Les balises sql de la JSTL

La JSTL (JavaServer Pages Standard Tag Library) fournit un ensemble de balises facilitant la manipulation des bases de données directement dans les pages **JSP**. Pour accéder à une base de données comme **Mysql** ,**PostgreSQL etc**, les balises SQL de la **JSTL** permettent d'exécuter des requêtes SQL sans avoir à utiliser explicitement des classes Java comme **`Connection`**, **`PreparedStatement`**, etc. .

1. Configuration de la connexion à PostgreSQL

Avant d'utiliser les balises JSTL SQL, assurez-vous d'avoir le pilote comme **PostgreSQL** dans le chemin de votre projet pour que JSTL puisse se connecter à la base de données.

2. Les balises principales de JSTL SQL

Les balises SQL de JSTL se trouvent dans le **namespace** `sql` :

`Syntaxe :`

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
```

2.1. `<sql:setDataSource>`

`<sql:setDataSource>` configure la connexion à la base de données. Elle doit être définie avant les autres balises SQL, car elle spécifie les informations nécessaires pour se connecter à PostgreSQL.

Exemple :

```
<sql:setDataSource
    var="db"
    driver="org.postgresql.Driver"
    url="jdbc:postgresql://localhost:5432/nom_de_la_base"
    user="votre_utilisateur"
    password="votre_mot_de_passe"/>
```

Ici :

- **`var`** définit le nom de la source de données.
- **`driver`** spécifie le pilote JDBC PostgreSQL.
- **`url`** fournit l'URL de connexion.
- **`user`** et **`password`** sont vos identifiants de connexion PostgreSQL.

2.2. `<sql:query>`

La balise **`<sql:query>`** exécute des requêtes SQL de sélection (`SELECT`) sur la base de données.

Exemple :

```
<sql:query dataSource="${db}" var="resultat">
    SELECT * FROM utilisateurs;
</sql:query>
```

- **`dataSource`** fait référence à la connexion définie par **`<sql:setDataSource>`.**
- **`var`** stocke le résultat de la requête dans une variable (ici **`resultat`**).

Pour afficher les résultats, utilisez une boucle JSTL comme suit :

```
<table>
    <tr><th>Nom</th><th>Email</th></tr>
    <c:forEach var="row" items="${resultat.rows}">
        <tr>
            <td>${row.nom}</td>
            <td>${row.email}</td>
        </tr>
    </c:forEach>
</table>
```

2.3. `<sql:update>`

`<sql:update>` exécute des requêtes SQL de mise à jour, insertion ou suppression (**`UPDATE`**, **`INSERT`**, **`DELETE`**).

Exemple d'insertion :

```
<sql:update dataSource="${db}" var="insertCount">
    INSERT INTO utilisateurs (nom, email) VALUES ('Toto ',
'toto@example.com');
</sql:update>

Nombre de lignes insérées : ${insertCount}
```

Dans cet exemple :

- **`dataSource`** spécifie la connexion à PostgreSQL.
- **`var`** stocke le nombre de lignes affectées (dans **`insertCount`**).

Exemple de mise à jour :

```
<sql:update dataSource="${db}" var="updateCount">
    UPDATE utilisateurs SET email = 'tata@example.com' WHERE
nom = 'toto';
</sql:update>
Nombre de lignes modifiées : ${updateCount}
```

Exemple de Suppression avec <sql:update>

Supposons que vous souhaitiez supprimer un utilisateur spécifique de la table utilisateurs en utilisant une condition basée sur l'identifiant (id).

```
<sql:setDataSource

    var="db"

driver="org.postgresql.Driver"

    url="jdbc:postgresql://localhost:5432/nom_de_la_base"

    user="votre_utilisateur"

    password="votre_mot_de_passe"/>

<sql:update dataSource="${db}" var="deleteCount">

    DELETE FROM utilisateurs WHERE id = 1;

</sql:update>

Nombre de lignes supprimées : ${deleteCount}
```

Utilisation de <sql:param> pour des Suppressions Dynamique

```
<sql:update dataSource="${db}" var="deleteCount">

    DELETE FROM utilisateurs WHERE id = ?

    <sql:param value="${param.id}"/>

</sql:update>



Nombre de lignes supprimées : ${deleteCount}
```

2.4. `<sql:param>`

`<sql:param>` permet de passer des paramètres de manière sécurisée pour éviter les injections SQL.

Exemple avec paramètres :

```
<sql:query dataSource="${db}" var="result">
    SELECT * FROM utilisateurs WHERE nom = ?
    <sql:param value="${param.nom}"/>
</sql:query>
```

Ici :

- **`?`** dans la requête est remplacée par le paramètre spécifié par **`<sql:param>`.**
- **`value`** est la valeur à passer à la requête. Elle peut provenir d'un formulaire ou d'une autre source.

Exemple complet d’utilisation de JSTL SQL avec PostgreSQL

```
<%@ page language="java" contentType="text/html; charset=UTF-8" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>

<html>
<head>
    <title>Utilisateurs - Liste</title>
</head>
<body>

<!-- Connexion à la base de données -->
<sql:setDataSource var="db" driver="org.postgresql.Driver"
    url="jdbc:postgresql://localhost:5432/nom_de_la_base"
    user="votre_utilisateur" password="votre_mot_de_passe" />

<!-- Exécution d'une requête SELECT -->
<sql:query dataSource="${db}" var="utilisateurs">
    SELECT * FROM utilisateurs;
</sql:query>

<!-- Affichage des résultats -->
<h2>Liste des Utilisateurs</h2>
<table border="1">
    <tr>
        <th>Nom</th>
```

```
        <th>Email</th>
    </tr>
    <c:forEach var="row" items="${utilisateurs.rows}">
        <tr>
            <td>${row.nom}</td>
            <td>${row.email}</td>
        </tr>
    </c:forEach>
</table>

</body>
</html>
```

`<sql:transaction>`

`<sql:transaction>` permet de regrouper plusieurs opérations SQL dans une transaction unique. Cela garantit que toutes les opérations incluses réussissent ou échouent ensemble, ce qui est essentiel pour maintenir l'intégrité des données.

Exemple

Imaginons que vous souhaitiez insérer un utilisateur et mettre à jour une table de journalisation dans la même transaction. Si l'une des opérations échoue, l'autre est annulée.

```
<sql:transaction dataSource="${db}">
    <sql:update var="insertUser">
        INSERT INTO utilisateurs (nom, email) VALUES ('toto',
'toto@example.com');
    </sql:update>

    <sql:update var="insertLog">
        INSERT INTO journal (action, description) VALUES
('Insertion', 'Nouvel utilisateur ajouté');
    </sql:update>
</sql:transaction>
```

Ici :

- Si l'insertion dans **`utilisateurs`** échoue, l'ajout dans `journal` ne sera pas exécuté.
- La transaction garantit que toutes les opérations s’exécutent avec succès ou pas du tout.

À noter :

- Utiliser `<sql:transaction>` est une bonne pratique pour les opérations critiques nécessitant l’intégrité des données (comme des opérations de type `INSERT` et `UPDATE` liées).
- Si une erreur survient dans une balise `sql` incluse, la transaction est automatiquement annulée (**rollback**).

`<sql:dataparam>`

`<sql:dataparam>` est une alternative à **`<sql:param>`** et est principalement utilisée pour transmettre plusieurs paramètres à une balise **`<sql:query>`** ou **`<sql:update>`**.

En fait, elle est moins couramment utilisée que **`<sql:param>`** parce que cette dernière reste plus simple et directe pour passer des paramètres, mais elle est utile lorsqu'on a besoin de spécifier plusieurs paramètres dans un contexte particulier.

Exemple

```
<sql:query dataSource="${db}" var="resultat">
    SELECT * FROM utilisateurs WHERE nom = ? AND email = ?;
    <sql:dataparam value="titi"/>
    <sql:dataparam value="titi@example.com"/>
</sql:query>
```

Dans cet exemple, chaque balise **`<sql:dataparam>`** est utilisée pour passer un paramètre dans la requête. Les paramètres sont évalués et insérés dans l'ordre où ils apparaissent. Les valeurs fournies ici sont `titi` et `titi@example.com`.

À noter :

- `<sql:dataparam>` est utile si les paramètres doivent être définis par avance ou lorsqu'il y a plusieurs paramètres à passer en une seule fois.
- Pour des requêtes nécessitant un nombre limité de paramètres (1-3), `<sql:param>` reste plus courant.

Ces balises augmentent la puissance et la sécurité des interactions SQL avec JSTL, surtout dans des situations où les transactions et les paramètres complexes sont indispensables.

Attention ,n'abusez pas de cette technique :

- **Performances** : JSTL SQL est pratique pour des démonstrations ou petites applications, mais pour des projets plus complexes, il est préférable d'utiliser **JDBC** ou un framework
- **Sécurité** : Utilisez toujours **`<sql:param>`** pour éviter les risques d'injections SQL.
- **Débogage** : Si vous rencontrez des erreurs, vérifiez votre URL JDBC et les permissions utilisateur de PostgreSQL.