Les règles de conception en SQL pour les requêtes

GAMME CONNECT

Historique de cette documentation

19/09/24	Création de la fiche documentaire.

Avenue des Censives - BP 50333 - 60026 BEAUVAIS Cedex - SAS au capital de 5 100 000 € - 327 733 432 RCS Beauvais

SOMMAIRE

Dans le cadre de l'obsolescence du moteur Microsoft SQL 2008R2 et la poursuite de la maintenabilité de nos progiciels, nous montons de version de compatibilité du moteur Microsoft SQL en Microsoft **SQL 2016 à partir de la version 2024-2 (24.20) d'ISACOMPTA ISAGI CONNECT.**

Certaines permissivités d'écriture de requête en Microsoft SQL 2008R2 ne sont plus tolérées par le moteur Microsoft SQL 2016. A ce titre, il est possible que certains **multiquids ou requêtes Microsoft SQL réalisées par vos équipes** rencontrent des problèmes de performances ou timeout à partir de la version 2024-2 (24.20).

Cette fiche documentaire présente les règles à respecter dans l'écriture de requêtes SQL (MULTIQUID ou SQL).

1. REGLES STANDARDS

1.1 Le SELECT

Evitez les SELECT * FROM t_table.

Explication : Sélectionnez explicitement les colonnes nécessaires.

Evitez les SELECTS à l'intérieur de SELECT.

```
Exemple : SELECT nom_colonne1,
Nom_colonne2,
(SELECT …. FROM t_table2 WHERE …)
FROM t_table1
```

Utilisez des filtres chaque fois que possible (WHERE et HAVING). Les clauses filtrent vos résultats et ne vous donnent que les données que vous souhaitez.

- Utilisez des conditions qui coïncident avec les indexes.

- Préférez l'utilisation de EXISTS à celle de IN dans une clause WHERE.

1.2 Les jointures

- Evitez de trop nombreuses jointures dans une requête.

- Utilisez les CTE pour simplifier des requêtes complexes (avec beaucoup de jointures ou avec le même ensemble de résultats plusieurs fois dans la même requête.).

```
Exemple : WITH CTE (colonne1, colonne2) AS
(SELECT t_colonn1,
T_colonn2
FROM t_table1)
SELECT …. FROM CTE
```

- Utilisez des tables temporaires plutôt que des CTE si les données d'une requête contenant de grande quantité de données sont utilisées dans plusieurs requêtes.

- Utilisez des tables variables plutôt que des tables temporaires pour des données simples et de petite taille.

Explication : En dessous de 1000 lignes Microsoft recommande de passer par des tables variables. Néanmoins il peut y avoir des problèmes de plan d'exécution car le moteur va toujours considérer 1 ligne dans les variables de tables.

- Si vous utilisez un filtre (WHERE) lors de l'exécution d'une jointure, veillez à l'appliquer aux deux tableaux de la jointure.

```
SELECT customer_id, email, order_id, order_total
FROM customers
INNER JOIN orders
ON customers.customer_id=orders.customer_id
WHERE customers.created_at > '1/1/2015'
AND orders.created_at > '1/1/2015'
```

- Lors de jointures entre des vues complexes imbriquées sur des vues complexes, utilisez le HASH JOIN pour améliorer les performances (**Cette solution n'est pas à systématiser et il est préférable de réécrire les requêtes en utilisant les recommandations précédentes**).

Exemple : SELECT colonneId,

colonne2, …

FROM view_table1 t1

INNER HASH JOIN view_table2 t2 ON t1.colonneId=t2.colonneTd

1.3 Le GROUP / ORDER BY

Placez les colonnes des deux clauses GROUP BY et ORDER BY dans le même ordre.

Ex : GROUP BY colonne1, colonne2

ORDER BY colonne1, colonne2

Evitez d'utiliser ORDER BY si non nécessaire. Cela alourdit le coût d'une requête. Préférez son utilisation aux résultats finaux.

2. REGLES SPECIFIQUES

2.1 Les vues

Evitez les fonctions scalaires dans les vues.

Préférez les jointures, les variables de table, les tables temporaires et les CTE **(Attention à ne pas cumuler trop de CTE**) et respecter les recommandations précédentes.

Explication : Les vues ne stockent que la requête, et non les données renvoyées par la requête. Pour chaque ligne de la table impliquée dans la requête, SQL Server doit entrer dans la fonction et exécuter la requête qui s'y trouve à nouveau.

Les données d'une vue sont calculées chaque fois que vous faites référence à la vue dans votre requête. Cela peut entrainer un coût en ressources si vous chercher la performance.

Passez par des Tables Temporaires pour consolider les données des vues va permettre au moteur d'avoir des statistiques et données propre. La réutilisation de ces Tables Temporaires va considérablement améliorer les requêtes.

Exemple : select Colonne1
, Colonne2
into #TT_Table1
from view_Table1

Dans le cadre de multiquid, veuillez utiliser des sous requêtes CTE si la vue peut être filtrée et n'est utilisée que dans une requête principale. Si la vue est utilisée dans plusieurs requêtes, veuillez créer une requête PARADOX.

Exemple : Liste des clients, liste des exercices…

Ces requêtes seront créées chacune dans une requête Paradox pour être utilisée dans les autres requêtes du multiquid.

2.2 Pagination

Pour faire de la pagination, préférez la méthode SEEK à OFFSET

3. ANNEXES

3.1 Types de jointures

Il existe plusieurs méthodes pour associer 2 tables ensemble. Voici la liste des différentes techniques qui sont utilisées :

- **INNER JOIN :** Jointure interne pour retourner les enregistrements quand la condition est vraie dans les 2 tables. C'est l'une des jointures les plus communes.

- **CROSS JOIN :** Jointure croisée permettant de faire le produit cartésien de 2 tables. En d'autres mots, permet de joindre chaque ligne d'une table avec chaque ligne d'une seconde table. Attention, le nombre de résultats est en général très élevé.

- **LEFT JOIN (ou LEFT OUTER JOIN) :** Jointure externe pour retourner tous les enregistrements de la table de gauche (LEFT = gauche) même si la condition n'est pas vérifiée dans l'autre table.

- **RIGHT JOIN (ou RIGHT OUTER JOIN) :** Jointure externe pour retourner tous les enregistrements de la table de droite (RIGHT = droite) même si la condition n'est pas vérifiée dans l'autre table.

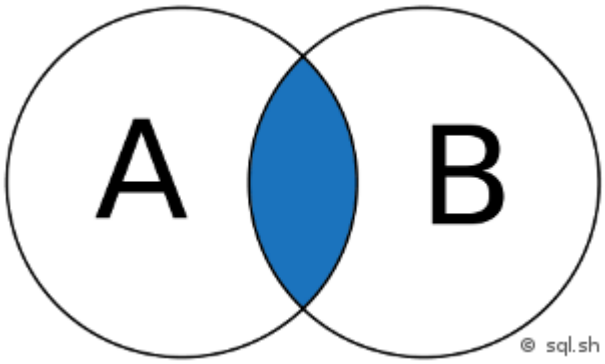
- **FULL JOIN (ou FULL OUTER JOIN) :** Jointure externe pour retourner les résultats quand la condition est vraie dans au moins une des 2 tables.

- **SELF JOIN :** Permet d'effectuer une jointure d'une table avec elle-même comme si c'était une autre table.

- **NATURAL JOIN :** Jointure naturelle entre 2 tables s'il y a au moins une colonne qui porte le même nom entre les 2 tables SQL.

- UNION JOIN : Jointure d'union.

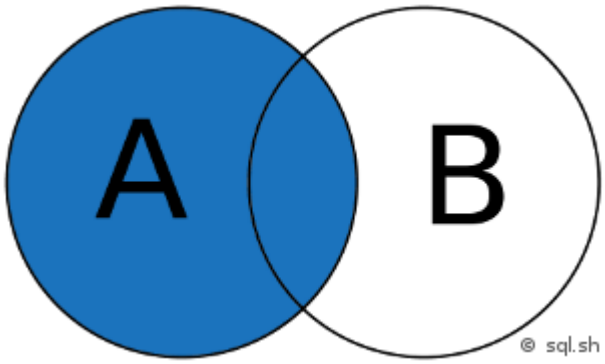
3.1.1 INNER JOIN



Intersection de 2 ensembles.

```
SELECT *
FROM A
INNER JOIN B ON A.key = B.key
```

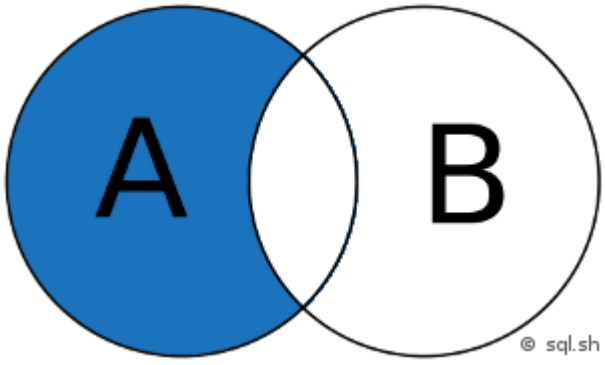
3.1.2 LEFT JOIN



Jointure gauche (LEFT JOIN).

```
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
```

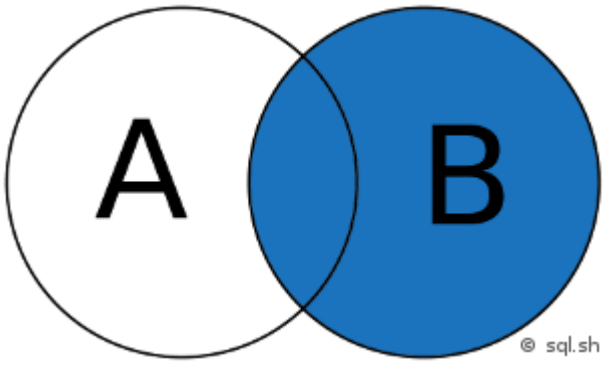
3.1.3 LEFT JOIN (sans l'intersection de B)



Jointure gauche (LEFT JOIN sans l'intersection B).

```
SELECT *
FROM A
LEFT JOIN B ON A.key = B.key
WHERE B.key IS NULL
```

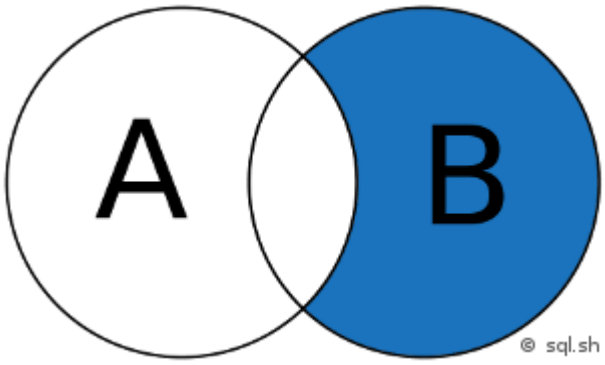
3.1.4 RIGHT JOIN



Jointure droite (RIGHT JOIN).

```
SELECT *
FROM A
RIGHT JOIN B ON A.key = B.key
```

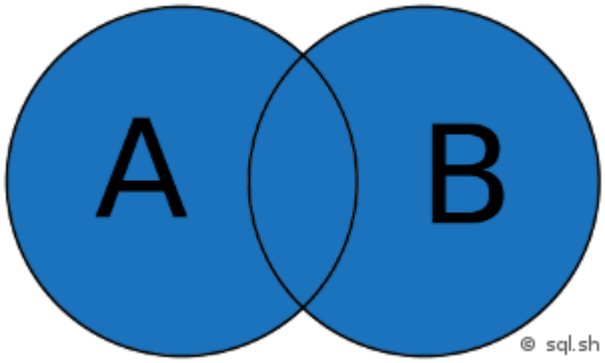
3.1.5 RIGHT JOIN (sans l'intersection de A)



Jointure droite (RIGHT JOIN sans l'intersection A)

```
SELECT *
FROM A
RIGHT JOIN B ON A.key = B.key
WHERE B.key IS NULL
```

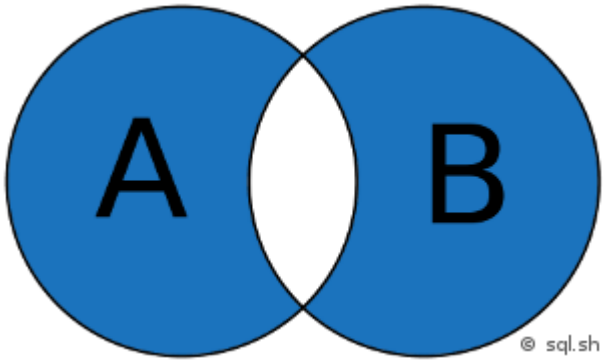
3.1.6 FULL JOIN



Union de 2 ensembles.

```
SELECT *
FROM A
FULL JOIN B ON A.key = B.key
```

3.1.7 FULL JOIN (sans intersection)



Jointure pleine (FULL JOIN sans intersection).

```
SELECT *
FROM A
FULL JOIN B ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```

3.2 Utilisation INNER JOIN \ EXISTS

D'une manière générale, **INNER JOIN** et **EXISTS** sont des choses différentes.

Le premier renvoie les doublons et les colonnes des deux tables, le second renvoie un enregistrement et, étant un prédicat (clause WHERE), renvoie les enregistrements d'une seule table.