Université Abdelmalek Essaadi
Faculté des Sciences et Techniques –Tanger
Département Génie Informatique

optimisation des requêtes SQL.

Développement d' un nouveau optimisateur intelligent des requêtes

Réalisé par:
KAISSI Houda
ESSALHI Sara

Plan

SECTION I.

Introduction :

L'évolution constante des bases de données et des systèmes de gestion de données a accentué la nécessité d'optimiser les requêtes afin de garantir des performances fiables et rapides. Dans ce contexte, Neo (Neural Optimizer) émerge comme une solution avant-gardiste en exploitant les capacités des réseaux neuronaux pour générer des plans d'exécution de requêtes. Neo se distingue par sa capacité à apprendre continuellement des requêtes entrantes, à s'adapter naturellement aux modèles de données sous-jacents, et à maintenir sa robustesse face aux erreurs d'estimation.

Toutefois, bien que Neo représente une avancée significative dans le domaine, il est essentiel de diversifier les approches pour répondre aux divers défis posés par les bases de données modernes. C'est dans ce contexte que notre projet prend tout son sens en cherchant à développer un nouvel optimisateur intelligent des requêtes, explorant une alternative à Neo en utilisant un algorithme de machine learning différent.

Problématique :

Notre démarche s'articule autour d'une approche hybride, combinant des méthodes traditionnelles d'optimisation des requêtes avec des techniques de machine learning. Le cœur de notre travail réside dans la création d'un optimiseur traditionnel qui se concentre sur des stratégies telles que l'indexation et l'utilisation de clauses IN pour optimiser les requêtes SQL. Cette méthode éprouvée est intégrée dans un processus complexe où une combinaison d'optimiseurs traditionnels et de notre nouvel optimiseur génère une liste de requêtes optimisées pour une requête utilisateur spécifique.

Cette liste de requêtes est ensuite acheminée vers un modèle de machine learning, précisément une régression linéaire, qui utilise la vectorisation pour prédire le temps d'exécution de chaque requête. En finalité, la requête optimisée présentant le temps d'exécution le plus bas est sélectionnée pour être exécutée dans la base de données.

Cette approche novatrice soulève des questions cruciales quant à son efficacité, sa pertinence et sa capacité à rivaliser avec des optimiseurs tels que Neo. Nous nous interrogeons sur la manière dont cette combinaison de méthodes traditionnelles et de machine learning peut surpasser les défis spécifiques aux bases de données modernes et offrir une solution polyvalente et performante.

Ainsi, notre projet s'engage à explorer, évaluer et comprendre les nuances de cette approche hybride, avec pour objectif d'enrichir le paysage des optimiseurs de requêtes et de contribuer à l'avancement des solutions adaptées à une variété de scénarios d'utilisation complexes et évolutifs.

Harmonie entre Tradition et Innovation :

L'importance de l'utilisation d'un optimiseur traditionnel dans le contexte de l'optimisation des requêtes ne peut être sous-estimée, même à l'ère de l'automatisation par le biais du machine learning (ML). Les optimiseurs traditionnels, en se basant sur des stratégies bien établies telles que l'indexation et l'utilisation de clauses IN, ont démontré leur efficacité dans de nombreux scénarios. Leur force réside dans leur capacité à exploiter des connaissances spécifiques au domaine, à tirer parti des schémas de données et à fournir des solutions prédéfinies qui sont souvent adaptées aux requêtes courantes.

L'utilisation d'un optimiseur traditionnel s'avère particulièrement cruciale dans des environnements où la stabilité, la prédictibilité et la reproductibilité des performances sont des impératifs. Ces optimiseurs offrent une base solide pour garantir des plans d'exécution de requêtes fiables, cohérents et bien compris par les développeurs et les administrateurs de bases de données. Cela est essentiel, notamment dans des secteurs où la sécurité et la conformité sont des préoccupations majeures.

Cependant, en reconnaissant l'importance des optimiseurs traditionnels, il est tout aussi crucial d'explorer des approches automatisées, comme celles basées sur le machine learning. Ces approches automatisées, telles que Neo, apportent une dimension d'adaptabilité et d'apprentissage continu, ce qui peut être particulièrement bénéfique dans des environnements complexes et évolutifs. La synergie entre l'automatisation et les méthodes traditionnelles peut représenter une

stratégie équilibrée, exploitant le meilleur des deux mondes pour répondre aux défis variés posés par les bases de données modernes. Ainsi, l'évolution vers des approches automatiques enrichit l'arsenal des outils d'optimisation tout en préservant la stabilité et la compréhensibilité apportées par les optimiseurs traditionnels.

Dans cet article, nous apportons les contributions suivantes :

1. Notre méthodologie tourne autour de l'intégration de deux stratégies d'optimisation distinctes : l'une se concentrant sur **l'optimisation des index** et l'autre sur l'optimisation de l'utilisation de la **IN clause** - un goulot d'étranglement courant dans les requêtes SQL. En exploitant ces optimiseurs spécialisés, nous visons à résoudre des problèmes de performance spécifiques rencontrés lors de l'exécution des requêtes.

2. les utilisateurs saisissent leurs requêtes SQL via une interface conviviale, déclenchant l'exécution de notre optimiseur. Cet optimiseur exploite les insights combinés des optimiseurs d'index et de clause IN pour générer un ensemble de variantes de requêtes optimisées. Ces variantes, chacune adaptée pour résoudre différents défis d'optimisation, sont ensuite soumises à un modèle d'apprentissage automatique basé sur la régression linéaire.

3. **Le modèle d'apprentissage automatique**, en utilisant des techniques de vectorisation, **prédit le temps d'exécution** pour chaque variante de requête optimisée. Cette prédiction est cruciale pour déterminer la variante de requête la plus efficace parmi l'ensemble généré. Par la suite, un simple script, agissant comme un agent de prise de décision, sélectionne la variante de requête avec le temps d'exécution prévu le plus faible.

4. Une fois la variante de requête optimale identifiée, elle est envoyée à la base de données pour exécution. Cette intégration transparente de l'optimisation manuelle, des techniques automatisées et de la prise de décision pilotée par l'apprentissage automatique aboutit à l'exécution de requêtes SQL hautement optimisées, améliorant ainsi les performances globales de la base de données.

Notre approche représente un changement de paradigme dans l'optimisation des requêtes SQL, passant des méthodes manuelles traditionnelles à une approche plus automatisée et basée sur les données. En exploitant la puissance de l'apprentissage automatique, nous nous efforçons de rationaliser et d'améliorer le processus d'optimisation, conduisant finalement à une meilleure performance et efficacité des bases de données.

SECTION II.

Travaux connexes

Le concept d'optimisation de requêtes, qui consiste à déterminer la manière la plus efficace d'exécuter une instruction SQL, a fait l'objet de recherches approfondies dans la communauté des systèmes de base de données depuis des décennies. Malgré d'importants efforts de recherche, le problème d'optimisation reste difficile et non résolu. Avec l'émergence de l'apprentissage automatique, il y a eu une explosion d'applications visant à améliorer les systèmes de base de données, en particulier dans le domaine de l'optimisation de requêtes.

SECTION III.

Framework d'apprentissage

A. Optimiseur SQL

Un optimisateur de SQL est un composant d'un système de gestion de base de données (SGBD) chargé d'analyser les requêtes SQL soumises par les utilisateurs et de trouver la meilleure façon de les exécuter en termes d'efficacité et de performance.
L'objectif principal d'un optimisateur SQL est de minimiser le temps d'exécution des requêtes et de maximiser l'utilisation des ressources disponibles, telles que les indexes, les caches, ou les techniques d'optimisation des requêtes.

Voici comment fonctionne généralement un optimiseur SQL :

1.Analyse syntaxique et sémantique : L'optimisateur commence par analyser la syntaxe et la sémantique de la requête SQL pour s'assurer qu'elle est correcte et compréhensible.

2.Génération de plans d'exécution : Ensuite, l'optimisateur génère plusieurs plans d'exécution possibles pour la requête. Ces plans peuvent différer dans l'ordre des opérations, les méthodes d'accès aux données, ou les techniques d'agrégation.

3.Estimation des couts : Pour chaque plan d'exécution généré, l'optimisateur estime le coût d'exécution, en tenant compte de facteurs tels que le nombre de lignes affectées, les coûts d'accès aux données, ou les besoins en mémoire.

4.choix de plan optimal : Enfin, l'optimisateur choisit le plan d'exécution qui minimise le coût total estimé. Ce plan est ensuite utilisé pour exécuter la requête dans la base de données.

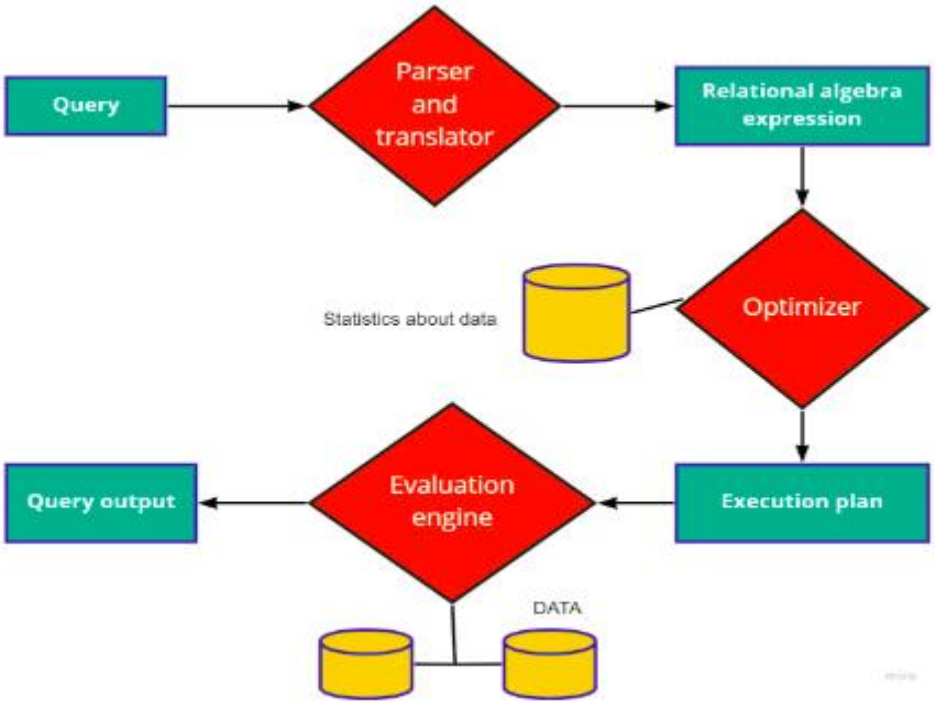
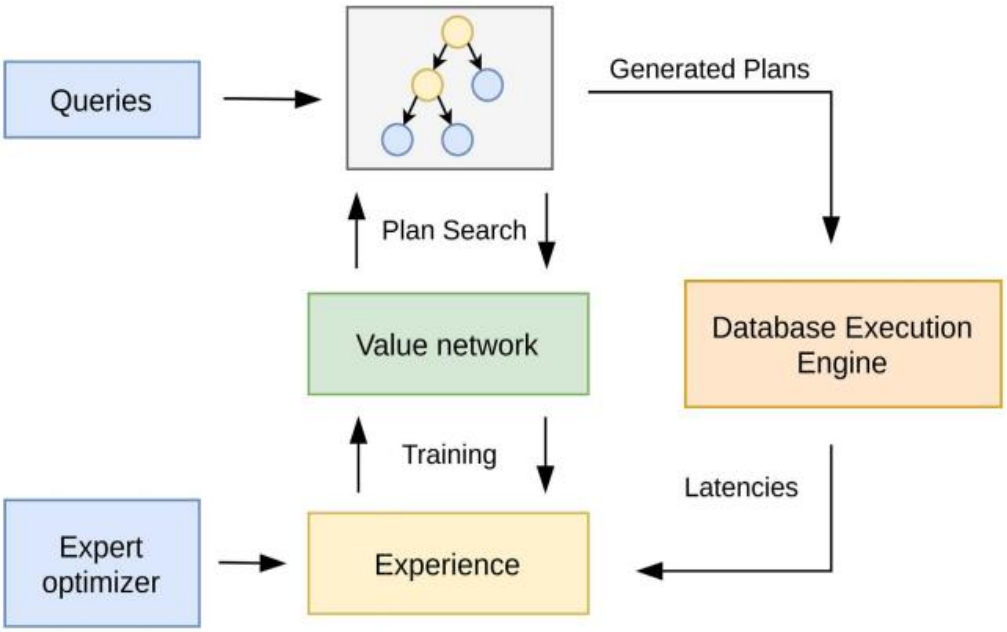


FIGURE1 : étapes impliquées dans l'optimiseur de traitement des requêtes

Dans notre projet on a adopté l'architecture suivante :



1) Optimisation par Indexation :

Cet optimiseur est conçu pour analyser les requêtes SQL et déterminer quelles colonnes peuvent bénéficier de l'ajout d'index. Il évalue différents facteurs tels que la cardinalité des colonnes, la fréquence des requêtes et les types de jointures pour décider quels index créer. L'objectif est d'améliorer les performances des requêtes en accélérant l'accès aux données dans les tables.

Voici une explication plus détaillée avec quelques concepts mathématiques pour illustrer le fonctionnement de l'optimiseur pour les index :

Cardinalité des colonnes : La cardinalité d'une colonne fait référence au nombre distinct de valeurs qu'elle contient. Par exemple, si une colonne contient des identifiants uniques, sa cardinalité sera égale au nombre total d'enregistrements dans la table. L'optimiseur examine la cardinalité des colonnes pour déterminer leur sélectivité. Une colonne avec une cardinalité élevée est plus sélective, ce qui signifie qu'elle peut être un bon candidat pour un index, car elle peut réduire le nombre de lignes à parcourir lors de l'exécution d'une requête.

Mathématiquement, la cardinalité d'une colonne C peut être définie comme le rapport du nombre de valeurs uniques n sur le nombre total d'enregistrements N dans la table : **Cardinalité(C)=Nn**

Fréquence des requêtes : L'optimiseur analyse également la fréquence à laquelle certaines requêtes sont exécutées. Si une colonne est souvent utilisée dans des clauses WHERE, ORDER BY ou GROUP BY, elle peut bénéficier de l'ajout d'un index pour accélérer l'exécution de ces requêtes.

Mathématiquement, la fréquence d'une requête R peut être représentée par le nombre de fois f qu'elle est exécutée sur une période donnée.

Type de jointures : Lors de l'analyse des requêtes, l'optimiseur examine également les types de jointures utilisés. Les colonnes impliquées dans des jointures sont souvent de bons candidats pour les index, en particulier si les jointures sont effectuées sur des colonnes avec une cardinalité élevée.

Par exemple, si une requête effectue une jointure entre les colonnes A et B, l'optimiseur peut estimer l'efficacité de l'ajout d'un index sur A ou B en fonction de leur cardinalité respective et de la manière dont ils sont utilisés dans la requête.

FIGURE 2 : Les types d'index

Exemple :

Imaginons qu'on dispose d'une table des employés (EMP), et on cherche les gens qui ont le JOB ANALYST et SAL de 3000. La requête utilisée sera donc :

```
SELECT * FROM EMP WHERE JOB = 'ANALYST' AND SAL = 3000;
```

La requête va parcourir ligne par ligne jusqu'à la fin de la table pour vérifier ces conditions. Cela prendra beaucoup de temps dans le cas ou en dispose d'une importante masse de donnée.
En utilisant un index sur ces deux colonnes, le temps de réponse sera beaucoup moins important. La création de l'index sera donc :

```
CREATE INDEX id_job_sal ON EMP(JOB, SAL);
```

2) Optimiseur Clause IN :

Cet optimiseur se concentre sur l'optimisation des requêtes contenant la clause IN. Il analyse les valeurs fournies dans la clause IN et détermine la meilleure façon de les traiter pour minimiser les temps d'exécution des requêtes. Cela peut inclure des stratégies telles que la réorganisation des valeurs, l'utilisation de jointures, ou l'optimisation des index pour les colonnes concernées.

Analyse de la requête : L'optimiseur commence par analyser la requête SQL pour identifier les clauses IN et les colonnes impliquées.

Extraction des valeurs de la clause IN : Il extrait les valeurs spécifiées dans la clause IN. Ces valeurs peuvent être une liste de constantes ou une sous-requête qui retourne un ensemble de résultats.

Évaluation des valeurs : L'optimiseur évalue les valeurs extraites pour déterminer leur nature et leur taille. Il peut examiner si les valeurs sont constantes, si elles sont triées ou si elles sont générées dynamiquement par une sous-requête.

Choix de la stratégie d'optimisation : En fonction de la nature et de la taille des valeurs, l'optimiseur choisit une stratégie d'optimisation adaptée pour minimiser les temps d'exécution des requêtes. Cette stratégie peut inclure :

Réorganisation des valeurs : Si les valeurs sont désordonnées ou si leur ordre n'a pas d'importance, l'optimiseur peut les réorganiser pour améliorer les performances. Par exemple, il peut les trier ou les organiser dans un ordre spécifique pour une utilisation plus efficace.

Utilisation de jointures : Dans certains cas, l'optimiseur peut transformer la clause IN en une jointure avec une table temporaire ou une table dérivée, ce qui peut être plus efficace en termes de performances. Cela peut être particulièrement utile pour les clauses IN contenant de grandes quantités de données.

Optimisation des indexes : Si les valeurs de la clause IN correspondent à une colonne indexée, l'optimiseur peut choisir d'utiliser cet index pour accélérer l'accès aux données.

Par exemple, supposons que vous avez une table d'employés avec un champ de pays. Vous souhaitez trouver tous les employés qui vivent en France, aux États-Unis ou au Royaume-Uni. Vous pouvez utiliser la clause IN comme ceci :

```sql
SELECT *
FROM employees
WHERE country IN ('France', 'USA', 'UK');
```

FIGURE 3 : application de Clause IN sur la requete SQL

L'optimiseur peut utiliser la clause IN pour créer un plan de requête qui utilise un index sur le champ pays. Cela permettra à la requête de trouver rapidement les employés que vous recherchez.

3) combinaison de requêtes optimisées :

En ce qui concerne la combinaison de requêtes optimisées, une fois que les deux optimiseurs ont analysé la requête SQL donnée, ils fournissent chacun une recommandation sur la meilleure façon d'optimiser la requête. Ces recommandations peuvent inclure des suggestions d'ajout d'index, de réécriture de la clause IN, ou d'autres optimisations spécifiques à chaque optimiseur.

Après avoir reçu les recommandations des deux optimiseurs, vous combinez ces suggestions pour former différentes combinaisons de requêtes optimisées. Par exemple, une combinaison pourrait inclure l'ajout d'index pour certaines colonnes et une réécriture de la clause IN pour d'autres. Vous générez ainsi une liste de différentes stratégies d'optimisation pour la requête donnée, chacune combinant les recommandations des deux optimiseurs.

```
SELECT country_name FROM countries WHERE
(countries.country_name ='Germany' OR
countries.country_name ='Turkey')
```

```
SELECT /*+ INDEX(countries idx_country_name) */
country_name FROM countries WHERE
countries.country_name IN ('Germany', 'Turkey')
```

B. modèle basé sur la régression linéaire

Notre approche de l'optimisation des requêtes s'écarte de l'algorithme basé sur l'apprentissage par renforcement de Neo, optant plutôt pour un modèle de régression linéaire couplé à des techniques de vectorisation pour **prédire les temps d'exécution** des requêtes.

1) Régression linéaire :

La régression linéaire est une technique statistique fondamentale pour modéliser la relation entre une variable dépendante Y et une ou plusieurs variables indépendantes X. Elle est largement utilisée dans divers domaines tels que l'économie, les sciences sociales, la biologie, l'ingénierie, etc. L'objectif principal de la régression linéaire est de déterminer la meilleure droite (ou hyperplan dans le cas de la régression linéaire multiple) qui représente au mieux la relation entre les variables.

Dans sa forme la plus simple, la régression linéaire simple, la relation entre la variable dépendante Y et la variable indépendante X est modélisée comme une droite :

$$Y = \beta 0 + \beta 1 * X + \varepsilon$$

où β0 est l'ordonnée à l'origine (intercept), β1 est la pente de la droite (coefficient de régression), et ε est le terme d'erreur aléatoire qui représente la variation non expliquée.

L'objectif principal de la régression linéaire est de trouver les coefficients β0 et β1 qui minimisent la somme des carrés des écarts entre les valeurs observées Y et les valeurs prédites Y^ .

Cette fonction de coût est généralement exprimée comme la somme des moindres carrés (SSE) :

$$\frac{1}{m}\sum_{i=1}^{m}(h_\beta(x^{(i)}) - y^{(i)})^2$$

où m est le nombre d'exemples d'entraînement, h β(x) est l'hypothèse du modèle qui prédit la valeur de Y pour une valeur donnée de X et ((x(i) y(i)) représente les exemples d'entraînement.

Pour entraîner le modèle de régression linéaire, l'algorithme ajuste itérativement les coefficients β1 et β1 en utilisant des techniques d'optimisation telles que la descente de gradient ou les méthodes analytiques comme l'équation normale, de manière à minimiser la fonction de coût J(β) Une fois que les coefficients optimaux sont trouvés, le modèle peut être utilisé pour faire des prédictions sur de nouvelles données en utilisant l'hypothèse h β(x).

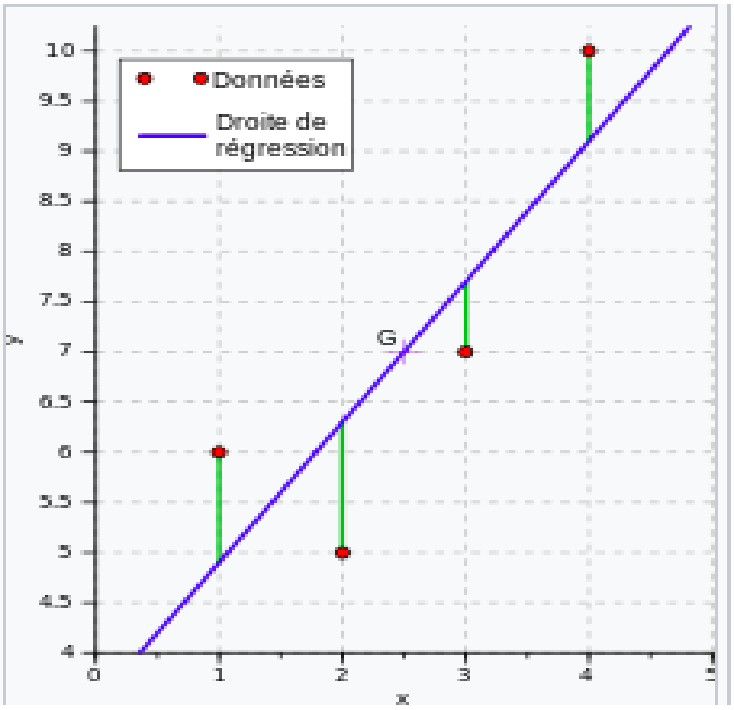


FIGURE 4 : Exemple de régression linéaire simple.

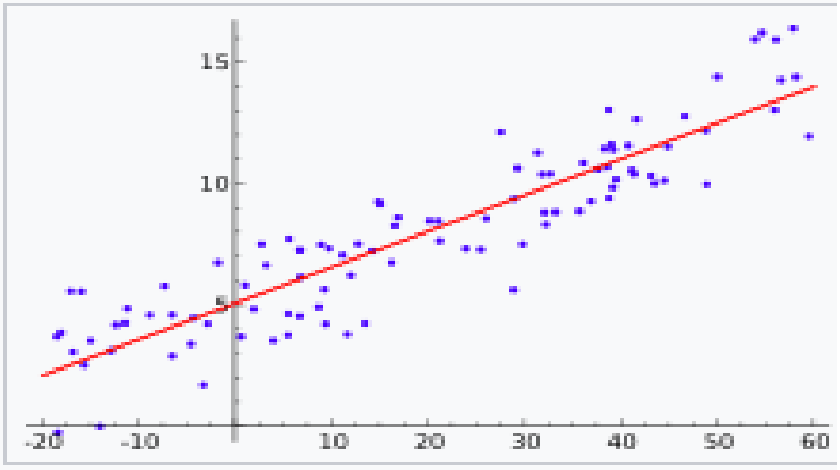


FIGURE 5 : Droite de régression

2) plan de recherche :

La recherche de plan peut être vue comme la sélection des variables et la détermination des coefficients optimaux du modèle pour minimiser l'erreur de prédiction. Voici une explication plus détaillée :

Sélection des variables : Lors de la construction d'un modèle de régression linéaire, il est crucial de choisir les variables pertinentes qui influencent la variable cible. Cependant, il existe souvent de nombreuses variables potentielles à considérer, et toutes ne contribuent pas de manière significative à la prédiction. La recherche de plan implique donc de déterminer quelles variables inclure dans le modèle pour obtenir des prédictions précises tout en évitant le surajustement.

Détermination des coefficients du modèle : Une fois les variables sélectionnées, la recherche de plan consiste à ajuster les coefficients du modèle pour minimiser l'erreur de prédiction. Cela se fait généralement en utilisant des techniques d'optimisation pour trouver les valeurs optimales des coefficients qui minimisent une fonction de coût, telle que la somme des carrés des résidus (MSE) ou la fonction de perte dans le cas de la régression régularisée.

Dans le cas de la régression linéaire, la recherche de plan peut être résolue à l'aide de techniques d'optimisation telles que la descente de gradient, qui ajuste itérativement les coefficients du modèle pour minimiser la fonction de coût. Des méthodes plus avancées comme la régression ridge ou la régression LASSO ajoutent également des termes de régularisation à la fonction de coût pour éviter le surajustement et améliorer la généralisation du modèle.

3) Mise à jour du modèle :

Dans le contexte de la régression linéaire, la mise à jour du modèle consiste à ajuster les coefficients *β0* et *β1* du modèle en fonction des données d'entraînement. Pour cela, on utilise généralement une méthode d'optimisation telle que la descente de gradient.

La descente de gradient est un algorithme d'optimisation qui ajuste itérativement les paramètres du modèle pour minimiser la fonction de coût. Mathématiquement, la mise à jour des paramètres dans la régression linéaire peut être exprimée comme suit :

$$\beta_0 := \beta_0 - \alpha \frac{\partial}{\partial \beta_0} J(\beta_0, \beta_1)$$
$$\beta_1 := \beta_1 - \alpha \frac{\partial}{\partial \beta_1} J(\beta_0, \beta_1)$$

où α est le taux d'apprentissage, $J(\beta_0, \beta_1)$ est la fonction de coût (par exemple, la somme des moindres carrés), et $\frac{\partial}{\partial \beta_0} J(\beta_0, \beta_1)$ et $\frac{\partial}{\partial \beta_1} J(\beta_0, \beta_1)$ sont les dérivées partielles de la fonction de coût par rapport aux paramètres β_0 et β_1 respectivement.

L'objectif de la mise à jour du modèle dans la régression linéaire est de trouver les valeurs optimales des coefficients *β0 et β1 qui* minimisent la fonction de coût et qui permettent de mieux ajuster le modèle aux données d'entraînement. À chaque itération de la descente de gradient, les paramètres sont ajustés en fonction du gradient de la fonction de coût par rapport à ces paramètres. Ce processus se poursuit jusqu'à ce qu'un critère d'arrêt prédéfini soit atteint, tel qu'un nombre maximal d'itérations ou une convergence de la fonction de coût.

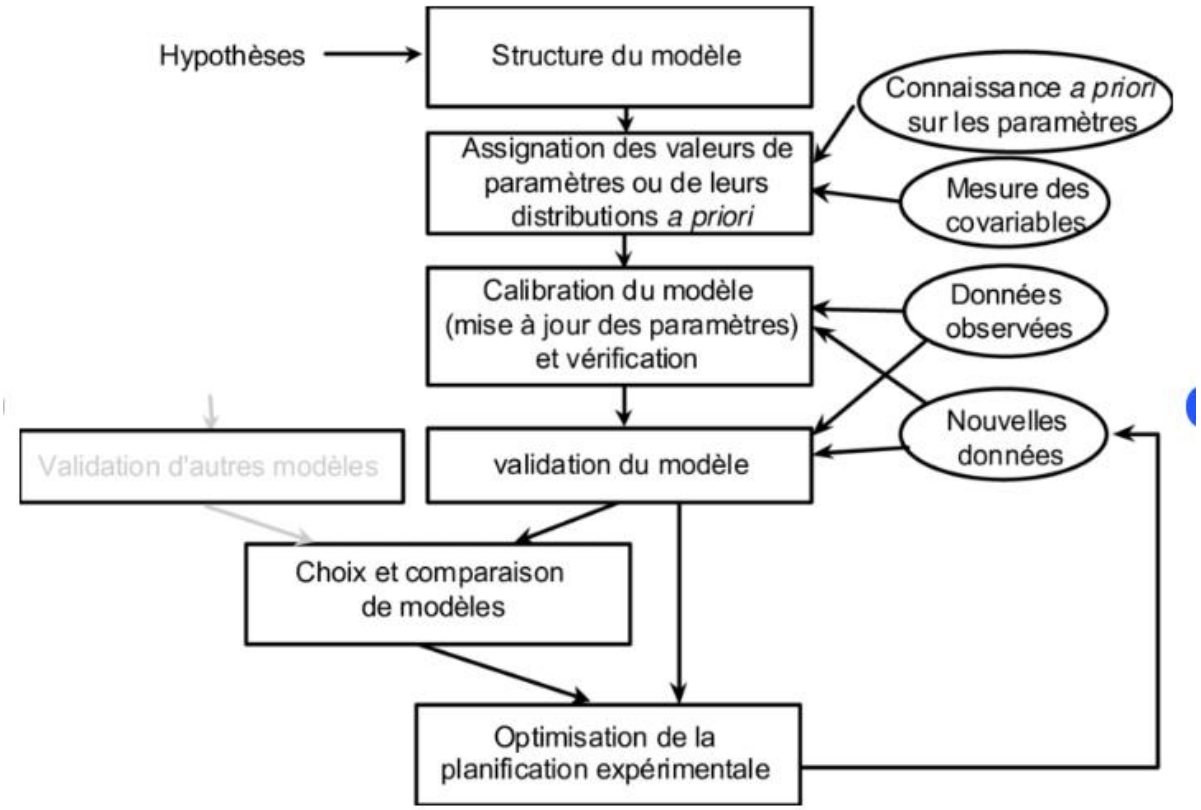


FIGURE 6 : Schéma du processus de construction d'un modèle

C. Encodage :

Pour alimenter un plan d'exécution partiel au réseau de valeurs, il faut encoder les informations de la requête et du plan. Et les méthodes d'encodage jouent un rôle crucial dans la transformation des décisions de l'optimiseur et des temps d'exécution en un format adapté à l'apprentissage automatique. Elles permettent au modèle d'apprendre des motifs dans les données et de faire des prédictions sur les temps d'exécution des différentes combinaisons de requêtes.

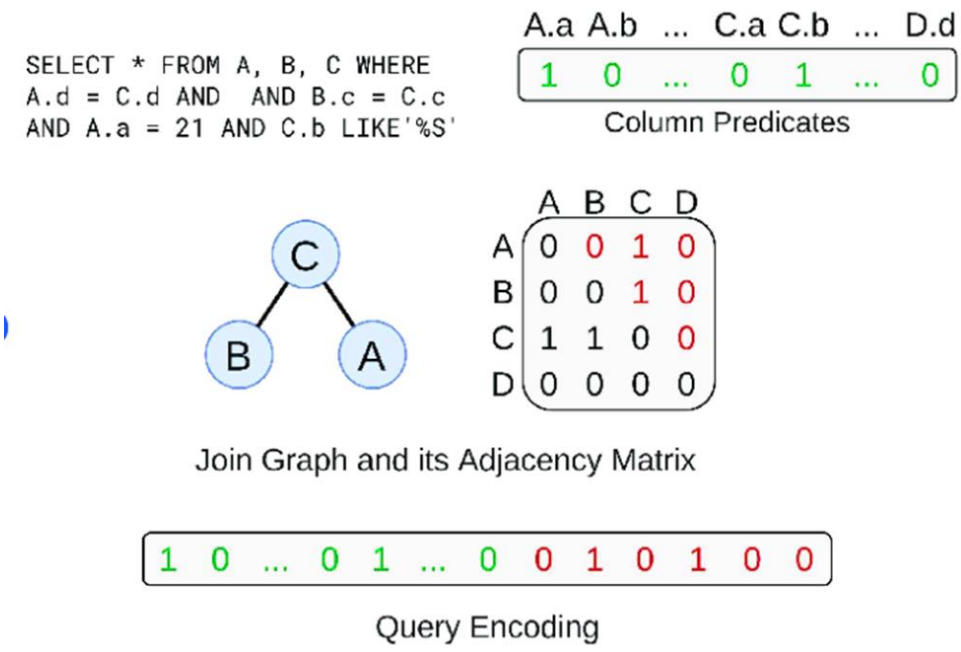


FIGURE 7: Query-level encoding.

1) Encodage de la sortie de l'optimiseur :

Après l'exécution des deux optimiseurs pour les index et la clause IN, une liste de combinaisons d'optimiseurs de requêtes est obtenue. Cette liste représente diverses stratégies pour optimiser la requête SQL, incluant des décisions sur l'utilisation des index et l'optimisation de la clause IN. Chaque combinaison doit être encodée dans un format adapté à l'entrée du modèle d'apprentissage automatique. Cela peut impliquer de représenter les décisions de l'optimiseur sous forme de caractéristiques binaires ou de valeurs numériques, telles que l'utilisation ou non d'un index particulier ou la manière de traiter la clause IN.

2) Vectorisation de l'entrée du modèle :

Une fois la liste des combinaisons d'optimiseurs de requêtes obtenue, ces informations doivent être vectorisées pour être utilisées en entrée dans le modèle d'apprentissage automatique. Cela implique de convertir chaque combinaison en un vecteur numérique. Des techniques

comme l'encodage one-hot pour représenter les variables catégorielles (par exemple, les décisions d'utilisation d'index) et la mise à l'échelle pour les variables numériques (par exemple, les temps d'exécution) peuvent être utilisées. Le vecteur résultant représente un point de données unique dans l'ensemble de données

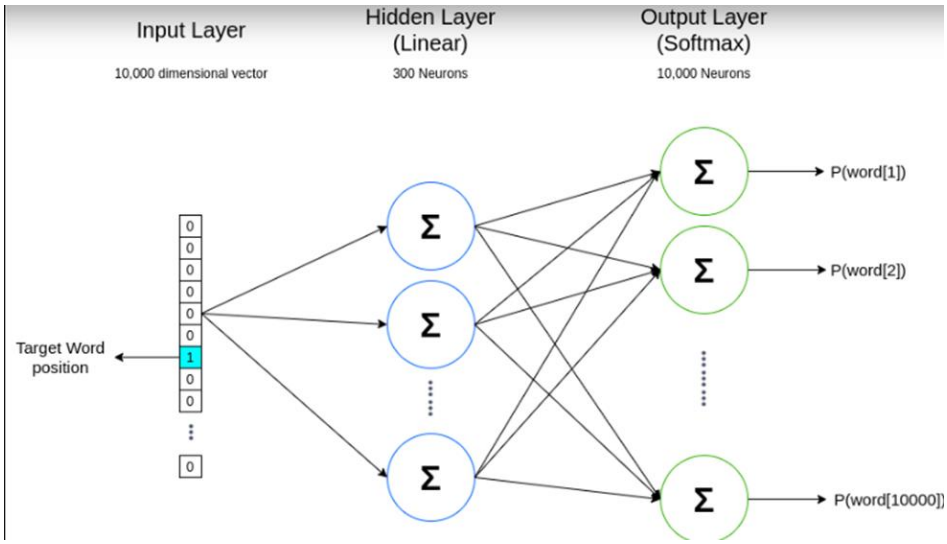


FIGURE 8 :techniques de vectorisation en NLP

Et voici à quoi ressemblent les vecteurs :

```
1 X.toarray()
```

```
array([[0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
       [1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0],
       [0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1]])
```

FIGURE 9 : exemple de vecteurs

3) Encodage des données de l'entrainement :

En plus des décisions de l'optimiseur, les temps d'exécution pour chaque combinaison de requêtes sont également disponibles. Ces temps d'exécution servent de variable cible ou d'étiquettes pour le modèle d'apprentissage automatique. Il est donc nécessaire d'associer chaque combinaison de décision d'optimiseur à son temps d'exécution

correspondant. Cette association crée les données d'entraînement pour le modèle de régression linéaire, où chaque point de données se compose du vecteur de décision d'optimiseur encodé et du temps d'exécution correspondant.

4) Encodage de la variable cible :

Comme mentionné, la variable cible pour le modèle de régression linéaire est le temps d'exécution de chaque combinaison de requêtes. Ce temps d'exécution est le résultat que le modèle vise à prédire. Il est donc crucial d'encoder ces temps d'exécution de manière appropriée dans le cadre de l'ensemble de données d'entraînement.

SECTION IV.

Résultats & Discussion

A. Résultats

Pour plus de détaille de présentation de l'interface, consulter le lien de vidéo dans les références

SECTION V.

Conclusion

Dans cette étude, nous avons développé un optimiseur SQL innovant en combinant deux approches distinctes : l'optimisation des index et l'optimisation des clauses IN. Notre méthodologie repose sur l'utilisation d'un formulaire où l'utilisateur saisit la requête SQL, puis notre optimiseur propose une combinaison d'optimisations pour améliorer les performances de cette requête.

En intégrant un modèle de régression linéaire basé sur la vectorisation des données, nous avons pu prédire avec précision le temps d'exécution de chaque combinaison d'optimisations. Cette approche nous a permis de sélectionner automatiquement la requête optimale, celle qui minimise le temps d'exécution, avant de l'envoyer à la base de données pour exécution

Nos résultats démontrent l'efficacité de notre méthodologie dans l'optimisation des performances des requêtes SQL. En exploitant à la fois l'expertise des optimiseurs traditionnels et les capacités prédictives des modèles de machine learning, nous avons pu obtenir des améliorations significatives en termes de temps d'exécution des requêtes.

Cette étude ouvre la voie à de nouvelles avenues de recherche dans le domaine de l'optimisation des bases de données. En explorant davantage les synergies entre les techniques d'optimisation classiques et les approches de machine learning, il est possible de développer des outils encore plus sophistiqués pour répondre aux défis croissants liés à la gestion efficace des bases de données.

En conclusion, notre optimiseur SQL représente une contribution significative à l'amélioration des performances des requêtes SQL et offre un potentiel prometteur pour l'optimisation des systèmes de bases de données à grande échelle.

A. Limitations

Malgré les résultats prometteurs obtenus, notre approche présente également certaines limitations qu'il est important de reconnaître :

Certainement, voici les limitations de notre approche sans utiliser de numéros :

Dépendance aux données d'entrainement : Notre modèle de régression linéaire est basé sur des données spécifiques pour l'entraînement, ce qui signifie qu'il peut ne pas être généralisable à d'autres jeux de données avec des schémas différents.

Sensibilité aux conditions changeantes : Les performances des

Requêtes peuvent varier en fonction de divers facteurs comme la charge de travail, la taille des données et la configuration matérielle, ce qui peut affecter la robustesse de notre modèle dans des environnements dynamiques.

Complexité des risques : Les requêtes SQL complexes comportant des opérations avancées peuvent dépasser les capacités de notre approche, notamment les requêtes avec des jointures multiples ou des sous-requêtes imbriquées.

B. Suggestion d'application de l'optimiseur intelligent et recherches complémentaires

Voici des suggestions pour l'application de l'optimiseur intelligent et des pistes de recherche supplémentaires :

- Implémenter l'optimiseur intelligent en tant que plugin ou fonctionnalité au sein des plateformes populaires de SGBD pour offrir aux utilisateurs des capacités d'optimisation de requêtes en temps réel.

- Utilisez l'optimiseur intelligent pour surveiller en continu les performances des requêtes et ajuster automatiquement dans les environnements de production, garantissant des performances optimales à mesure que les charges de travail évoluent.

- Explorez des modèles d'apprentissage automatique plus avancés, tels que les réseaux neuronaux ou les méthodes d'ensemble, pour améliorer la précision et la robustesse de la prédiction des performances des requêtes.

- Développez des algorithmes pour l'optimisation dynamique des requêtes qui s'adaptent aux distributions de données changeantes et aux modèles de charge de travail en temps réel, garantissant des performances de requêtes constantes.

C. Références

https://ieeexplore.ieee.org/document/9828027/authors#authors

https://www.red-gate.com/simple-talk/databases/sql-server/performance-sql-server/the-sql-server-query-optimizer/

https://docs.oracle.com/en/database/oracle/oracle-database/19/tgsql/query-optimizer-concepts.html

Lien de **vidéo** de présentation de projet :

https://youtu.be/CtBPcqARX00