HINTS

CONSULTA PARAMETRIZADA DE NODE POSTGRES

Si está creando aplicaciones que incluyen consultas SQL, hay algunas razones convincentes para usar consultas parametrizadas en lugar de consultas ad-hoc en su código. Las consultas parametrizadas no solo reducen el riesgo de ataques de inyección SQL, sino que también mejoran el rendimiento porque no es necesario prepararlas cada vez que se ejecutan. En este artículo, vimos un ejemplo usando Node con una consulta parametrizada de Postgres. Con este ejemplo como guía, podrá implementar consultas parametrizadas en sus propias aplicaciones de Nodo.

Por ejemplo:

```
const pg = require("pg");
const cs = "postgres://postgres:1234@localhost:5432/some_db";
const client = new pg.Client(cs);

client.connect();

const sql = "SELECT * FROM employee WHERE salary > $1";
const values = [55000];
console.log(`SQL statement #1: ${sql}`);

client.query(sql, values).then(res => {
  const data = res.rows;
  data.forEach(row => console.log(row));
});

const sql = "SELECT * FROM employee WHERE salary > $1 AND id ==
$2";
const values = [55000, 1];
console.log(`\nSQL statement #2: ${sql}`);

client.query(sql, values).then(res => {
  const data = res.rows;
  data.forEach(row => console.log(row));
});
```

PostgreSQL no admite parámetros para identificadores. Si necesita tener nombres dinámicos de bases de datos, esquemas, tablas o columnas (por ejemplo, en declaraciones DDL), use el paquete **pg-format** para manejar estos valores y asegurarse de que no tenga una inyección de SQL.

MEDIDAS PREVENTIVAS PARA ATAQUES DE INYECCIÓN SQL

Primero, debemos abordar la validación de la entrada del usuario implementada en nuestro código de interfaz de usuario. Esta validación sería nuestra primera capa de defensa contra los malos actores y serviría como un mecanismo de interacción receptivo para los usuarios que luchan con la intuición de la interfaz.

Asegúrese de que los valores proporcionados por el usuario estén delimitados y desinfectados para cada campo en consecuencia. Eso significa, por ejemplo, que, si un campo de entrada está destinado a recibir correos electrónicos, no permite que el usuario envíe el formulario si se establece una dirección de correo electrónico no válida o ningún valor.

En segundo lugar, la validación de entrada se puede implementar en el nivel de control, donde se realizan la mayoría de los cálculos. Esta estrategia podría ser tan simple como revalidar y, cuando corresponda, desinfectar la entrada del usuario antes de que llegue a la capa del modelo. Además, también ayuda agregar una biblioteca de terceros como "node-postgres" que implementa el escape automáticamente.

PROTECCIÓN DE LA CAPA DE DATOS

Una vez que se solucionan los problemas en el nivel superior, podemos proteger la capa de la base de datos. Para hacer eso, todo lo que necesitamos hacer es implementar lo que se conoce como marcadores de posición de consulta o marcadores de posición de nombre. Estos marcadores de posición, indicados con el símbolo “?”, dígale a la capa de interfaz que escape automáticamente de la entrada que se le pasó antes de que se inserte en la consulta.