Etude de la complexité de requêtes de SQLShare - étude statistique et visualisation

Tien Thanh LE

[1] INSA Centre Val de Loire, Campus Blois
[2] Université de Tours, Campus Blois
letienthanh091199@gmail.com

Abstract. Ces dernières années, la collecte et l'analyse des tendances quotidiennes des utilisateurs doivent être prises au sérieux. Cet article suivre le travail de C.Moreau et al, qui présente un ensemble d'analyses visant à mieux comprendre la complexité d'une requête de SQLShare afin de classifier les compétences des utilisateurs. Avec le but de trouver qu'est qu'on en commun ces explorations entre les cluster d'exploration, ce papier reprend les travaux préliminaires de S.Jain et al et les travaux en cours de C.Moreau et al dans le but de réaliser un état de l'art sur les indicateurs de permettant de mesurer la complexité et expérimenter avec les clusters d'explorations de SQLShare.

Keywords: Complexité des requêtes · Explicabilité · Clustering de séquences · Exploration de données · Visualisation

1 Introduction

Des nos jours, l'importance de l'analyse du comportement des utilisateurs devient de plus en plus importante, ce qui apporte de nombreux avantages à la fois économiquement et dans la recherche. Plus précisément dans le cadre de cette étude, le point important est l'étude des requêtes que les differents utilisateurs ont effectuées sur une base de données SQL.
Ce projet poursuivra le travail de C.Moreau et al sur le mining Workload SQL endu portant sur l'étude de clusters d'explorations d'analyse de données pour mieux classifier les compétences des utilisateurs. Bien au contrait, on voulait ouvrir à l'autre aspects. Cependant, comme il y a tellement d'aspects différents à considérer, ce travail se concentrera uniquement sur la complexité des requêtes (longueur de requêtes et usage de clauses avancées).
La première partie du projet sera dédié à l'étude d'indicateurs pour mesurer la complexité d'une requête SQL et juger lesquels sont applicables aux requêtes de SQLShare. Ces indicateurs serviront ensuite à comprendre et à expliquer les comportements typiques des utilisateurs. La deuxième partie consiste à appliquer les indicateurs retenus lors de la première partie afin d'expliquer les 12 clusters d'explorations issues de SQLShare. Les résultats seront restitués sous la forme d'indicateurs statistiques et visuels, pouvant s'inspirer des indicateurs décrits dans les études de C.Moreau et al et, de A. Vashistha et S. Jain.

Pour mettre en place un modèle répondant au mieux à ce sujet, plusieurs questions de recherche sont posées :

- Quels indicateurs permettant de mesurer la complexité d'une requête SQL ?
- Comment nous appliquons ces indicateur dans la contexte du SQLShare ?

Après avoir posé les spécificités des different métrique pour mesurer la complexité, nous décrirons l'étape de les analyse, jugerons lesquels sont applicables et comprendront les comportements typiques des utilisateurs. Pour finir, nous répondrons aux questions de recherche fixées précédemment. La section suivante présentera quelques définitions de base, qui vous aideront à fournir un bon départ.

2 Préliminaires

La première partie du rapport sera consacrée à rappeler quelques définitions relatives aux domaines abordées dans cette étude.

SQLShare SQLShare est le résultat de traivail depuis de plusieurs années d'un expérience SQL-as-a-Service [1] et est conçu pour être un nouveau système pour mieux prendre en charge la gestion et l'analyse des données. Dans le but de mettre en place une infrastructure de base de données permanente et de faciliter autant que possible les aspects de modification des données et/ou des exigences ainsi que la manipulation de fichiers avec des scripts en langage de programmation comme R, Python, MATLAB, etc. [1]
Il est très clair que SQLShare est bénéfique pour les data scientists et les chercheurs [1,3] en automatisant ou en éliminant les étapes d'installation, de déploiement, de conception de schéma, de réglage physique et de diffusion des données pour simplement télécharger des données, écrire des requêtes et partager les résultats. Une courte enquête réalisée par S. Jain et al montre que 6 (sur les 33 utilisateurs qui ont répondu) ont estimé qu'un autre système de base de données standard pourrait répondre à leurs besoins de gestion de données, 18 sur 33 ont déclaré qu'aucun autre outil ne fonctionnerait pour leurs exigences et 23 des 33 utilisateurs ont mentionné que "la facilité de téléchargement et de nettoyage des données" et "la possibilité de partager" étaient les raisons pour lesquelles ils trouvaient SQLShare le plus utile.
Cependant, ces bienfaits sont compensés par le fait que le système SQLShare n'est pas destiné aux grands ensembles de données [1] et le workload SQLShare est publié avec le but de reutilisation. Dans ce contexte, il est composée de 10.668 requêtes SQLL, concernant 2.809 explorations, réalisés par 97 utilisateurs sur la plateforme SQLShare utilisateur [3]. Mais à côté de cela, comme le prétendent Jain et al dans leur article, cette workload est la seule contenant principalement des requêtes écrites à la main sur des ensembles de données téléchargés par l'utilisateur [1]. C'est-à-dire ce workload est capable de refléter

une véritable activité humaine interactive sur l'ensemble de données et qu'elle présente également une certaine complexité et diversité, ce qui se révèle utile pour d'autres chercheurs.

La complexité de la requête SQL Dans la complexité descriptive, une requête est une correspondance entre les structures d'une signature et les structures d'un autre vocabulaire [4]. La complexité de la requête est souvent mesurée en termes de ressources requises par un serveur de base de données pour exécuter la requête. Donc pour le mesurer, nous devons prendre à la fois le respectif de l'utilisateur ainsi que de la machine. Les mesures courantes de la complexité logicielle incluent la complexité cyclomatique (une mesure de la complexité du flux de contrôle) et la complexité de Halstead (une mesure de la complexité de l'arithmétique) [6]. Par exemple:

- Le "flux de contrôle" dans une requête SQL est mieux lié aux opérateurs AND et OR dans la requête.
- La "complexité de calcul" est mieux liée aux opérateurs tels que SUM ou JOINS implicites.

Noté que la mesure de la complexité cyclomatique reflète le nombre de décisions d'un algorithme en comptabilisant le nombre de chemins linéairement au travers d'un programme [6]. Comme notre travail prendra le travail de A. Vashistha et S. Jain [2], ils trouvent une bonne manière pour l'appliquer dans leur travail.
Le but des mesures de complexité est de caractériser à quel point il est difficile pour une personne de comprendre la requête, notre objectif principal sera donc l'aspect utilisateur. Nous prendrons leur définition de la complexité de la requête comme la charge cognitive sur un utilisateur lors de l'écriture de la requête.
En bref, nous ferons référence à la complexité des requêtes comme à la complexité d'une requête du point de vue d'un utilisateur plutôt que du système.

Exploration des données L'exploration des données est la première étape de l'analyse des données, où les utilisateurs explorent un grand ensemble de données de manière non structurée pour découvrir les modèles, les caractéristiques et les points d'intérêt initiaux. Ce processus n'est pas destiné à révéler toutes les informations contenues dans un ensemble de données, mais plutôt à aider à créer une image globale des tendances importantes et des principaux points à étudier plus en détail [7].
C.Moreau et al ont proposé un ensemble d'indicateurs pour analyser les explorations de données [3] :par répartition statistique (distribution de longueur et distribution d'état), par description vectorielle (la norme, la corrélation et l'analyse des composants), par transitions (matrice origine-destination) et par scattrine et valeurs aberrantes (UMAP).
Dans les travaux de C.Moreau et al, ils ont présenté leurs idées pour l'exploration des clusters [3]. En utilisant différents sous-ensembles de fonctionnalités de requête, ils ont étudié 2 variantes de clustering Pour étudier la complexité, nous intéressons aux clauses les plus complexes tout au long de l'exploration et pour utiliser le

maximum de valeurs de caractéristiques, les explorations contenant des requêtes complexes doivent être considérées comme complexes [3].

- Le premier variante vise à déterminer si les opérations SQL déterminent la complexité de la requête. Ils émettent l'hypothèse que des fragments de requête plus riches déterminent des clauses complexes plus riches.
- La deuxième variante propose un nouveau clustering mélangeant toutes les fonctionnalités issues des vecteurs d'opération et de complexité. Ils ont ensuite testé plusieurs stratégies de sélection et de normalisation des caractéristiques

Pour le premier variante, ils implémentent ces méthodes en utilisant une combinaison de CED, UMAP et DBSCAN. DBSCAN est adapté à la topologie résultant de CEF et UMAP lorsqu'il est appliqué sur le workload SQLShare. Ceci est ensuite testé avec des explorations one-shot et avec des plus explorations plus longues (2 requêtes ou plus) pour obtenir les clusters différenciés.
Et pour la deuxième variante de clustering, ils ont testé avec de nombreux paramètres: test d'exclusion des caractéristiques du vecteur de longueur, vecteurs de clause, agrégation de plusieurs caractéristiques parmi différentes stratégies; calculation séparément la similarité des vecteurs d'opération, de longueur et de clause, en combinant les scores de similarité avec différents types de fonctions d'agrégation.

3 Mesurer la complexité des requêtes SQL

Le workload SQLShare se compose de requêtes écrites par des experts et des non-experts et a un workload unique, c'est-à-dire que ses données ont été collectées indépendamment de tout autre projet ;et à la fois, il est donc très variée. La partie suivante de ce rapport sera consacrée à analyser plus en détail les métriques d'un workload SQL et à trouver un moyen de les adapter au travail d'étude du comportement d'analyse de C.Moreau et al.
Avant de commencer à analyser les métriques de mesurer la complexité de requête SQL dans SQLShare Workload, je tiens à rappeler que les mesures courantes de complexité en programmation incluent la complexité cyclomatique (une mesure de la complexité du flux de contrôle) et la complexité de Halstead (une mesure de la complexité de l'arithmétique) [5]. Par exemple, le "flux de contrôle" dans une requête SQL est mieux lié aux opérateurs "AND" et "OR" dans le requête et la "complexité de calcul" est mieux liée aux opérateurs tels que SUM ou JOINS implicites.
Les mesures de complexité cyclomatique sont également utilisées pour estimer le nombre de défauts dans un programme et déterminer la complexité d'un code. La complexité cyclomatique est le nombre de chemins linéairement indépendants dans le code source d'un programme et elle est calculée à l'aide du graphe de contrôle de flux du programme [6] : les noeuds du graphe correspondent à des groupes indivisibles de commandes d'un programme, et une arête dirigée relie deux nœuds si la deuxième commande peut être exécutée immédiatement

après la première commande. La complexité cyclomatique peut également être appliquée à des fonctions, modules, méthodes ou classes individuels au sein d'un programme. Cet aspect concerne plus probablement l'ordinateur et non le comportement de l'utilisateur, nous ne considérerons donc probablement pas ce paramètre dans une étude plus approfondie.
Le but des mesures de complexité est de caractériser à quel point il est difficile pour une personne de comprendre la requête, et non à quel point elle peut être évaluée efficacement donc ce que l'optimiseur SQL fait aux queries, à mon avis, n'est pas pertinent.
En fin,en ce qui concerne l'analyse des métriques de complexité, nous nous souvenons toujours qu'aucune requête ne vaut la peine d'être prise en compte (par exemple une requête "soigneusement" rédigée vaut-elle plus de considération qu'une requête "bizzare" écrite au hasard ?). Tant que les rêquetes reflètent une certaine vérité sur la complexité et que nous pouvons les comparer les uns par rapport aux autres. De cette façon, nous pouvons organiser toutes les requêtes en fonction de leur complexité, les trier et nous concentrer sur les différents groupes. A. Vashishtha et S. Jain. ont fait un excellent travail d'application dans le contexte du workload SQL, qui sont decrites dans la sous-section suivante.

3.1 Travaux connexes des metriques pour mesurer la complexité des requêtes SQL [2]

Pour leur projet, la première chose qu'ils ont faite a été de trouver en ensemble de requêtes représentatives à partir de workload SQLShare pour utiliser cet ensemble de requêtes pour déterminer quelles métriques contribuent le plus à la complexité des requêtes. En utilisant leurs propres outils pour l'analyse globale de SQLShare, tels que l'outil QWLA (Query WorkLoad Analysis) [8]. Une partie de sa mise en œuvre effectue les opérations dans l'ordre suivante: prendre un ensemble de requêtes sur le SQLShare en tant qu'entrée, générer des plans de requête XML et enfin les analyser pour extraire des métriques intéressantes des requêtes, telles que le nombre d'opérateurs et le nombre d'expressions. Ensuite, pour s'assurer d'avoir un sous-ensemble final de requêtes les plus diversifiées, ils ont analysé l'ensemble de données de requête SQLShare, examiné la liste ordonnée de requêtes en fonction de chaque métrique et généré des échantillons à partir de chaque liste, capturant des requêtes avec des valeurs de métriques élevées et basses. En supprimant ces doublons de leur liste de 180 requêtes, ils obtiennent un ensemble de requêtes résultant de 117 requêtes uniques. Cet ensemble est appelé ensemble représentatif de requêtes.
Sur la base de leurs travaux antérieurs, ils ont proposé un coefficient de variété V par rapport à un environnement Env composé d'un ensemble de données D, d'un catalogue S et d'un ensemble de tâches T. Ce coefficient de variété V est représenté par la formule suivante

$$V_Env(D;S;T) = \Omega catalog(S) + \Omega code(S,T) + \Omega manual(D,S,T) + \Omega execution(D,T)$$

Avec $\Omega catalog(S)$ est l'effort pour comprendre le catalogue, $\Omega code(S,T)$ est l'effort pour écrire le code, $\Omega manuel(D,S,T)$ est l'effort pour effectuer tout

travail manuel, et $\Omega execution(D,T)$ est l'effort d'exécuter le code. Comme il existe plusieurs métriques qui pourraient être dérivées d'une requête pour estimer $\Omega code(S,T)$ et un bon nombre de ces métriques sont également pertinentes pour mesurer la complexité de la requête, A. Vashistha et S. Jain. ont donc décidé de se baser sur leur institution et leur expérience pour sélectionner les métriques suivantes pour avoir un impact sur la complexité de la requête :

- Le nombre de tables dans un requête ou le nombre de colonnes dans un requête
- La longueur d'une requête
- Le nombre d'opérateurs dans une requête comme Scan, Join, Filter
- Le nombre d'opérateurs d'expression dans une requête comme LE, LIKE, GT, OR, AND, Count
- Le temps d'exécution d'une requête

Ils mènent ensuite une enquête auprès d'experts en bases de données pour identifier l'importance relative des métriques pour la complexité des requêtes, puis analysent les résultats de l'enquête sur trois aspects:

- **Agreement between Experts** : En effectuant le test Kendall's W, ils peuvent déterminer l'accord entre les différents sous-ensembles de notre groupe d'experts. Ils ont évalué le W de Kendall pour 33 sous-groupes et la valeur de W pour chaque sous-groupe est présenté dans la figure 1.

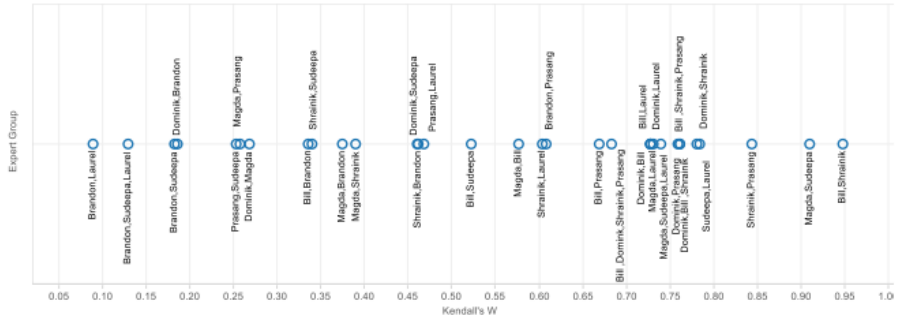

Fig. 1. Mesurer l'accord entre les experts

- **Important Metrics** : Dans le but de mesurer à quel point les métriques sont similaires ou différentes les unes des autre, ils ont effectué le test de Friedman et n'ont trouvé aucune preuve convaincante que les mesures diffèrent les unes des autres. Cela s'est peut-être produit à cause de données insuffisantes.
- **Correlation among Metrics** : Ils ont effectué des tests de coefficient de corrélation de rang Kendall et de coefficient de corrélation de rang de Spearman pour mesurer la corrélation entre les métriques et ont constaté qu'il existe en effet une corrélation négative statistiquement significative entre le temps d'exécution d'une requête et le nombre d'expressions.

Ensuite, ils ont calculé le score de complexité pour 117 requêtes distinctes dans l'ensemble représentatif. En considérant l'effort cognitif dans l'écriture des requêtes comme la mesure de la complexité, ils ont utilisé le **Ground Truth Coding** avec le processus de rating se composait de trois passages: les auteurs ont évalué toutes les requêtes indépendamment; puis les auteurs ont revu le score de l'autre pour mettre en évidence les notes incohérentes données par l'autre et enfin les auteurs ont revu les requêtes mises en évidence par l'autre auteur pour réviser leur note ou conserver l'ancienne note. En considérerant cet effort cognitif et attribuer un nombre plus élevé, ils indiquent une complexité plus élevée. Ces scores de complexité seront ensuite utilisés comme vérité terrain pour d'autres expériences. Ils ont donc effectué 3 moyens d'évaluer la complexité des requêtes suivant:

- **Experts' Ranking** : Demander aux experts d'évaluer chacune des six métriques sur une échelle pour indiquer dans quelle mesure la métrique détermine la complexité d'une requête selon leurs expériences. Puis calculer un poids normalisé pour chaque métrique en analysant les scores donnés et au final, calculer le score de complexité comme la somme des multiplications de poids pour chaque métrique et sa valeur.
- **Halstead Mesures**: Ils ont adapté la mesure de complexité de Halstead pour déterminer la complexité de la requête dans l'ensemble représentatif en utilisant une formule faisant intervenir le nombre d'opérateurs distincts, le nombre d'opérandes distincts, le nombre total d'opérateurs et le nombre total d'opérandes
- **Regression for Query Complexity** Ils ont exécuté une régression linéaire avec le score de complexité dépendant des 6 métriques : nombre de tables, nombre de colonnes, longueur de la chaîne de la requête, nombre d'opérateurs, nombre d'expressions et durée d'exécution de la requête. Ici A. Vashistha et S. Jain ont utilisé le *Ground Truth Coding*.

A côté des travaux de A. Vashishtha et S. Jain, il convient de mentionner l'ouvrage "Towards a Metrics Suite for Object Oriented Design" écrit par Shyam R. Chidamber et Chris F. Kemerer [9] . Dans leur travail, ils ont proposé des métriques pour mesurer la complexité des logiciels ainsi que leur évaluation. Ces métriques sont spécifiquement conçues pour la conception orientée objet et je trouve qu'elles sont difficiles à appliquer à notre cas. Notre workload SQLShare est conçue pour être une base de données relationnelle et a trop peu de ressemblance avec une base de données orientée objet, donc ces métriques sont moins représentatives de la complexité que nous visons.

3.2 Application dans le continuation de travail Mining du workload SQL

La prochaine partie sera dédiée à la mise en œuvre de ces métriques dans notre workload SQLShare, donc obtenir les résultats et les analyser statistiquement et visuellement.

Suite aux travaux de C.Moreau et al, nous avons obtenu un fichier de sortie qui contient les informations des requêtes faites par les utilisateurs de SQLShare. Ce fichier est enregistré au format xml et a une structure relativement standard en tant que fichier de base de données. Voici un exemple de la façon dont les informations sur une requête sont stockées dans ce fichier

```
<exploration>
  <request>
    <id>0</id>
    <string_request>SELECT max(latitude) FROM
        [690].[All3col]</string_request>
    <user>690</user>
    <projections>
      <projection>max(latitude)</projection>
    </projections>
    <selections />
    <function_aggregates>
      <function_aggregate>max(latitude)</function_aggregate>
    </function_aggregates>
    <tables>
      <table>[690].[all3col]</table>
    </tables>
    <columns>
      <column>latitude</column>
    </columns>
    <GroupByClauses />
    <OrderByClauses />
    <TOPClauses />
    <columns />
  </request>
```

Ce fichier est organisé en utilisant différentes balises qui définissent ensuite la portée d'un élément. Ces balises peuvent également être utilisées pour déclarer les paramètres requis pour l'analyse de l'environnement ou pour insérer des autres instructions. Poue ce fichier, il existe trois types de balises :

- balise d'ouverture : $< nom\_balise >$
- balise de fermeture : $< /nom\_balise >$
- balise vide : $< nom\_balise/ >$

la façon dont ce fichier stocke les informations de manière organisée est qu'il comporte de nombreuses explorations, chaque exploration contient de nombreuses requêtes et les informations sur la requête sont décrites dedans. Une balise d'ouverturet immédiatement suivie d'une balise de fermeture comme indiqué et le texte qui apparaît entre la balise d'ouverture et la balise de fermeture sont les informations que nous utiliserons pour l'exploration. Grâce à l'exemple ci-dessus, nous pouvons obtenir des informations à partir de cette requête comme:

- $< id >$: Cette requête est marquée avec l'id 1

– $< string\_request >$: La requête d'origine faite au serveur
– $< user >$: Cette requête est faite par l'utilisateur 690
– $< tables >$: La table affectée dans cette requête est [690].[all3col]
– $< columns >$: La colonne affectée par cette requête est latitude

Et les balises d'élément vide peuvent être utilisées pour tout élément qui n'a pas de contenu. Cela signifie que nous n'aurons pas besoin de nous soucier de ces balises car elles ne fourniront pas d'informations utiles pour le processus d'analyse. Comme dans l'exemple ci-dessus, nous pouvons voir qu'il y a des balises $< GroupByClauses/ >$, $< OrderByClauses/ >$, $< TOPClauses/ >$, cela signifie que cette requête n'aura pas de clauses Group By ou Order By et TOP. . Nous pouvons vérifier via la balise $< string\_request >$ et voir facilement que cette requête n'a pas ces clauses.

Application des méthode pour calculer les métriques de complexité Inspiré des travaux de A.Vashistha et S. Jain, nous avons parlé de 3 métriques importantes qui seront utilisées pour calculer la complexité des requêtes dans notre cas. Dans cette section, je vais proposer mon approche pour calculer ces métriques en extrayant les informations du fichier xml. J'ai utilisé donc principalement Python à l'aide de la bibliothèque xml.etree.ElementTree pour extraire les informations nécessaires.

– **Table Touch** : Cette métrique représente le nombre de table référencée dans une requête. Dans notre fichier xml, les informations nécessaires à son sujet ont déjà été extraites et marquées avec la balise $< tables >$. Par exemple dans la requête ci-dessus, nous pouvons voir que la balise $< table >$ n'apparaît qu'une seule fois, c'est-à-dire qu'il n'y a qu'une seule table affectée, donc sa Table Touch sera 1.
– **Column Touch** : De la même manière avec la Table Touch, cette métrique représente le nombre de colonnes affectées dans une requête. Dans notre fichier xml, les informations nécessaires à son sujet ont déjà été extraites et marquées avec la balise $< columns >$ et dans notre exemple, Column Touch sera 1.
– **Longueur** : Comme son nom l'indique, cette métrique concerne le nombre total de caractères dans une requête. Pour le calculer, j'ai importé le magasin de requêtes dans $< string\_request >$ au format texte et en utilisant la fonction len() pour compter le nombre de caractères dans cette chaîne de caractères.
– **Nombre d'opérateurs** : Cette métrique représente le nombre total d'opérateurs dans une requêt comme Scan, Joint, Filter. En suivant la liste des opérateurs SQL fournie par database.guide [10], j'ai trié puis choisi les bons opérateurs comme suit: SELECT, UNION, CASE, OVER, JOIN (Left Outer Join, Right Outer Join, Inner Join, Full Join), WHERE, HAVING, GROUP BY, ORDER BY, EXCEPT, INTERSECTION, PIVOT/UNPIVOT. En fin, j'ai résumé le nombre d'apparition de ces opérateurs dans la requête.

- **Nombre d'opérateurs d'expression** : Poue cette métrique, j'ai traité de la même manière que le précédent sur mais dans ce cas, je me concentre sur le nombre d'expressions dans la requête. J'ai donc choisi une liste d'opérateurs d'expression SQL fournie par database.guide [10] comme suit: ALL, AND, ANY, BETWEEN, EXISTS, IN, LIKE, NOT, OR, SOME, LE, GT, COUNT, WITH, TOP, PARTITION BY, SUM, MOD, MAX, MIN.
- **Runtime** : Cette métrique n'est pas réalisable dans notre cas. De plus, si l'on considère son application lors du calcul du score de complexité, elle est relativement petite (dans le classement par experts, le score standard pour l'exécution est de 0,11, le plus petit; le même avec la régression pour la formule de complexité de la requête, son coefficient n'est que de 8,89E-0,7 en comparaison avec le deuxième plus petit b = 0,000168). C'est pourquoi j'ai décidé de ne pas considérer cette métrique dans notre calcul.

Après avoir obtenu ces métriques, j'ai ensuite appliqué pour calculer le score de complexité en utilisant différentes approches. Reprenant la même partie avec A. Vashishtha et S. Jain, j'ai calculé 3 scores de complexité différents.

- **Classement des experts**: J'ai calculé le score de complexité comme la somme des multiplications du poids normalisé pour chaque métrique et sa valeur.

$$Score = \sum W_i * V_i$$

Table 1. Tableau des poids normalisés

Table Touch	Column Touch	Longeur	Nombre opérateurs	Nombre expressions
0.18	0.14	0.19	0.19	0.2

- **Mesures de Halstead**: J'ai utilisé cette formule adaptée pour calculer le score de complexité en utilisant le nombre d'opérateurs et d'opérandes.

$$Score = n1/2 * N2/n2 * log_2(n1 + n2)$$

Ici, n1 est le nombre d'opérateurs distincts ou le nombre d'opérateurs et d'expressions, n2 est le nombre d'opérandes distincts ou de colonnes distinctes référencés dans une requête. N1 est le nombre total d'opérateurs et N2 est le nombre total d'opérandes. Pour le mettre dans la formule relative aux métriques que j'ai montrées auparavant :
$n1 = nombre\_operators + nombre\_expression$ et $n2 = Column\_Touch$
avec $N1 = \sum n1$ et $N2 = \sum n2$ Ce que je veux noter pour ce score, c'est qu'il est possible que ce score ait la valeur 0 (il n'y a pas d'opérateurs ni d'opérateurs d'expression dans la requête) et/ou infinitif (il n'y a pas de colonnes affectées dans la requête). Dans le second cas, je prendrai une valeur un peu plus élevée que la valeur maximale calculée à travers le Column Touch dans les données.

– **Régression pour la formule de complexité** : En utilisant une régression avec validation croisée 10 fois, A. Vashishtha et S. Jain ont estimé les valeurs d'un ensemble de coefficients à appliquer pour la formule obtenue à l'aide de la régression. Dans notre cas, cette formule prend la forme ci-dessous

$$Score = a * Table\_Touch + b * Column\_Touch + c * Longueur$$

$$+d * Nombre\_operateurs + e * Nombre\_expressions$$

Table 2. Coefficients issus de la régression

a	b	c	d	e
-0.00248	0.000168	0.001571	0.012903	0.000355

J'ai ensuite obtenu les données de résultat rassemblés dans une dataframe, ce qui sera utile pour une analyse plus approfondie. Tout mon travail peut être trouvé le reproduit en utilisant ce lien Google Colab `https://colab.research.google.com/drive/1_BBbWTCLNnQ8kdgBPNskDp41hYCHM8fy?usp=sharing`.

id	exploration	uid	tab_touch	col_touch	length	nb_operators	nb_expressions	experts_ranking	regression	halstead
0	1	690	1	1	41	1	1	8.50	0.075357	8.018801e+04
1	1	690	1	2	95	1	1	18.90	0.160359	5.059300e+04
2	1	690	1	2	167	2	3	33.17	0.287084	1.775406e+05
3	1	690	1	2	218	3	4	43.25	0.380463	2.806580e+05
4	1	690	1	2	192	3	4	38.31	0.339617	2.806580e+05

Fig. 2. Dataframe contient les informations nécessaires

Le premier problème auquel j'ai été confronté est que bien que j'aie répertorié certains opérateurs et opérateurs d'expression, certains d'entre eux peuvent ne pas être utilisés dans toutes les explorations. Un profilage des données est donc nécessaire pour les éliminer. En même temps, il est utile d'avoir une vue d'ensemble sur nos données.
Le résultat est qu'il y a bien des opérateurs inutilisés, qui sont INTERSECTION, UNPIVOT, ANY et SOME. Cela nous donne aussi d'autres informations comme: Il y a un total de 451 explorations. Les 3 explorations avec le plus grand nombre de requêtes sont la 164e (937 requêtes), 365 (539 requêtes) et 328 (536 requêtes) ainsi que les explorations avec seulement 1 requête comme 4 ou 10. Il y a un total de 97 utilisateurs. Le 3 utilisateur le plus actif avec le plus grand nombre de requêtes effectuées est 1123 (1620 requêtes), 412 (1597 requêtes) et 1314 howe (883 requêtes).

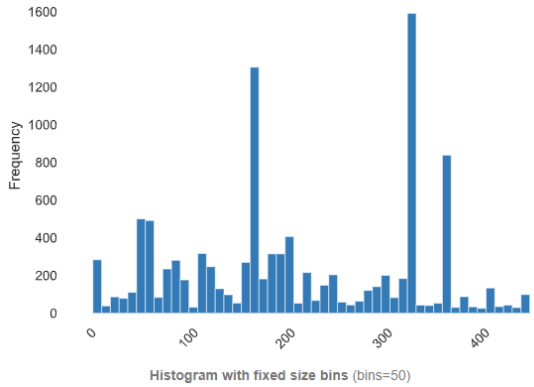

Fig. 3. Nombre des requêtes dans chaque exploration

Ensuite, je veux avoir une vue plus approfondie des propriétés des requêtes. Je refais donc aussi la même chose pour les métriques de complexité calculées. Le Table Touch nous indique combien de tables sont affectées dans une requête et dans notre ensemble de données, les 4 apparences le plus courant est 1 (8025 fois qui prend 75,2%), 2 (1868 fois qui prend 17,5%), 3 (296 fois qui prend 2,8%) et 0 (213 fois ce qui prend 2,0%). Le même analyste pour le Column Touch, le nombre de colonnes affectées dans chaque requête, j'obtiens les 4 apparences le plus courant est 2 (2240 fois qui prend 21,0%), 1 (2211 fois qui prend 20,7%), 3 (1231 fois qui prend 11,5%) et 0 (1020 fois qui prend 9,6%).

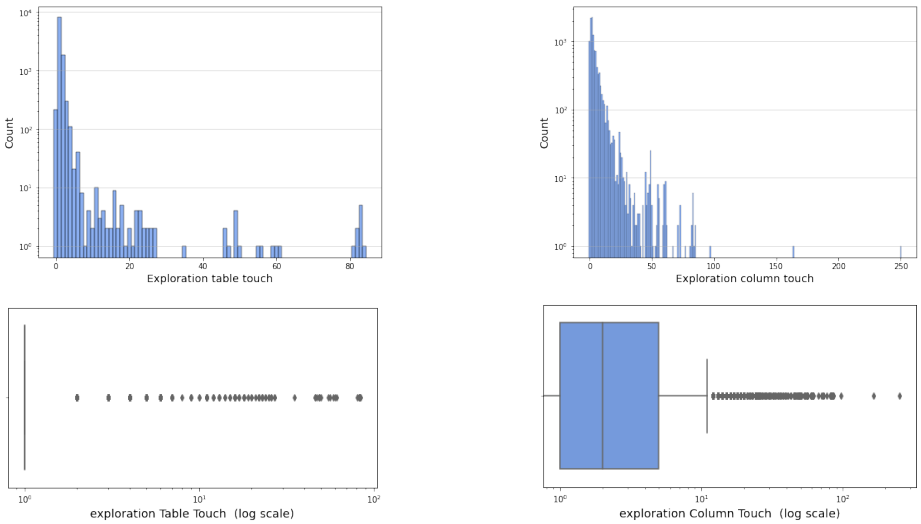

Fig. 4. Distribution de Table Touch et Column Touch

Pour la longueur de la requête, il a une plage de valeurs relativement diversifiée avec la valeur minimale de 8 et la valeur maximale de 22315 et la moyenne de 267,5.

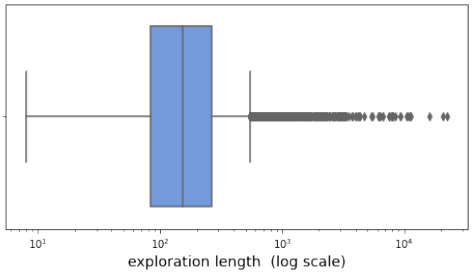

Fig. 5. Distribution de la longueur

Un rapide aperçu de la distribution du score de complexité calculé à l'aide de 3 méthodes différentes permet également selon l'exploration d'apporter également quelques premières observations sur les données.

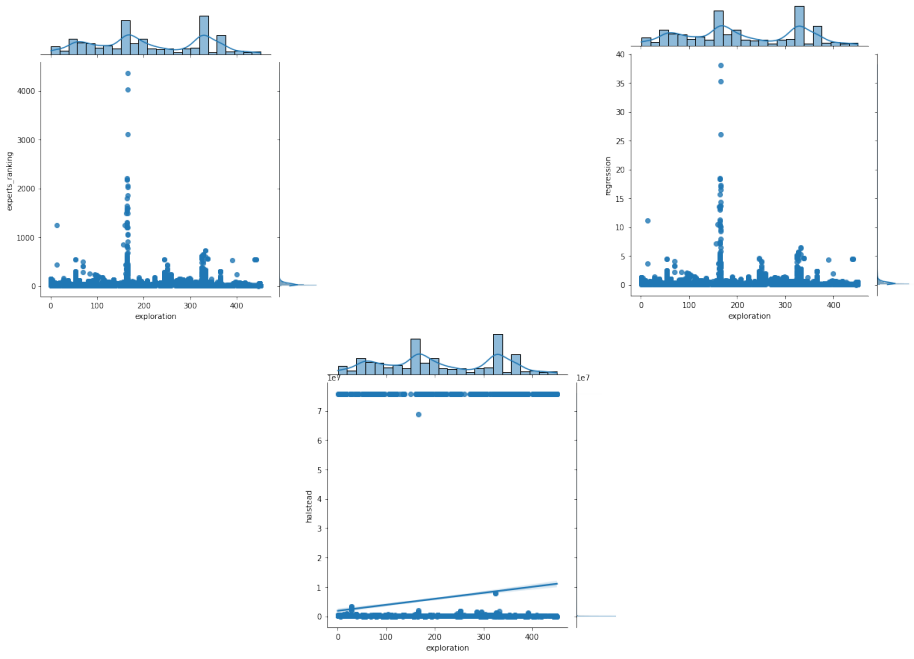

Fig. 6. Classement des experts(gauche) - Formule de régression(droit) - Mesure Halstead(bas)

En partageant une formule similaire avec le coefficient de chaque métrique. la distribution du score calculé à l'aide du classement des experts et de la régression par formule semblent partager de nombreuses similitudes. Mais dans le cas de la mesure de Halstead, parce que sa formule n'est liée qu'au nombre d'opérateurs et d'opérandes, il y a une grande différence dans sa distribution.
Nous pouvons expliquer cela par la différence de point de vue dans chaque score. Le classement des experts a une mesure de poids par les experts en données et la régression est avec validation croisée 10 fois, ils sont du point de vue de l'utilisateur tandis que la mesure de la complexité halstead est du point de vue informatique en s'inspirant des mesures de complexité cyclomatiques, qui sont utilisées pour estimer le nombre des défauts d'un programme et de la complexité d'un code [2]. Par exemple, la longueur d'une requête a un grand impact dans le calcul de ces scores, dans les deux cas, elle a un coefficient relativement élevé dans la formule mais il est mentionné pour le calcul le score de Halstead Mesure.

Comme le montre la matrice de corrélation illustrée dans la figure ci-dessous, certaines des caractéristiques sont corrélées.

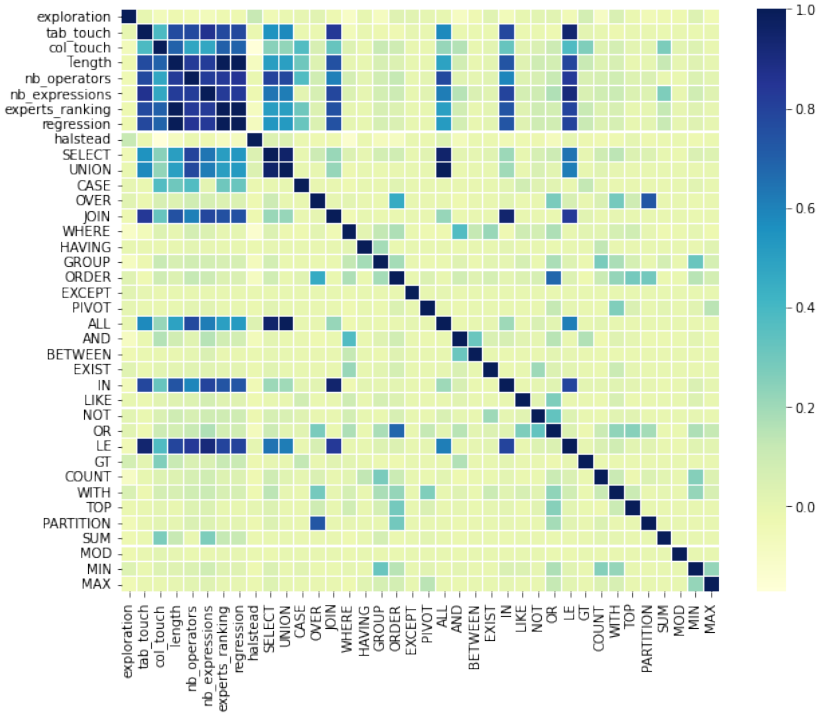

Fig. 7. Matrice de corrélation entre toutes les fonctionnalités de requête

Je ne commente ici que les corrélations majeures. Comme nous l'avons déjà expliqué précédemment, la longueur d'une requête a un impact énorme sur les scores basés sur le classement des experts et la régression. Nous pouvons facilement voir qu'ils sont corrélés les uns aux autres.

Méthode de clustering

K-means Clustering
Pour mieux classer les différents groupes de requêtes, l'étape suivante consiste à implémenter une méthode de clustering. Tout d'abord, j'ai essayé de regrouper en fonction de chaque score seul, un par un pour trouver s'il existe des informations sur lesquelles nous pouvons exploiter. L'algorithme de choix est le clustering K-means et il classe avec succès 4 clusters lorsqu'il est appliqué dans le cas du score de classement des experts et du score de régression et 3 clusters pour le cas du score de Mesure Halstead.

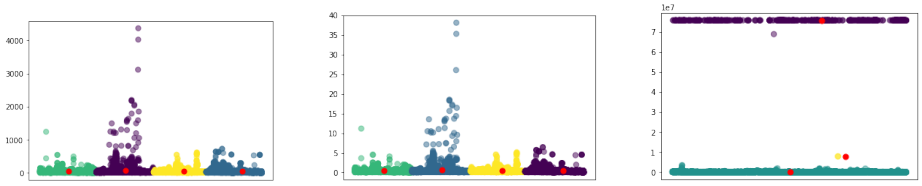

Fig. 8. Clustering par K-means: Classement des experts- Regression - Mesure Halstead

Parmi les clusters Experts Score et Regression, il existe un cluster qui a notamment un score de complexité plus élevé que l'autre. Du point de vue des utilisateurs, cela peut s'expliquer par leur courbe d'apprentissage. Avec le temps, lorsqu'un utilisateur commence à écrire des requêtes SQL, il ne peut pas en faire une trop compliquée. C'est pourquoi dans le premier cluster, la plupart du temps a un faible score de complexité. Il arrive parfois que ce score augmente, cela pourrait hypothétiquement que des utilisateurs essaient de tester des requêtes plus complexes qu'ils ont trouvées ailleurs. Après ce début difficile, l'utilisateur a appris toutes les bases et souhaite les tester lui-même en faisant des requêtes beaucoup plus compliquées. Ce faisant, ils ajoutent de plus en plus d'opérateurs, d'expressions même de manière non optimisée, ce qui rend le score de complexité très élevé. Dans le cluster suivant, les utilisateurs commencent à apprendre à rédiger des requêtes de manière plus optimisée en utilisant les opérateurs avec précaution, ce qui fait baisser le score pendant cette période. Dans le dernier, les utilisateurs ont appris à écrire des requêtes bonnes et optimisées, ils peuvent parfois en écrire une plus complexe sans trop compliquer la requête.
Le problème se produit lorsque vous essayez d'appliquer la mesure halstead, qui est davantage basée sur le côté informatique. D'après ce que je peux observer,

la plupart des requêtes ont des points faibles car elles n'affectent que 0 ou 1 colonnes. Il existe des cas où les requêtes sont beaucoup plus complexes et nous voyons qu'il est dans la même période de pic avec le classement des experts ou la régression. En haut, il y a une "ligne", cela représente le score lorsque la valeur de CT est égale à 0 ou disons d'une autre manière, ce sont les requêtes qui n'affectent aucune colonne. Cette "ligne" qui traverse le graphique est assez simple à expliquer car le fait qu'un utilisateur écrive une requête qui ne contient aucune colonne est tout à fait normal.

Combinaision de CED, UMAP et DBSCAN proposé par C.Moreau et al.
Inspiré des travaux de C.Moreau et al, une autre façon est en utilisant la méthode de clustering est d'utiliser la combinaison de CED, UMAP et DBSCAN. Cela a été prouvé par des tests utilisant les workloads OLAP [3]. Des explications supplémentaires sur le fonctionnement de cette combinaison sont décrites dans leur travail "Mining SQL Workloads for Learning Analysis Behavior" (July 10,2021). Dans mon travail, j'ai simplement réappliqué les fonctions créées par eux avec une touche pour bien adapter à mes fonctions actuelles.

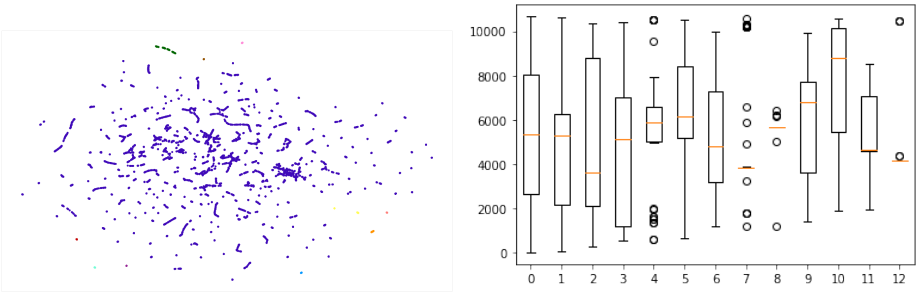

Fig. 9. Clustering selon les fonctionnalités fournies avec le box plot la norme des scores dans chaque cluster

Ce que nous obtenons est un très gros cluster au milieu et d'autres clusters beaucoup plus petits entourent ce gros. Une façon d'expliquer ce résultat est que la plupart des utilisateurs ici sont des utilisateurs "standard", qui ont une bonne connaissance de la façon d'écrire une bonne requête SQL mais n'écrivent pas ou rarement une requête compliquée.

4 Conclusion

La complexité des requêtes est intrinsèquement difficile à quantifier. Dans ce travail, mon objectif principal est d'appliquer les méthodes précédentes et les résultats d'autres études pour essayer de découvrir plus de modèles dans la charge de travail de SQLShare. En appliquant les métriques proposées précédemment,

je peux confirmer que lors de la mesure de la complexité d'une requête, il existe des facteurs plus importants pour la complexité de la requête (c'est-à-dire le nombre d'opérateurs et d'expressions) et des facteurs moins importants (c'est-à-dire le temps d'exécution de la requête).
Ce projet a été réalisé sur une vraie charge de travail SQLShare, permettant l'extraction des informations des utilisateurs, permettent d'étudier plus sur la complexité d'une requête ainsi que le comportement des utilisateurs,

References

1. S. Jain, D. Moritz, D. Halperin, B. Howe, and E. Lazowska. Sqlshare: Results from a multi-year sql-as-a-service experiment. In SIGMOD, 2016.
2. A. Vashistha and S. Jain. Measuring query complexity in sqlshare workload. https://uwescience.github.io/sqlshare/pdfs/Jain-Vashistha.pdf.
3. Clément Moreau, Clément Legroux, Verónika Peralta, Mohamed Ali Hamrouni: Mining SQL Workloads for Learning Analysis Behavior. Submitted to Information Systems, July 2021.
4. Neil, Immerman: Descriptive Complexity. New York, NY: Springer New York. ISBN 9781461205395. OCLC 853271745. (1999)
5. Halstead, Maurice H. : Elements of Software Science. Amsterdam: Elsevier North-Holland, Inc. ISBN 0-444-00205-7.(1977)
6. Thomas J. McCabe, Arthur H. Watson: Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric. NIST Special Publication 500-235. 1996 http://www.mccabe.com/pdf/mccabe-nist235r.pdf
7. Vijay Kotu, Bala Deshpande, in Data Science (Second Edition), 2019 https://www.sciencedirect.com/topics/computer-science/data-exploration
8. A. Vashistha, S. Jain. Measuring : "Query Workload Analysis"
9. Shyam R. Chidamber, Chris F. Kemerer: Towards a Metrics Suite for Object Oriented Design , ACM 89791-446-5/91/0010/0197 (1991) https://dspace.mit.edu/bitstream/handle/1721.1/49292/towardsmetricssu00chid.pdf?s..
10. https://database.guide/sql-operators/ Posted on November 20, 2020 by Ian