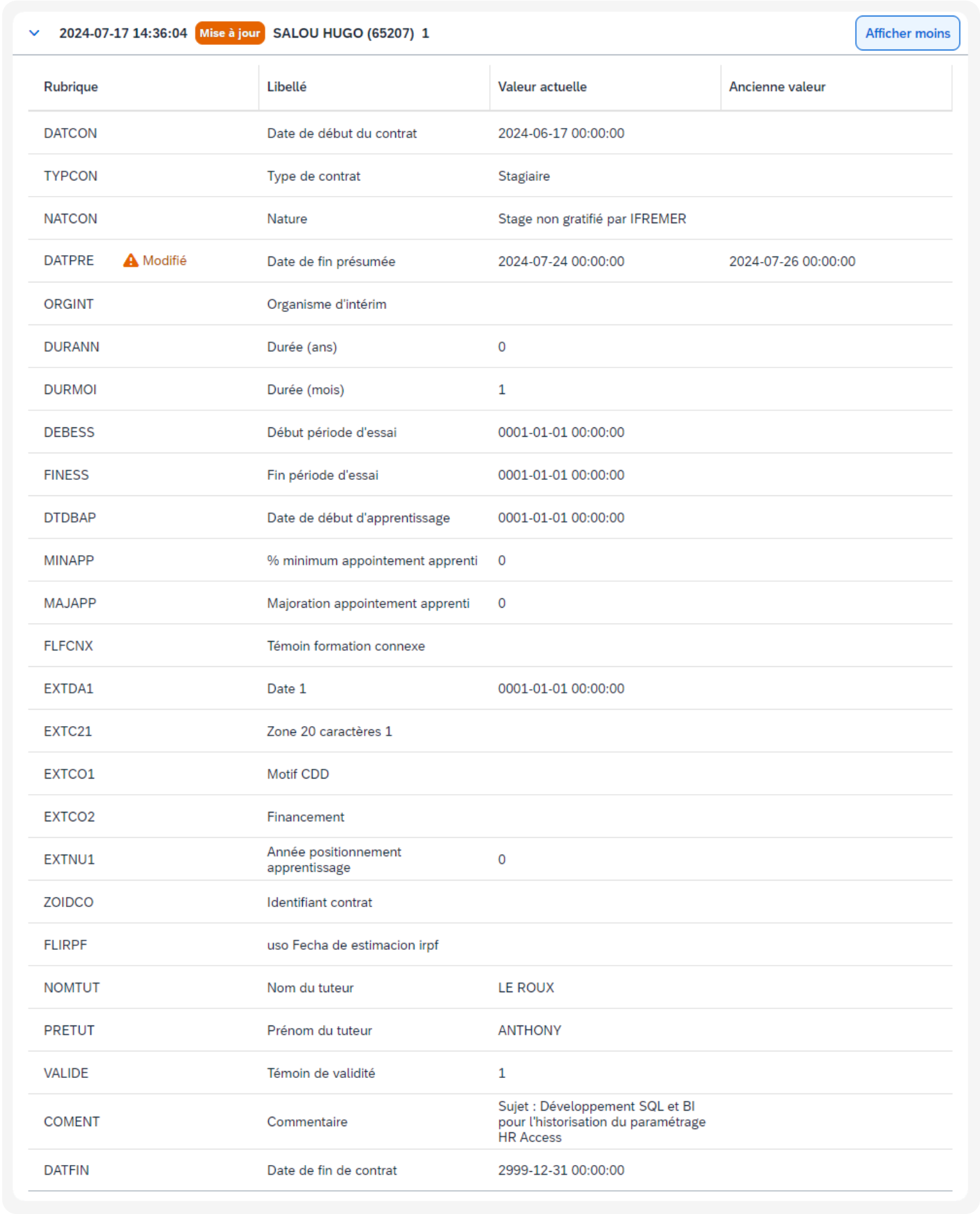
Figure 5 – *L'écran « détails » (ici pour FDO-DRV-CAL → ZDEC – Ordre de calcul des rubriques)*

En appuyant sur le chevron à gauche, on affiche le détail des occurrences modifiées. Pour les insertions, il s'agit de toute l'occurrence ; pour les mise à jour, il s'agit des rubriques modifiées ; pour les suppressions, ce chevron n'apparaît pas.

Il est possible d'afficher l'entièreté de l'occurrence, dans le cas d'une mise à jour, en appuyant sur le bouton « Afficher plus ».

Figure 6 – *Le détail d'une occurrence mise à jour (uniquement les rubriques modifiées)*

Dans la Figure 6, on montre une occurrence mise à jour, il s'agit ici d'une modification d'un contrat dans ZDC0. Lorsqu'on appuie sur « Afficher plus », on obtient un affichage comme celui de la Figure 7 (page 15).



2024-07-17 14:36:04 Mise à jour SALOU HUGO (65207) 1

Afficher moins

Rubrique	Libellé	Valeur actuelle	Ancienne valeur
DATCON	Date de début du contrat	2024-06-17 00:00:00	
TYPCON	Type de contrat	Stagiaire	
NATCON	Nature	Stage non gratifié par IFREMER	
DATPRE Modifié	Date de fin présumée	2024-07-24 00:00:00	2024-07-26 00:00:00
ORGINT	Organisme d'intérim		
DURANN	Durée (ans)	0	
DURMOI	Durée (mois)	1	
DEBESS	Début période d'essai	0001-01-01 00:00:00	
FINESS	Fin période d'essai	0001-01-01 00:00:00	
DTDBAP	Date de début d'apprentissage	0001-01-01 00:00:00	
MINAPP	% minimum appointement apprenti	0	
MAJAPP	Majoration appointement apprenti	0	
FLFCNX	Témoin formation connexe		
EXTDA1	Date 1	0001-01-01 00:00:00	
EXTC21	Zone 20 caractères 1		
EXTCO1	Motif CDD		
EXTCO2	Financement		
EXTNU1	Année positionnement apprentissage	0	
ZOIDCO	Identifiant contrat		
FLIRPF	uso Fecha de estimacion irpf		
NOMTUT	Nom du tuteur	LE ROUX	
PRETUT	Prénom du tuteur	ANTHONY	
VALIDE	Témoin de validité	1	
COMENT	Commentaire	Sujet : Développement SQL et BI pour l'historisation du paramétrage HR Access	
DATFIN	Date de fin de contrat	2999-12-31 00:00:00	

Figure 7 – *Le détail d'une occurrence mise à jour (toutes les rubriques)*

L'occurrence *complète* n'est pas affichée : seules les colonnes « intéressantes » sont affichées dans l'interface. Par « intéressante », on entend une colonne qui ne contient pas qu'une même valeur pour toute la colonne.

Pour déterminer ces colonnes « intéressantes », une procédure (qui est nommée `update_colonnes_non_vides`) est lancée tous les jour. Elle permet de remplir la table `colonnes_non_vides_hra`, dont la structure est définie ci-dessous.

COLONNES NON VIDES
→ **Structure** → **Information** → **Rubrique**

Table 4 – *Structure de la table* `colonnes_non_vides_hra`

Il est important de préciser que cette table n'est utilisée *que* pour l'affichage. Toutes les colonnes sont historisées, même si elles sont vides ou marquées « inintéressantes » (elle ne contiennent que des valeurs par défaut).

Afin de récupérer les données des occurrences modifiés (et leur détail), une requête comme la requête ci-dessous est envoyée au serveur.

```
/changes/details/505131/CO?structure=ZD
                          &CDREGL=FDO
                          &CDSTCO=DRC
                          &CDCODE=
                          &dateSart=2024-07-07%2000:00:00
                          &dateEnd=2024-07-21%2023:59:59
```

Le serveur réalise quatre requêtes SQL :

- récupération des occurrences modifiées (à l'aide de la table historisation occurrences) ;
- récupération des clés-valeurs modifiées (à l'aide de la table historisation clés-valeurs) ;
- récupération des libellés des rubriques (à l'aide de la table `DI61`) ;

- récupérations des rubriques « intéressantes » (à l'aide de la table colonne non vides).

Les résultats de ces quatre requêtes sont ensuite modifiés pour obtenir un résultat comme celui en Code 1 (page 18)

Parfois, la valeur d'une rubrique est un code. Par exemple, on a `TYPCON=ST` dans le contrat d'un stagiaire. Afin d'avoir un visuel plus « métier », on aimerai avoir écrit « Type de contrat : stagiaire » dans l'interface. Pour cela, on utilise la table `DI80` comme détaillé dans l'Annexe A. Le code « brut » reste visible en *tooltip*.

III. A. 5. Écran « graphiques ».

Sur cet écran, on retrouve différent graphiques qui permettent de voir quels dossiers ont reçus un nombre important de modifications, et à quel date. Cet affichage, pour l'instant, ne fonctionne qu'avec ZD.

Les graphiques sont réalisés avec la librairie *open-source* **D3.js**. Cette librairie permet de réaliser tout type de graphiques, mais demande une implémentation « à la main » : il faut dire qu'on place un axe x, un axe y, *etc.*

On affiche donc, comme montrée en Figure 8, un histogramme du nombre de modifications par jour (en échelle log).

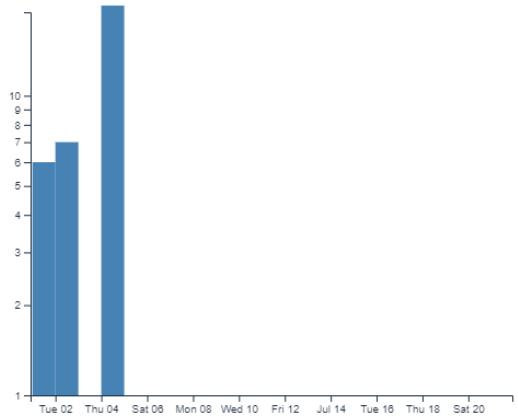


Figure 8 – *Histogramme du nombre de modifications par jour (échelle log)*

```
{
  "status":"OK",
  "data":{
    "changes":[
      {
        "DATEDEB":"2024-07-17 14:36:04",
        "NUDOSS":65207,
        "ZONTRI":"17/06/2024",
        "NULIGNE":1,
        "TYPEMVT":"maj",
        "LABEL":"SALOU HUGO",
        "DATA":"{...occurrence complète...}",
        "keyValues":[
          {
            "key":"DATPRE",
            "oldValue":"2024-07-26 00:00:00",
            "newValue":"2024-07-24 00:00:00",
            "oldCode":"2024-07-26 00:00:00",
            "newCode":"2024-07-24 00:00:00"
          }
        ]
      }
    ],
    "labels":{
      "APPLEV":"Apprenticeship  level",
      "APPSTA":"Apprenticeship status",
      ...
    },
    "columns":[
      "COMENT",
      "DATCON",
      "DATFIN",
      ...
    ]
  }
}
```


Code 1 – *Résultat de la requête « détails »*

On montre également, comme représenté en Figure 9, un camembert et un graphique à bulles groupées qui représentent les dossiers les plus modifiés (en nombre de rubriques modifiées). Les libellés sont également disponible en *tooltip*. En cas d'appui sur un dossier, la page « détails » s'ouvre.

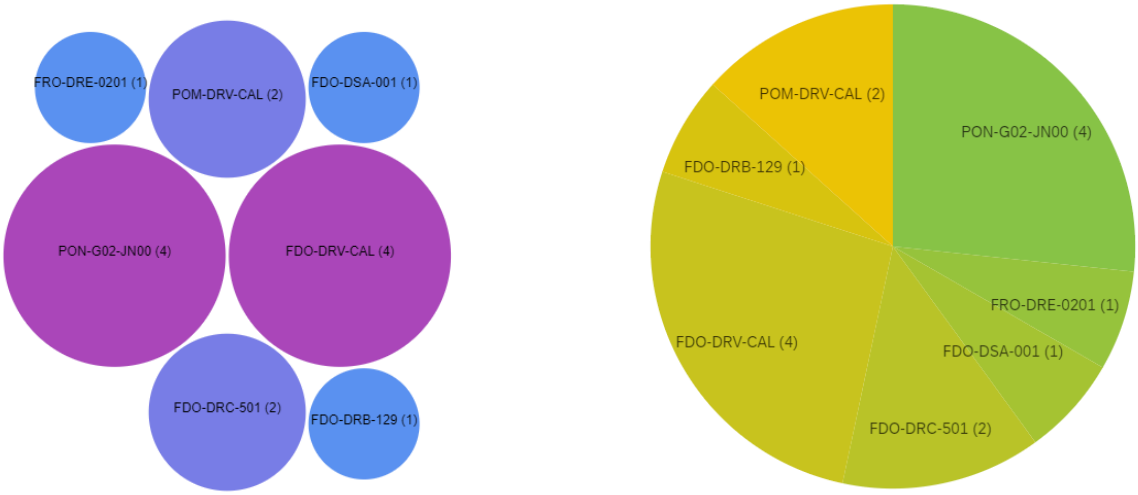


Figure 9 – *Graphiques camembert et à bulles groupées représentant les dossiers les plus modifiés*

Afin de récupérer les données nécessaires pour les graphiques, une requête

```
/changes/summary?structure=ZD
                &CDREGL=
                &CDSTCO=
                &CDCODE=
                &dateSart=2024-07-07%2000:00:00
                &dateEnd=2024-07-21%2023:59:59
```

et renvoie comme résultat :

- jour par jour, le nombre de changements de rubriques ;
- dossier par dossier, le nombre de changements de rubriques.

Ces résultats proviennent majoritairement de historisation clé-valeur.

III. B. Interface « métier ».

Deux écrans sont disponibles en « interface métier ». Ils n'utilisent cependant pas l'historisation mais affichent directement les valeurs.

Afin d'obtenir des libellés pour les différentes valeurs des rubriques, on utilise également la table `DI80`. Son utilisation est décrite en Annexe A.

III. B. 1. Écran « rubriques de paie » (DRC).

L'écran « DRC » se décompose en cinq panels : « recherche », « position dans le bulletin », « interface comptable », « assiettes alimentées », et « alimentation de la DSN ».

1. **Panel « Recherche ».** Ce panel sert à sélectionner la réglementation et le code.

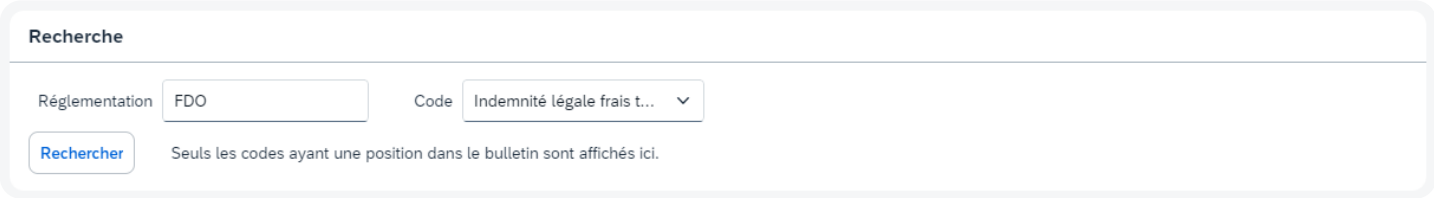


Figure 10 – *Le panel « Recherche » dans l'écran « Rubriques de paie »*

Le champ « code » est en *autocomplete*, mais il n'affiche que les rubriques de paie ayant une position dans le bulletin (afin de ne pas surcharger la ComboBox, ce qui ralentirait *très fortement* l'interface). On limite également aux positions avant « EA », et entre les positions « MA » et « ZZ » (uniquement les rubriques HDN, PSS et PSV sont affichés avec une position « ZZ »).

La requête de l'*autocomplete* du champ « code » n'est pas la même que dans l'interface console. En effet, nous limitons les codes à ceux dans le répertoire « DRC ». La requête est donc de la forme ci-dessous.

```
/drc/autocomplete
```

Lors de la recherche, les données des quatre panels suivants sont récupérées par la requête :

```
/drc?CDREGL=FDO
    &CDCODE=ITZ
```

Sur le serveur, quatre requêtes sont effectuées (une par panel) pour récupérer les données nécessaires aux panels ci-dessous.

2. **Panel « Position dans le bulletin ».** On utilise la rubrique `POSAGR` de la table `ZDAO`, qu'on affiche en utilisant la `DI80` (*c.f.* Annexe A) pour afficher des libellés au lieu de codes « brutes ». Le code « brut » reste accessible dans le *tooltip*.

Figure 11 – *Le panel « Position dans le bulletin » dans l'écran « Rubriques de paie »*

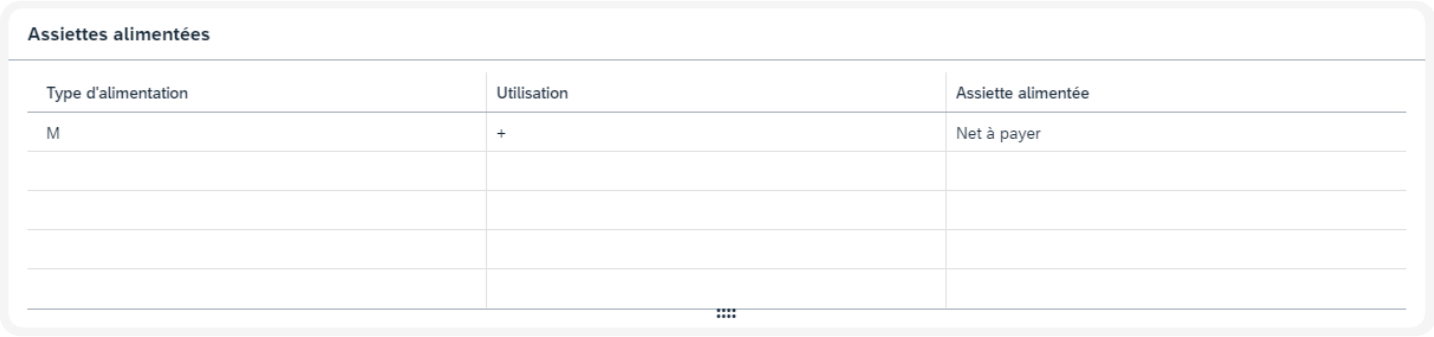
3. **Panel « Interface comptable ».** On utilise les données de la table `ZD4M`, qu'on affiche en utilisant la `DI80` (*c.f.* Annexe A) pour afficher des libellés au lieu de codes « brutes ». Le code « brut » reste accessible dans le *tooltip*. Certains libellés semblent manquant (nature de dépense et débit / crédit). Également, pour le compte, on affiche le code comptable en brut en plus du libellé.

   En sélectionnant une colonne, on peut filtrer les lignes affichées.

Figure 12 – *Le panel « Interface comptable » dans l'écran « Rubriques de paie »*

4. **Panel « Assiettes alimentées ».** On utilise les données de la table `ZDCQ`, qu'on affiche en utilisant la `DI80` (*c.f.* Annexe A) pour afficher des libellés au lieu de codes « brutes ». Le code « brut » reste accessible dans le *tooltip*. Certains libellés semblent manquant (type d'alimentation et utilisation).

   En sélectionnant une colonne, on peut filtrer les lignes affichées.



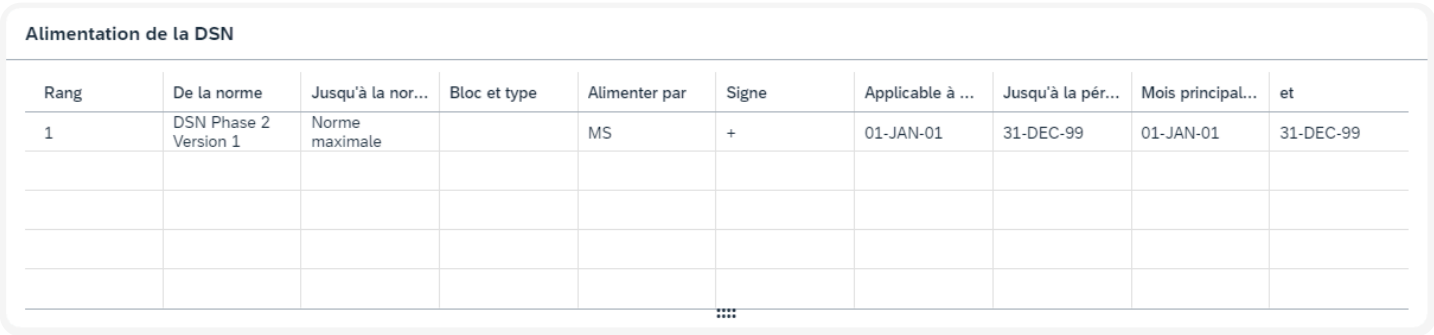
Assiettes alimentées

Type d'alimentation	Utilisation	Assiette alimentée
M	+	Net à payer

Figure 13 – *Le panel « Assiettes alimentées » dans l'écran « Rubriques de paie »*

5. **Panel « Alimentation de la DSN ».** On utilise les données de la table `ZD7P`, qu'on affiche en utilisant la `DI80` (*c.f.* Annexe A) pour afficher des libellés au lieu de codes « brutes ». Le code « brut » reste accessible dans le *tooltip*. Certains libellés semblent manquant (« alimenter par », par exemple).

   En sélectionnant une colonne, on peut filtrer les lignes affichées.



Alimentation de la DSN

Rang	De la norme	Jusqu'à la nor...	Bloc et type	Alimenter par	Signe	Applicable à ...	Jusqu'à la pér...	Mois principal...	et
1	DSN Phase 2 Version 1	Norme maximale		MS	+	01-JAN-01	31-DEC-99	01-JAN-01	31-DEC-99

Figure 14 – *Le panel « Alimentation de la DSN » dans l'écran « Rubriques de paie »*

III. B. 2. Écran « constantes de paie » (DS9).

L'écran « DS9 » se décompose en trois panels : « recherche », « sélection/liste des valeurs » et « affichage graphique ».

1. **Panel « Recherche ».** Ce panel sert à sélectionner la réglementation et le code.

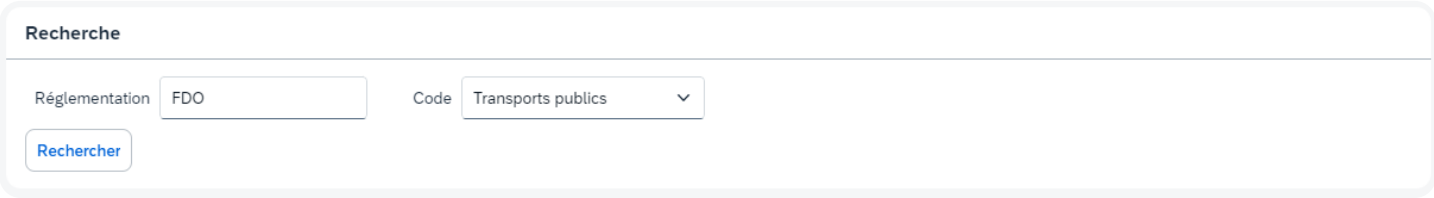


Figure 15 – *Le panel « Recherche » dans l'écran « Constantes de paie »*

Le champ « code » est en *autocomplete*, sa requête n'est pas la même que dans l'interface console. En effet, nous limitons les codes à ceux dans le répertoire « DS9 ». La requête est donc de la forme ci-dessous.

```
/ds9/autocomplete
```

Lors de la recherche, les données du panel suivant sont récupérées par la requête :

```
/ds9?CDREGL=FDO
    &CDCODE=260
```

2. **Panel « Sélection / Liste des valeurs ».** On utilise les données de la table `ZDCU`, qu'on affiche dans un tableau arbre. On peut donc déplier les libellés, puis les mot clés (les libellés de ces mot-clés ne sont pas dans la `DI80`).

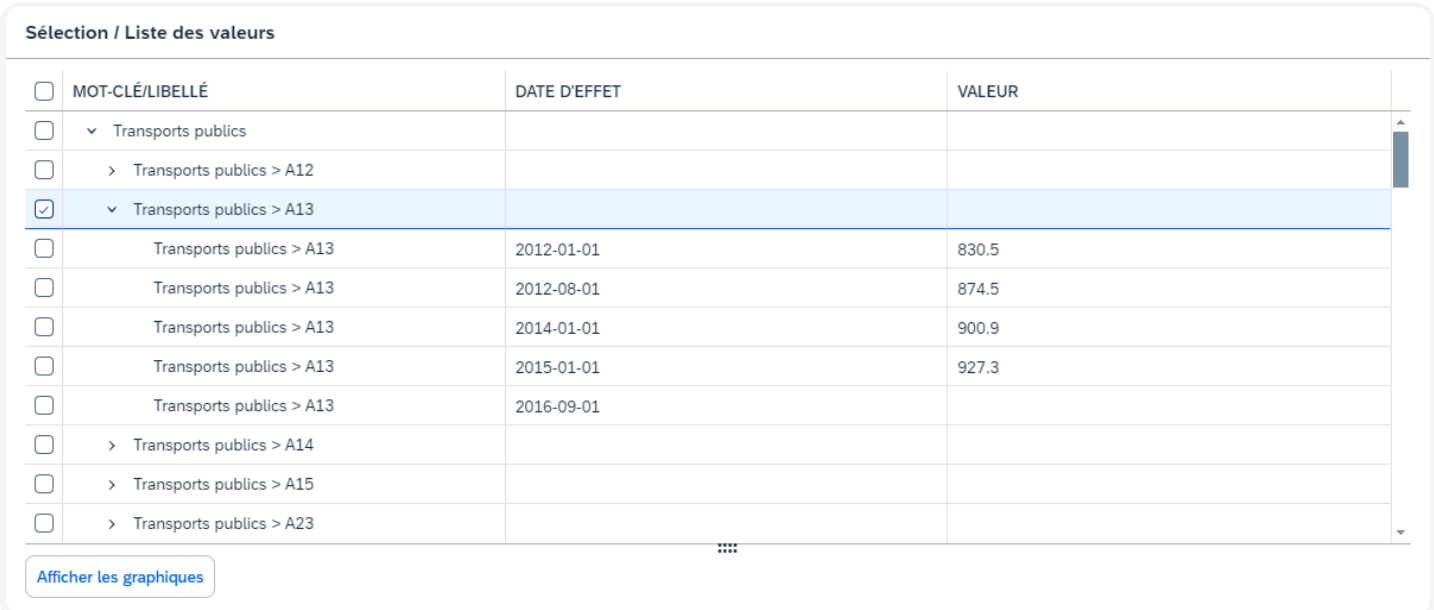


Figure 16 – *Le panel « Sélection / Liste des valeurs » dans l'écran « Constantes de paie »*

En sélectionnant une colonne, on peut filtrer les lignes affichées. En sélectionnant (en cochant) les lignes, on peut afficher des valeurs. Il suffira d'appuyer ensuite sur « Afficher les graphiques » pour que les valeurs cochées soient affichées dans le graphe du panel suivant.

3. **Panel « Affichage graphique ».** On affiche graphiquement les valeurs sélectionnées dans le panel précédent. Chaque courbe correspond à un mot clé sélectionné. L'axe x s'adapte à la période des données sélectionnées.

   Ce graphe est réalisé avec la librairie **D3.js**.

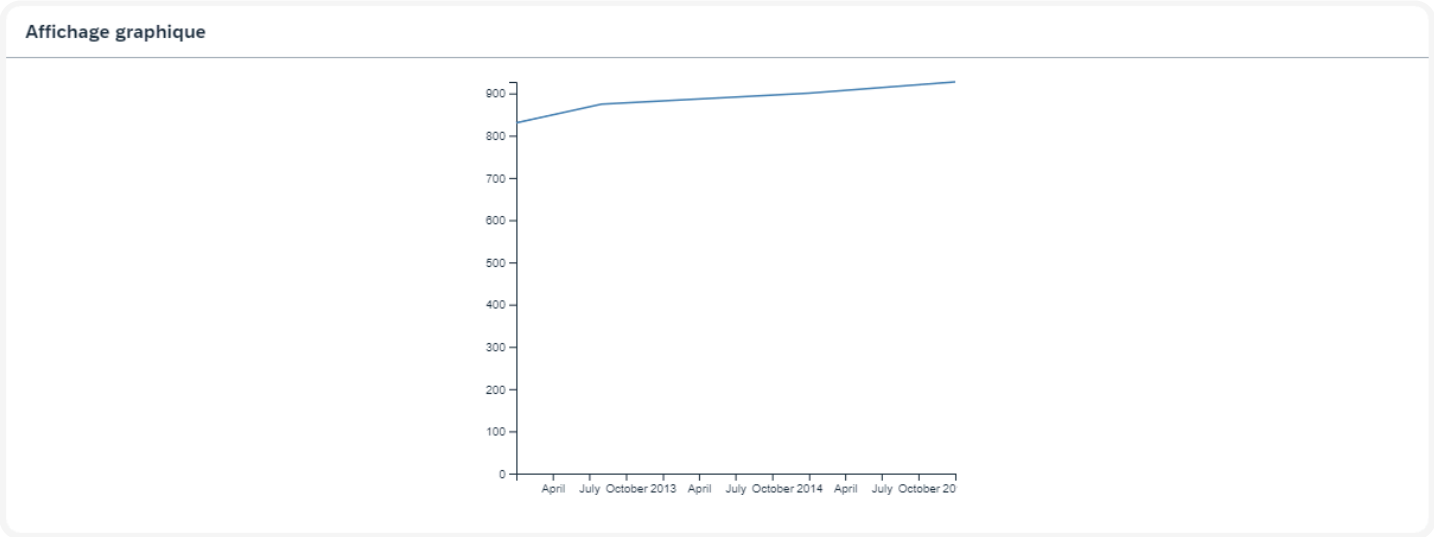


Figure 17 – *Le panel « Affichage graphique » dans l'écran « Constantes de paie »*

Annexe A. Utilisation de la table DI80.

La table DI80 permet d'associer un dossier à une valeur dans une rubrique. Ceci permet, en particulier, d'associer un libellé « métier » plutôt qu'un code technique.

La table DI80 contient de multiples champs, mais seuls certains nous intéressent ici :

CDSTDO. le code de structure ;
CDINFO. le code d'information (*i.e.* la table) ;
CDRUBR. le code de la rubrique (*i.e.* le nom de la colonne) ;
CDSCCO. le code du répertoire associé.

Afin de trouver le libellé du dossier associé dans ZD, il est nécessaire de déterminer trois valeurs :

1. la réglementation
2. le répertoire
3. le code

- Le répertoire nous est donné par la table DI80 : il s'agit de la valeur du champ CDSCCO.
- Pour la réglementation, on utilise la fonction PL/SQL reglementation. Elle demande une réglementation d'origine (généralement, il s'agit du champ SOCDOS dans la table [structure]00), et un répertoire (c'est également la valeur de CDSCCO dans la DI80).
- Pour le code, il s'agit de la valeur du champ dans la rubrique d'origine.

Par exemple, la rubrique TYPCON dans la table ZYCO a une valeur « **ST** ». On regarde dans la table DI80, et on trouve que le répertoire associé est le répertoire « **UIP** ». La fonction reglementation renvoie, dans cet exemple-ci, la réglementation « **FDO** ». On en déduit que le dossier associé est « **FDO-UIP-ST** », et le libellé associé est « Stagiaire ».

Fonction PL/SQL di_liblon_valeur.

Afin de pouvoir utiliser facilement les données de la DI80, une fonction PL/SQL est disponible di_liblon_valeur, qui donne le libellé d'une valeur étant donnés la structure, l'information, la réglementation d'origine, la rubrique et la valeur.

En cas de problèmes (plusieurs répertoires associés ou aucun dans la table `DI80`), on renvoie la valeur « brute ».

Une optimisation simple est de tester si la valeur est suffisamment courte pour être un code de dossier ZD (dans l'implémentation, on dit qu'un code fait moins de 50 caractères).

Fonction PL/SQL `di_liblon_valeur_json`.

Dans l'historisation, on stocke une ligne d'une table sous forme d'un objet JSON. Dans l'affichage, il est intéressant de pouvoir avoir les libellés correspondant aux valeurs de l'occurrence, sans avoir à les stocker dans l'historisation.

Pour cela, la fonction PL/SQL `di_liblon_valeur_json` permet, étant donné un objet JSON sous forme de `clob` (ainsi que la structure, l'information et la réglementation d'origine) d'obtenir les libellés associés. Pour cela, pour chaque clé du JSON, on ajoute une nouvelle clé au format « `[clé]_LABEL` » qui correspond au libellé de la valeur. On n'ajoute cette clé *que* lorsqu'un libellé est disponible. On utilise, pour cela, la fonction `di_liblon_valeur`.

Annexe B. Export des données en fichier Excel.

Comme indiqué à la fin de la Section III. A. 2, un export Excel des différentes modifications effectuées. La génération de ce fichier Excel a lieu sur le serveur : on diminue le délai de génération du fichier (car moins d'intermédiaires pour les requêtes SQL).

Clé	Libellé	Ancienne valeur	Nouvelle valeur
2024-07-17 14:36:04 [Mise à jour] SALOU HUGO (65207) 1			
DATPRE	Date de fin présumée	2024-07-26 00:00:00	2024-07-24 00:00:00

Figure 18 – *Exemple de fichier Excel exporté*

Une première étape est de récupérer les données. Pour cela, on effectue plus ou moins les mêmes requêtes SQL que pour l'affichage des données dans l'écran « détails » (Section III. A. 4).

Une fois les données récoltées, on utilise la librairie `exceljs`. Elle permet de contrôler finement le rendu du fichier Excel, tout en ajoutant les données sans trop de difficultés.

Les deux fonctions `worksheet.addRow` et `worksheet.addRows` permettent d'ajouter une ligne de donnée dans la feuille du Excel.

On peut ensuite, cellule par cellule, ou en groupe de cellules, définir le style de rendu : bordure, remplissage, police, alignement, *etc.* On peut également fusionner des cellules avec la fonction `mergeCells`.

Bien plus de détails sont disponibles dans la documentation **ExcelJS**.

Figure 19 – *Le logo du projet*