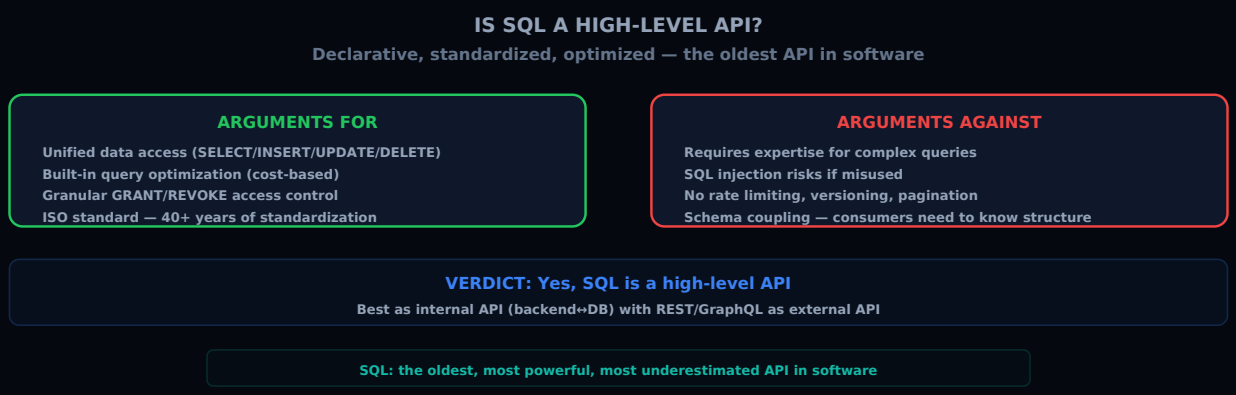
Le SQL est-il une API de haut niveau ?

Sylvain ARBAUDIE · 18 mai 2025

SQL ARCHITECTURE API OPINION



La question qui dérange

Dans l'architecture logicielle moderne, tout passe par des APIs. REST, GraphQL, gRPC — les applications communiquent via des interfaces bien définies, documentées, versionnées. Mais il existe une API qui précède toutes les autres, qui est omniprésente, et que personne n'appelle vraiment "API" : **SQL**.

SQL est un langage déclaratif standardisé (ISO/IEC 9075) qui permet d'interagir avec des données structurées. Vous déclarez ce que vous voulez, pas comment l'obtenir. Le moteur de base de données s'occupe de l'exécution. C'est, par définition, une interface de programmation — une API.

Alors pourquoi personne ne traite SQL comme une API ?

SQL comme API : les arguments pour

Accès unifié aux données

SQL offre un point d'accès unique à des données complexes. Qu'il s'agisse de lire une ligne, de joindre dix tables, d'agréger des millions de lignes ou de modifier des données en masse, l'interface est la même : des instructions SQL envoyées via un protocole réseau (le protocole MariaDB / MySQL, par exemple).

C'est exactement ce que fait une API REST : exposer un point d'accès unifié à des ressources, avec une sémantique claire (GET = lire, POST = créer, etc.). SQL fait la même chose avec SELECT, INSERT, UPDATE, DELETE.

Optimisation intégrée

Quand vous appelez une API REST, c'est votre code backend qui décide comment exécuter la requête. Quand vous envoyez une requête SQL, c'est l'optimiseur de la base de données qui choisit le meilleur plan d'exécution.

L'optimiseur SQL analyse la requête, évalue les statistiques des tables, considère les index disponibles et génère un plan d'exécution optimal. C'est une couche d'abstraction que peu d'APIs offrent : vous dites "quoi", le système décide "comment".

Contrôle d'accès granulaire

SQL intègre un système de contrôle d'accès natif. Les GRANT et REVOKE permettent de contrôler finement qui peut faire quoi sur quelles données. C'est l'équivalent d'un système d'autorisation intégré à l'API.

```
GRANT SELECT (product_name, price) ON catalog.products TO 'api_user'@'%';
```

Cet utilisateur peut lire le nom et le prix des produits, mais pas les coûts internes ni les marges. Le contrôle d'accès est au niveau de la colonne.

Standardisation

SQL est un standard ISO. Malgré les variations entre les implémentations (MariaDB, MySQL, PostgreSQL, Oracle), le cœur du langage est commun. Un développeur qui connaît SQL peut interagir avec n'importe quelle base relationnelle.

C'est un avantage que peu d'APIs ont. REST n'est pas un standard strict (c'est un style architectural), GraphQL est un standard plus récent, gRPC est lié à Protocol Buffers. SQL a plus de 40 ans de standardisation.

SQL comme API : les arguments contre

Expertise requise

SQL est puissant mais complexe. Écrire un SELECT simple est accessible à tout développeur. Écrire une requête performante avec des jointures complexes, des CTEs récursifs et des window functions demande une expertise significative.

Une bonne API REST/GraphQL abstrait cette complexité : le consommateur n'a pas besoin de savoir comment les données sont organisées physiquement. Avec SQL, le consommateur doit comprendre le schéma, les relations et les contraintes de performance.

Injection SQL

Le risque d'injection SQL est le talon d'Achille de SQL-en-tant-qu'API. Si le consommateur construit ses requêtes par concaténation de chaînes, les données sont en danger.

```
# DANGEREUX — injection SQL possible
query = f"SELECT * FROM users WHERE name = '{user_input}'"

# SÉCURISÉ — requête paramétrée
query = "SELECT * FROM users WHERE name = %s"
cursor.execute(query, (user_input,))
```

Les APIs REST/GraphQL n'ont pas ce problème fondamental : l'interface est séparée des données par construction.

Pas de gestion des connexions

SQL ne gère pas la notion de session HTTP, de rate limiting, de pagination standardisée, ni de versioning d'API. Le protocole MariaDB / MySQL gère les connexions, mais il n'y a pas d'équivalent aux headers HTTP, aux codes de statut 4xx/5xx, ni aux mécanismes de cache HTTP.

Ces fonctionnalités doivent être implémentées au-dessus de SQL, via des proxies (MaxScale, ProxySQL) ou des couches applicatives.

Couplage au schéma

Exposer SQL directement couple le consommateur au schéma physique de la base. Si vous renommez une colonne, ajoutez une table ou modifiez une relation, toutes les requêtes SQL du consommateur doivent être mises à jour. Une API REST ou GraphQL isole le consommateur de ces changements via une couche d'abstraction.

Le verdict : oui, mais...

SQL est fonctionnellement une API de haut niveau. Il remplit les critères essentiels : interface standardisée, accès déclaratif aux données, optimisation intégrée, contrôle d'accès.

Mais c'est une API qui nécessite de l'expertise pour être utilisée correctement, qui expose des risques de sécurité si elle est mal utilisée, et qui ne fournit pas les mécanismes de gestion modernes (versioning, rate limiting, pagination).

La meilleure approche est probablement hybride :

- **SQL comme API interne** entre les services backend et la base de données, avec des vues et des procédures stockées comme couche d'abstraction
- **REST/GraphQL comme API externe** pour les consommateurs qui n'ont pas besoin de connaître le schéma physique

SQL n'est pas mort. SQL n'est pas un outil du passé. C'est une API — la plus ancienne, la plus puissante, et la plus sous-estimée de l'écosystème logiciel.

Cet article a été initialement publié sur Medium.