TEXT CLASS REVIEW

TEMAS A TRATAR EN EL CUE:

- Lenguaje SQL (Structured Query Language).
- Procesamiento de la respuesta.
- Captura de errores en consultas.
- Manejo de cursores en Node.

PROCESAMIENTO DE LA RESPUESTA

El objeto result()

Éste devuelve el objeto para cada consulta exitosa.

Propiedades:

result.rows: Array<any>

Cada resultado tendrá una matriz de filas. Si éstas no se devuelven, la matriz estará vacía; de lo contrario, contendrá un elemento por cada fila devuelta por la consulta. Por defecto, node-postgres crea un mapa desde el nombre hasta el valor de cada columna, devolviéndole un objeto similar a json para cada fila.

result.fields: Array<FieldInfo>

Cada resultado tendrá una matriz de campos. Ésta contiene el nombre, y el ID del tipo de datos de cada campo en el resultado. Dichos campos se ordenan en el mismo orden que las columnas, si se está utilizando arrayMode para la consulta.

Ejemplo:

```
const { Pool } = require('pg')
const pool = new Pool()
const client = await pool.connect()
const result = await client.query({
  rowMode: 'array',
  text: 'SELECT 1 as one, 2 as two;',
```

```
})
console.log(result.fields[0].name) // one
console.log(result.fields[1].name) // two
console.log(result.rows) // [ [ 1, 2 ] ]
await client.end()
```

result.command: string

El último tipo de comando ejecutado: INSERT UPDATE CREATE SELECT, entre otros.

result.rowCount: int

El número de filas procesadas por el último comando.

No solo refleja el número de filas devueltas de una consulta. Por ejemplo: una declaración de actualización podría actualizar muchas filas (un valor tan alto de result.rowCount), pero result.rows.length sería cero. Para verificar una respuesta de consulta vacía en una consulta SELECT, use result.rows.length === 0.

CAPTURA DE ERRORES EN CONSULTAS

Para la captura de los mensajes emitidos por el servidor PostgreSQL, se asignan códigos de error de cinco caracteres, los cuales siguen las convenciones del estándar SQL para códigos "SQLSTATE". Las aplicaciones que necesitan saber qué condición de error se ha producido, generalmente deberían probar el código de error, en lugar de mirar el mensaje de error textual. Por ejemplo, alguna lista de códigos de Violación de restricción de integridad:

Class 23 — Integrity Constraint Violation	
23000	integrity_constraint_violation
23001	restrict_violation
23502	not_null_violation

23503	foreign_key_violation
23505	unique_violation
23514	check_violation
23P01	exclusion_violation

La lista completa se puede obtener aquí.

Por ejemplo, si queremos detectar errores causados por violaciones de restricciones únicas, en un método de inserción de datos, podemos adecuar una función de la siguiente manera:

```
function createPerson(body, callBack){
  // Esta función agrega una persona que se registra en la base de
datos.

var id;
var sql = 'INSERT INTO member (name, phone, email ) VALUES ($1, $2,
$3) RETURNING id;';
db.query(sql, [body.name, body.phone, body.email]).then(res => {
      id = res.rows[0].id;
      if (id) {
        callBack(body);
        console.log("Nuevo miembro con id:  " + id);
      }
  }).catch(e => {
      if (e.code == '23505'){
        console.log("\n ERROR! \n persona con nombre: " + body.name +
" " + " y telefono #: " + body.phone + " es un miembro duplicado.
\n");
        callBack("Duplicado");
        return;
      }
                console.log("\n ERROR! \n persona con nombre: " +
body.name + " " + " y telefono #: " + body.phone      + " no se puede
agregar. \n", e);
      callBack(false);
      return e;
  })
}
```

Partiendo de la lista de errores y códigos en Postgresql, que fueron encontrados en el enlace anterior, observamos que obtenemos un error cuando hay una violación de clave única, con el código "23505".

En este caso, agregamos en la marca de bloque catch para capturar si el error tiene un código de "23505".

MANEJO DE CURSORES EN NODE

¿Qué es un cursor?

Estos se pueden usar para leer de manera eficiente grandes conjuntos de resultados, sin cargarlo todo en la memoria antes de tiempo. Son útiles para simular una lectura de datos de estilo de "transmisión", o salir antes de un gran conjunto de resultados. El cursor se pasa a client.query, y se envía internamente de una manera muy similar a cómo se envían las consultas normales, pero en cambio, la API que presenta para consumir el conjunto de resultados es diferente.

Instalación de paquete

```
$ npm install pg pg-cursor
```

Constructor

```
new Cursor(text: String, values: Any[][, config: CursorQueryConfig])
```

La Instancia de un nuevo Cursor, es una de Submittable, y debe pasarse directamente al método client.query.

```
const { Pool } = require('pg')
const Cursor = require('pg-cursor')
const pool = new Pool()
const client = await pool.connect()
const text = 'SELECT * FROM my_large_table WHERE something > $1'
const values = [10]
const cursor = client.query(new Cursor(text, values))
cursor.read(100, (err, rows) => {
  cursor.close(() => {
    client.release()
  })
```

```
})


interface CursorQueryConfig {
  //  por defecto, las filas aparecen como un par clave/valor para
cada fila
  // pase la cadena 'matriz' aquí para recibir filas como una matriz
de valores
  rowMode?: string;
  // analizadores de tipos personalizados solo para este resultado de
consulta
  types?: Types;
}
```

LECTURA E ITERACIÓN SOBRE CURSORES

```
cursor.read(rowCount: Number, callback: (err: Error, rows: Row[],
result: pg.Result) => void) => void
```

Leer rowCount de filas de la instancia del cursor. Devuelve el callback, y es llamado cuando las filas estén disponibles, cargadas en la memoria, analizadas, y convertidas a tipos de JavaScript.

Si el cursor ha leído hasta el final de los conjuntos de resultados, todas las llamadas posteriores a cursor#read devolverán una matriz de filas de longitud 0.

Llamar a read en un cursor que ha leído hasta el final. Por ejemplo:

```
const { Pool } = require('pg')
const Cursor = require('pg-cursor')
const pool = new Pool()
const client = await pool.connect()
const cursor = client.query(new Cursor('select * from
generate_series(0, 5)'))
cursor.read(100, (err, rows) => {
  if (err) {
    throw err
  }
  assert(rows.length == 6)
  cursor.read(100, (err, rows) => {
    assert(rows.length == 0)
  })
})
```

CIERRE DE CURSORES

```
cursor.close(callback: () => void) => void
```

Se utiliza para cerrar el cursor antes de tiempo. Si se desea dejar de leer desde el cursor, antes de que se devuelvan todas las filas, llame a this.