Relations et filtres

2019/2020

APERCU

Dans ce TP vous serez amenés à mettre en place des relations entre objets ainsi que des filtres pour les requêtes.

OBJECTIFS

1. Mettre en place des relations
2. Mettre en place des filtres

1. Mise en place de relations

Dans ce TP nous allons mettre en place des requêtes relationnelles entre l'objet **users** et l'objet **data**. Avant de procéder à la suite du TP, veuillez créer une nouvelle table **data** dans votre base de données avec trois colonnes **id** et **user_id** et **value**. Avec **id** un **int** en auto incrémentation, **user_id** une **clef étrangère** de la table **users** et **value** un **varchar** de 255 caractères.

Nous souhaitons donc, accéder aux objets **data** d'un objet **users** grâce à une requête. Il y a deux requêtes possibles: demander tous les objets **data** d'un utilisateur, ou demander un objet **data** spécifique d'un utilisateur. Pour cela avec **NodeJS** il nous faut donc deux routes:

- /users/:id/data
- /users/:id_users/data/:id_data

Ce qui nous donne le code suivant:

```
app.get('/users/:id/data', function(req, res) {
    let id = req.params.id;
    let query = `SELECT data.id, data.value FROM users INNER JOIN data ON
users.id = data.user_id WHERE users.id=${id}`;
    db.query(query, function(err, result, fields) {
        if (err) throw err;

        res.send(JSON.stringify(result));
    });
});
app.get('/users/:id_user/data/:id_data', function(req, res) {
    let id_user = req.params.id_user;
    let id_data = req.params.id_data;
    let query = `SELECT data.id, data.value FROM users INNER JOIN data ON
users.id = data.user_id WHERE users.id=${id_user} AND data.id=${id_data}`;
    db.query(query, function(err, result, fields) {
        if (err) throw err;

        res.send(JSON.stringify(result));
    });
});
```

Nous utilisons ici le mot clef SQL **INNER JOIN** qui nous permet de sélectionner les lignes qui ont des valeurs correspondantes dans les deux tables. Nous utilisons aussi le mot clef **WHERE** afin d'affiner notre requête avec les conditions qui nous intéressent.

2. Mise en place de filtres

Afin de permettre un meilleur tri de nos requêtes, nous allons intégrer des filtres. Les filtres peuvent être classés en plusieurs thèmes:

- Les conditions: elles permettent de chercher les objets dont un champ spécifique contient une valeur choisie.
- L'ordre: il permet d'organiser les objets dans l'ordre choisi.
- Les champs: ils permettent de sélectionner les variables de l'objet.
- La pagination: elle permet de choisir le nombre d'objets ainsi que de les décaler.

Tout d'abord, le tri dans nos requêtes ne se fera pas par une route spécifique mais par les données supplémentaires optionnelles de l'URL. Ces données sont transmises grâce au caractère de séparation **?** après le nom de domaine et la route, puis sont organisées dans une chaîne de requêtes déterminée par une clef et une valeur séparée par le caractère =. Chaque paire de clef/valeur est ensuite séparée par le caractère **&**. Exemple:
http://example.com/route**?key1=value1&key2=value2**
Ici nous avons donc deux clefs **key1** et **key2** avec respectivement les valeurs **value1** et **value2**.

Avec **NodeJS**, on peut accéder à ces valeurs avec l'objet **req.query**:

```
app.get('/query', function(req, res) {
    res.send(JSON.stringify(req.query));
});
```

Maintenant, procédons au traitement de ces données supplémentaires.

2.1 Filtre par condition

```
app.get('/data', function(req, res) {
    let query = "SELECT * FROM data";
    let conditions = ["user_id", "value"];

    for (let index in conditions) {
        if (conditions[index] in req.query) {
            if (query.indexOf("WHERE") < 0) {
                query += " WHERE";
            } else {
                query += " AND";
            }

            query +=
`${conditions[index]}='${req.query[conditions[index]]}';
        }
    }

    db.query(query, function(err, result, fields) {
        if (err) throw err;

        res.send(JSON.stringify(result));
    });
});
```

Ici la requête de base correspond à la recherche de toutes les lignes de la table **data**. Ensuite la variable **conditions** est utilisée afin de spécifier les champs possibles à filtrer. Pour chacun des champs spécifiés, si ce champ est contenu dans les données supplémentaires optionnelles de la requête, alors on l'ajoute dans la requête SQL. Mais avant, si le mot-clef **WHERE** n'est pas dans la requête, alors on le rajoute.

2.2 Filtre d'ordre

```
app.get('/data', function(req, res) {
    let query = "SELECT * FROM data";

    if ("sort" in req.query) {
        let sort = req.query["sort"].split(",");
        query += " ORDER BY";

        for (let index in sort) {
            let direction = sort[index].substr(0, 1);
            let field = sort[index].substr(1);

            query += ` ${field}`;

            if (direction == "-")
                query += " DESC,";
            else
                query += " ASC,";
        }

        query = query.slice(0, -1);
    }

    db.query(query, function(err, result, fields) {
        if (err) throw err;

        res.send(JSON.stringify(result));
    });
});
```

Ici la requête de base correspond à la recherche de toutes les lignes de la table **data**. Si l'option **sort** est bien dans les paramètres de la requête, alors les champs en sont extraits et ajoutés dans la requête grâce à la boucle. Le dernier caractère de **query** est supprimé à cause de la virgule.

Remarque: On ne teste pas si la variable **direction** vaut + car ce caractère symbolise un espace dans un URL.

2.3 Filtre des champs

```
app.get('/data', function(req, res) {
    let query = "SELECT * FROM data";

    if ("fields" in req.query) {
        query = query.replace("*", req.query["fields"]);
    }

    db.query(query, function(err, result, fields) {
        if (err) throw err;

        res.send(JSON.stringify(result));
    });
});
```

Ici la requête de base correspond à la recherche de toutes les lignes de la table **data**. Si l'option **fields** est bien dans les paramètres de la requête, alors les champs sont ajoutés directement dans la requête.

2.4 Filtre par pagination

```
app.get('/data', function(req, res) {
    let query = "SELECT * FROM data";

    if ("limit" in req.query) {
        query += ` LIMIT ${req.query['limit']}`;

        if ("offset" in req.query) {
            query += ` OFFSET ${req.query['offset']}`;
        }
    }

    db.query(query, function(err, result, fields) {
        if (err) throw err;

        res.send(JSON.stringify(result));
    });
});
```

Ici la requête de base correspond à la recherche de toutes les lignes de la table **data**. Si l'option **limit** est bien dans les paramètres de la requête, alors la limite est appliquée dans la requête. Si l'option **offset** est aussi utilisée alors un décalage est appliqué à la requête.

Remarque: En SQL, le mot-clef **OFFSET** est uniquement utilisable si le mot-clef **LIMIT** est utilisé, c'est pour cela que la condition pour le paramètre **offset** est englobé dans la condition du paramètre **limit**.