

SQL 4 : SELECT imbriqués

Thibaut Cantaluppi

January 16, 2025

SELECT imbriqués

Il est possible d'utiliser le résultat d'un **SELECT** à l'intérieur d'un autre **SELECT**, souvent dans un **WHERE** ou **HAVING**.

SELECT imbriqués

Dans `eleve(nom, note, ...)`, comment trouver le nom de l'élève ayant la note maximum?

SELECT imbriqués

Dans `eleve(nom, note, ...)`, comment trouver le nom de l'élève ayant la note maximum?

```
SELECT nom FROM eleve  
WHERE note = (SELECT MAX(note) FROM eleve);
```

Autre solution

Dans `eleve(nom, note, ...)`, comment trouver le nom de l'élève ayant la note maximum, sans utiliser de **SELECT** imbriqué?

Dans `eleve(nom, note, ...)`, comment trouver le nom de l'élève ayant la note maximum, sans utiliser de `SELECT` imbriqué?

```
SELECT nom FROM eleve  
ORDER BY note DESC  
LIMIT 1;
```

SELECT imbriqués

Dans `Country(name, pib, population, ...)`, comment trouver le nom des pays ayant un PIB par habitant supérieur à la moyenne mondiale?

SELECT imbriqués

Dans Country(name, pib, population, ...), comment trouver le nom des pays ayant un PIB par habitant supérieur à la moyenne mondiale?

```
SELECT name FROM Country
WHERE (pib / population) > (SELECT AVG(pib / population)
  FROM Country);
```


SELECT imbriqués

Dans `planete(nom, etoile, ...)`, comment trouver les noms des planètes dans le même système que la planète Proxima b?

SELECT imbriqués

Dans `planete(nom, etoile, ...)`, comment trouver les noms des planètes dans le même système que la planète Proxima b?

```
SELECT p1.name FROM planete AS p1
WHERE p1.etoile = (SELECT p2.etoile
FROM planete AS p2 WHERE p2.nom = 'Proxima b');
```

Autre solution

Dans `planete(nom, etoile, ...)`, comment trouver les noms des planètes dans le même système que Proxima b, sans utiliser de **SELECT** imbriqué?

Autre solution

Dans `planete(nom, etoile, ...)`, comment trouver les noms des planètes dans le même système que Proxima b, sans utiliser de `SELECT` imbriqué?

```
SELECT p1.name FROM planete AS p1, planete AS p2
WHERE p1.etoile = p2.etoile AND p2.name = 'Proxima b';
```

Architecture d'une base de données

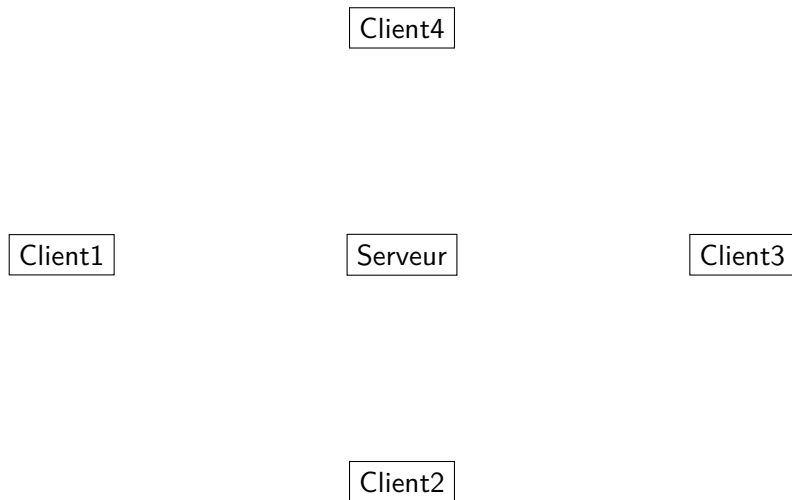
Souvent une base de données doit être accessible par de nombreux utilisateurs.

Comment le faire en pratique?

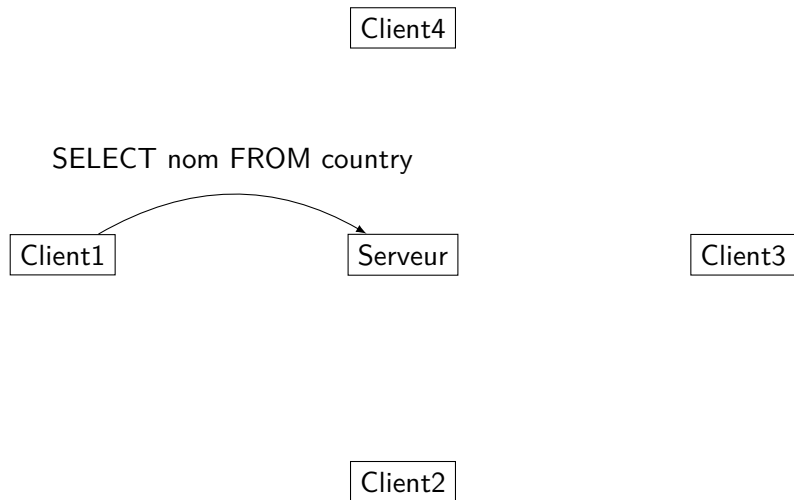
La méthode la plus simple consiste à avoir un **serveur** hébergeant la base de données.

Des clients peuvent alors s'y connecter et faire des requêtes.

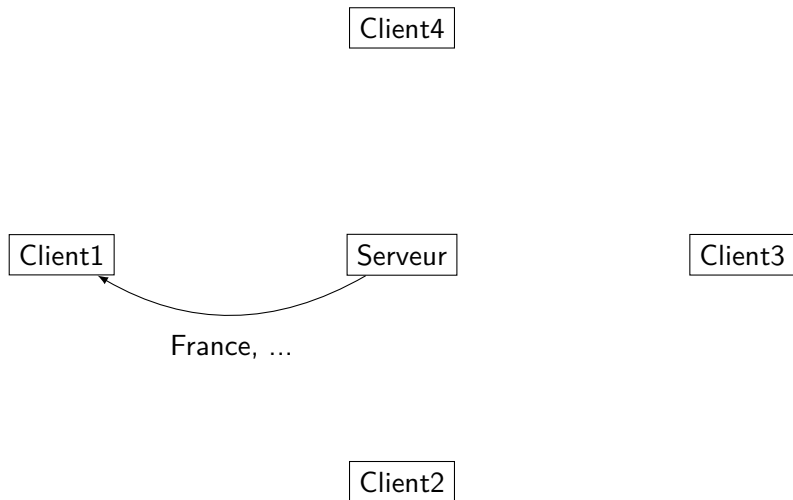
Architecture client-serveur



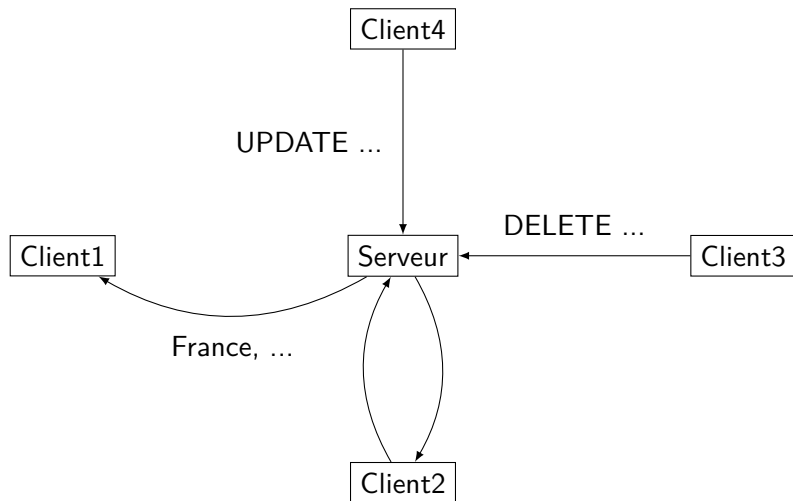
Architecture client-serveur



Architecture client-serveur



Architecture client-serveur



L'architecture client-serveur est omniprésente, et pas seulement pour les bases de données :

- 1 Web : consultation de pages web (HTTP)
- 2 Messagerie électronique (POP, IMAP, SMTP)
- 3 ...

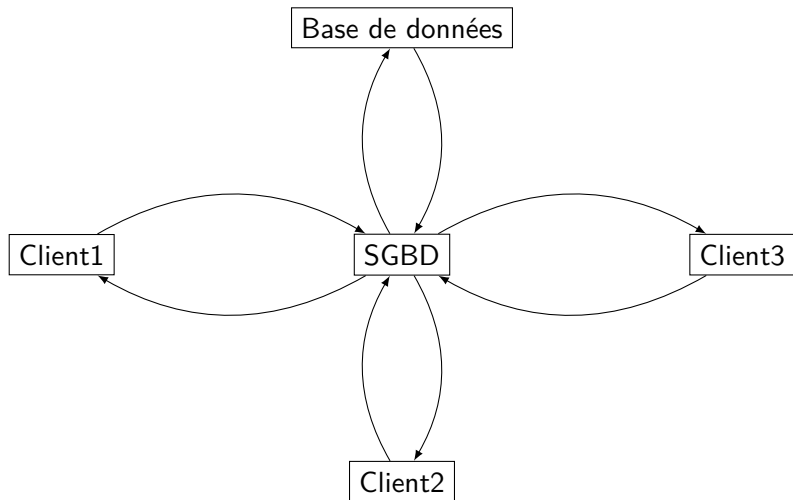
Quelques soucis :

- ❶ Le serveur peut planter au milieu d'une modification
- ❷ Deux personnes peuvent modifier en même temps la base
- ❸ Comment protéger la base de données d'accès non autorisés?

L'**architecture 3-tiers** consiste à dupliquer le serveur en :

- 1 Un serveur contenant la base de données, non accessible par les clients
- 2 Un **système de gestion de bases de données (SGBD)** : une interface (souvent graphique) entre clients et base de données, s'occupant de faire les requêtes et renvoyant le résultat aux client.

Architecture 3-tiers



Intérêts de l'architecture 3-tiers :

- ❶ Le SGBD est le seul pouvant modifier la base, et vérifie que les requêtes des clients ne sont pas malveillantes.
- ❷ Le client n'a pas besoin de connaître SQL, si le SGBD propose une interface graphique.

Souvent, le client communique avec le SGBD via une application web.

Avantage : pas besoin d'installer quoi que ce soit, utilisable sur tout système d'exploitation (Windows, Linux, Mac...).

Question 15 :

Pour les questions 15 à 18 on considère une base de données utilisée dans l'agence de location immobilière CHEZ MOI à Toulouse. Cette base de données contient les deux tables suivantes :

APPARTEMENTS(APT_ID,APT_PRO,APT_VILLE,APT_TARIF,APT_SURF)

PROPRIETAIRES(PRO_ID,PRO_NOM,PRO_PRENOM,PRO_ADRESSE,PRO_TEL)

La première regroupe les données sur les appartements : leur identifiant, l'identifiant de leur propriétaire, la ville, le montant du loyer et la surface.

La deuxième regroupe les données sur les propriétaires des appartements : leur identifiant, le nom, le prénom, leur adresse et le numéro de téléphone.

Les clés primaires sont soulignées.

A quoi servent ces clés ?

Question 16 :

Que fait la requête SQL suivante :

```
SELECT * FROM APPARTEMENTS WHERE APT_SURF>40;
```

- A) Elle sélectionne tous les champs de la table APPARTEMENTS.
- B) Elle sélectionne les données de la table APPARTEMENTS concernant les appartements dont la surface est strictement supérieure à 40.
- C) Elle compte le nombre d'appartement de surface supérieure à 40 dans la table APPARTEMENTS.
- D) Elle n'est pas valide.

Question 17 :

Lesquelles des requêtes suivantes permettent d'obtenir l'identifiant des propriétaires des appartements dont le loyer est le plus élevé ?

- A) `SELECT APT_PRO FROM APPARTEMENTS WHERE APT_TARIF = (SELECT MAX(APT_TARIF) FROM APPARTEMENTS);`
- B) `SELECT APT_PRO FROM APPARTEMENTS WHERE MAX(APT_TARIF);`
- C) `SELECT APT_PRO FROM APPARTEMENTS WHERE APT_TARIF = (SELECT MAX(APT_TARIF) FROM PROPRIETAIRES);`
- D) `SELECT APT_PRO FROM PROPRIETAIRES WITH MAX(APT_TARIF);`

Question 18 :

Quel opérateur est le plus adapté pour obtenir le numéro de téléphone des propriétaires des appartements dont le loyer est le plus élevé.

- A) Une jointure.
- B) Une division cartésienne.
- C) Une intersection.
- D) Il est impossible d'obtenir ces données car les tables contenant les numéros de téléphones et le montant des loyers sont distinctes.