Sémantique formelle des contraintes qualitatives

StandardQL — une approche déclarative pour valider les données

Loïc Herman

17 avril 2025

HEIG-VD — PLM

Sommaire

Contexte

Depuis les données des registres fédéraux

Dans le cadre du registre fédéral des bâtiments et des logements (RegBL), l'Office fédéral de la statistique définit 7 entités et $\sim$ 800 caractères (attributs) sur ces entités[1]. Cela comprend :

- **Attributs obligatoires** : identifiants uniques, coordonnées, adresses
- **Attributs structurels** : nombre d'étages, surface habitable, type de chauffage
- **Attributs temporels** : dates de mise à jour, historique des modifications
- **Relations spatiales** : commune, quartier, parcelle cadastrale

1. https ://www.housing-stat.ch/catalog/fr/4.2/final

Vers une formalisation de contraintes de qualités

La définition d'un tel ensemble d'attributs vient avec un besoin d'en valider la qualité, tant en typage qu'en termes relationnels.

L'OFS définit donc un ensemble de $\sim$ 500 règles de qualité sous différentes classes :

- **Automatismes** : règles devant démarrer une action automatique
- **Obligation d'annonce** : information manquante selon un critère spécifique
- **Exigence de qualité** : validations de cohérence et contraintes d'intégrités
- **Spécification technique** : définitions formelles du domaine des attributs

Vers une formalisation de contraintes de qualités

Pour simplifier la compréhension, l'OFS définit pour chaque règle un pseudo-code comparable à du code SQL permettant d'avoir une version déclarative des règles.

Listing 1 – Pseudocode for rule BQ9980

```
GSTAT=1007
WHERE GABBJ NOT NULL
```

Le langage StandardQL s'inspire du format du pseudo-code pour proposer un langage déclaratif facilement éditable par les gestionnaires d'applications permettant d'offrir une couverture satisfaisante des contraintes avec une retranscription minime.

StandardQL – la version formelle du pseudo-code

Objectifs du langage

Le langage doit remplir les critères suivants :

- Permet de modéliser un ensemble de règles représentant des contraintes, éventuellement dépendantes, sur un ou plusieurs attributs d'un registre,
- Peut facilement être modifié par des personnes aisées en informatique, mais pas nécessairement développeurs,
- Expressivité ciblée, par exemple, avec les conditions multiples sur un seul attribut,
- Extensibilité de l'environnement d'exécution avec l'ajout dynamique de fonctions

Non-objectifs du langage

Le langage ne cherche pas à faire :

- **Traduction automatique complète** : certaines modifications légères sur le pseudo-code sont à prévoir
- **Règles d'automatismes** : hors-scope pour la validation de données
- **Compatibilité complète *day-one*** : certaines règles ne pourront juste pas être traduites, celles-ci devront être implémentées manuellement

Paradigme des contraintes de qualité

StandardQL est un langage déclaratif qui :

- Au lieu de spécifier comment vérifier une contrainte, on spécifie ce que la contrainte doit satisfaire,
- Propose une syntaxe qui s'adapte aux données, centrée autour de la notion d'attributs,
- Comble le fossé entre l'expression informelle des règles par l'OFS et leur implémentation technique,

Paradigme des contraintes de qualité

StandardQL atteint ce paradigme par :

- Une extensibilité assurée avec la possibilité de définir dynamiquement des fonctions depuis l'environnement d'exécution, séparément de la syntaxe
- Une gestion dynamique des attributs en les traitant comme des fonctions, permettant de parcourir les relations entre entités

Grammaire formelle

La structure générale du langage est toutefois définie formellement comme suit :

- Le *Program* ne peut être constitué que d'un seul *Statement*
- Un *Statement* est constituté d'une *Expression*, éventuellement complémentée d'une condition sous forme d'*Expression*
- Une *Expression* modélise une ou plusieurs contraintes (*Condition*) ou relations logiques (égalité, vérité) sur un attribut
- Les contraintes sont des *Condition* sur un seul attribut qui peuvent également être combinées au sein d'une même *Expression*

Le *Program* représente alors la production de plus haut niveau pour la représentation d'une règle à évaluer.

Grammaire formelle

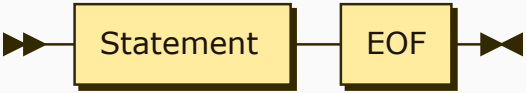

Figure 1 – Définition de *Program*

Grammaire formelle

Figure 2 – Définition de *Statement*

Grammaire formelle

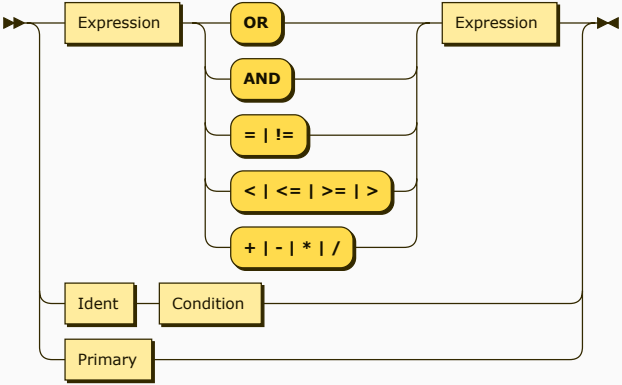

Figure 3 – Définition de *Expression*

Grammaire formelle

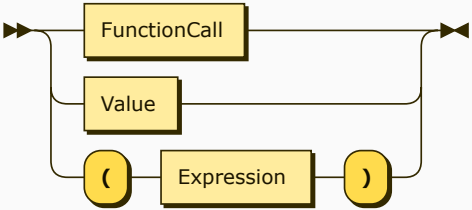

Figure 4 – Définition de *Primary*

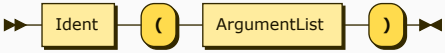

Figure 5 – Définition de *FunctionCall*

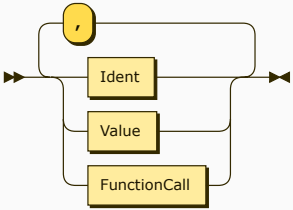

Figure 6 – Définition de *ArgumentList*

Grammaire formelle

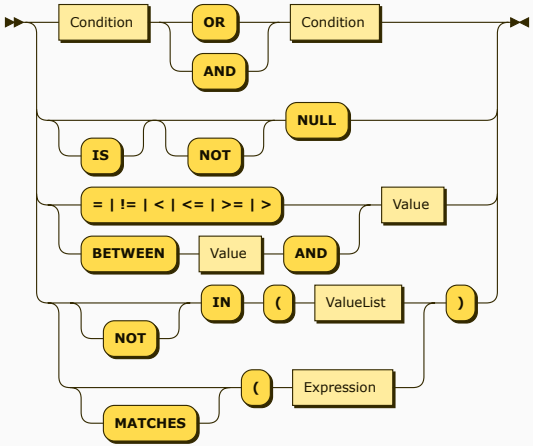

Figure 7 – Définition de *Condition*

Grammaire formelle

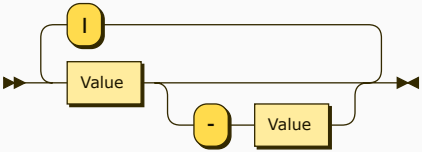

Figure 8 – Définition de *ValueList*

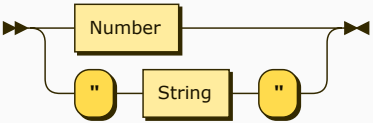

Figure 9 – Définition de *Value*

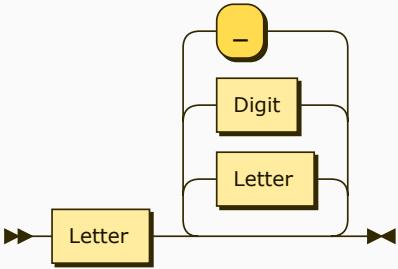

Figure 10 – Définition des *Ident*

Système de types

$$\begin{aligned} \tau \quad ::= \quad & \mathsf{int} \mid \mathsf{float} \mid \mathsf{string} \mid \mathsf{bool} \mid \mathsf{date} \\ \mid \quad & [\tau] \mid \mathsf{entity}(\mathsf{e}) \mid \mathsf{null} \end{aligned}$$

Figure 11 – Types utilisés dans le typage dynamique de StandardQL

Système de types

Les identifiants sont des fonctions, de sorte que :

$$\frac{\Gamma(\mathrm{id}) = \mathbf{entity}(e) \to \tau}{\Gamma \vdash \mathrm{id} : \mathbf{entity}(e) \to \tau} \quad \text{(ID)}$$

Appliqué pour un attribut donné :

$$\vdash \texttt{GSTAT} : \mathbf{entity}(\texttt{BUILDING}) \to \mathbf{int}$$

Le statut du bâtiment (`GSTAT`) est disponible et sa fonction associée retourne le statut sous forme d'entier lorsqu'elle est appelée depuis un contexte où l'entité principale est le bâtiment.

De même manière, pour un identifiant dont on connait la relation :

$$\frac{\Gamma(\mathrm{id}) = \mathsf{entity}(e) \to \mathsf{entity}(e_k) \to \tau}{\Gamma \vdash \mathrm{id} : \mathsf{entity}(e) \to [\tau]} \quad \text{(REL-ID)}$$

Appliqué pour un attribut donné :

$$\frac{\Gamma(\mathtt{WSTAT}) = \mathsf{entity}(\mathtt{BUILDING}) \to \mathsf{entity}(\mathtt{DWELLING}) \to \mathsf{int}}{\vdash \mathtt{WSTAT} : \mathsf{entity}(\mathtt{BUILDING}) \to [\mathsf{int}]}$$

Le statut du logement (`WSTAT`) est disponible et sa fonction associée retourne le statut sous forme de liste d'entiers lorsqu'elle est appelée depuis un contexte où l'entité principale est le bâtiment.

Système de types

Pour travailler avec les résultats multiples, l'environnement par défaut propose quelques fonctions standards comme `LENGTH`, `COUNT`, `SUM`, `MIN`/`MAX`, etc.

Un consommateur de la librairie de validation peut définir d'autres fonctions simplement en les ajoutant dans le contexte Γ.

La clause `MATCHES` est particulière, les appels de fonctions au premier niveau se voient augmentés d'une référence vers l'identifiant référencé.

```
LPARZ NULL OR (LENGTH<=12 AND COUNT(LEADINGZEROS)<3)
```

deviendra, en version élaborée

```
LPARZ()=NULL OR (LENGTH(LPARZ())<=12 AND
    COUNT(LEADINGZEROS(LPARZ()))<3)
```

Feuille de route

Implémentation du parser

Le parser sera implémenté avec la technique des *parser combinators* en Haskell.

Le Haskell offre beaucoup d'avantages pour la définition élégante d'un parser grâce au pattern matching et plusieurs librairies qui permettent de construire le parser en modules réutilisables avec des messages d'erreurs informatifs.

Il n'y a pas besoin d'exécuter le Haskell depuis Java, car la « compilation » peut être faite en amont et l'interface Java devra uniquement se baser sur le résultat de cette compilation.

Élaboration du langage externe

Le langage externe offre beaucoup de flexibilité sur la façon de définir plusieurs contraintes sur un même attribut, mais en termes d'évaluation cela double le nombre de fois qu'il faut définir la gestion pour une même logique.

L'élaboration viendra transformer le langage externe, défini formellement, en un langage interne plus concis.

Dans l'essentiel, toutes les *Condition* seront transformées en leur équivalent plus long et plus verbeux sous forme d'*Expression*.

Intégration à une maquette de SIBAT

Pour assurer la gestion de la collecte des données auprès des communes vaudoises, l'État de Vaud développe le Système d'information des Bâtiments (SIBAT).

Dans le cadre de l'implémentation, une version « maquette » sera implémentée permettant de valider le fonctionnement des définitions dynamiques de fonctions et de la bonne exécution des règles interprétées.

Pour suivre l'architecture habituelle de l'État de Vaud, la maquette sera en Java et un protocole de communication entre le résultat de l'élaboration et l'interface Java sera mis en place.

Questions